Hybrid Link-State, Path-Vector Routing

M. Abdul Alim
Computer Laboratory, University of Cambridge
15 JJ Thomson Avenue, Cambridge, UK
Abdul.Alim@cl.cam.ac.uk

Timothy G. Griffin
Computer Laboratory, University of Cambridge
15 JJ Thomson Avenue, Cambridge, UK
Timothy.Griffin@cl.cam.ac.uk

ABSTRACT

Traffic engineering in Internet backbones can be improved by off-line optimization algorithms that automatically configure link weights used by routing protocols. However, with existing protocols large networks often require some type of partitioning in order to scale the routing protocol, and these partitions actually complicate the metrics to the extent that link-weight optimization is no longer practical. In this paper we study how an algebraic specification of a path problem can be naturally decomposed into simpler sub-problems where each sub-problem can then be solved independently without changing the global metric being used network-wide. In addition, we go on to study four possible combinations of link-state and distance-vector mechanisms in this setting. In particular, we attempt to clarify the trade-offs between fast convergence of link-state and low space requirements of distance-vector. The results provide a framework for analyzing existing mechanisms and for designing more reliable and robust routing protocols.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]; C.2.2 [**Network Protocols**]: Routing protocols

General Terms

Theory, Algorithms

Keywords

Routing protocols, Algebraic path problem, Link-state, Path-vector

1. MOTIVATION

Most Internet routing protocols respond to topology changes but not to shifts in traffic load [1]. Typically, link weights are adjusted at network management time scales to improve network performance. Recent research on traffic engineering has shown how off-line algorithms can be used to configure link weights to solve various network-wide optimization problems [2, 3].

For campus, corporate, or service provider backbones, the most common intra-domain protocols used are OSPF and IS–IS. Both of these protocols provide mechanisms to partition networks into regions in order to allow the protocols to scale to very large networks. However, with both protocols the introduction of regions actually *complicates* the metric used for finding best paths. Furthermore, this complication precludes the use of the automated traffic engineering tools just mentioned.

In this paper we address this issue by untangling the path problem being solved from the mechanisms used to solve the problem. We study how an algebraic specification of a path problem can be decomposed into simpler sub-problems where each sub-problem can be solved with a link-state or path-vector approach without changing the global metric being used (Sections 2 and 3). Informally, this decomposition partitions the network into *regions* and a *core* that spans the regions.

Dijkstra's shortest-path first (SPF) algorithm is associated with *link-state* routing protocols such as OSPF and IS–IS that are commonly used in enterprise networks because of the fast convergence of the SPF algorithm. On the other hand, *path- or distance-vector* routing protocols such as RIP and the Border Gateway Protocol (BGP) use the distributed Bellman-Ford (DBF) algorithm [1]. There is a clear trade-off — link-state protocols converge faster but require more memory at each router, while distance-vector protocols are slower to converge but require much less memory.

We develop an analytic cost model for four possible combinations of link-state and path-vector mechanisms to be used inside regions and in the core (Section 5.3). We have shown that hybrid algorithms have better space-time trade-offs than using the same algorithm. Moreover, we have observed that the SPF inside

AINTEC'10, November 15–17, 2010, Bangkok, Thailand.

routing	S	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$
least-cost	$\mathbb{N}^\infty$	min	$+$	∞	0
max-capacity	$\mathbb{N}^\infty$	max	min	0	∞
most-reliable	$[0,\ 1]$	max	$\times$	0	1
usable-path	$\{0,\ 1\}$	max	min	0	0

Figure 1: A few simple semirings.

regions and the DBF in the core graph is the best alternative for practical networks. We then present simulation results of hybrid algorithms in Section 5.3. Related work and open problems are discussed in Section 6.

2. PRELIMINARIES

The model we develop is not restricted to shortest-path routing, but can be applied to a very large class of routing metrics. For background, this section contains a brief review of this theory. The reader should consult [4, 5] for more details.

The standard shortest-path routing uses weights drawn from the structure sp $= (\mathbb{N}^\infty,\ \min,\ +,\ \infty,\ 0)$, where $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$. This structure, and many of its associated algorithms, can be generalized to a large class of algebraic structures called *semirings*. Semirings have the form $(S, \oplus, \otimes, \overline{0}, \overline{1})$, comprised of a carrier set S, an associative and commutative additive operation $\oplus$, an associative multiplicative operation $\otimes$, an additive identity $\overline{0}$, and a multiplicative identity $\overline{1}$. The binary operations satisfy the distributivity laws — $\otimes$ must distribute over finite $\oplus$-sums. In addition, $\overline{0}$ must be an annihilator for $\otimes$. That is, $\forall s \in S : s \otimes \overline{0} = \overline{0} \otimes s = \overline{0}$. Figure 1 presents a few simple semirings corresponding to metrics that are useful in finding optimal paths in weighted graphs.

Given a semiring $(S, \oplus, \otimes, \overline{0}, \overline{1})$ and a graph graph $G = (V, E)$, a *weight function for* G is a mapping $w \in E \to S$. (Below, n is the number of vertexes in V.) If $p = v_0, v_1, \cdots, v_k$ is a path in G, then the weight of p, denoted $w(p)$, is the product of arc weights, $w(v_o, v_1) \otimes w(v_1, v_2) \otimes \cdots \otimes w(v_{k-1}, v_k)$. The empty path is given weight $\overline{0}$. For most semirings useful in network routing (and all of the examples in Figure 1) the $\oplus$ operation is both *idempotent* ($\forall s \in S : s \oplus s = s$) and *selective* ($\forall s, t \in S : s \oplus t \in \{s, t\}$). In such cases the relation $a \leq b \equiv a = a \oplus b$ is a total order, and $\oplus$ can be used to select *best* path weights with respect to this order.

Given a weighted graph, the *all-pair best path problem* is to find the best path weight over all paths from i to j for all $i, j \in V$. Put another way, we would like to find a matrix $\mathbf{R}$ such that

$$\mathbf{R}(i,j) = \bigoplus_{p \in \mathcal{P}(i,\ j)} w(p).$$

where $\mathcal{P}(i,\ j)$ is the set of all paths from node i to node j. This may be accomplished using matrix methods by first defining the semiring of $n \times n$ matrices over S. If $\mathbf{X}$ and $\mathbf{Y}$ are two matrices, then $\mathbf{C} = \mathbf{X} \oplus \mathbf{Y}$ is defined to be the matrix such that $\mathbf{C}(i,j) = \mathbf{X}(i,j) \oplus \mathbf{Y}(i,j)$. The product $\mathbf{C} = \mathbf{X} \otimes \mathbf{Y}$ is defined to be the matrix

$$\mathbf{C}(i,j) = \bigoplus_{1 \leq q \leq n} \mathbf{X}(i,q) \otimes \mathbf{Y}(q,j).$$

The notation $\mathbf{XY}$ is used for $\mathbf{X} \otimes \mathbf{Y}$. The *closure* of matrix $\mathbf{X}$, denoted $\mathbf{X}^*$, is defined as

$$\mathbf{X}^* = \mathbf{I} \oplus \mathbf{X} \oplus \mathbf{X}^2 \oplus \cdots \oplus \mathbf{X}^k \oplus \cdots \tag{1}$$

where $\mathbf{X}^k$ is the k-fold matrix product of $\mathbf{X}$. Note that $\mathbf{X}^*$ may not exist, or may not be computable.

The function w is then captured in an $n \times n$ *adjacency matrix* $\mathbf{A}$, where $\mathbf{A}(u,v) = w(u,v)$ when $(u,v) \in E$ and $\mathbf{A}(u,v) = \overline{0}$ when $(u,v) \notin E$. Given suitable conditions on the semiring or on the graph, the closure of $\mathbf{A}$ exists and is the solution we are looking for,

$$\mathbf{A}^*(i,j) = \bigoplus_{p \in \mathcal{P}(i,\ j)} w(p). \tag{2}$$

For many semirings useful in network routing (and all of the examples in Figure 1) the multiplicative identity is also an annihilator for $\oplus$ ($\forall s \in S : s \oplus \overline{1} = \overline{1} \oplus s = \overline{1}$). This implies that paths with loops are not considered in the sum and therefore we have

$$\mathbf{A}^* = \mathbf{I} \oplus \mathbf{A} \oplus \mathbf{A}^2 \oplus \cdots \oplus \mathbf{A}^{n-1}. \tag{3}$$

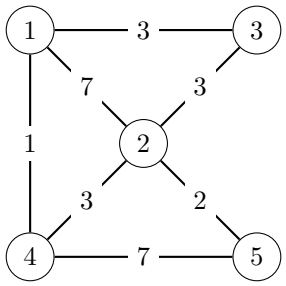


Figure 2: A shortest-paths example.

Figure 2 presents a simple graph where weights are drawn from the semiring sp. The adjacency matrix is

$$\mathbf{A} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & 7 & 3 & 1 & \infty \\ 7 & \infty & 3 & 3 & 2 \\ 3 & 3 & \infty & \infty & \infty \\ 1 & 3 & \infty & \infty & 7 \\ \infty & 2 & \infty & 7 & \infty \end{bmatrix} \end{array}$$

and the solution to the all-pair shortest paths problem is given by the matrix

$$\mathbf{A}^* = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 4 & 3 & 1 & 6 \\ 4 & 0 & 3 & 3 & 2 \\ 3 & 3 & 0 & 4 & 5 \\ 1 & 3 & 4 & 0 & 5 \\ 6 & 2 & 5 & 5 & 0 \end{bmatrix} \end{array}$$

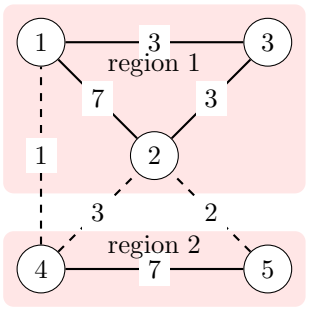


Figure 3: The example with two partitions.

There are many algorithms for computing $\mathbf{A}^*$ [4]. Some of these algorithms are generalizations of of familiar algorithms such as the Floyd-Warshall, Bellman Ford, or Dijkstra's (the last two can be used to compute $\mathbf{A}^*$ one row or column at a time). See Section 4 for a discussion of distributed computation.

3. ALGEBRAIC PARTITIONING

A natural way of partitioning a graph $G = (V, E)$ into m regions is to partition the set of vertexes into m non-empty disjoint sets, $\pi = \{V_1,\ V_2,\ \ldots,\ V_m\}$. where $V = V_1 \cup V_2 \cup \cdots \cup V_m$ where $r \neq s$ implies $V_r \cap V_s = \emptyset$. We refer to each V_r as a *region* in the graph G. Let n_r be the size of V_r.

Such a partition induces a partition on the set of arcs E, with partitions $E_{r,s} = \{(u, v) \in E \mid u \in V_r \wedge v \in V_s\}$ for all $1 \leq r, s \leq m$. Note that such sets may be empty. Each region V_r is associated with a set of *border nodes*, $V_r^{\mathrm{b}} \subseteq V_r$, defined as

$$V_r^{\mathrm{b}} \equiv \{u \mid \exists s \neq r, v \in V \text{ such that } \ (u, v) \in E_{r,s}\}.$$

Assuming that each partition V_s is a set of contiguous integers, the associated arc partition induces a block partition of the adjacency matrix,

$$\mathbf{A} \ = \ \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \cdots & \mathbf{A}_{1,m} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & \cdots & \mathbf{A}_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{m,1} & \mathbf{A}_{m,2} & \cdots & \mathbf{A}_{m,m} \end{bmatrix}.$$

Each $\mathbf{A}_{r,s}$ is the $n_r \times n_s$ matrix associated with the weights of the set of arcs $E_{r,s}$. Note that each sub-matrix $\mathbf{A}_{r,r}$ along the diagonal corresponds to a region in G (not required to be connected).

For example, suppose we partition the graph of Figure 2 into two regions, $\pi = \{V_1,\ V_2\}$, where $V_1 = \{1,\ 2,\ 3\}$ and $V_2 = \{4,\ 5\}$. The corresponding block-partitioned adjacency matrix is

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \quad \begin{array}{cc} 4 & 5 \end{array} \\ \left[\begin{array}{ccc|cc} \infty & 7 & 3 & 1 & \infty \\ 7 & \infty & 3 & 3 & 2 \\ 3 & 3 & \infty & \infty & \infty \\ \hline 1 & 3 & \infty & \infty & 7 \\ \infty & 2 & \infty & 7 & \infty \end{array}\right] \end{array}$$

Figure 3 presents this partitioned graph where dashed lines correspond to inter-region arcs.

Define the *region matrix*, $\mathbf{R}$, to be the block-diagonal matrix associated with all regions and intra-region arcs,

$$\mathbf{R} \ = \ \begin{bmatrix} \mathbf{A}_{1,1} & \overline{\mathbf{0}} & \cdots & \overline{\mathbf{0}} \\ \overline{\mathbf{0}} & \mathbf{A}_{2,2} & \cdots & \overline{\mathbf{0}} \\ \vdots & \vdots & \ddots & \vdots \\ \overline{\mathbf{0}} & \overline{\mathbf{0}} & \cdots & \mathbf{A}_{m,m} \end{bmatrix}.$$

Here, each $\overline{\mathbf{0}}$ represents a sub-matrix of the appropriate size containing only the value $\overline{0}$. Note that

$$\mathbf{R}^* \ = \ \begin{bmatrix} \mathbf{A}_{1,1}^* & \overline{\mathbf{0}} & \cdots & \overline{\mathbf{0}} \\ \overline{\mathbf{0}} & \mathbf{A}_{2,2}^* & \cdots & \overline{\mathbf{0}} \\ \vdots & \vdots & \ddots & \vdots \\ \overline{\mathbf{0}} & \overline{\mathbf{0}} & \cdots & \mathbf{A}_{m,m}^* \end{bmatrix}$$

which captures the solution to the routing problem within each region in isolation. How can these solutions to sub-problems be used to compute $\mathbf{A}^*$?

First, we define the *boundary matrix*, $\mathbf{B}$, to be the adjacency matrix of those arcs not included in $\mathbf{R}$,

$$\mathbf{B} \ = \ \begin{bmatrix} \overline{\mathbf{0}} & \mathbf{A}_{1,2} & \cdots & \mathbf{A}_{1,m} \\ \mathbf{A}_{2,1} & \overline{\mathbf{0}} & \cdots & \mathbf{A}_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{m,1} & \mathbf{A}_{m,2} & \cdots & \overline{\mathbf{0}} \end{bmatrix}.$$

Suppose that $\mathbf{R}^*$ has been computed, then we can derive a set of *virtual arcs* between border nodes within each region, which we call *transit arcs*. Given two border nodes in region r, $u, v \in V_r^{\mathrm{b}}$, if $\mathbf{A}_{r,r}^*(u,\ v) \neq \overline{\mathbf{0}}$, then $(u,\ v)$ is a transit arc in region r. Note that $(u,\ v)$ may or may not be in E. We can then define the *transit matrix* $\mathbf{T}$ to capture the weights of transit arcs,

$$\mathbf{T}(u,\ v) = \begin{cases} \mathbf{A}_{r,r}^*(u,\ v) & \text{if } (u, v) \text{ is a transit arc in region } r \\ \overline{\mathbf{0}} & \text{otherwise} \end{cases}$$

The border arcs together with the transit arcs define the *core graph* with respect to a partition π. Thus,

$$\mathbf{C} = \mathbf{B} \oplus \mathbf{T}$$

is the adjacency matrix of the core graph.

For our running example we have

$$\mathbf{R} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \quad \begin{array}{cc} 4 & 5 \end{array} \\ \left[\begin{array}{ccc|cc} \infty & 7 & 3 & \infty & \infty \\ 7 & \infty & 3 & \infty & \infty \\ 3 & 3 & \infty & \infty & \infty \\ \hline \infty & \infty & \infty & \infty & 7 \\ \infty & \infty & \infty & 7 & \infty \end{array}\right] \end{array}$$

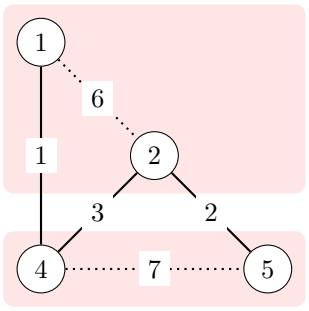


Figure 4: The core graph with virtual arcs.

and

$$\mathbf{B} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{ccc|cc} \infty & \infty & \infty & 1 & \infty \\ \infty & \infty & \infty & 3 & 2 \\ \infty & \infty & \infty & \infty & \infty \\ \hline 1 & 3 & \infty & \infty & \infty \\ \infty & 2 & \infty & \infty & \infty \end{array}\right] \end{array}$$

Figure 4 illustrates the core graph of the running example, where dotted lines are transit (virtual) arcs.

THEOREM 3.1.

$$\mathbf{A}^* = \mathbf{R}^* \oplus \mathbf{R}^*\mathbf{C}^*\mathbf{R}^* \tag{4}$$

Proof: From Conway [6] we have the identity

$$(\mathbf{X} \oplus \mathbf{Y})^* = \mathbf{X}^*(\mathbf{Y}\mathbf{X}^*)^*. \tag{5}$$

By construction, it is clear that $\mathbf{A} = \mathbf{R} \oplus \mathbf{B}$, and so we obtain $\mathbf{A}^* = \mathbf{R}^*(\mathbf{B}\mathbf{R}^*)^*$. From Equation (1) we have

$$(\mathbf{B}\mathbf{R}^*)^* = \mathbf{I} \oplus \mathbf{B}\mathbf{R}^* \oplus (\mathbf{B}\mathbf{R}^*)^2 \oplus \cdots \oplus (\mathbf{B}\mathbf{R}^*)^k \oplus \cdots$$

When we look at the $(i,j)-th$ entry of $(\mathbf{B}\mathbf{R}^*)^k$ we have a summation of product terms of the form

$$\mathbf{B}(i,q_1)\mathbf{R}^*(q_1,q_2)\mathbf{B}(q_2,q_3)\cdots\mathbf{B}(q_{k-1},q_k)\mathbf{R}^*(q_k,j),$$

where the q_t are ranging over all nodes. The only non-zero product terms are those where each $\mathbf{R}^*(q_t,q_{t+1})$ is not $\overline{0}$, and so (q_t,q_{t+1}) is a transit arc, and each $\mathbf{B}(q_s,q_{s+1})$ is not $\overline{0}$, and so (q_s,q_{s+1}) is a border arc. The only exception is the last expression in each term, $\mathbf{R}^*(q_k,j)$. Therefore $(\mathbf{B}\mathbf{R}^*)^* = \mathbf{I} \oplus \mathbf{C}^*\mathbf{R}^*$, because the sum over all such product terms is exactly the same as exploring all paths in the core graph. (End of proof.)

4. HYBRID PROTOCOLS

Equation (4) can be read as a scheme for the *decomposition* of the computation of $\mathbf{A}^*$:

1. Compute $\mathbf{R}^*$
2. Construct the matrix $\mathbf{C} = \mathbf{B} \oplus \mathbf{T}$
3. Compute $\mathbf{C}^*$
4. Construct $\mathbf{A}^*$ as $\mathbf{R}^* \oplus \mathbf{R}^*\mathbf{C}^*\mathbf{R}^*$.

Note that *different algorithms* can be used in the first and the third steps. In the context of network routing, it is interesting to consider how the work of the first and the third steps could be done by different *distributed* algorithms (routing protocols).

We first need to consider a more general problem. The solution to a matrix equation of the form

$$\mathbf{F} = \mathbf{A}\mathbf{F} \oplus \mathbf{M}$$

is $\mathbf{F} = \mathbf{A}^*\mathbf{M}$. In terms of routing, the matrix $\mathbf{M}$ could model a *mapping* where $\mathbf{M}(q,j)$ contains a metric associated with some destination j that is *external to* the graph represented by $\mathbf{A}$ [7].

In the context of Internet routing we want to compute $\mathbf{F}$ in a distributed manner. Let us model each router as vertex $i \in V$ Typically, the router associated with node i will only compute the i-th row $\mathbf{F}(i,\ \_)$. With this interpretation, the next-hops associated with the entries of $\mathbf{F}(i,\ \_)$ encode the *forwarding table* at router i. Matrix $\mathbf{M}$ could represent the attachment of static routes, or it might be produced as the forwarding table of another protocol [7].

A synchronous approximation to distributed algorithms in the Bellman-Ford family can be given by the following iterative algorithm.

$$\begin{array}{rcl} \mathbf{F}^{(0)} & = & \mathbf{M} \\ \mathbf{F}^{(k+1)} & = & \mathbf{A}\mathbf{F}^{(k)} \oplus \mathbf{M} \end{array}$$

At step $k+1$ node i can compute the i-th row of $\mathbf{F}^{(k+1)}$ as

$$\mathbf{F}^{(k+1)}(i,j) = (\bigoplus_q \mathbf{A}(i,q)\mathbf{F}^{(k)}(q,j)) \oplus \mathbf{M}(i,j)$$

Care must be taken to avoid *counting to infinity* in a distributed implementation. One easy solution is to include a path of router identifiers with each route announcement and eliminate those with loops (similar to ASPATHS in BGP).

In link-state protocols, the matrix $\mathbf{A}$ is constructed at each node via link-state flooding. Then each node i computes one row $\mathbf{A}^*(i,\ \_)$ using an efficient algorithm such as Dijkstra's SPF. In order to compute the i-th row of $\mathbf{F} = \mathbf{A}^*\mathbf{M}$, each node needs access to the mapping $\mathbf{M}$. In OSPF and IS–IS this kind of information is piggy-backed on link-state announcements.

We can now restate our decomposition in terms of equations that must be solved. First solve the region routing problem

$$\mathbf{F}_1 = \mathbf{R}\mathbf{F}_1 \oplus \mathbf{I}$$

for $\mathbf{R}^*$, then solve

$$\mathbf{F}_2 = \mathbf{C}\mathbf{F}_2 \oplus \mathbf{F}_1$$

for core routing $\mathbf{C}^*$ and for exporting region routes to the core $\mathbf{C}^*\mathbf{R}^*$. Finally, solve

$$\mathbf{F} = \mathbf{F}_1 \oplus \mathbf{F}_1\mathbf{F}_2.$$

for importing region external routes to regions $\mathbf{R}^*(\mathbf{C}^*\mathbf{R}^*)$. If core routing is done via link-states, then the entries of $\mathbf{F}_1$ can be disseminated across the core as piggy-backed otherwise by iteration. Similarly, $\mathbf{F}_2$ can be disseminated across regions.

5. COST MODEL AND SIMULATION

We are interested in estimating asymptotic space and computation costs for four combinations of link-state with the SPF and path-vector with the DBF (*viz.*, the Dijkstra's SPF both inside regions and in the core (D-over-D), the Dijkstra's SPF inside regions and the distributed Bellman-Ford in the core (B-over-D), the distributed Bellman-Ford inside regions and the Dijkstra's SPF in the core (D-over-B), and the distributed Bellman-Ford both inside regions and in the core (B-over-B)). We will ignore the communication costs of disseminating routing information across the network. Suppose the graph $G = (V, E)$ has $n = |V|$ nodes and $a = |E|$ arcs. Let the graph be partitioned into m regions such that region r has n_r nodes and a_r arcs. Also let there be b_r boundary nodes in region r with a total of $n_\mathrm{b} = \sum b_r$ boundary nodes, and a_b inter-region arcs. According to Section 3, the core graph has n_b nodes and at most $a_\mathrm{c} = a_\mathrm{b} + \sum b_r(b_r - 1)$ arcs when all regions are connected, thus region r has $b_r(b_r - 1)$ transit arcs.

5.1 Analytic Cost Model

In distributed algorithms, $\mathbf{R}^*$ is computed by computing $\mathbf{A}^*_{r,r}$ for partition r independently and putting them together. Each node computes one row of $\mathbf{A}^*_{r,r}$ using either the SPF or the DBF algorithm. Since all the nodes run the routing algorithm simultaneously, so the worst-case computation cost is the computation cost of a node in the largest region, G_{max}. Suppose $t(G_r, X)$ is the computation cost of running routing algorithm X at a node in the graph G_r, $t_\mathrm{x}(\pi, X)$ is the maximum computation cost at any border node to export region internal routes to the core graph using algorithm X, and $t_\mathrm{i}(\pi, X)$ is the maximum computation cost at any non-border node to import region external routes using algorithm X. Then the distributed computation cost of our hybrid algorithms can be estimated with

$$\tau(\pi, X, Y) = t(G_{max}, X) + t(G_\mathrm{c}, Y) + t_\mathrm{x}(\pi, Y) + t_\mathrm{i}(\pi, X) \quad (6)$$

where G_{max} is the largest region sub-graph, G_c is the core graph, X and Y are instances of either the SPF or the DBF algorithms and X is used inside regions and Y is used in the core graph.

By substituting t with s for space in above formulation, we can estimate the distributed space cost of our hybrid algorithm with

$$\sigma(\pi, X, Y) = s(G_{max}, X) + s(G_\mathrm{c}, Y) + s_\mathrm{x}(\pi, Y) + s_\mathrm{i}(\pi, X) \quad (7)$$

We know the computation complexity of the SPF algorithm is $O(n \ln n + a)$ and the DBF algorithm is $O(dn^2)$ where d is the maximum degrees of neighborhood of a node. Computation cost of the SPF for exporting routes to the core can be estimated to $O(\sum b_r n_r)$ $\approx mb_r n_r$ as each border node exports n_r routes to non-border nodes from region r and there are b_r border nodes in region r. Similarly, import cost can be estimated to $O(b_{r_1} \sum n_{r_2}) \approx O(b_{r_1} n)$ as each border node in region r_1 imports all routes across the network $\sum n_{r_2} = n$ into region r_1 and there are b_{r_1} border nodes in region r_1. For the DBF algorithm, export cost is $O(d \sum b_r n_r)$ and import cost is $O(db_{r_1} \sum n_{r_2}) \approx O(db_{r_1} n)$ since a node processes routes sequentially from all of its neighbors in the DBF. Table 1 summarizes the computation costs of running the SPF and the DBF algorithms.

X	$t(G, X)$	$t_\mathrm{x}(\pi, X)$	$t_\mathrm{i}(\pi, X)$
Dijkstra (D)	$n \ln n + a$	$mb_r n_r$	$b_r n$
Bellman-Ford (B)	dn^2	$dmb_r n_r$	$db_r n$

Table 1: Computation costs (in big O).

Again, we know the space complexity of the SPF algorithm is $O(n+a)$ and the DBF algorithm is $O(n+d)$. We can estimate the space requirement of link-state mechanism for export operation as $O(\sum b_r n_r)$ since a border node needs to store link-state information for all region internal routes announced by all border nodes. Similarly, we can estimate the space requirement of link-state mechanism for import operation as $O(b_{r_1} \sum n_{r_2}) \approx$ $O(b_{r_1} n)$ since a non-border node needs to store link-state information for all routes announced by all border nodes in the region. For the DBF algorithm, the space requirement for import and export operations is to store routes to all nodes (*i.e.*, $\sum n_r = n$). Table 2 summarizes the space costs (in big O) for running the SPF and the DBF algorithms. By substituting values of X and

X	$s(G, X)$	$s_\mathrm{x}(\pi, X)$	$s_\mathrm{i}(\pi, X)$
Dijkstra (D)	$n + a$	$mb_r n_r$	$b_r n$
Bellman-Ford (B)	$n + d$	n	n

Table 2: Space costs (in big O).

Y and values of $t(G, X)$, $t_\mathrm{x}(\pi, X)$, and $t_\mathrm{i}(\pi, X)$ from Table 1 to Equation (6), we have computation costs (in big O) of our hybrid algorithms as shown in Table 3.

Similarly, by substituting values of X and Y and values of $s(G, X)$, $s_\mathrm{x}(\pi, X)$, and $s_\mathrm{i}(\pi, X)$ from Table 2 to

Y-over-X	Computation cost $\tau(G, X, Y)$
D-over-D	$n_r \ln n_r + a_r + n_\mathrm{b} \ln n_\mathrm{b} + a_\mathrm{c} + b_r n + m b_r n_r$
B-over-D	$n_r \ln n_r + a_r + b_r n + d_\mathrm{c} n_\mathrm{b}(n_\mathrm{b} + n)$
D-over-B	$d_r n_r(n_r + n) + n_\mathrm{b} \ln n_\mathrm{b} + a_\mathrm{c} + m b_r n_r$
B-over-B	$d_r n_r(n_r + n) + d_\mathrm{c} n_\mathrm{b}(n_\mathrm{b} + n)$

Table 3: Hybrid computation costs (in big O).

Equation (7), we have the space costs (in big O) of our hybrid algorithms as shown in Table 4.

Y-over-X	Space cost $\sigma(\pi, X, Y)$
D-over-D	$n_r + a_r + n_\mathrm{b} + a_\mathrm{c} + m b_r n_r + b_r n$
B-over-D	$n_r + a_r + b_r n + d_\mathrm{c} + n_\mathrm{b} + n$
D-over-B	$d_r + n_r + n_\mathrm{b} + a_\mathrm{c} + m b_r n_r + n$
B-over-B	$d_r + n_r + d_\mathrm{c} + n_\mathrm{b} + n$

Table 4: Hybrid space costs (in big O).

5.2 Analytic plots

Since there are many parameters in our cost model, we analyze the behavior of our system with the following assumptions:

$d_r = d_\mathrm{c} = d \ll n$	d is a small constant
$a = nd$	a is constant multiple of n
$n_r = n/m$	partitions are of equal size
$a_r = nd/m$	no. of arcs in a partition
$b_r = c \ll n_r$	b_r is a small constant
$n_\mathrm{b} = m b_r = mc$	total no. of border nodes
$a_\mathrm{b} = md$	a_b is constant multiple of m

Note that for a fixed-sized network, as the number of partitions m increases, the size of regions n/m decreases and the size of the core cm increases. As a result, the computation cost of regions routing decreases while the computation cost of core routing increases. The computation cost for exporting region internal routes to the core and the computation cost for importing region external routes to regions do not change very much as the number of partitions increases.

Fig. 5 presents the space-time complexity plots for four hybrid algorithms for $n = 100$, $d = 5$, and $c = 2$. The x-axis in each of the plots represents the number of (almost) equally sized partitions. That is, when x is 0, there are no partitions (or the network is one large partition), whereas for $x = 25$ the network has 25 partitions with 4 nodes in each partition.

In Fig. 5(a), we observe that with partitioning the overall computation cost of D-over-D does not change very much, but there is a significant reduction in overall space requirement for some small number of partitions. In contrast, there is a significant reduction in overall computation cost of B-over-B for some small number of partitions since for small m, n^2/m^2 is larger than mn. However, overall space requirement does not change very much as the number of partitions increases as seen in Fig. 5(d).

We know that the SPF is fast and requires large space, while the DBF is slow but requires low space. Fig. 5(b) and Fig. 5(c) present performance when these very different algorithms are combined into one hybrid algorithm. They clearly illustrate the trade-offs between time and space. For example, in Fig. 5(b), as we increase the number of partitions, we see a clear improvement in space utilization at the cost of computation time. Similarly, in Fig. 5(c) we see a clear benefit in computation time at the cost of increased space utilization. In both cases the trade-offs are more dramatic for some small number of partitions.

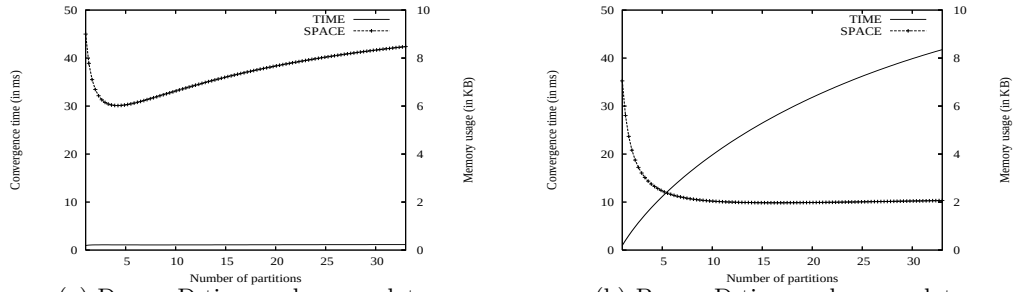
5.3 Simulation

In order to examine the behavior, we have implemented simulators of our hybrid algorithms. We have two subroutines, one for the SPF algorithm and the other for the Bellman-Ford algorithm, which take a graph and a source node to compute the SPT rooted at the source node and return the routing table. Algorithm X is used inside regions and algorithm Y in the core graph. Algorithm 1 lists the pseudo-code for simulation of our hybrid algorithms, where $\mathbf{A}_r^*$ is the region routing table at a node in region r, $\mathbf{A}_\mathrm{c}^*$ is the core routing table at a border node. $\hat{\mathbf{A}}_r^*$ is the routing table at a border node in region r after export operation and $\bar{\mathbf{A}}_r^*$ is the routing table at a non-border node after import operation.

Algorithm 1 Hybrid simulation algorithm.

```
    for each region G_r of G do
 2:   for each node v of G_r do
        A*_r = X(G_r, v)
 4: for each region G_r of G do
      for all border nodes (u, v) of G_r do
 6:     if A*_r(u, v) ≠ ∞ then
          Add virtual arc A*_r(u, v) to G_c
 8: for each node v of G_c do
      A*_c = Y(G_c, v)
10: for all regions G_{r1}, G_{r2} of G, r1 ≠ r2 do
      for all border nodes u of G_{r1} and v of G_{r2} do
12:     for each leaf node w of G_{r2} do
          Â*_{r1}(u, w) = A*_c(u, v) ⊗ A*_{r2}(v, w)
14: for each region G_r of G do
      for each leaf node u and border node v of G_r do
16:     for each route Â*_r(v, w), w ≠ u do
          Ā*_r(u, w) = R*_r(u, w)⊕
18:                    A*_r(u, v) ⊗ Â*_r(v, w)
```

For the link-state algorithm, we have constructed link-state packets for all the nodes and links of the input



(a) D-over-D time and space plots

(b) B-over-D time and space plots

(c) D-over-B time and space plots

(d) B-over-B time and space plots

Figure 5: Analytic time-space plots for fixed-size networks against increasing number of partitions.

graph, stored them in a link-state database, and measured space taken by the database. We have run the SPF algorithm on the link-state database for each node in the graph and taken the maximum amount of time of all the nodes. On the other hand, for the path-vector algorithm, we have measured space requirements for storing links to neighboring nodes and routes to all the destinations and we have taken the maximum of all nodes. We have run Bellman-Ford algorithm on the input graph and taken the maximum of the convergence time of all the nodes. We have added times and spaces respectively from each step to estimate the total amount of time and space needed by each of our four algorithms.

5.3.1 *Topology generation*

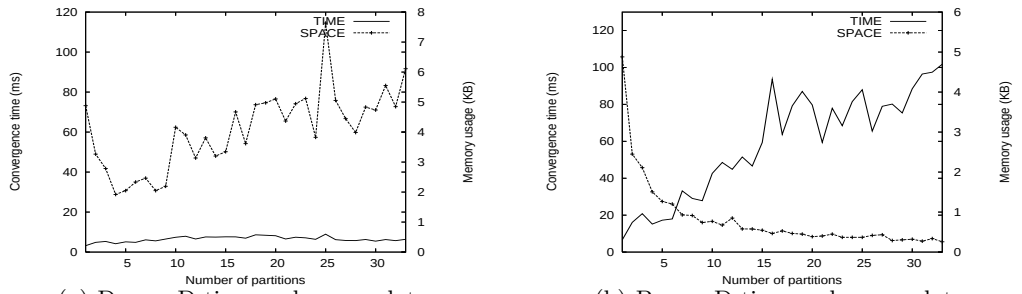
We have used BRITE [8] topology generator for generating topologies for our simulation. For a given number of nodes, say n, we have generated n graphs with 1 to n partitions. We have used BRITE's Router Waxman model with n routers for no partitioning, for n partitions, we have used AS Waxman model, and for other partitioned graphs, we have used Top-Down model. Note that for m partitions when n is not an integer multiple of m, we have first generated a graph with m regions where each region has $\lfloor n/m \rfloor$ nodes and then randomly distributed n mod m nodes to different partitions.

5.3.2 *Simulation results*

We have simulated the system for a fixed number of nodes and varying the number of partitions from one to number of nodes in the partitions and recorded convergence time and memory usage for each combination. Fig. 6 plots times and space against number of partitions for 100 nodes of our hybrid algorithms. x-axis represents the number of partitions, left y-axis represents convergence time, and right x-axis represents memory usage. It is observed that simulation plots are similar to that of analytical plots of Fig. 5. For some small number of partitions, 4 - 9 in Fig. 6 convergence times and memory usage of our hybrid algorithms are minimum.

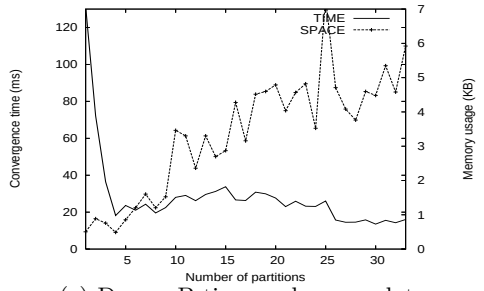
As we have discussed earlier, the convergence time does not change very much for D-over-D (Fig. 6(a)), but the memory usage is minimum for some small number of partitions, here 4-9 partitions for 100 nodes. Therefore, if we want to design a network for fast convergence and consider space, we should use D-over-D . It is clear from Fig. 6(d) that memory usage for B-over-B , which is very low, does not vary significantly, but the convergence time drops sharply as the number of partitions increases. Still convergence time of B-over-B is much higher than that of D-over-D . Therefore, if memory is our main concern we could use B-over-B to get some convergence time benefit by partitioning instead of using a single path-vector domain.

Fig. 6(b) indicates that the memory usage falls sharply as the number of partitions increases, at the same time the convergence time increases slowly for B-over-D . On the other hand, Fig. 6(c) indicates that the convergence time falls sharply and the memory usage increases rapidly for D-over-B as the number of partitions increases. Therefore, both D-over-B and B-over-D have significant trade-offs in time and space as number of

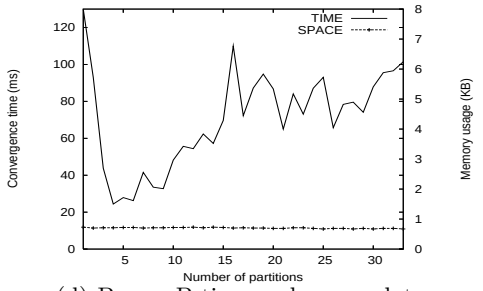


(a) D-over-D time and space plots

(b) B-over-D time and space plots



(c) D-over-B time and space plots



(d) B-over-B time and space plots

Figure 6: Simulation time-space plots for fixed-size networks against increasing number of partitions.

partitions increases.

6. DISCUSSION

Recently, a hybrid link-state path-vector protocol for inter-domain routing was proposed, HLP [9]. We leave it as an open problem whether or not HLP can be modelled algebraically. The difficulty here is that BGP metrics cannot be modelled with semirings because algebraic realistic models of BGP do not obey the distributivity properties. This would invalidate much of the reasoning done in Sections 2 and 3. Furthermore, the partitions in HLP actually overlap — HLP is essentially a link-state domain for each "customer/provider cone" in the network and path-vectoring domain with respect to peering links. Routers can be in multiple cones, and link-state flooding is restricted to conform to a simple model of commercial relationships. The shortcoming of this approach is that it ties the mechanism very tightly to an over-simplified model of inter-domain policies. It is difficult to see how to extend such a model to the more expressive, but still safe, policies described in [10].

Acknowledgement

The first author has been supported by grants from Boeing and Cisco Systems, while the second author is partially supported by those grants and EPSRC (UK) grant EP/F002718/1. Both authors would like to thank Alexander Gurney, Vilius Naudžiūnas, and Philip Taylor for their helpful discussions and feedback.

7. REFERENCES

[1] R. Perlman. *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols*. Addison-Wesley Professional, 2 edition, 2000.

[2] Bernard Fortz. Internet traffic engineering by optimizing OSPF weights. In *in Proc. IEEE INFOCOM*, pages 519–528, 2000.

[3] Bernard Fortz, Jennifer Rexford, and Mikkel Thorup. Traffic engineering with traditional IP routing protocols. *IEEE Communications Magazine*, 40:118–124, 2002.

[4] M. Gondran and M. Minoux. *Graphs, Dioids, and Semirings*. Springer, 2008.

[5] J. S. Baras and G. Theodorakopoulos. *Path problems in networks*. Morgan & Claypool, 2010.

[6] J. H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.

[7] John N. Billings and Timothy G. Griffin. A model of internet routing using semi-modules. In *RelMiCS*, 2009.

[8] A. Medina et al. BRITE: Universal topology generation from a user's perspective. Technical report, Boston University, USA, April 2001.

[9] L. Subramanian et al. HLP: A Next Generation Inter-domain Routing Protocol. In *Proc. ACM SIGCOMM*, 2005.

[10] Timothy G. Griffin. The stratified shortest-paths problem. In *COMSNETS*, 2010.