Theoretical Approaches to Node Assignment

David G. Andersen

December 23, 2002

1 Abstract

In this paper, we provide a survey and explanation of several methods for approximating the NP-hard problem of finding a *multiway separator* in a graph. This is motivated by the problem of assigning nodes in an ethernet switch connected testbed without violating bandwidth constraints. As a critical component of the multiway separator problem, we discuss in some detail the *sparse cut* problem, with several modern $O(\log k)$ approaches, which result in realizable $O(\log^2 k)$ approximations to the multiway separator problem, and some even better theoretical bounds. Finally, we conclude with a less formal discussion of the mapping of the more general typed-nodes testbed problem to the multiway separator problem.

2 Introduction

The Flux Testbed project at the University of Utah is designing a network emulation testbed, consisting of PCs with multiple ethernet interfaces connected to switches. The switches have (effectively) infinite internal bandwidth. These switches are then connected to a master switch with links of finite bandwidth. A user of the testbed provides a description of the desired logical network they wish to emulate. One problem that has arisen is the assignment of physical nodes to logical nodes. An incorrect assignment could result in too much traffic going between switches, resulting in unintended interference between supposedly unrelated links in the user's experiment. In the most general case, there may be multiple types of nodes which the user wishes to allocate, but for now we focus on the uniform node case. The full testbed assignment problem is discussed in appendix A.

The testbed assignment problem is NP-hard. It is similar to the *multiway separator problem*, the *minimum cut linear arrangement problem*, VLSI layout problems, and embeddings of processor networks in parallel machines. However, unlike these problems, optimization is a secondary concern in the testbed assignment problem; a feasible assignment which respects both bandwidth constraints and the number of nodes available is the most desired solution - we can't accept a partially successful solution.

3 Problem Analysis

Regrettably, the testbed allocation problem is NP-hard. It reduces quite simply to the multiway separator problem by setting all node types equal, and asking for a mapping where the external node clusters connect to the internal nodes with a certain bandwidth. A binary search can then be used to close down that bandwidth such that an optimum solution is found in polynomial time with a solution to the testbed allocation problem. A similar reduction can be used to the minimum-degree graph partitioning problem [5].

4 Attacking the Problem

4.1 Brute-Force Approaches

4.1.1 Backtracking

There are many approaches which can be taken to solve this problem. The most obvious is the use of a brute-force backtracking algorithm. With sufficiently clever heuristics, this may be a feasible approach. Early implementation work by Chris Alfeld at Utah showed both the practicality and impracticality of this. On a simplified model similar to that proposed for this work, a heuristic approach was able to solve some medium sized problems of about 20 nodes in under 1 minute. However, the goal of the testbed project is to scale to ˜ 200 node simulations, so more scalability is needed. Perhaps more importantly, the backtracking algorithm takes exponential time regardless of heuristics to determine that there is no solution to the problem.

4.1.2 Simulated Annealing

Simulated annealing [6] is a potentially promising approach in practice. It is easily implemented and produces good results in other, similar, domains such as cellular base station location [1]. However, simulated annealing suffers from two drawbacks. Since graph theory is extensively studied, we have extensive domain-specific information available for our problem, which simulated annealing fails to utilize. Second, it may be very difficult to design an evaluation function which leads to quick convergence. Simulated annealing can still be quite slow, and may again take large amounts of time to conclude that it was unable to find a solution.

4.2 Approximation Algorithms

Because of the problems with the brute force approches, an approximation algorithm is an attractive alternative if it provides reasonable approximation bounds, and in polynomial time can tell us if there is no acceptable partition for the nodes. Our basic problem is one of graph partitioning, and in recent years, several approaches have been developed to approximate this problem. In the rest of this paper, we examine one metric for graph partitioning, the *sparse cut*, show its relationship to the testbed embedding problem, and discuss several methods for finding approximate sparse cuts, and from there, approximations to the k-multiway cut.

5 Sparse Cuts

5.1 Multicommodity Flow Problems

In order to understand the sparse cut problem, we must first understand the context in which sparse cuts are discussed. This problem is called the Multicommodity Flow problem.

The MCF problem is similar to a maxflow problem, but instead of only one source and one sink, we have many pairs of sources and sinks, which send *different* types of flow between them. The types of flow are termed commodities. Each node can act as a source and a sink for zero or more of these commodoties. Each source has some *supply* of each commodity, and has a *demand* for each commodity. The object of the MCF problem is to attempt to satisfy the most possible demand while remaining within the capacity bounds of the edges. Each commodity i has a source and a sink (s_i, t_i). A simple multicommodity flow problem is shown in figure 1

Note that any vertex may have a supply of many commodities, and a demand for many commodities.

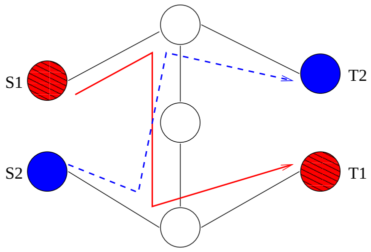

Figure 1: A simple multicommodity flow problem. There are two commodities. S1 is a source of the first commodity, and T1 has a demand for that commodity. Likewise, S2 is a source of the second commodity, and T2 has a demand for that commodity. The lines show the paths which the flow would need to take to reach the demands

5.2 Understanding Sparse Cuts

If we view (source,sink) pairs as sources of traffic, then the goal of a sparse cut is to stop as much traffic from getting through as possible, at minimum cost of cutting edges. One example of a sparse cut is in assigning tollbooths to roads: you make money per-driver by forcing them to go through a tollbooth, but you have to pay to errect the booth. Your profit margin is determined by the ratio of tolls to booth costs. In the general multicommodity flow case, the ratio of demand separated to the cost of separating it is called the *sparsity ratio* of the set S, and is referred to as $\rho(S)$.

Looking at this more formally, we have a graph $G = (V, E)$. Each edge has a cost $c(e)$ (for each edge $e \in E$). We have k commodities, and each one has a source and a sink. We term this pair (s_i, t_i) for a commodity i. The demand associated with this commodity i is $d(i)$.

When we make a cut across this graph, we chop the graph into two distinct sets of vertices, U and $\overline{U}$. We had to cut a set of edges which crossed between the sets, $e \in (U, \overline{U})$. The total cost is the sum of the costs of those edges, $c(e)$. With this cut, we also managed to separate the sources and sinks of some of the commodities. We'll call this subset of the commodities I. We define the sparsity ratio as the cost/benefit ratio from the cut:

$$\rho(S) = \frac{\sum_{e \in (U, \overline{U})} c(e)}{\sum_{i \in I} d(i)}$$

Unfortunately, finding a cut which maximizes this ratio is NP-hard, but approximations exist.

5.3 Uniform Multicommodity Flow Problems

In their work, Leighton and Rao [8] use a special type of MCF they term a *uniform multicommodity flow problem*. In the UMFP, there is one commodity between every pair of vertices in the graph (not just connected vertices). The demand for this commodity is equal to one. The result of this is that the sparsity ratio of a cut on this graph is then equal to the cost of the edges we cut to separate the nodes divided by the number of nodes "chopped off" from the rest of the graph. The expression for the sparsity ratio ρ simplifies to:

$$\rho(S) = \frac{\sum_{e \in (U, \overline{U})} c(e)}{|U||\overline{U}|}$$

In figure 2, we show the ρ values of different cuts across a uniform multicommodity flow problem with edge costs of 1.

The relationship of a sparse cut to the testbed allocation problem should now be apparent: The sparse cut gives us a way to chop the "best" pieces off of the graph. However, "best" in this case means

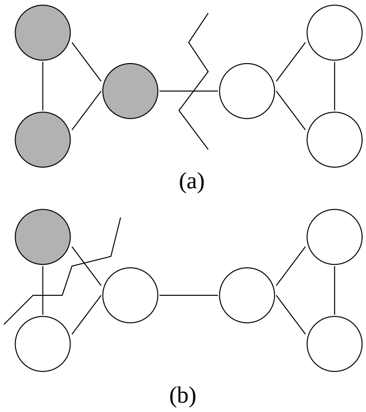

Figure 2: Two cuts across a simple graph. The first cut is a sparse cut, where we cut one edge, and separate three nodes, so $\rho = \frac{1}{9}$. The second cut is not an optimal cut, for which $\rho = \frac{2}{5}$

the smallest $\rho(S)$ cuts, so we need to determine a method for using a sparse cut to map to a testbed allocation. We'll discuss this after explaining how to find a sparse cut.

5.4 The Dual of an MCF

The algorithms below use the dual of the multicost flow problem as a part of their solution, so it's important to understand what the dual means. The primal represents a maximized assignment of flow to edges, subject to the edge capacities, which ships flow from the sources to the sinks.

The dual of our maxflow is another graph with edges of a certain length between the vertices. We can think of these edges as pipes. Each pipe can carry a certain amount of flow based upon how wide it is; that width is the capacity (formerly the cost) of the edge, $c(e)$. Thus, the volume of a pipe is equal to its width $c(e)$ times its length, $d(e)$. We constrain the volume of the entire system to equal 1: $\sum_{e \in E} d(e)c(e) = 1$.

The total volume of flow in a network must not exceede the total volume of the pipes that carry the flow. We know that the optimum value of the dual is equal to the optimum value of the primal, so the question for the dual problem is an assignment of distances to vertices which minimizes the volume of the network.

6 Approaches to the sparse cut

6.1 Rao's Planar Cuts

In 1987, Satish Rao developed the first algorithm for finding optimal separators in planar graphs [10]. While his dynamic programming approach to finding sparse cuts was specific to planar graphs, the resulting algorithms for finding k-ratio separators and approximate k-ratio separators applies to general graphs as well. As we'll see, the algorithm for finding an approximate k-ratio separator becomes very important in one approach to the multiway cut problem. The algorithm is quite simple, and very understandable. Paraphrasing from [10]:

1. Find the optimal k-ratio quotient circuit separator
2. If the result is a k-ratio separator, then exit. Else, remove the nodes separated by the cuts defined in part 1, recompute how much weight will need to be removed from the graph, and repeat this process on the resulting smaller graph until we find a k-ratio separator.

It is important to note that this requires an *optimal* k-ratio quotient separator, and results in a $log(n)$ approximation to the optimal k-ratio separator. However, by relaxing the k-ratio requirement to a ck-ratio requirement, we reduce the cost to a linear function of optimal cost for the k-ratio separator. That is, we find a cut which acts as a ck-ratio separator (not as good as a k-ratio separator), but we pay c times the the optimal cost for the k-ratio separator.

6.2 Leighton-Rao

Rao's work set the stage for the next major development. The next year, Leighton and Rao developed their max-flow min-cut theorem for UMFPs [8]. Like a traditional maxflow problem, a multicommodity flow problem can be solved in polynomial time as a linear program. We know from the max-flow min-cut theorem for traditional maxflows that the flow across the minimum cut is equal to the maxflow of the graph. However, this relationship does not hold for MCF problems. This means that a polynomial solution to the UMFP does not give you a min-cut on the graph. However, Leighton and Rao showed the first bounds on how far off the UMFP is as an estimate of the sparse cut. Their results, where S is the value of the sparse cut and f is the flow of the UMFP, are:

$$\Omega(\frac{S}{\log n}) \leq f \leq S$$

This means that using the UMFP as an estimator, we can obtain a sparse cut within a factor of $\log n$ of the optimal value.

Leighton-Rao is an important theoretical advance, but the algorithm developed for finding the cuts has been supplanted by more recent work. Therefore, I only briefly sketch the operation of their algorithm.

Leighton-Rao operates by first solving the multicommodity flow problem, and its dual. As noted earlier, the dual of the UMFP provides a "worst possible" distance function on the edges. From here, the algorithm uses a modified shortest path algorithm to attempt to route flux between the pairs of vertices along paths with short length. Conceptually, computing the dual of the algorithm has pulled some vertices apart from each other and left clusters of these vertices together, with the distances defined by the results of the dual problem. By routing flow on shortest paths, the algorithm keeps the flow away from these long links and within clusters of nodes which are more tightly interconnected, producing a cut which does not go through the more tightly connected clusters. In approximating the cut in this manner, the factor of $O(\log n)$ is lost.

6.3 Metric Embeddings

Aumann and Rabani developed a new, and more practical, algorithm for finding sparse cuts in multicommodity flow graphs [2]. Their algorithm produces an $O(\log k)$ approximation (equivalent to the $O(\log n)$ bound found by Leighton-Rao for the uniform case). In a manner similar to Leighton-Rao, the algorithm first uses the solution to the dual to assign distances to the edges. However, the algorithm then uses these distance measures to express the graph as a set of points in metric space (with k dimensions, where k is the number of commodities), via an exact transformation.

Expressed as a set of points in metric space, the problem can be viewed as one of separating the clusters of points with a hyperplane which "slices through" the edges connecting those clusters. Of course, this is still an NP-hard problem when viewed this way.

The beauty of the Aumann and Rabani approach is that it then translates this metric space from L_∞ to L_1, but in doing so, loses a factor of $O(\log k)$ in accuracy. Once the embedding is in L_1, however, it has reduced dimensionality from k to $O(\log^2 k)$.

Once this embedding has been created, the algorithm finds a random hyperplane which is perpindicular to one of the axes in the embedding. This hyperplane will cut through some of the edges in the graph; obviously, longer edges will have a higher probability of being cut than shorter edges. With hope that honesty points apply here, with math that I'm not sure I grok quite yet, Aumann and Rabani show expected capacity of this cut is equal to the max-flow of the graph.

In a similar approach, Linial, London, and Rabinovich [9] showed a two ways of obtaining a metric embedding. Their methods are based upon an exponential algorithm called Bourgain's algorithm. It is a deterministic algorithm, but isn't particularly practical. We use a function $d(x, A)$ where x is a vertex, and A is a set of vertices. $d(x, A)$ is the minimum distance betwen x and the closest vertex in A. Note that d produces a vector pointing between the two vertices. For every subset A of V, we determine a coordinate in the new embedding, with the normalized distance between each vertex in the graph x and the vector $d(x, A)$.

To avoid the exponential size problems with Bourgain's algorithm, LLR present two methods. The first, the *random subsets embedding* method, finds am embedding in randomized polynomial time. For a graph with n vertices, for every $k < n$ which is a power of two, they randomly select $O(\log n)$ subsets of the vertices where the subsets have cardinality k. For each subset $A \subset V(G)$, map each vertex x in the matrix to $d(x, A)$. For every A we select, this produces a different coordinate for each x. Since we do this for each power of two under n, we end up creating $\log n$ different representations of the original vertices; by using $\log n$ subsets, the result is that we map each original vertex into a new formulation on $\log^2 n$ dimensions.

LLR's second approach uses semidefinite programming to find an approximate embedding in deterministic polynomial time. Until slightly more recent work [4], this was the most promising avenue for implementation.

7 Multiway separators

As previously mentioned, the testbed problem is extremely similar to the *multiway separator* problem, in which the goal is to partition a graph into k groups of similar size with the removal of the minimum subset of edges.

The approximation algorithms for multiway separators use the sparse cut as a subroutine, and apply a divide-and-conquer methodology to the two components resulting from such a cut. The first such algorithm was developed by Leighton et al. in 1990 [7], and used the Leighton-Rao cut approximator. The method was simple, and understandable.

7.1 Simple construction of a k-way separator

Given a set of connected vertices V_i, find a minimum-quotient cut for an approximate $\frac{1}{k}, \frac{k-1}{k}$ separator. This will produce two subgraphs. While the size of any subgraph is larger than k, repeat this process recursively on it. This produces an approximation to the optimal k-way separator within a factor of $\log n \log \frac{n}{k}$ when partititioning into fewer sets of larger size, where $k \geq \log n$, or a factor of $\log^2 n$ when producing more sets of smaller size, where $k \leq \log n$.

7.1.1 Proof of approximation bounds

The resulting optimality of the result is nicely bounded. I explain Leighton's proof here. (Parts of this proof are taken verbatim from [7], because they said it better than I could).

For this proof, we have a graph G with n nodes, which has been divided up into m sets $V_1, V_2, ..., V_m$. Consider an optimal partitioning of any set V_i into sets of size $\leq k$, which has cost OPT_i^i, and an *optimal* $\frac{1}{3}, \frac{2}{3}$ partition of the same graph, which has cost B_i. Set V_i has size n_i.

Lemma 1 *If $k \leq \frac{n_i}{3}$, then $B_i \leq OPT_k^i$*

Explanation and proof: Lemma 1 states that if OPT_k^i splits into 3 or more partitions, its cost is at least that of the optimal $\frac{1}{3}, \frac{2}{3}$ partition.

The k-partition gives us m subsets of nodes, which we label $a_1, ..., a_m$. We'll sort these in increasing size, so $|a_1| \geq |a_2| \geq |a_3|$ and so on. Now, we walk down this list and assign alternate records into one of two subets, V_{i1} and V_i2. a_1 goes into V_{i1}, a_2 into V_i2, a_3 into V_{i1} and so on. $|V_{i1}| \geq |V_i2|$ because it gets the first pick of the possibly larger values. Note, though, that we effectively reverse the order of picking if we simply remove a_1 from V_{i1}. Therefore, $|V_{i1}| - a_1 \leq |V_{i2}|$. We know that $|a_1| \leq \frac{n_i}{3}$ because that was the partitioning criteria for forming OPT. Therefore, we know that V_{i1} and V_{i2} are within $\frac{n_i}{3}$ of each other, so we can successfully bundle the results of the optimal partition into two subsets which form a $\frac{1}{3}, \frac{2}{3}$ partition.

Next, consider the globally optimum partition OPT_k on the entire graph G.

Lemma 2 *If $k \leq min\{\frac{n_1}{3}, ..., \frac{n_m}{3}\}$, then $OPT_k \geq \sum_{i=1}^{m} B^i$.*

Explanation and proof: If k is $\leq$ the smallest subset in the current partition, then the cost of the optimal partition which results in no partition larger than k is at least as expensive as the sum of the $\frac{1}{3}, \frac{2}{3}$ partitions which created the current partition.

Consider the set of edges that form the OPT_k solution. In each set V_i, we remove the edges which would be in the OPT_k partition. Since $k \leq \frac{n_i}{3}$, and in OPT_k no connected subset has size $\geq k$, we know that we have partitioned all V_i sets into subsets at most $\frac{1}{3}$ their original size. By lemma 1, this is a $\frac{1}{3}, \frac{2}{3}$ partition of the original sets V_i.

We have a sparse cut approximation which can give us a $\frac{1}{3}, \frac{2}{3}$ partition within a factor of $O(\log n)$. If $k > \log n$, then we have $O(\log \frac{n}{k})$ levels of recursion before we have chopped the graph sufficiently. By lemma 2, we have a lower bound on the optimum for creating just larger $\frac{1}{3}, \frac{2}{3}$ partitions, so we know our bound is thus $O(\log n \log \frac{n}{k})$. When $k \leq \log n$, there are $O(\log n)$ (worst case!) levels in the partition tree. This results in very tiny final sets. For any connected component of size between k and $3k$ ($\log n$ and $3 \log n$), the optimal solution must remove at least one edge. This subgraph can have a maximum of $\log^2 n$ edges, so if we were to nuke them all, we would only lose us a factor of $O(\log^2 n)$ times the optimal. From the proof perspective, then, we can pay $O(\log^2 n)$ to reduce our graph to subgraphs of size $3k$, and $O(\log^2 n)$ to reduce those subgraphs to size k. In practice, two things help us. First, the subsets are small enough that it may be feasible to actually find an optimum partition of the subgraph. Second, since the testbed has a design goal of $2^7 - 2^10$ total nodes, we will likely rarely exceed the bounds for $O(\log n \log \frac{n}{k})$.

7.2 Better bounds via spreading metrics

A more direct solution to the k-multiway partition is explored in [4]. Like earlier work on sparse cuts, the authors use a metric embedding of the dual. However, instead of seeking near-$\frac{1}{2}$ separators and applying a divide-and-conquer approach, Even et al. directly remove subsections of the graph which correspond to the proper size subgraph, and only recurse on the remaining large part of the graph. As a result, they are able to achieve an $O(\log n)$ approximation to the k-multiway partition problem, with improved time bounds.

Their algorithm operates by first finding a spreading metric which assigns edge lengths to the edges in the graph. While this LP has an exponential number of constraints, the authors suggest the construction of a nonexponential separation oracle, which can be used by the ellipsoid method to solve the program in polynomial time. We'll examine the unweighted case, where all vertices have weight 1 (the problem we must solve for the testbed).

Their LP computes the spreading metric for a ρ-separator, a set of edges whose removal causes the resulting graph to be partitioned into subsets of at most size $\rho \cdot |V|$.

$$min \sum_{e \in E} c(e) \cdot d(e)$$

subject to the constraints:

$$\forall S \subset V, \forall v \in S : \sum_{u \in S} (dist(v,u) - 1) \geq -\rho \cdot |V|$$

Realizing that it's unfair to spout math without explanation, what does this combined constraint mean? For every proper, nonempty subset of the vertices V of our graph, the sum of the distances of any vertex to the rest of the vertices must be greater than the maximum subset size allowed by our ρ. (Note that this is a simplification of the weighted case, but it's sufficient for our problem). Evan et al. describe different separation oracles for the unweighted and weighted cases.

Once the spreading metric (distance values on the edges) has been computed, we must then compute a cut based upon this metric. First, we decided upon a ρ value defining the size of the cut we wish to make, and a related value ρ' which defines the tolerance of our cut. As in the earlier sparse cuts, we find a ρ' separator at some function of the cost of a ρ separator. To do so, we pick any starting vertex $v \in V$, and run Dijkstra's shortest paths algorithm from it. Since the distance of subsequently added nodes with Dijkstra is a nondecreasing function, we run until the distance of the vertex we would add next is greater than our threshold value, $r = \frac{\rho' - \rho}{\rho'}$. Conceptually, this procedure begins drawing a spheroid around the starting vertex. The radius limit for the spheroid is defined by the size (and tolerance) of the cut we wish to make.

This algorithm appears to be the most readily implementable of all of the algorithms discussed thus far. While it remains a reasonably complex algorithm, it should be possible to create an implementation with a reasonably fast ellipsoid LP solver.

8 General Testbed Problem

8.1 Additional Constraints

Note that the general testbed problem is *not* the k-multiway graph partitioning problem. The testbed problem has a second set of constraints, node types and capacities, which must be satisfied in addition to the bandwidth constraints. Thus, we can immediately observe that the set of feasible solutions to the testbed problem is a *subset* of the feasible solutions to the graph partitioning problem. One of the major problems with the heuristic methods discussed in section 4.1 is that they require exponential time to determine that a problem is infeasible. For problems which are infeasible due to bandwidth utilization, a k-multiway partition can determine this in polynomial time. (In fact, for problems whose bandwidth utilization is greater by a factor of $O(\log n)$, we can answer the question definitively).

It remains for me to determine the best way to simultaneously fit the node constraints and the bandwidth constraints of the general testbed mapping problem. The obvious approach is to divide the graph into sections smaller than the node capacity, and then group these resulting subgraphs in such a way to make the distribution work properly. Intuitively, this should quite well, because the existence of multiple nearly equivalent (but fairly disjoint) partitionings implies the subsets produced by one partition are not densely connected, but can instead be internally re-partitioned with relatively low cost. [3] hints at this technique when they use a similar method for finding k-way separators from a ρ-separator.

The resulting assignment of node types is still an NP-complete assignment problem, but because the nodes on which it operates have already been clumped into fairly large groups, the overall number of assignments which must be made is quite small (likely less than 32), so an heuristic approach should find a good solution rapidly even for large graphs.

8.2 Non-star layout

Nothing in the general testbed requires that the physical layout of the switches be a star pattern. However, if we pick an arbitrary inter-switch link to represent the "middle" of the graph, we can compute the weights of the graph on each side of that link, and recursively divide the graph in this manner. Unfortunately, this kind of partitioning leaves us with far less flexibility in assigning requests to node clusters so as to satisfy the other constraints. This suggests that for the design of the testbed problem, the star formulation will maximize flexibility, so long as sufficient switch capacity is available.

8.3 On-line global optimality

The work presented herein results in an approximation to the optimal solution for a single user of the testbed. However, the assignments on a testbed are a dynamically changing set, and will be used on-line. The question remains, then, of the best way to allocate testbed mappings on-line to attempt to maximize the remaining usefulness of the testbed. This is a secondary concern to those discussed in this paper, but further information on this problem would be helpful.

9 Conclusions

A critical basis of the multiway separator approximator is the sparse cut on multicommodity flows, and many $O(\log n)$ approximations to this cut exist. Recursive sparse cuts achieve an $O(\log^2 n)$ approximation to the k-multiway separator, and spreading metric-based achieve $O(\log n)$ approximations. For the testbed assignment problem, the multiway separator approximation is a very valuable tool. In polynomial time, it can tell us if the requested input is not viable for the general assignment problem, and it provides a strong basis for approximating a solution if one exists.

From an implementation standpoint, recent spreading metric approaches by Even et al. provide the most pragmatic approach to solving both the direct k-multiway separator problem and the sparse cut problem.

References

[1] H. Anderson and J. McGeehan. Optimizing microcell base station locations using simulated annealing techniques. In *Proceedings of the 44th Vehicular Technology Society Conference*, pages 858–862, Stockholm, June 1994.

[2] Y. Aumann and Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput.*, 27(1):291–301, Feb. 1998.

[3] G. Even, J. S. Naor, S. Rao, and B. Schieber. Divide-and-conquer approximation algorithms via spreading metrics (pending journal submission).

[4] G. Even, J. S. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. *SIAM J. Comput.*, 28(6):2187–2214, 1999.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness.* W. H. Freeman, San Fransisco, CA, 1979.

[6] L. Ingber. Very fast simulated re-annealing. *Journal of Mathematical Computer Modelling*, 12:967–973, 1989. URL http://www.ingber.com/asa89_vfsr.ps.gz.

[7] T. Leighton, F. Makedon, and S. Tragoudas. Approximation algorithms for vlsi partition problems. In *Proceedings of the 1990 IEEE International Symposium on Circuits and Systems*, pages 2865–2868, 1990.

[8] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *29th Annual Symposium on Foundations of Computer Science*, pages 422–431, White Plains, New York, 24–26 Oct. 1988. IEEE.

[9] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. In *35th Annual Symposium on Foundations of Computer Science*, pages 577–591, Santa Fe, New Mexico, 20–22 Nov. 1994. IEEE.

[10] S. Rao. Finding near optimal separators in planar graphs. In *28th Annual Symposium on Foundations of Computer Science*, pages 225–237, Los Angeles, California, 12–14 Oct. 1987. IEEE.

A Testbed Assignment Problem

In this section, we describe the general testbed assignment problem in more detail. An understanding of the general problem is not essential to the rest of the paper, but helps to understand the motivation behind this work.

A.1 Problem Inputs

There are two inputs to the system. The first is a reasonably static set of inputs which describe the physical topology. The second is the user's inputs for their desired emulation environment.

I believe that multiple assignments can be handled by subtracting out the existing simulations from the physical topology, and reducing the bandwidths as necessary. This may not result in a globally optimal solution, but since this assignments occur on-line, we are constrained in what we can do anyway.

- A description of the physical topology:
  - A list of nodes, with their parameters.
    - Machine type
    - Number of network interfaces
    - Forwarding capacity (for multiple interface machines)
  - A list of switches
  - A list of links, with their bandwidth.
- A description of the desired emulated topology:
  - A list of user nodes with their type
  - A list of links

A.2 Output

The output of this algorithm should be two things:

- A list of all the nodes the user is using
- A list of the links and link combinations needed.
- A list of the amount of traffic across the physical links so we can compute the remaining bandwidth on inter-switch links.
- A list of the delay node configurations needed

Restrictions on interactions:

- Some machine types may also be able to act as type "delay". A machine being used for any delay channels must be used only to provide delay channels.

- The total throughput through a delay node must be $\leq$ to its capacity.
- The bandwidth over inter-switch links must be $\leq$ their capacity.

A pictoral representation of the inputs and outputs to the system can be see in figure 3.

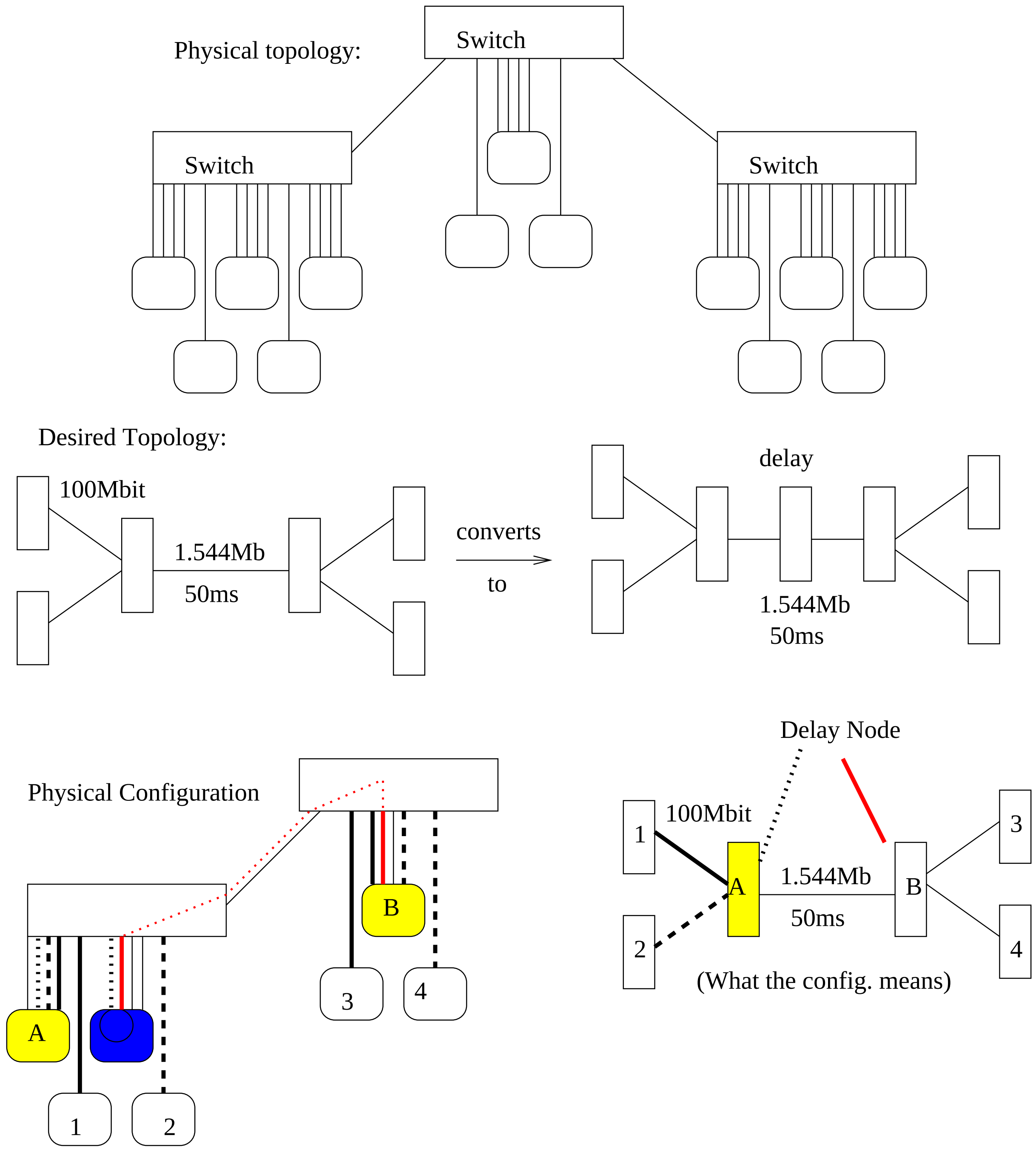

Figure 3: A pictoral representation of the inputs and outputs from the algorithm. The top picture is the input physical topology. The middle right figure is the needed logical topology output from the preprocessor. The lower left picture shows a mapping of the desired topology to the physical topology, with an explanation at right.