The Network Testbed Mapping Problem

Rick McGeer[1], David G. Andersen[2], and Stephen Schwab[3]

[1] Hewlett-Packard Laboratories, Palo Alto, CA,
rick.mcgeer@hp.com,
[2] Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA,
dga@cs.cmu.edu,
[3] Cobham Analytic Solutions, Columbia, MD,
Stephen.Schwab@cobham.com

Abstract. The *Network Testbed Mapping Problem* is the problem of mapping an emulated network into a test cluster such as Emulab or DETER. In this paper, we demonstrate that the Network Testbed Mapping Problem is $\mathcal{NP}$-complete when there is constrained bandwidth between cluster switches. We demonstrate that the problem is trivial when bandwidth is unconstrained, and note that a number of new proposals for data center networking have removed this barrier. Finally, we consider new heuristics in the bandwidth-limited case.

Key words: network emulation, network embedding, graph partitioning

1 Introduction

The *Network Testbed Mapping Problem* is the fundamental combinatiorial problem faced by network test environments such as Emulab [10] and DETER [1]. This is the problem of embedding instances of an emulated or virtual test network, which consist of a collection of nodes, links, and LANs, onto a physical cluster using virtual LANS so that the inter-node bandwidth requirements of the test network is satisfied. This problem is thought to be difficult, since inter-node bandwidth in a typical cluster often depends on the relative placement of the nodes within the cluster. Nodes which are placed on the same switch have sufficient bandwidth with adequately provisioned switches; however, since switch-switch bandwidth is typically much less than the aggregate bandwidth each switch allocates to its attached nodes, nodes attached to different switches in the data center have limited inter-node bandwidth.

For this reason, network testbed mapping is typically solved by heuristic methods. For example, Emulab and DETER use simulated annealing, a proba-

* This research partially supported by the Defense Advanced Reseach Projects Agency under DARPA Subcontract 09-2080 for DARPA National Cyber Range Phase I - Prime Contract HR0011- 09-C-0039. Approved for Public Release, Distribution Unlimited. The views, opinions, and/or findings contained in this article are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.

bilistic hill-climbing algorithm originally used for VLSI placement in the 1980's. Simulated annealing, described below, works by starting from some solution and attempting to find a better, lower cost, solution. In this, it is similar to a large number of perturbative or progressive-improvement solutions. It differs from many in that it permits "hill-climbing" perturbations which temporarily lead to a worse solution but ultimately yield a superior solution. The intuition is that pure improvement perturbative methods such as the Greedy method find only local optima; permitting occasional "upward" moves which find a poorer solution offers the prospect of finding a better global solution than one can find from pure improvement methods.

The attraction of perturbative methods in general is that a perturbative algorithm works on *any* combinatiorial optimization problem. All one needs is a metric to measure the quality of a solution, an initial solution, and an ablity to move from one solution to another; given these three elements, the algorithm iterates repeatedly and eventually produces a better solution.

For example, in the case of the network testbed mapping problem, the initial solution is an assignment of nodes to switches; the metric is the deficit in actual vs. desired inter-node bandwidth; and a move is the exchange of pairs of nodes or the reassignment of a single node to a switch.

Given how simple perturbative methods are to use, one might wonder why they are not universally used. The answer is that the generality of perturbative methods means that they can only give fairly weak guarantees of solution quality. Despite the best efforts of many theorists, simulated annealing only gives stochastic assurances of quality, and the characteristics of the solution space on which simulated annealing works well is largely conjectural. Sorkin [14] persuasively argued that simulated annealing was likely to do well on self-similar solution spaces, but one cannot *a priori* determine which problems or instances have self-similar solution spaces.

Furthermore, it is easy to demonstrate hard problems for which simulated annealing will not do well. Consider, for example, an instance of SAT[6] with relatively few satisfying assignments. Starting from a random assignment, the odds that simulated annealing's random exploration of the solution space will find a satisfying assignment in reasonable time is vanishing; and in the case of an unsatisfiable instance, simulated annealing will prove nothing until it has explored the entire space.

Simulated annealing has proven to be succesful in testbeds at the scale of Emulab and DETER[12]. However, recent data suggests that it may not scale well in significantly larger settings [8][3]. Other interesting approaches have been proposed in the recent literature, notably [11], who exploited the fact that in many network graphs isomorphic subgraphs are detected. [16] broke the ground of observing that modification of the underlying substrate network can make the problem more tractable. In this paper, we show that some classes of underlying network substrate make this problem easy.

Given that the next generation of testbeds is projected to handle emulated test networks more than an order of magnitude larger than Emulab or DETER,

this problem is worth revisiting. In particular, we would like to investigate the following questions:

1. When *is Network Testbed Mapping $\mathcal{NP}$-hard*? It is widely believed to be hard, but to date the reductions appearing the literature have been sketches in papers devoted to heuristic development. While this is quite common, it doesn't tell us precisely *why* NTM is hard, which often illustrates where and when it can be solved.
2. Though Network Testbed Mapping is $\mathcal{NP}$-hard, are there any circumstances under which it is easy? Can we design and engineer our range cluster networks to make it easy?
3. Though Network Testbed Mapping is $\mathcal{NP}$-hard, is it amenable to non-perturbative methods which can give stronger guarantees of the relative quality of solutions as compared to simulated annealing? For example, some variants of VLSI placement can be solved by the *Linear Programming with Randomized Rounding*[15] technique, which can give strong probabilistic guarantees. While approximation and probabilistic techniques typically are not conserved across polynmial reductions (or else every problem in $\mathcal{NP}$ would be easy to approximate, a strong and manifestly false result), often the nature of a reduction may provide a heuristic guide to good approximation and probabilistic techniques.

In this paper we consider these questions.

2 The Network Testbed Mapping Problem

The formal network testbed mapping problem is a simplification of the actual mapping problems solved within real testbeds. We assume, e.g., that all nodes are homogenous which means we can freely assign nodes to any switch. We ignore details of the interconnect fabric between the physical switches, assuming that each switch has a dedicated connection of some bandwidth to every other switch. These two assumptions simplify the problem to enable us to prove theorems about it, without assuming the problem away.

For example, one can easily capture heterogeneity of nodes by making the capacity (maximum port count) of each switch a vector, as opposed to scalar, quantity. The independence of inter-switch bandwidth is certainly a simplifying assumption, but for the purposes of complexity results it is unnecessary to introduce a more sophisticated model in the problem description – if this formulation of the problem is hard, then the general problem will be as well. Furthermore, the simplified problem generalizes very naturally to the general problem, as will be seen below.

Problem 1 The Network Testbed Mapping Problem. Given: *A network of switches, $s_1, ..., s_n$ with (port) capacities $C_1, ..., C_n$ and interswitch bandwidth capacities $B_{11}, ...B_{1n}, B_{21}, ..., B_{nn}$, and a test network of nodes $N_1, ..., N_m$ with internode bandwidth requirements $b_{11}...b_{mm}$.* Question: *is there an injective assignment $\mathcal{A} : N \rightarrow s$ such that:*

$$|\mathcal{A}(u) = i| \leq C_i \ \forall i, 1 \leq i \leq n \tag{1}$$

and:

$$\sum_{\mathcal{A}(u)=i, \mathcal{A}(v)=j} b_{uv} \leq B_{ij} \ \forall i, j \tag{2}$$

(Where the summation is taken over all $\mathcal{A}(u), \mathcal{A}(v)$ satisfying the equalities.) We say any mapping that satisfies (1) and (2) is feasible.

The *capacity* of a switch is the number of edge nodes to which it can connect; the *bandwidth capacity* of a pair of switches is the total bandwidth between them. The two conditions on the problem are therefore that the assignment function not assign too many nodes to any switch, and the total assigned bandwidth between any pair of switches doesn't exceed the available bandwidth between that pair of switches. (We note that this formulation of the problem identifies only one edge link per edge node. In practice, a real test cluster would support n edge links per edge node. If we assume that the physical test cluster wiring connects all links from an edge node to a single switch, then the problem is unchanged. The formulation generalizes easily by permitting $\mathcal{A}(u)$ to be a fixed sized set for each u, where $|\mathcal{A}(u)|$ is the number of outgoing connections from u. The bandwidth constraint 2 becomes slightly messier, since there are now multiple paths between u and v. In practice, one cannot discuss this sensibly without some knowledge of the routing discipline used in the multiple path case. If routing between terminals is deterministic, as it usually is, then one can treat each NIC of each node as a separate node).

Remark 1 *The definition of network testbed mapping can extend to the conventional tree-of-switches network in a data center. Each switch in a tree-of-switches network has an upward bandwidth capacity U_i, a downward bandwidth capacity D_i, a set of descendant switches c_i, and a set of switches r_i for which it is the least common ancestor. The second constraint equation (2) in NTM is replaced by a pair of constraints, where the upward capacity must exceed the total bandwidth exiting the tree rooted at this switch. Formally, let*

$$u_i = \sum_{\mathcal{A}(u) \in c_i, \mathcal{A}(v) \notin c_i} b_{uv}$$

Then: $U_i \geq u_i$. Similarly, the downward capacity must exceed the total bandwidth exiting the sub-tree + the total bandwidth passing through this switch as the root switch. Formally, let

$$d_i = \sum_{\mathcal{A}(u) \in r_i, \mathcal{A}(v) \in r_i} b_{uv}$$

then

$$D_i \geq u_i + d_i$$

where u_i is defined as above.

Observation 1 *NTM is in $\mathcal{NP}$*

Proof. Given an assignment $\mathcal{A}$, conditions (1) and (2) are checked in linear time straightforwardly.

We will now demonstrate that NTM is $\mathcal{NP}$-hard, by reducing the known $\mathcal{NP}$-hard problem Minimum Graph Bisection.

Problem 2 *Minimum Graph Bisection.* Given: *an unweighted, directed graph* $G(V, E)$*, integer* K. Question: *is there a partition of* V *into sets* V_1, V_2*,* $||V_1| - |V_2|| \leq 1$*, such that:*

$$|(v_1, v_2) \in E | s.t. v_1 \in V_1, v_2 \in V_2 \leq K$$

See, for example, http://tracer.lcc.uma.es/problems/bisect/bisect.htm.

Theorem 1 *Network Testbed Mapping is* $\mathcal{NP}$*-complete.*

Proof. We reduce Mimimum Graph Bisection. Given an instance $G(V, E)$, integer K of Minimum Graph Bisection, derive an instance of Network Testbed Mapping as follows. We define two switches, s_1 and s_2, with capacities $C_1 = \lfloor(|V|/2)\rfloor, C_2 = \lceil(|V|/2)\rceil$, and interswitch bandwidth capacities $B_{12} = B_{21} = K$. Our test network of nodes is $N_1, ..., N_{|V|}$ (one node per graph node), with $b_{uv} = 1$ if (u, v) is an edge in E, $b_{uv} = 0$ otherwise.

Plainly, the derivation of an instance of Network Testbed Mapping is linear. Let $\mathcal{A}$ be the assignment which solves the Network Testbed Mapping instance. Let $V_1 = \{u | \mathcal{A}(u) = 1\}, V_2 = \{v | \mathcal{A}(v) = 2\}$. V_1 and V_2 are a bisection of V, and V_1 and V_2 are derived in linear time from $\mathcal{A}$. Further,

$$\sum_{\mathcal{A}(u)=1, \mathcal{A}(v)=2} b_{uv} \leq K$$

since $\mathcal{A}$ is a solution to the Network Testbed Assignment problem, we have:

$$|(u, v) | u \in V_1, v \in V_2 \leq K$$

so (V_1, V_2) is a solution to the Minimum Graph Bisection instance

This suffices to show that the Network Testbed Mapping is $\mathcal{NP}$-complete. In the next section, we will consider special cases where the problem is easy, and following that we will consider approximation and heuristic algorithms for the general case.

3 Polynomial-Time Special Cases

The reduction in the previous section permitted switches of arbitrary capacity and outgoing bandwidth. In practice, of course, switches have finite capacity and interswitch bandwidth is largely a function of the network topology. Further, for many network topologies, such as the common tree-of-switches fabric we discussed earlier, interswitch bandwidth is not independent but competitive.

Some network topologies, however, permit easy solution of the Network Testbed Assignment problem. We characterize this set here.

In this discussion, we will assume network fabrics constructed from homogeneous switches with a specific bandwidth capacity per port, and the same port bandwidth is used on the network interface cards (NICs) of the various nodes. This simplification, and appropriate choice of units permits us to take B_{ij} as a non-negative integer and $0 \leq b_{uv} \leq 1$, and to take a node as equivalent to a unit of bandwidth. This has no substantive effect on the results below, but does permit us to state the results clearly and without introducing spurious constants.

Definition 1 *A network fabric is said to be* bandwidth-unconstrained *if and only if:*

1. *Inter-switch bandwidth capacities are independent; assigning bandwidth between one pair of switches doesn't affect available bandwidth between a different pair*
2. *For each pair of switches i, j:*

$$B_{ij} \geq \max(C_i, C_j)$$

where B_{ij} and C_i are taken from the definition of the Network Testbed Mapping problem.

Several scalable network fabrics have been proposed in the literature recently, notably the data center networks proposed by Al-Fares et al. [2] and by Scott et al. [13]. These conditions make network testbed mapping trivial.

Theorem 2 *Consider* any *Network Testbed Mapping problem where the fabric is bandwidth unconstrained, in particular $B_{ij} \geq \max(C_i, C_j)$ for all i, j.* Every *assignment $\mathcal{A} : N \to S$ such that $|\mathcal{A}(u) = i| \leq C_i \ \forall i$ is feasible.*

Proof. The constraint $|\mathcal{A}(u) = i| \leq C_i$ merely says that we can't assign more nodes to a switch than it can take, so consider *any* assignment $\mathcal{A}$ that meets this constraint. We must show for any pair of switches i, j, that:

$$\sum_{\mathcal{A}(u)=i, \mathcal{A}(v)=j} b_{uv} \leq B_{ij}$$

But this is trivial. Under our notational convention, $0 \leq b_{uv} \leq 1$ for all u, v, so:

$$\sum_{\mathcal{A}(u)=i, \mathcal{A}(v)=j} b_{uv} \leq |\mathcal{A}(u) = i, \mathcal{A}(v) = j|$$

and the constraint $|\mathcal{A}(u) = i| \leq C_i \ \forall i$ implies $|\mathcal{A}(u) = i, \mathcal{A}(v) = j| \leq \max(C_i, C_j)$, so:

$$\sum_{\mathcal{A}(u)=i, \mathcal{A}(v)=j} b_{uv} \leq |\mathcal{A}(u) = i, \mathcal{A}(v) = j| \leq \max(C_i, C_j)$$

and the fact that the bandwidth is unconstrained ensures:

$$\max(C_i, C_j) \leq B_{ij}$$

This theorem strongly motivates the use of scalable network architectures as building blocks for network testbeds.

4 Heuristic Approaches

The premise of Theorem 2 is in fact stronger than necessary. It is simply a premise that can be stated entirely in terms of cluster topology, independent of the details of the test network to be embedded. One can achieve the same objective by significantly underpromising bandwidth to embedded nodes. In particular, if

$$\max_{u,v} b_{uv} \leq \min_{i,j} \frac{B_{ij}}{\max(C_i, C_j)} \tag{3}$$

then any assignment is feasible.

The proof is quite similar to the proof of Theorem 2. In fact Theorem 2 is the special case of (3) where:

$$\max_{u,v} b_{uv} \leq 1 \leq \min_{i,j} \frac{B_{ij}}{\max(C_i, C_j)}$$

Since by convention

$$\max_{u,v} b_{uv} \leq 1$$

and the premise of Theorem 2 is

$$1 \leq \min_{i,j} \frac{B_{ij}}{\max(C_i, C_j)}$$

These two inequalities are sufficient to maintain the general inequality needed for the theorem, but not necessary. Indeed, one can ensure the general invariant by a variety of means. The general invariant is a restriction of bandwidth demands of the test network relative to the bandwidth capacity of the cluster. For example, the DieCast system of Gupta et al.[7] articially enhances the relative capacity of a switch infrastructure to a test network by slowing down the test network's system clocks.

5 Network Testbed Mapping on a Leaf DAG

An interesting topology for network testbed mapping is a variant of a tree called a *leaf Directed Acyclic Graph*, or *leaf DAG*. A leaf DAG is a multi-rooted directed acyclic graph where internal nodes have a single parent, but leaves are permitted to have multiple parents. This specific class of topologies is often chosen for data center networks, because it permits alternate paths from the leaves while retaining the autoconfiguration properties of standard Ethernet networks. The Emulab topology, for example, is a leaf DAG [4]. It is reproduced in Figure 1.

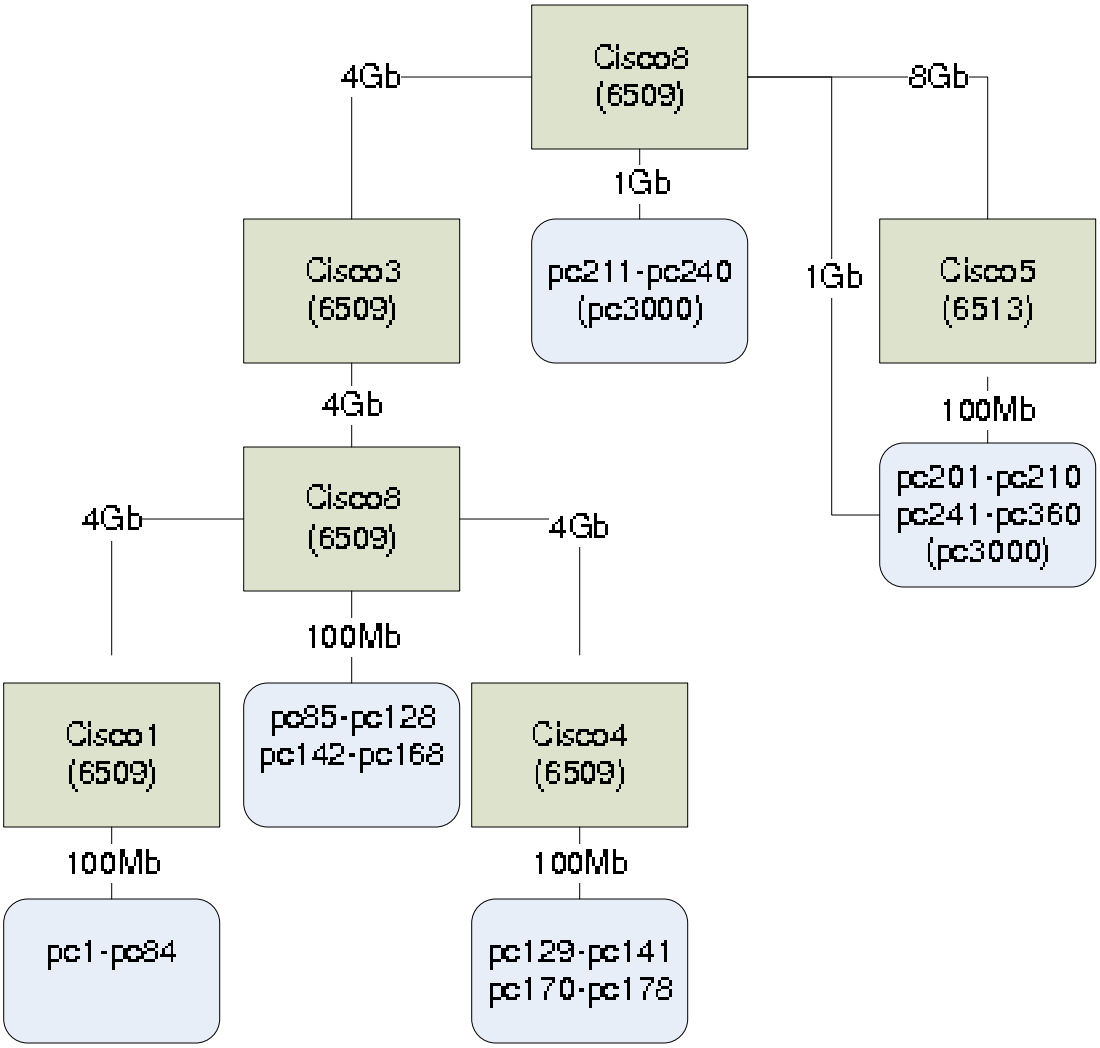

Fig. 1. Emulab Topology

The switches in this figure form a tree of depth three; Cisco8 is the root switch, with Cisco3 and Cisco5 as its (switch) children. Cisco1 and Cisco4 are children of Cisco3. Each switch has leaves attached. The network is a leaf DAG because pc201-210 and pc241-360 are multihomed, attached to both Cisco8 and Cisco5.

The reduction given above suggests a strong similarity between the network testbed mapping problem and the graph partitioning problem. The latter problem has been studied extensively, particularly in the field of VLSI placement. The first procedure to be suggested was the Kernighan-Lin procedure [9]. This procedure bipartitioned a graph to minimize the weight of the edges crossing between the two partitions. It is shown in Figure 2. This procedure requires the

computation of the *gain* to be obtained by interchanging a pair of nodes v_1, v_2 where $v_1 \in V_1$ and $v_2 \in V_2$. The gain is simply the change in the sum of the weights of the edges crossing the partition. An invariant of this procedure is that each node can be moved at most once, and hence we must differentiate between those nodes that have been moved and those that have not. Hence there are four sets of nodes in this procedure; V_1, the set of unmoved nodes on the left side of the partition, U, the nodes on the left side of the partition that were moved from the right, and the corresponding sets V_2 and V on the right side of the partition. So the gain from exchanging v_1 and v_2 is the weight of the edges that crossed the partition before the exchange minus the edges that crossed the partition after the exchange. The sum before exchange is just:

$$\sum_{v \in V_2 \bigcup V} w(v_1, v) + \sum_{u \in V_1 \bigcup U} w(v_2, u) - w(v_1, v_2)$$

with the subtraction of $w(v_1, v_2)$ to eliminate double-counting. Similarly, the sum after exchange is just:

$$\sum_{v \in V_2 \bigcup V} w(v_2, v) + \sum_{u \in V_1 \bigcup U} w(v_1, u) - w(v_1, v_2)$$

The total gain, G_{v_1,v_2}, is just the difference:

$$G_{v_1,v_2} = \sum_{v \in V_2 \bigcup V} (w(v_1, v) - w(v_2, v)) + \sum_{u \in V_1 \bigcup U} (w(v_2, u) - w(v_1, u)) \quad (4)$$

The Kernighan-Lin procedure has been well-studied. Its cost is dominated by the need to recompute the potential gain at each step. Assuming fixed, low degree for each node in the graph, computing the gain for each pair is a constant-time operation. Since there are $O(n^2)$ pairs, where n is the number of nodes in the graph, each iteration takes time $O(n^2)$. Each iteration reduces the size of V_1 and V_2 by one node each; there are $O(n/2)$ nodes initially in each partition, the algorithm is $O(n^3)$.

The algorithm in Figure 2 may be iterated so long as cost improves; in this case the global runtime is $O(cn^3)$, where c is the initial partition cost.

In 1982, Fidducia and Mattheyses improved the performance of the Kernighan-Lin procedure to linear time[5]. Fidducia and Mattheyses made two key innovations to improve the performance of Kernighan-Lin. First, Fiduccia-Mattheyses does not *exchange* nodes as Kernighan-Lin does, but simply moves individual nodes from one side to another. The sides alternated to keep the partition balanced. Second, the gain for moving each node is precomputed, and nodes are stored in an array *indexed by the gain for moving the node*. This permits efficient selection of the best node to move. The valid indexes of the array are $-dw, ..., dw$, where d is the maximum degree of a node and w is the maximum edge weight. In the case of VLSI design (specifically, VLSI placement), which inspired the Fiduccia-Mattheyses procedure, typically $w = 1$ and d was guaranteed small due to physical considerations.

```
KernighanLin(G, V1, V2):
    U = V = ∅
    foreach pair v1, v2, v1 ∈ V1, v2 ∈ V2
        gain(v1, v2) = G_{v1,v2}
    bestCost = Σ_{v1∈V1, v2∈V2} w(v1, v2)
    bestPartition = V1, V2
    currentCost = bestCost
    while V1 ≠ ∅ and V2 ≠ ∅
        choose v1 ∈ V1, v2 ∈ V2 such that gain(v1, v2) is maximized
        V1 = V1 − v1, V2 = V2 − v2
        U = U + {v2}, V = V + {v1}
        currentCost = currentCost - gain(v1, v2)
        if currentCost < bestCost
            bestPartition = (V1 ⋃ U, V2 ⋃ V)
            bestCost = currentCost
        foreach pair v1, v2, v1 ∈ V1, v2 ∈ V2
            gain(v1, v2) = G_{v1,v2}
    return bestPartition
```

Fig. 2. Kernighan-Lin Algorithm

When nodes are moved, only the gain of neighbors need to be updated, and the neighbors themselves need to be moved to the appropriate list. Assuming bounded degree, gain updates can be computed in constant time. Careful attention to data structures permits the move operation to be done in constant time. The procedure is shown in Figure 3.

As with Kernighan-Lin, the Fiduccia-Mattheyses procedure can be iterated while it continues to improve cost. The total complexity for the original procedure is bilinear in the graph size and the initial cost.

We can adapt the Fidducia-Mattheyses procedure to the network testbed mapping problem. Two significant modifications must be made to the algorithm:

1. The original procedure assumed that both the degree of each node and the weight of each edge were bounded by small constants. The former assumption holds in the network testbed mapping problem (if we treat a LAN as a single large node); the latter does not. In most network testbed mapping instances, internode bandwidth can be specified to any value up to one gigabit a second. Even if we make the simplifying assumption that bandwidth is only specified in units of a megabit a second, that still gives us a `gain` array on the order of several thousand entries, almost all empty, and this size cannot be neglected in considering the complexity of the algorithm
2. The Fidducia-Mattheyses procedure assumed that each node must reside on only one side of the partition. For leaf-DAGs, this isn't the case; in particular, referring to Figure 1, we see that we can assign up to 130 nodes to both sides of the root partition (represented by the switch Cisco8).

```
FidduciaMattheyses(G, V1, V2):
    U = V = ∅
    foreach node v ∈ G:
        g = gain from moving v across the partition
        add v to gain[g]
    currentCost = Σ_{u∈V1, v∈V2} w(u, v)
    bestCost = currentCost
    bestPartition = (V1, V2)
    while V2 ≠ ∅ and V1 ≠ ∅:
        choose the highest gain node u from V1 and move it into V
        update the gains of each neighbor v of u
        choose the highest gain node s from V2 and move it into U
        update the gains of each neighbor t of s
        currentCost = currentCost - (gain(u) + gain(s))
        if currentCost < bestCost:
            bestCost = currentCost
            bestPartition = (V1 ⋃ U, V2 ⋃ V)
    return bestCost, bestPartition
```

Fig. 3. Fidducia-Mattheyses Procedure

In order to cope with these two changes to the underlying use case of the procedure, we offer two modifications to the Fidducia-Mattheyses procedure.

1. We replace the array of gain by an exponential trie. This data structure, fundamentally a tree indexed by each digit of the magnitude of the gain, gives a guarantee of $O(\log dw)$ to find the node of maximal gain and $O(\log dw)$ for trie updates, giving an algorithm with a total complexity of $O(n \log dw)$ and space $O(n + \log dw)$. The conventional Fidducia-Mattheyses procedure has a time complexity of $O(ndw)$ and a space complexity of $O(n + dw)$. This suffices to make the Fidducia-Mattheyses procedure efficient when dw is large.
2. In addition to moving nodes, we permit a further operation: *clone*. The *clone* operation assigns a node to both sides of the partition, exactly as required for a leaf DAG. When a node is assigned to both sides of a partition, none of its edges cross the partition; essentially, for purposes of subsequent partition calculations, the node has been deleted from the graph. This is done in a preprocessing step, by deleting the cloned nodes from the initial partitions V_1, V_2. A further parameter, C, gives the total number of cloned nodes.
   Cloning is incorporated directly into the Fidducia-Mattheyses algorithm. The *clone gain* of a node is set equal to its weighted degree. Nodes of highest weighted degree are eliminated successively from the partition V_1, V_2, and the *clone gain* recomputed after each deletion. Further, other nodes are moved from side to side during this process to ensure that the invariant – $|V_1 \bigcup U| = |V_2 \bigcup V|$ is maintained. At each deletion, C is decremented. When it reaches zero, the preprocessing step halts, and the conventional Fidducia-Mattheyses procedure resumes.

The revised Fidducia-Mattheyses procedure is shown in Figure 4. The first revision, to accomodate a large range of gain values, is left opaque here for reasons of brevity and clarity. The cloning operation appears as a preprocessor step.

```
FidduciaMattheysesWithCloning(G, V1, V2, C):
    U = V = ∅
    sortedNodes = v ∈ G sorted in decreasing order by total connections
    place top C nodes in sortedNodes into both U and V and delete from V1,V2,G
    foreach node v ∈ G:
        g = gain from moving v across the partition
        add v to gain[g]
    while |V1| > |V2| − 1:
        choose node v from V1 with highest gain, move to V
        recompute gains of neighbors
    while |V2| > |V1| − 1:
        choose node u from V2 with highest gain, move to U
        recompute gains of neighbors
    (S,T) = FidduciaMattheyases(G, V1, V2)
    return (S⋃U, T⋃V)
```

Fig. 4. Fidducia-Mattheyses Procedure with Cloning

Given the partitioning procedure, the assignment problem on a leaf DAG is straightforward; one simply partitions at the root, and recursively partitions at each subsequent level.

6 Experiments

A number of preliminary experiments were run using the revised Fidducia-Mattheyses procedure, without cloning, in Python 2.6. Great care should be taken in interpreting these results. Runtimes are quite short, especially in comparison to the Simulated Annealing procedures in the literature. However, it should be noted that the revised Fidducia-Mattheyses procedure does only a subset of the actions of the standard mapping procedures. In particular, it does not consider heterogeneity, nor multihoming. We view this procedure less as a replacement for the standard SA procedures, than as a preprocessor to help the SA procedure start from a known, fairly good solution.

In the first set of experiments, a set of structured graphs were constructed, with known partition properties. For these graphs, the weight of each node was set to one, and edges were assigned as follows for nodes $v_0, ..., v_n$

$$w(v_i, v_{i+1}) = 10, i = 1 \bmod 2, i < n/2$$
$$w(v_i, v_{i+2}) = 50, i = 1 \bmod 2, i < n/2 - 1$$

$$w(v_i, v_{i+3}) = 50, i = 1 \bmod 2, i < n/2 - 2$$
$$w(v_i, v_{i+1}) = 50, i = 1 \bmod 2, n > i \geq n/2$$
$$w(v_i, v_{i+2}) = 10, i = 1 \bmod 2, n - 1 > i \geq n/2$$
$$w(v_i, v_{i+3}) = 10, i = 1 \bmod 2, n - 2 > i \geq n/2$$

The intent in this case was to construct a set of nodes with a known best partition. In particular, the outdegree of each node in this graph is four, and the best partition is 120.

We obtained the following results on the structured graph experiment, for a single partition. In all experiments, the maximum capacity of each side of the partition was set to 60% of the total graph weight.

The implementation was done in Python 2.6 under Windows Vista. All times for all experiments were measured on an HP Elitebook 2530p laptop. Times reported are user time, measured with the builtin os.times() function in Python. In all experiments, we report the number of nodes in the graph (the column MaxNum), the number of iterations of the Fidducia-Mattheyses procedure required for convergence, the total time, the initial partition weight, the final (post F-M procedure weight), and the total improvement.

MaxNum	Iterations	Time (ms)	Initial Weight	Final Weight	% Improvement
100	4	0	1480	60	95.95
200	4	0	2670	120	95.51
300	6	0	4080	110	97.30
400	5	0	5870	110	98.13
500	6	15.6	6050	170	97.19
600	5	0	8220	150	98.18
700	7	0	10260	170	98.34
800	7	0	11390	150	98.68
900	6	15.6	13490	470	96.52
1000	7	15.6	15820	160	98.99

Fig. 5. Results for the Structured Graph Experiment

.

These results are somewhat promising, but the graphs involved are highly structured and artificial. To see how the procedure performs on less structured graphs, we ran two more sets of experiments. For a second set of experiments, we ran on a sequence of random graphs, where each node was given a random number of connections, exponentially distributed with a mean of five connections per node. Weights were randomly given, again with an exponential distribution with a mean of 50. Results are given in figure 6.

These results still show improvement over a random distribution, but they are not nearly as dramatic as the structured graph results.

For a third set of experiments, we constructed random graphs using a normal distribution of connections (mean 5, standard deviation 1) with connection

MaxNum	Iterations	Time (ms)	Initial Weight	Final Weight	% Improvement
100	3	0	10495	3282	68.73
200	5	15.600	23290	8934	61.64
300	3	0	32534	13959	57.09
400	6	15.600	45520	19429	57.32
500	4	31.200	55782	23103	58.58
600	14	78.001	70979	30760	56.66
700	6	15.600	76513	32958	56.92
800	5	0	87914	36076	58.96
900	4	46.8	105298	44836	57.42
1000	5	62.4	116133	50541	56.48

Fig. 6. Results for the Exponential Random Graph Experiment

.

weights normally distributed (mean 50, standard deviation 10). The results are shown in Figure 7

MaxNum	Iterations	Time (ms)	Initial Weight	Final Weight	% Improvement
100	7	31.2	11243	5579	50.38
200	11	15.6	22190	10830	51.19
300	5	15.6	32758	16833	48.61
400	6	0	44414	22254	49.89
500	4	15.6	55008	28793	47.66
600	6	15.6	69878	34143	51.14
700	5	15.6	76520	39243	48.72
800	6	62.4	86433	44697	48.29
900	7	31.2	101490	52529	48.24
1000	8	31.2	111873	55381	50.50

Fig. 7. Results for the Normal Random Graph Experiment

.

Results are somewhat promising, and the relatively low runtimes even for large graphs indicates that this technique has some promise, perhaps as a preprocessor to the standard simulated annealing technique.

As a final experiment, we continued a full partition recursively of the network, until we reached 12 nodes in a partition – a number arbitrarily set as the notional capacity of a switch. We then summed the total edge weight of connections which crossed a partition, effectively the total interswitch bandwidth of the fabric. We measured this for both a random assignment of nodes to switches in the tree, and then a partition using Fidducia-Mattheyses. The results are presented in Figure 8

The results are reflective of the individual partition experiments: the Fidducia-Mattheyses procedure is effective on structured graphs, and less so on random graphs.

	Structured Graph			Random Exponential			Random Normal		
MaxNum	Random	Fidducia	Imprvmnt	Random	Fidducia	Imprvmnt	Random	Fidducia	Imprvmnt
100	24100	3840	84.07	104752	85942	17.96	112117	95622	14.71
200	67390	7560	88.78	295521	259936	12.04	306664	281520	8.20
300	112620	13560	87.96	540842	400591	25.93	533112	408881	23.30
400	166990	20600	87.66	879114	783680	10.86	804577	753200	6.39
500	225380	38200	83.05	1152150	845132	26.65	1129426	906998	19.69
600	271800	30960	88.61	1312668	1061238	19.15	1311184	1083681	17.35
700	318740	60200	81.11	1573438	1359245	13.61	1536886	1388424	9.66
800	409090	43720	89.31	2096036	1928859	7.98	1933719	1832996	5.21
900	495840	142360	71.29	2617836	2082327	20.46	2439612	2024681	17.01
1000	556160	77210	86.12	2693960	2032458	24.56	2669548	2251410	15.66

Fig. 8. Total Bandwidth Experiments for All Graph Classes

.

7 Future Work

While a formal proof of the complexity of the network testbed mapping problem is important, future work remains in characterizing variants of the problem. In particular, the complexity bounds on solving generalizations of this problem for federations of testbed clusters will ultimately be relevant to the scalability of algorithms or heuristics for automatically mapping test networks onto large, constrained federations.

8 Acknowledgments

The authors gratefully acknowledge the assistance of the TridentCom general and program committees, and in particular many thoughtful comments on early versions of this paper from Rob Ricci.

References

1. The deter testbed: Overview. `http://www.isi.edu/deter/docs/testbed.overview.htm`.
2. Mohammad al Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity datacenter architecture. In *Proc. SIGCOMM*, 2008.
3. J. Duerig, R. Ricci, J. Byers, and J. Lepreau. Automatic ip address assignment on network topologies. Technical Report Flux Technical Note FTN-2006-02, University of Utah, 2006.
4. Emulab. Emulab topology diagram. http://www.emulab.net/doc/topo.pdf.
5. C. M. Fidducia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. 19th Design Automation Conference*, pages 175–181, 1982.
6. Michael R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co Ltd, January 1979.

7. Diwaker Gupta, Kashi V. Vishwanath, and Amin Vahdat. Diecast: testing distributed systems with an accurate scale model. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 407–422, Berkeley, CA, USA, 2008. USENIX Association.
8. Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shahi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the emulab network testbed. In *Usenix*, 2008.
9. B. W. Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–307, 1970.
10. Jay Lepreau. Emulab network emulation testbed. `http://www.emulab.net`.
11. J. Lischka and H. Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures (VISA)*, August 2009.
12. Robert Ricci, Chris Alfeld, and Jay Lepreau. A solver for the network testbed mapping problem. *Computer Communications Review*, 2003.
13. Malcom Scott, Andrew Moore, and Jon Crowcroft. Addressing the scalability of ethernet with moose. In *First Workshop on Data Center - Converged and Virtual Ethernet Switching (DC-CAVES), ITC 21*, 2009.
14. Gregory B. Sorkin. Efficient simulated annealing on fractal energy landscapes. *Algorithmica*, 6:367–418, 1991.
15. Aravind Srinivasan. Approximation algorithms via randomized rounding: A survey. In *Series in Advanced Topics in Mathematics, Polish Scientific Publishers PWN*, pages 9–71. Publishers, 1999.
16. M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtualnetwork embedding: Substrate support for path splitting and migration. *ACM SIGCOMM Computer Communications Review*, 38(2), April 2008.