Fast and Flexible Buffer Trees that Navigate the Physical Layout Environment

Charles J. Alpert[1], Milos Hrkic[2], Jiang Hu[3], and Stephen T. Quay[1]
[1]IBM Corp. 11400 Burnet Road, Austin, Texas 78758
[2]University of Illinois at Chicago, 851 S. Morgan St., Chicago, Illinois 60607
[3]Texas A&M University, 320G WERC, College Station, Texas 78743
alpert@us.ibm.com, mhrkic@cs.uic.edu, jianghu@ee.tamu.edu, quayst@us.ibm.com

ABSTRACT
Buffer insertion is an increasingly critical optimization for achieving timing closure, and the number of buffers required increases significantly with technology migration. It is imperative for an automated buffer insertion algorithm to be able to efficiently optimize tens of thousands of nets. One must also be able to effectively navigate the existing layout, including handling large blockages, blockages with holes specifically for buffers, specially allocated buffer blocks, placement porosity, and routing congestion. The algorithm must also be flexible enough to know when to use and when not to use expensive layout resources. Although several previous works have addressed buffer insertion in the presence of blockages, this is the first to present a complete solution that can manage the physical layout environment.

Categories and Subject Descriptors
B.7.2 [**Integrated Circuits**]: Design Aids - Placement and routing; J.6 [**Computer-aided Engineering**]: Computer-aided design.

General Terms
Algorithms, Performance, Design

Keywords
Buffer insertion, physical synthesis, global routing.

1. INTRODUCTION
Interconnect's domination of system performance has made buffering a critical component of modern VLSI design methodologies. Cong [5] expects that close to 800,000 repeaters will be needed for designs in the 50 nanometer node. Saxena et al. [12] argues that the number of repeaters will rise even faster, e.g., reaching about 15% of the total cell count for intrablock communications for the 65 nanometer node. Consequently, the impact of buffers on the layout environment can no longer be ignored. Efficiently finding a high-quality buffered Steiner route which can manage the environmental constraints is a very challenging problem.

Construction of thousands of buffer trees can potentially cause several design headaches. Figure 1(a) illustrates the "alley" problem, in which space is limited between two large fixed blocks. The space between blocks is highly desired by global routes that cross over the blocks, leading to both placement and routing congestion. This becomes especially problematic if the alley is filled with buffers for non-critical nets that could have potentially avoided the alley. Figure 1(b) shows the buffer "pile-up" phenomenon. Several nets may desire buffers to be inserted in the black region, yet since this area is blocked, the buffers are inserted as close to the boundary as possible. As more nets are optimized, these buffers pile up and spiral out further from their ideal locations. This could be avoided by only allowing buffers for the most critical nets to use these scarce resources. One must manage the layout environment during optimization.

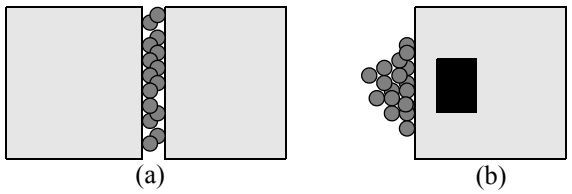

Figure 1 Buffer insertion on thousands of nets can potentially (a) fill up constrained "alleys" and (b) cause buffer "pile-ups".

As technology continues to scale, the optimum distance between consecutive buffers continues to decrease. In hierarchical design, this means allocating spaces within macro blocks for buffering of global nets. An example is shown in Figure 2(a). The space for buffers is potentially limited, so non-critical nets should be routed around the blocks while critical ones can use the holes. Long non-critical nets still require buffers to fix slew and/or capacitance violations. In addition, these nets could be critical, but have a wide range of possible buffering solutions that may bring them into the non-critical group. In the figure, the top net is non-critical and requires three buffers, while the bottom net is critical and needs only two by exploiting holes punched in the block.

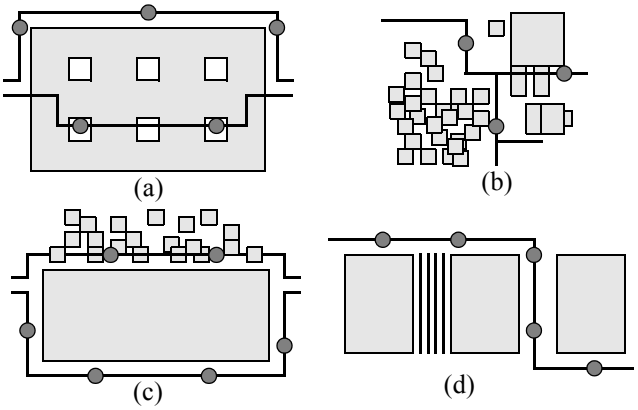

Figure 2 Some environmental based constraints include (a) holes in large blocks, (b) navigating large blocks and dense regions, (c) distinguishing between critical and non-critical preferred routes, and (d) avoiding routing congestion.

DAC 2004, June 7-11, 2004, San Diego, California, USA.

Even without holes in block, designs may have pockets of low density for which inserting buffers is preferred, as shown in Figure 2(b). In the figure, the Steiner route is located in the low density part of the chip, which makes the buffers inserted along the route also use low density regions. Figure 2(c) shows an example where one may be willing to insert buffers in high density regions if a net is critical. The 2-buffer route above the block yields faster delays than the 4-buffer route below the block that is better suited for non-critical nets. Finally, Figure 2(d) shows routing congestion between two blocks; the preferred buffered route avoids this congestion without sacrificing timing.

Most previous works on buffer insertion that consider the environment focus on either avoiding large blockages or using a set of fixed buffer locations, which means they do not handle the types of constraints shown in Figures 1 or 2. The authors of [6] and [13] both propose algorithms that handle a set of very limited fixed buffer locations; they are too inefficient to be used on thousands of nets. Zhou et al. [15] use dynamic programming to insert buffers for nets with routing and physical obstacles. Both [9] and also [8] solve this problem with shortest paths. By themselves, these approaches are not suitable for optimizing thousands of nets since they are too inefficient and only apply to 2-pin nets.

The only approach that addresses various constraints (and not just blockages and buffer blocks) on large-scale industry designs (tens of thousands of nets with multiple fanouts) is the work of [1]. The P-tree [10] and S-tree approaches [7] may potentially be extended for environmental navigation, yet the problems of excessive runtime and modeling the environment in the data structures still remain. Our proposed approach is a major upgrade of [1], as described below. This work starts from the premise that only a marginal runtime degradation is permissible. The re-routing must at least be as fast, if not faster, than van Ginneken's algorithm, for a designer to use it regularly.

2. ALGORITHMIC OVERVIEW

We call the algorithm in [1] PABST (Porosity Aware Buffered Steiner Trees) and call our proposed algorithm BEN (Buffer trees that are Environmentally Navigated). Both approaches share the same overall flow and strategy:

- **Step 1**: Construct a timing-driven Steiner tree (e.g., [2]) that is ignorant of the environment.
- **Step 2**: Re-route the Steiner tree to preserve its topology while navigating environmental constraints.
- **Step 3**: Insert buffers using a resource-aware variant [11] of van Ginneken's algorithm [14].

2.1 Plate Expansion

PABST addresses the re-routing of Step 2. The design area is divided into tiles where each tile has a cost reflecting its placement and/or routing congestion. The Steiner tree is then broken into disjoint *2-paths*, i.e., paths which start and end with either the source, a sink, or a Steiner point such that every internal node has degree two. For example, the nets shown in Figure 3(a) and (b) both decompose into three 2-paths. Finally, each 2-path is re-routed in turn to minimize cost, starting from the sinks and ending at the source. The new Steiner tree is assembled from the new 2-path routes.

A key idea of PABST is that Steiner points are allowed to migrate from their original locations, as long as they do not deviate outside of a specified "plate" region. During maze re-routing, one considers routing to any tile in the plate instead of just the original tile. Figure 3(a) shows a routing tree after Step 1. The striped tile is the Steiner point, and the shaded region shows a 5 x 5 plate centered at the original Steiner point. Figure 3(b) shows a Steiner tree that might result after re-routing. The Steiner point has moved to a different location within the plate; where it ends up depends on the cost function that is optimized. The dotted region shows the potential search space for the re-routing of the 2-path from the Steiner point to the source. In this case, the bounding box containing the two endpoints was expanded by one tile.

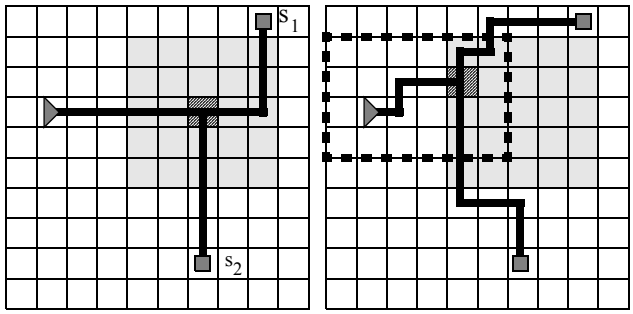

Figure 3 Example of a three-pin net (a) before and (b) after re-routing. The shaded square region is the "plate" and the dotted region is the solution search space for the final 2-path.

Having a plate of reasonable size allows the route to move outside of a potentially heavily congested area. If one sets the plate size to be the entire tiled area, this becomes equivalent to an S-Tree [7] type of approach. PABST accomplishes tile to tile routing by dynamic programming in which both a load and cost is propagated from one tile to the next (as in [7]). A congestion penalty is assessed only when a buffer is "virtually" inserted. At each tile non-dominated buffer solutions are pruned.

2.2 Technical Contributions

BEN utilizes the same strategy as PABST, but the underlying plate-to-plate routing algorithm is completely different. This was necessary because PABST suffers from the following weaknesses:

- It is too slow. Having multiple candidates propagated from tile to tile means that the re-routing step is inherently more expensive than van Ginneken's algorithm since the solution space is larger, yet the same type of candidate propagation and pruning is being employed.
- Tilings are too coarse. If one uses a fine tiling of the grid, the runtimes can become prohibitive especially because the plate sizes must also be large.
- Plates are too small. A finer tiling requires a larger plate to escape from a high cost region. Since the complexity of PABST increases quadratically with plate size, one cannot afford to use a fine tiling and a large plate size.
- Distinctions between critical and non-critical nets are missing. The algorithm may need to generate different solutions for critical and non-critical nets. This has to be embedded within the algorithm.

BEN addresses all these weaknesses and it is actually much simpler than PABST.

Recognizing that performing dynamic programming and pruning during re-routing is inherently too expensive, we permit at most one candidate per tile. This results in a more traditional maze routing algorithm; the right cost function is paramount. Also, instead of performing plate to plate routing of a sequence of tile to tile routes, the entire optimization is performed in a single pass. This allows one to use as large a plate as necessary, for almost no runtime penalty. In PABST, when left (L) and right (R) branches of a Steiner tree are merged, the candidate solutions for L and R must be merged together to form a new larger set of solutions. For BEN, only the costs of the left and right subtrees need to be combined to form a new initial cost for the next 2-path optimization that begins from the Steiner point.

Finally, we show how one can parameterize the cost function to trade-off critical and non-critical nets. The result is that BEN is both extremely efficient and flexible.

3. COST FUNCTION

One motivation behind this work was recognizing that PABST was inherently too slow. Within our industrial physical synthesis tool called PDS[1], buffer insertion typically takes 5-20% of the total CPU time. This runtime consists of Steps 1 and 3 in Section 2. Adding the PABST algorithm for Step 2 causes the total CPU time of physical synthesis to more than double. While this may be tolerable in some instances, it is too severe for large scale usage. Thus, we allow at most one solution per tile. To realize the desired behavior, we construct the cost function as follows.

Each tile has a cost that reflects its environment, in particular, the cost of potentially inserting a buffer and routing through the tile. Let $e(t)$ be the environmental cost of using tile t, where zero is for a free tile and one represents a fully utilized tile. For example, let $d(t)$ be the *density* (cell area divided by total area available) of tile t. Let $r(t)$ be its *routability* (used tracks divided by total tracks available) where we distinguish between horizontal and vertical routability. For $0 \le \alpha \le 1$, we use the cost function:

$$e(t) = \alpha d(t)^2 + (1-\alpha) r(t)^2 \quad (1)$$

The terms are squared to increase the penalty as either the density or routing congestion becomes close to one.

3.1 Cost for Non-Critical Nets

One set of nets requires buffering to fix electrical violations (such as slew, capacitance, or noise). One wants the net to avoid highly dense areas or routing congestion. However, one still wants to minimize wirelength to some degree. Consider

$$cost(t) = 1 + e(t) \quad (2)$$

This implies that a tile blocked for routing and/or density has cost twice that of a tile that uses no resources. The constant of one can be viewed as a "delay component" (which we revisit in Section 4.2).

A tile that corresponds to a Steiner point must accumulate the costs of the L and R children into a single cost. If $cost(L)$ and $cost(R)$ are their respective costs, then we simply use the merged cost function

$$cost(t) = cost(L) + cost(R) \quad (3)$$

[1] PDS (Placement Driven Synthesis) has been used for timing closure on hundreds of ASIC parts and on several microprocessors.

Since these are non-critical nets, all sinks are treated equally by having initial cost zero.

3.2 Cost for Critical Nets

For a second set of critical timing nets, the cost impact of the environment is immaterial. We seek the absolute best possible slack. When a net is optimally buffered (assuming no obstacles), its delay is a linear function of its length [4]. Of course, this solution must be realizable (see Section 4.2). To minimize delay, we simply minimize the number of tiles to the most critical sink. Thus, we have $cost(t) = 1$. When merging branches, one wants to choose the branch with worst slack, so the merged cost $cost(t)$ is: $max(cost(L), cost(R))$. To initialize the slack, a notion of which sink is critical is needed. Since our cost function basically counts tiles as delay, the required arrival time (RAT) must be converted to tiles. Let DpT be the minimum delay per tile achievable on an optimally buffered line. For a sink s, the $cost(s)$ is initialized to $-RAT(s)/DpT$. The more critical a sink, the higher its initial cost. The objective is to minimize cost at the source.

3.3 An Example

Consider re-routing the net in Figure 3(a) using the critical net cost function. Assume that $RAT(s_1) = -100$ ps, $RAT(s_2) = -300$ ps, and that an optimally buffered line takes 100 ps to cross a tile, i.e., $DpT = 100$. Then, $cost(s_1) = 1$ and $cost(s_2) = 3$. This means s_2 begins with higher cost, and the cost function guarantees a shortest path from the source to s_2.

Figure 4(a) shows one of several possible solutions using this cost function. The Steiner point could have been chosen from any within the dotted region. The tile in the lower left corner of the plate cannot be chosen since it would increase the length of the route to the top sink by too much. The cost at the Steiner point is 9 the cost at the source is 13. Figure 4(b) shows what re-routing would take place using the non-critical cost function, in which the "blob" represents an area of high cost. If $e(t)$ is 1.0 for tiles inside the blob and 0.2 for areas outside the blob, then the cost at the Steiner point is 12, and the total cost at the source is 25.2.

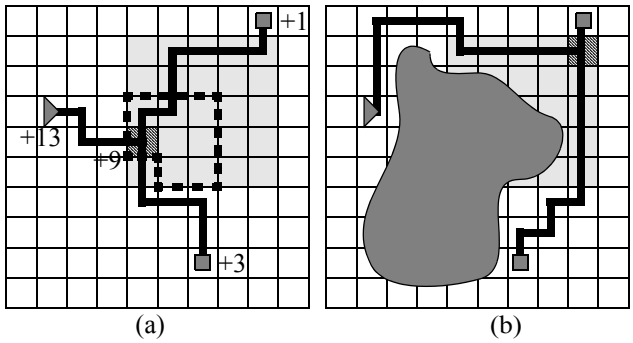

Figure 4 Examples of the (a) critical and (b) non-critical net cost functions. The shaded area represents a region of high cost.

3.4 General Cost Function

We wish to trade-off between the critical and non-critical cost functions. Let $0 \le K \le 1$ be the trade-off parameter, where $K = 1$ corresponds to a non-critical net and $K = 0$ corresponds to a critical net. The cost function for tile t is

$$cost(t) = 1 + K \cdot e(t) \quad (4)$$

For critical nets, merging branches is a maximization

function, while it is an additive function for non-critical nets. These ideas can be combined with K to yield:

$$cost(t) = \begin{array}{c} max(cost(L), cost(R)) + \\ K \cdot min(cost(L), cost(R)) \end{array} \quad (5)$$

Finally, the sink initialization formula becomes

$$cost(s) = (K-1)RAT(s)/DpT \quad (6)$$

Thus, K trades off the cost function, the merging operation, and sink initialization. In practice, we suggest using K as follows. First, optimize all nets that need buffering with $K = 1$, which limits the use of scarce resources. After performing a timing analysis, those nets that still have negative slack can be re-optimized with a smaller value of K, e.g., 0.7. This process of re-optimizing and gradually reducing K can continue until, say, $K = 0.1$. As seen by our experimental results, reducing all the way to $K = 0$ is not suggested, since its total ignorance of the environment also hampers performance.

4. SLEW THRESHOLD CONSTRAINT

Consider the non-critical net in Figure 5(a) in which there is no routing congestion, and the shaded area is a block for which buffers cannot be inserted. If the detour is long enough, then the non-critical cost function will prefer the solution that goes over the block. However, making this jump may prevent the net from fixing a slew violation.

A common design methodology practice [3] requires a fixed slew constraint on every gate in the design to ensure a reasonable initial timing and to maintain signal integrity. This slew constraint can be translated into a length constraint either empirically or analytically, which can be converted to a number of tiles T. In our example, if $T < 4$ then the route going over the blockage is not permitted.

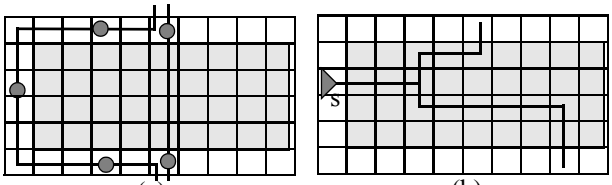

Figure 5 Examples of possible solutions considering placement blockage for (a) a non-critical and (b) a critical net.

To enforce the slew threshold constraint during re-routing, we store the number of consecutive blocked tiles $B(t)$ a solution for tile t has seen to date. A tile t is blocked if $d(t) \geq 1$. When a new tile t' is encountered that is blocked, $B(t')$ is set to $B(t) + 1$, and when t' is unblocked, we have $B(t') = 0$.

Let T_L and T_R be the number of consecutive blocked tiles on the left and right branches from a Steiner point t. To compute the consecutive blocked tiles from a Steiner point, one could use $B(t) = T_L + T_R$. In Figure 5(b), $T_L = 4$ and $T_R = 8$. So for the source s, $B(s) = 15$. If $T = 13$, then this Steiner route is not permissible. This is somewhat pessimistic since the longest blocked path is just 11 tiles. However, if one uses $B(t) = max(T_L, T_R)$, this leaves a value of $B(s) = 11$, which ignores the potential extra capacitance of the left branch. As a compromise, we use $B(t) = \sqrt{T_L^2 + T_R^2}$. At the Steiner point, the consecutive tile count is 8.9 at the Steiner point and $B(s) = 11.9$. This provides a reasonably good model of the capacitance required by the driving buffer to meet its slew constraint.

5. TILE COST FUNCTION FOR DELAY

One serious problem remains with BEN as described so far: for critical nets, a linear delay may not be achievable. If a large block exists over the route, as in Figure 6, the critical net will be routed with a straight shot instead of around the block since the distance is shorter. A large enough block causes the buffers to be spaced further than optimal, i.e., the route around the block could very well be faster than the route through the block.

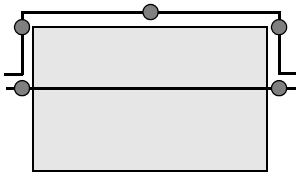

Figure 6 Example showing the need for a tile delay function.

One solution to this problem is a tight slew threshold constraint. Call the optimum tile spacing between buffers L and the slew tile constraint T. If $T = L$, then the route that goes over the block is forbidden; the route will be forced to go around. Using $T = L$ is unnecessarily restrictive. Assume that $T > L$ and that the width W of the block lies between L and T. If W is close to L, then routing over the block may yield the best for delay, but if W is closer to T, the route around may be preferred. The cost function must be able to recognize which solution is better.

The problem of estimating delay given that the route may cross over blocks of varying size is quite difficult; yet, a simple, the work in [4] concludes that it does not matter *where* blockages appear on a line: only the size of the blocks affect delay. Further, one can safely ignore any blocks with $W < L$ since van Ginneken's algorithm is smart enough to choose the size buffers as necessary if blockages force irregular buffer spacing. An optimal buffering of a net that crosses a block with width $W > L$ will almost assuredly have a buffer inserted right before and right after the block. Hence, one can use a linear delay model for the route whenever traversing blocks with width less than L and use the Elmore delay, otherwise.

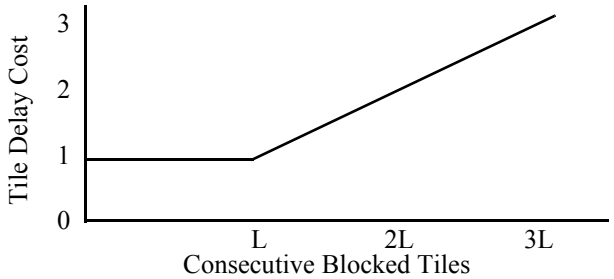

Figure 7 Tile delay cost function.

These conclusions imply that the correct cost of a tile for critical nets is still one when $B(t) < L$; each subsequent blocked tile after L becomes increasingly expensive. Because delay is quadratic in unbuffered wirelength, the cost function of each additional tile increases with the derivative of delay. Figure 7 illustrates the function, which we label $TDC(B(t))$. Then, Equation (4) now becomes

$$cost(t) = 1 + K \cdot e(t) + (1 - K)TDC(B(t)) \quad (7)$$

Using the prefix $1 - K$ reduces the impact of the tile delay cost as the net becomes less criticalSince BEN uses maze routing instead of dynamic programming, it can use a finer tiling and explore a much larger solution space than PABST. Figure 8 shows an example in which the dotted bounding box is the space searched by PABST and the dashed line is the search space of BEN. BEN expands the searchable routing regions based on the actual environmental constraints. If a region cannot be buffered or routed, BEN expands the search space sufficiently to explore solutions that avoid the blocked tiles.

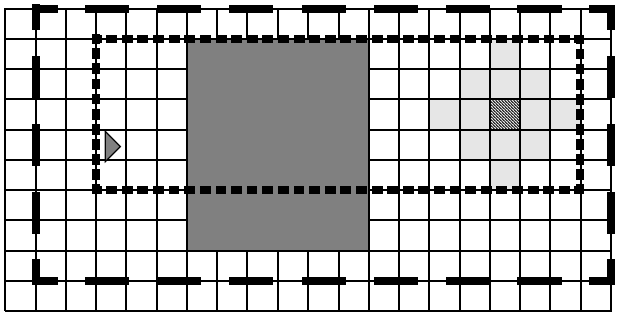

Figure 8 Example of the plates and region search space.

BEN also uses diamond shaped plates as opposed to square ones used by PABST. A diamond is a Manhattan circle and more accurately reflects the region one needs to explore to move a Steiner point to an alternate location.

6. EXPERIMENTAL RESULTS

We now demonstrate BEN's speed and effectiveness. We do not compare directly to PABST since it cannot handle all the type of constraints discussed in the introduction and since its runtime is an order of magnitude larger. We simply demonstrate that BEN is flexible, fast, and successful at constructing Steiner routes that navigate the environment.

6.1 Demonstrating the General Cost Function

To demonstrate how the behavior of BEN changes as a function of K, we constructed a test case from an industry design with 70 thousand objects. The cells were spread out, and a large block was inserted into the design to create a placement obstacle. In addition small holes were punched in the block to create areas where buffers could be inserted.

Figure 9 shows the corresponding density map for a given 7-pin net. The source is marked with a white x, sinks are marked with dark squares, the white dots are potential buffer insertion locations, and the diamonds are the inserted buffers. The top route is the solution with $K = 1.0$, the bottom route is for $K = 0.1$. Observe that the top route totally avoids the large block, leading to a 4134 ps slack improvement over the unbuffered solution (after performing Steps 1-3). When $K = 0.1$, the route successfully finds the holes and places buffers in them where it deems it appropriate, improving slack by 4646 ps. The different solutions imposed by changing K can be clearly seen.

We ran a total of 500 nets (with 5204 total sinks) on this design using different values of K to see how K affects performance. The results for all 500 nets are summarized in Table 1. The baseline represents running only Steps 1 and 3, i.e., no optimization of the environment is performed. We measure the total number of buffers inserted and the average slack improvement over the zero buffer solutions.

Each buffering approach significantly improves slack for all nets. The improvement for the baseline is lower compared to the other results since it cannot find the holes in the block. Also, BEN returns poor solutions for $K = 0.0$ since it completely ignores $e(t)$. Note the general improvement in slack and reduction in total buffers as K decreases. The latter effect is caused by more direct Steiner routes. By the time K reaches 0.3 the value of reducing K any further is not noticeable. In practice, we recommend using $K = 0.1$ for the most critical nets and $K = 1.0$ for non-critical nets.

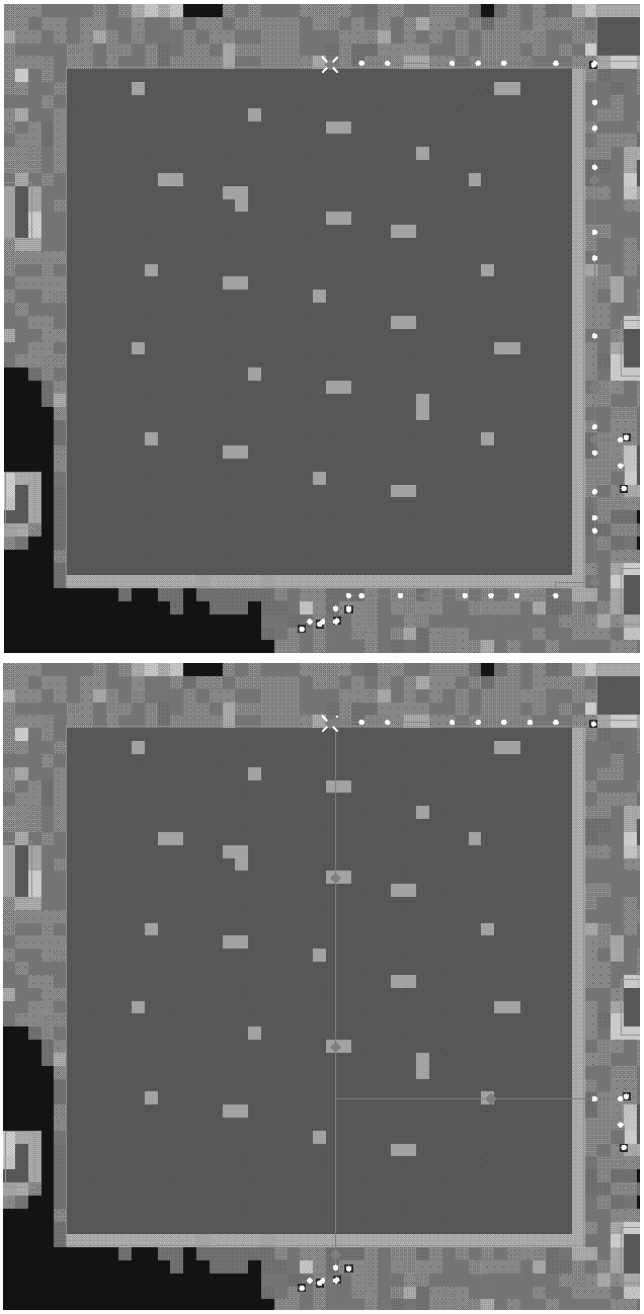

Figure 9 Test case containing a large block with holes. A net with $K = 1.0$ is shown above and with 0.1 is shown below.

6.2 Results from an Industry Design

Next, we consider an industrial ASIC design with about 330 thousand objects. We selected 1000 long nets with 2704 total sinks to buffer in three modes, one with no Step 2 environment navigation (BASE), one with $K = 0.1$ (CRIT), and one with $K = 1.0$ (ELEC). We also ran each mode on four different size tilings of the chip.

Alg.	# Bufs	Slack imp	Alg	# Bufs	Slack imp
baseline	1477	1849 ps	K=0.5	1927	2103 ps
K=1.0	1975	2072 ps	K=0.4	1922	2084 ps
K=0.9	1967	2081 ps	K=0.3	1917	2118 ps
K=0.8	1947	2085 ps	K=0.2	1923	2120 ps
K=0.7	1932	2093 ps	K=0.1	1930	2118 ps
K=0.6	1926	2108 ps	K=0.0	3333	1884 ps

Table 1 Average slack improvement as a function of K.

Table 2 shows the total number of buffers, average slack improvement, and total CPU time for Steps 1, 2, and 3 (on an IBM p690). The results improve as the number of rows increases from 83 to 166, yet for finer tilings, BEN does not improve performance. As one would expect, the slack improves more for CRIT than ELEC. For tilings with 166 rows and above, ELEC outperforms BASE because it avoids obstacles that force buffers to be spaced too far apart. Note that BEN is much faster than Step 1 or 3 for coarse tilings, and it is still reasonable even for very fine tilings. The additional runtime for van Ginneken's algorithm for ELEC and CRIT is caused by their ability to avoid blocks which leads to additional potential buffer insertion locations. This extra runtime for Step 3 could be mitigated by more careful selection of potential buffer locations.

Tiling Size	Alg	# Bufs	Slack imprv	CPU(s)		
				Steiner	BEN	VG
83	BASE	5917	5070 ps	367	0	258
x	ELEC	7160	5022 ps	367	32	677
82	CRIT	7109	5116 ps	367	29	621
166	BASE	5962	5050 ps	367	0	250
x	ELEC	7153	5068 ps	367	139	693
164	CRIT	7070	5134 ps	367	126	648
249	BASE	5926	5045 ps	367	0	264
x	ELEC	7186	5081 ps	367	425	697
247	CRIT	7078	5134 ps	367	352	655
332	BASE	5978	5034 ps	367	0	254
x	ELEC	7230	5078 ps	367	759	695
329	CRIT	7066	5132 ps	367	660	653

Table 2 Slack and CPU summary for 1000 nets.

Next, we examine how well BEN utilizes resources for the tiling with 249 rows. For this tiling, BASE, ELEC, and CRIT insert 5871, 7299, and 7137 buffers, respectively. For each insertion in a tile t, we record the $e(t)$ cost. The distribution of all buffer insertions as a function of $e(t)$ is shown in Table 3. For example, for tiles with cost between 95% and 99.99%, BASE, ELEC and CRIT insert 318, 25 and 144 buffers, respectively. One can see that both ELEC and CRIT are more successful at inserting buffers in lower cost tiles than BASE. ELEC conserves resources better than CRIT due to its more environmentally aware cost function.

7. CONCLUSIONS

We have presented BEN, a fast and flexible buffering algorithm that handles a wide variety of environmental constraints. Even when distinguishing between critical and non-critical nets, BEN yields better solutions in terms of slack and uses fewer critical resources than a flow which does not use BEN. Future work seeks both improvements in the runtime of the overall flow and effective and seamless integration into physical synthesis. We also plan to model other constraints in the environmental cost, such as power density. In addition, we believe BEN is appropriate to embed into both global routing and design planning.

100 e(t)	BASE	ELEC	CRIT	100 e(t)	BASE	ELEC	CRIT
0-4.99	0	0	0	50-54.99	368	487	524
5-9.99	158	378	282	55-59.99	398	394	374
10-14.99	252	729	625	60-64.99	360	375	337
15-19.99	236	730	714	65-69.99	298	312	280
20-24.99	220	523	566	70-74.99	437	270	274
25-29.99	196	435	449	75-79.99	544	270	252
30-34.99	196	408	427	80-84.99	418	206	187
35-39.99	241	372	364	85-89.99	369	168	176
40-44.99	256	449	436	90-94.99	396	142	151
45-49.99	264	513	516	95-99.99	318	25	144

Table 3 Distribution of buffers as a function of environmental cost. Here $e(t)$ is multiplied by 100.

References

[1] C. J. Alpert, G. Gandham, M. Hrkic, J. Hu, and S. T. Quay, "Porosity Aware Buffered Steiner Tree Construction", *ISPD*, April 2003, pp. 158-165.

[2] C. J. Alpert, G. Gandham, M. Hrkic, et al., "Buffered Steiner Trees for Difficult Instances", *IEEE Transactions on CAD*, 21(1), pp. 3-14, Jan. 2002.

[3] C. J. Alpert, J. Hu, S. S. Sapatnekar, and P. G. Villarrubia, "A Practical Methodology for Early Buffer and Wire Resource Allocation," *IEEE/ACM DAC*, 2001, pp. 189-195.

[4] C. J. Alpert, J. Hu, S. Sapatnekar, and C.-N. Sze, "A Fast Oracle for Interconnect Delay Prediction", ACM Tau Workshop, 2004, pp. 39-45.

[5] J. Cong, "Challenges and Opportunities for Design Innovations in Nanometer Tech.", *SRC Working Papers*, 1997.

[6] J. Cong and X. Yuan, "Routing Tree Construction under Fixed Buffer Locations", *IEEE/ACM DAC*, 2000, pp. 379-384.

[7] M. Hrkic and J. Lillis, "S-tree: A Technique for Buffered Routing Tree Synthesis", *IEEE/ACM DAC* 2002, pp. 98-103.

[8] A. Jagannathan, S.-W. Hur, and J. Lillis, "A Fast Algorithm for Context-Aware Buffer Insertion", *DAC* 2000, pp. 368-373.

[9] M. Lai and D. F. Wong, "Maze Routing with Buffer Insertion and Wiresizing", *IEEE/ACM DAC*, 2000, pp. 374-378.

[10] J. Lillis, C.-K. Cheng, T.-T. Y. Lin, and C.-Y. Ho, "New Performance Driven Routing Techniques With Explicit Area/ Delay Tradeoff and Simultaneous Wire Sizing", *IEEE/ACM DAC,* 1996, pp. 395-400.

[11] J. Lillis, C.-K. Cheng and T.-T. Y. Lin, "Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model", *IEEE J. Solid-State Circuits*, 31(3), 1996, 437-447.

[12] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick, "The Scaling Challenge: Can Correct-by-Construction Design Help?", *Proc. ISPD*, 2003, pp. 51-58.

[13] X.Tang, R. Tian, H. Xiang, and D. F. Wong, "A New Algorithm for Routing Tree Construction with Buffer Insertion and Wire Sizing under Obstacle Constraints," *IEEE/ACM ICCAD*, 2001, pp. 49-56.

[14] L. P. P. P. van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay", *Intl. Symposium on Circuits and Systems*, 1990, pp. 865-868.

[15] H. Zhou, D. F. Wong, I.-M. Liu, and A. Aziz, "Simultaneous Routing and Buffer Insertion with Restrictions on Buffer Locations," *IEEE/ACM DAC*, 1999, pp. 96-99.