Provably Good Moat Routing

Joseph L. Ganley

Cadence Design Systems, Inc., 2615 John Milton Drive, Oak Hill, Virginia 20171

James P. Cohoon

Department of Computer Science, University of Virginia, Charlottesville, Virginia 22903

Abstract

Moat routing is the routing of nets between the input/output pads and the core circuit. In this paper, it is proved that moat routing is NP-complete under the routing model in which there are no vertical conflicts and doglegs are disallowed (i.e., every net is routed within a single track). This contrasts with the fact that channel routing is efficiently solvable under these restrictions. The paper then presents an approximation algorithm for moat routing that computes moat routing solutions that are guaranteed to use at most three times the optimal number of tracks. Empirical results are presented indicating that for a number of industrial benchmarks, the algorithm produces solutions that are near optimal and that use significantly fewer tracks than previous moat routing strategies.

Keywords: Approximation algorithms, computational complexity, moat routing.

1 Introduction

The final stage in detailed routing is typically to route the connections between the input/output pads and the core circuit. The area between the core and the pads is called the *moat*, and this routing task is consequently called *moat routing*.

A moat routing instance consists of a number of nets whose pins lie on either or both of the inside perimeter of the padframe and the outside perimeter of the core circuit area. The moat between the pads and the core is divided into a number of concentric *tracks*, similar to a channel routing instance except that each track forms a circle rather than a line segment. A set of pads, a circuit core, and the moat between them are illustrated in Figure 1.

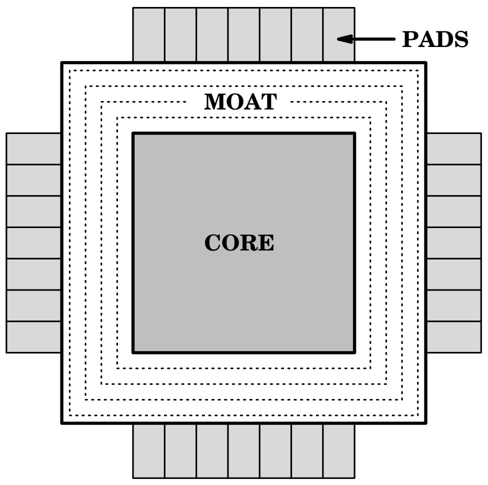

Fig. 1. A moat routing instance.

In this paper we assume the use of a routing model in which doglegs are not allowed, i.e., every net is routed within a single track. Furthermore, we assume that there are no vertical constraints (as Wang [13] points out, the pins on the pads are typically spaced sufficiently far apart that any vertical constraints can be eliminated). We henceforth refer to this model simply as the *restricted routing model*.

As in previous works [10,13], we use a two-layer model in which the tracks lie in one layer and the radial connections between the core or pad pins and the tracks lie in the other layer.

The remainder of this paper is organized as follows. Section 2 describes some concepts regarding intersection graphs and channel routing algorithms. In Section 3, it is proved that under the restricted routing model, moat routing is NP-complete (whereas the restricted routing model renders channel routing efficiently solvable). Section 4 describes a technique for determining a lower bound on the number of tracks required for a given moat routing instance. Section 5 then describes an approximation algorithm that computes a moat routing that uses at most three times the optimal number of tracks. In Section 6, empirical evidence is provided indicating that for a number of industrial benchmarks, the approximation algorithm performs well with respect to lower bounds and previous moat routing strategies. Finally, Section 7 concludes with some ongoing work.

2 Terminology

The *graph K-colorability* problem is defined as follows: given a graph, is it possible to assign a color to each vertex such that at most K colors are used and such that the endpoints of every edge are colored differently?

An *interval graph* is a graph in which the vertices correspond to intervals on a line and in which there is an edge between every pair of vertices whose intervals intersect[1]. Under the restricted routing model, the channel routing problem corresponds directly to the problem of coloring an interval graph. Each interval corresponds to a net, and its endpoints are the minimum and maximum x coordinates of the pins in the net (assuming without loss of generality that the channel is horizontal). A K-coloring of this interval graph corresponds directly to a channel routing solution using K tracks, and vice versa. Each color corresponds to a track, and since no pair of intervals of the same color intersect, all intervals of like color can be routed within a single track. The K-coloring problem can be efficiently solved in an interval graph, and thus a channel routing solution that is optimal within the restricted routing model can be efficiently computed.

One classic algorithm that does so is the *left-edge algorithm* of Hashimoto and Stevens [8]. The left-edge algorithm proceeds as follows: sort the intervals according to the x coordinates of their left endpoints. Then process the nets in this sorted order, inserting intervals into tracks in a greedy fashion: each interval is inserted into the first track in which it fits, or if it fits in none of the current tracks, then it is inserted into a new track.

The *density* of a channel routing instance is the maximum number of intervals that intersect any vertical line, i.e., the size of a maximum clique in the corresponding interval graph. Clearly the density of an instance is a lower bound on the number of tracks required to route the instance. In fact, interval graphs are *perfect graphs*, meaning that the maximum clique size is equal to the minimum number of colors required to color the graph. Thus, the optimal number of tracks is precisely equal to the density, and the left-edge algorithm computes an optimal channel routing solution under the restricted routing model.

A *circular arc graph* is similar to an interval graph except that the vertices correspond to arcs on a circle rather than intervals on a line. Unlike interval graphs, circular arc graphs are not perfect, meaning that the minimum number of colors required to color a circular arc graph is not necessarily equal to its density (though density is still clearly a lower bound). One might suspect that moat routing is analogous to coloring a circular arc graph in the same manner that channel routing is analogous to coloring an interval graph. In the next section, we prove that a moat routing algorithm can indeed be used to solve the K-colorability problem in a circular arc graph. Unfortunately, since circular arc graph coloring is NP-complete [5], this reduction implies

[1] All graph-theoretical concepts discussed here are described in Golumbic [6].

that moat routing under the restricted routing model is NP-complete as well. This contrasts with the fact that channel routing is efficiently solvable.

3 Computational Complexity

In this section, we prove that a moat routing algorithm can be used to solve the K-coloring problem in a circular arc graph. Since coloring a circular arc graph is NP-complete [5] and there is a polynomial-time reduction from circular arc graph coloring to moat routing, the moat routing problem is NP-complete as well.

Theorem 1 *Moat routing is NP-complete under the restricted routing model.*

Proof. Inclusion in NP is obvious. NP-completeness is shown by a reduction from circular arc graph coloring.

An instance of the coloring problem for circular arc graphs is a circular arc graph G and an integer K. The question is whether G can be colored using K or fewer colors.

Assume without loss of generality that the endpoints of the arcs that comprise G are all distinct [7]. Sort the endpoints of the n arcs in clockwise order, and call them $p_1, p_2, \ldots, p_{2n}$. We now build a moat routing instance that mirrors the structure of G.

All pins in the moat routing instance lie on the perimeter of the core. They are arranged in $2n$ groups, each of which contains at most n pins. The groups are arranged around the core perimeter in the same order as the corresponding points on the circle. Let t_{ij} denote the j^{th} pin in the i^{th} group of pins. A net R_j is built for each arc $a_j = (p_L, p_R)$ in G, in which the pins are t_{ij} for all $p_L \leq i \leq p_R$ (index arithmetic is performed modulo $2n$, i.e., $a < b$ is understood to imply that a is counterclockwise of b). This construction is illustrated in Figure 2.

We claim that a K-track moat routing solution to the construction described above exists if and only if G is colorable using K colors.

- ($\Rightarrow$) *If G is colorable using K colors, then a K-track solution exists for the moat routing instance constructed as described above.* For each set of arcs given the same color, route the corresponding nets within a single track clockwise from p_L to p_R. Such a routing is valid since no two arcs with the same color intersect.

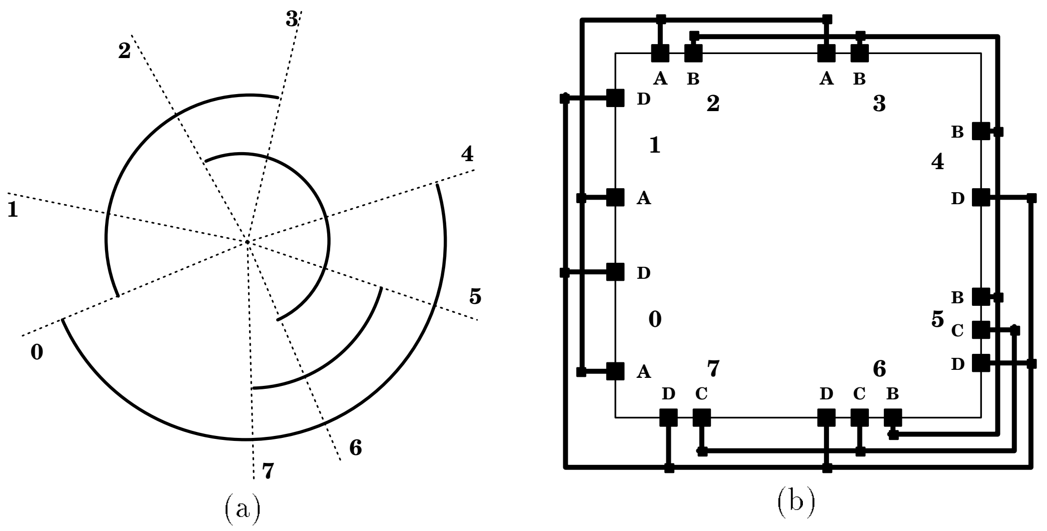

Fig. 2. (a) A 3-colorable circular arc graph and (b) a corresponding 3-track moat routing instance.

($\Leftarrow$) *If a K-track solution exists for the moat routing instance constructed as described above, then G is colorable using K colors.* The routing for each net is an entire track minus the section between two adjacent pins. If the missing section lies between pins t_{ij} and $t_{i+1,j}$ such that $p_L \leq t_{ij} < t_{i+1,j} \leq p_R$, then the net must intersect the routing of every other net, and thus it is routed within its own distinct track. Thus, any net whose routing does not correspond to the circular arc from which it was constructed can be rerouted so that it does correspond to its circular arc without increasing the number of colors required. Therefore, we can transform any K-track moat routing solution into a K-track solution in which the routing of each net corresponds directly with the circular arc from which it was constructed. Each arc is then given a color corresponding to the track in which its corresponding net is routed, and the resulting coloring is a valid K-coloring of G.

This transformation is performed in $O(n^2)$ time. Since the circular arc graph coloring problem is NP-complete [5] and is reducible in polynomial time to the moat routing problem, the moat routing problem is NP-complete. $\square$

4 Lower bounds

Due to the circular nature of the moat, many ideas from the channel routing literature cannot be applied directly. In a (horizontal) channel, a routing algorithm must assign a track to each net, but the horizontal span of the routing is determined by the instance, i.e. there is only one possible horizontal span for each net. For moat routing, the circular moat creates a possibility of many different routing paths, independent of the assignment of nets to tracks. Specifically, a net with m pins can be routed in m different ways within the same track, each corresponding to a complete track with the span between two adjacent pins removed.

Since nets in a moat routing instance can be routed in multiple ways, the notion of density as it applies to channel routing cannot be directly generalized to moat routing. However, we derive a slightly different lower bound by considering pairs of points around the moat. Consider a pair of lines, each of which extends from the core circuit area perpendicular to its border. Each such pair separates the moat into two channels.

Find a pair of such lines such that the number of nets with pins in both channels is maximum. We say that these nets are *cut* by the pair of lines, and denote the maximum number of cut nets as N. Every routing of a cut net must intersect at least one of the lines, so the minimum number of tracks required for the moat routing is at least $\lceil N/2 \rceil$.

5 An Approximation Algorithm

Further exploration of the ideas in the previous section leads to an approximation algorithm for the moat routing problem. (This is a modified version of the algorithm presented by the authors in [4].) As in Section 4, find a pair of lines that cuts a maximum number of nets. These lines divide the moat into a pair of channels; call them the *left* and *right* channels. If a net has pins in both of these channels, then we say it is *cut*; otherwise it is *uncut*.

Choose an arbitrary cut, and let N denote the set of cut nets. Now route every uncut net within the channel that contains its pins (using, for example, the left-edge algorithm [8]), and route the cut nets arbitrarily. Let k denote the number of tracks in an optimal moat routing, and let X be the set of uncut nets for which the optimal routing is not within the cut channel that contains its pins. The moat width incurred by the uncut nets is thus at most $k + |X|$, and obviously the moat width incurred by the cut nets is at most $|N|$.

For a lower bound, note that every net in X, since it is not routed within its cut channel, must cross both lines in the cut in an optimal moat routing, and thus contributes 1 to the lower bound. Each net in N must cross at least one of the lines in the cut, and thus the nets in N contribute $\lceil |N|/2 \rceil$ to the lower bound.

Using these upper and lower bounds, we determine that the worst-case ratio of the moat width produced by this approximation algorithm to the optimal moat width is

$$\frac{k + |X| + |N|}{k} \leq 1 + \frac{|X| + |N|}{|X| + \lceil |N|/2 \rceil} \leq 3.$$

That is, this approximation algorithm produces a solution whose moat width is at most 3 times optimal.

If there are n nets, then this algorithm requires $O(n \log n)$ time.

Though an arbitrary routing of the cut nets is sufficient to satisfy the approximation bound, in practice one would like to compute a good routing. We turn once again to circular arc graphs to devise an effective heuristic for routing the cut nets.

Construct a circular arc graph G as follows: For each net R_i, denote the pins in R_i as $t_0, t_1, \ldots, t_{|R_i|-1}$, in clockwise order. For each R_i, add to G the arcs $[t_{(j+1) \bmod |R_i|}, t_j]$ for all $0 \leq j < |R_i|$. Note that each arc includes all the pins in R_i. Note also that if $|R_i| > 2$, then the arcs constructed from R_i are all pairwise intersecting. If the n nets contain a total of m pins, then G contains m arcs.

Now find a *maximum independent set (MIS)* in G. There are m arcs in G, so if the arc endpoints are sorted (requiring $O(m \log m)$ time) then an MIS can be computed in $O(m)$ time [9]. Since all intervals in a net with 3 or more pins are pairwise intersecting, the MIS cannot contain more than one arc from a net R_i unless $|R_i| = 2$. If it contains both arcs from a net R_i with $|R_i| = 2$, then the size of the MIS is 2. In this case, if the arc endpoints are sorted, then an MIS that does not contain two arcs from the same net can be found in $O(n)$ time. This is dominated by the general case, so an MIS that does not contain 2 arcs from the same net is computed in $O(m \log m)$ time.

Once the MIS has been computed, route each net for which there is an arc in the MIS according to that arc. Remove all arcs that were constructed from nets thus routed, and repeat the process until no arcs remain. We call this heuristic the *iterated maximum independent set (IMIS)* heuristic. Since sorting requires $O(m \log m)$ time, each pass requires $O(m)$ time, and at most $O(n)$ passes are performed, the IMIS heuristic runs in $O(m \log m + nm)$ time.

In order to better use the space among the uncut nets, we have implemented the algorithm as follows: G contains at least one arc for every net. For a cut net, G contains the arcs corresponding to every possible routing path, as described above. For an uncut net, G contains only the single arc corresponding to its routing within the left or right channel.

Using the IMIS heuristic to route the nets in the approximation algorithm increases the time complexity of the approximation algorithm to $O(n \log n +$

			Lower	Approximation		Left Edge	
Name	Nets[3]	Pins[3]	Bound	Tracks	Time	Tracks	Time
biomed	40	138	13	14	0.02	16	0.01
fract	9	33	2	2	0.01	2	0.01
industry1	237	1086	50	58	0.51	69	0.09
industry1a	112	688	35	40	0.17	45	0.02
industry2	36	530	10	10	0.05	10	0.01
industry3	89	466	40	68	0.26	70	0.01
primary1	71	211	9	10	0.02	11	0.01
primary2	80	298	11	13	0.04	15	0.01
struct	30	97	8	9	0.01	9	0.01

Table 1. Experimental results.

$m \log m + nm) = O(m \log m + nm)$.

6 Experimental Results

We have implemented our approximation algorithm in order to compare its performance in practice with the lower bounds described in Section 4 and with the performance of previous moat routing strategies.

For the latter comparison, we implemented an algorithm using a strategy similar to that of Wang [13]. The main idea in Wang's algorithm is to choose the *shortest* routing path for each net, and then to assign these routing paths to tracks using the left-edge algorithm.

The circuits we use are the SIGDA standard-cell benchmark suite [11]. The moat routing instances are determined by performing placement and global routing using TimberWolf version 6.0 [12].

The results of these tests are summarized in Table 1. As these results indicate, the approximation algorithm produces moat routing solutions that are nearly optimal and that in most cases use substantially fewer tracks than the left-edge strategy. In fact, for these benchmarks, the approximation algorithm produces solutions that use an average of 6.5% fewer tracks than the left-edge algorithm.

The improved moat routing solutions computed by our approximation algo-

[3] Statistics are for the moat routing instance, not the entire circuit.

rithm come at the expense of a slight increase in running time compared to the left-edge algorithm. However, the algorithm is still quite fast, especially relative to the time required for the entire placement and routing process.

Figure 3 shows a moat routing of the benchmark instance primary2, computed using our approximation algorithm.

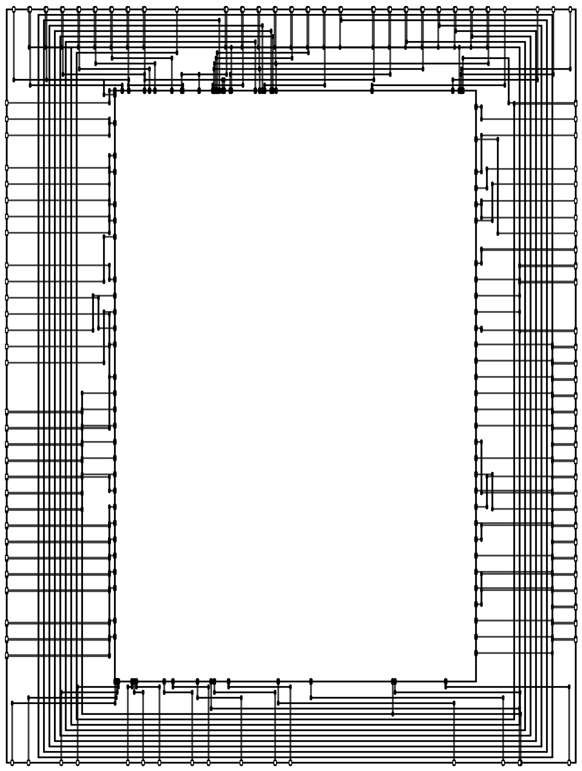

Fig. 3. A moat routing of the benchmark primary2, computed by the approximation algorithm. For illustrative purposes, the width of the moat is exaggerated relative to the size of the circuit core.

7 Current Work

Though we have shown that our approximation algorithm quickly produces near-optimal solutions for a number of industrial benchmarks, we are currently working on several modifications to improve its effectiveness further.

We have used the two-layer restricted routing model in which doglegs are disallowed, i.e. every net is routed in one track. Further reductions in track usage may result from allowing doglegs and supporting routing models with more than two layers.

An instance is *pad bound* if the moat routing can be accomplished within the smallest possible padframe. Otherwise, the instance is *core bound.* These two types of instance are illustrated in Figure 4.

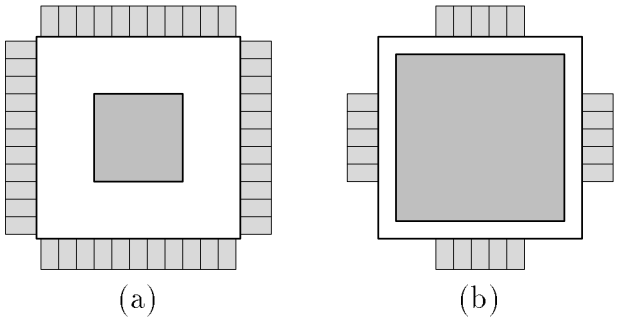

Fig. 4. (a) A pad-bound instance and (b) a core-bound instance.

High-quality moat routing of pad-bound instances is often less important than for core-bound instances, since the moat width is determined by the size of the padframe rather than by the moat routing itself. However, often an instance is pad-bound in one direction but not in the other (for example, the primary2 instance is pad-bound horizontally but not vertically, as can be seen in Figure 3). In such instances, more nets should be routed in the portions of the moat whose width is determined by the padframe, so that fewer nets need be routed in the portions that are not pad-bound, thus reducing the total circuit area.

More generally, we are working on modifying our algorithm to allow the use of an uneven number of tracks on the four sides of the moat.

Finally, we are examining special cases of the moat routing problem. Of particular interest is the special case in which every net contains two pins. We have been unable to modify the NP-completeness proof of Section 3 to cover this case, and we are working toward either doing so or devising a polynomial-time algorithm for moat routing of two-pin nets. We note that a similar problem, minimum-congestion routing in a cycle, is polynomial-time solvable for two-pin nets [1] but NP-complete for nets with more than two pins [4].

Acknowledgments

The authors were partially supported by National Science Foundation grants MIP–9107717 and CCR–9224789, and JLG was partially supported by a Virginia Space Grant Fellowship. Their support is greatly appreciated. In addition, we are grateful to the anonymous referees for many helpful suggestions, in particular for pointing out reference [9], which greatly improved the running time of our algorithm.

A previous version of this paper appeared in *Proceedings of the Sixth Great Lakes Symposium on VLSI* [3].

References

[1] A. Frank, T. Nishizeki, N. Saito, H. Suzuki, and É. Tardos. Algorithms for routing around a rectangle. *Discrete Applied Mathematics*, 40:363–378, 1992.

[2] J. L. Ganley. *Geometric Interconnection and Placement Algorithms.* PhD thesis, Department of Computer Science, University of Virginia, Charlottesville, Virginia, 1995.

[3] J. L. Ganley and J. P. Cohoon. A provably good moat routing algorithm. In *Proceedings of the Sixth Great Lakes Symposium on VLSI*, pages 86–91, 1996.

[4] J. L. Ganley and J. P. Cohoon. Minimum-congestion hypergraph embedding in a cycle. *IEEE Transactions on Computers*, 46:600–602, 1997.

[5] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic and Discrete Methods*, 1:216–227, 1980.

[6] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, New York, 1980.

[7] U. I. Gupta, D. T. Lee, and J. Y.-T. Leung. Efficient algorithms for interval graphs and circular arc graphs. *Networks*, 12:459–467, 1982.

[8] A. Hashimoto and J. Stevens. Wire routing by optimizing channel assignment within large apertures. In *Proceedings of the Eighth Design Automation Conference*, pages 155–163, 1971.

[9] W.-L. Hsu and K.-H. Tsai. Linear time algorithms on circular-arc graphs. *Information Processing Letters*, 40:123–129, 1991.

[10] R. K. McGehee. A practical moat router. In *Proceedings of the Twenty-fourth Design Automation Conference*, pages 216–222, 1987.

[11] B. T. Preas. Benchmarks for cell-based layout systems. In *Proceedings of the Twenty-fourth Design Automation Conference*, pages 319–320, 1987.

[12] C. Sechen and A. Sangiovanni-Vincentelli. The TimberWolf placement and routing package. *IEEE Journal of Solid-State Circuits*, 20:510–522, 1985.

[13] D. C. Wang. Pad placement and ring routing for custom chip layout. In *Proceedings of the Twenty-seventh Design Automation Conference*, pages 193–199, 1990.

Biographies

Joseph L. Ganley received his Ph.D. in Computer Science in 1995 from the University of Virginia. Since then, he has been a member of the research and development staff at Cadence Design Systems, working on software and technology for custom IC physical design. He is the author or co-author of over 25 papers on VLSI physical design automation and graph and geometric algorithms. He is a member of ACM, SIGACT, SIGDA, and Tau Beta Pi. He can be reached at ganley@cadence.com, and his web page is at http://ganley.home.ml.org/.

James P. Cohoon received his B.S. in Mathematics from Ramapo College, M.S. in Computer Science from Pennsylvania State University, and Ph.D. in Computer Science from the University of Minnesota in 1982. He is a former member of AT&T Bell Laboratories and is currently an Associate Professor with the Department of Computer Science at the University of Virginia in Charlottesville, Virginia. His professional interests include VLSI design automation algorithms, computational geometry, probabilistic search, and computer science education. He is the author of more than 60 papers in these fields. He is a member of the ACM Publications and SIG Boards and is past chair of SIGDA. He has served on the programming committees for such conferences as DAC, ICCAD, and ICCD, and was co-organizer of the first ACM Design Automation Workshop in Russia. His honors include a Fulbright Fellowship, his department's Best Teaching Award, and SIGDA's Leadership and Outstanding Service awards. He can be reached at cohoon@virginia.edu. His web page is at http://www.cs.virginia.edu/cohoon/.