A NOVEL GEOMETRIC ALGORITHM FOR FAST WIRE-OPTIMIZED FLOORPLANNING

Peter G. Sassone, Sung K. Lim

School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332, USA

ABSTRACT

As the size and complexity of VLSI circuits increase, the need for faster floorplanning algorithms also grows. In this work we introduce Traffic, a new method for creating wire- and area-optimized floorplans. Through the use of connectivity grouping, simple geometry, and efficient data structures, Traffic achieves higher result quality than Simulated Annealing (SA) in a fraction of the time. This speed allows designers to explore a large circuit design space in a reasonable amount of time, rapidly evaluate small changes to big circuits, and quickly produce initial solutions for other floorplanning algorithms.

1. INTRODUCTION

Despite the amount of academic and industrial research in the area, the challenge of block packing is even tractable by a child: given a set of rectangles, arrange them into the smallest area. This problem is relevant to many fields, from truck loading to OS process scheduling. Additional constraints such as wire-minimization or fixed-position blocks make the challenge more complex for VLSI circuit floorplanning. However, even without these additional constraints, floorplanning is difficult and requires heuristics to efficiently solve.

We introduce a two-phase algorithm for block floorplanning called Traffic (Trapezoidal Floorplanning for Integrated Circuits), which seeks to floorplan through simple geometry. The first phase groups blocks by global and local connectivity using a modified partitioning algorithm. The second phase forms trapezoidal shapes from these grouped blocks. Trapezoids, when given similar slopes on their diagonals, are easily tileable. We use this principle to tessellate these shapes across the floorplan. Since the algorithm and data structures are very simple, each run takes only a thousandth of the time of a Simulated Annealing (SA) run and achieves very good results. Taking the best of many Traffic runs improves the solution quality further while still taking far less time than even a single SA run.

The quality and speed of Traffic indicates many applications. First is as a final floorplanner, as taking the best of thousands of runs achieves high result quality in a reasonable amount of time. More significantly, our algorithm allows the circuit design space to be appraised quickly. Engineers using Traffic can quickly evaluate the physical implications of different circuit configurations (i.e., 10 large blocks versus 1000 smaller blocks) or different architectural details (i.e., 16-entry register file versus 32-entry). Traffic can also be used to produce initial solutions for other floorplanning algorithms, possibly mitigating their prohibitive runtimes.

ICCAD'03, November 11-13, 2003, San Jose, California, USA.

This paper is organized as follows. Section 2 addresses previous work in the area of floorplanning. Sections 3 and 4 describe the two phases of the Traffic algorithm. Section 5 presents the experimental parameters we use for our results. Section 6 shows circuit results compared with Simulated Annealing. Finally, Section 7 concludes and addresses future work.

2. RELATED WORK

Floorplanning has been studied extensively in the past two decades due to its theoretical and practical importance. Given a VLSI circuit consisting of both fixed or flexible blocks (some of the blocks can be pre-placed at some locations) and a net-list interconnecting these blocks, floorplanning constructs a layout indicating the position and shape of each flexible block such that all nets can be routed and total layout area is minimized.

There are two types of floorplans: slicing and non-slicing. A slicing floorplan [14][16][17] is one that can be obtained by recursively cutting a rectangle into two parts by either a vertical line or a horizontal line. A non-slicing floorplan [2][6][10][15] is one that is not necessarily slicing. In general, a non-slicing floorplan can describe any type of packing.

Most of the existing floorplanning algorithms are iterative in nature – start with some initial solution and gradually improve its quality by performing various local moves. A popular choice for exploring the solution space has been Simulated Annealing [1][9], where a new solution is selectively accepted based on some probability in a cost function. Moreover, the major focus of recent advances on floorplanning has been on the development of an efficient solution representation and its fast evaluation for SA-based optimization approaches. Unfortunately, however, it has been a widely accepted fact that SA-based algorithms suffer from a prohibitively long runtime and require tedious parameter tuning.

To address these concerns, many authors have introduced prediction algorithms [4][5][7][11][13] which quickly estimate the area and wiring needed by a completed floorplan. Though these algorithms are often very fast and accurate, they are only a heuristic—there is no guarantee that a floorplan can be created with the output results.

Ranjan et al. [12] propose improving the speed of Simulated Annealing by computing the cost functions a priori in a predictor. They then use these values, along with top-down slicing and a final stage of SA, to quickly produce floorplans. Their result quality is comparable to SA, and their speedup is significant.

3. CONNECTIVITY PHASE

Figure 1 shows the pseudocode for the Traffic algorithm. As stated earlier, this two phase algorithm first groups blocks (lines 1-3) and then compacts them (lines 4-12). This section describes the initial grouping phase which is based on net-connectivity. This process is divided into *global net grouping* and *local net grouping*, though the division between these terms is grey. Global nets connect distant blocks in a floorplan; however, given a different floorplan of the same blocks, a different set of nets may be considered global.

3.1 Global Grouping

To minimize longer wires that will be present in any floorplan, we first partition the blocks using a method called Linear Partitioning and Placement (LPP) which is similar to the partitioning algorithm introduced in [3]. In this scheme, the net-graph is divided into partitions, but the partitions (not the blocks in the partitions) are assumed to be situated in a line. Thus, a connection from partition one to four incurs a higher cost than a connection from partition one to two. This produces linearly ordered partitions, which is analogous to block linear ordering [8]. As the next section will show, this linear order is advantageous to us since the Traffic physical algorithm will operate on each partition separately then stack them together to form a final floorplan.

We round the result from Equation 1 to determine the best number of partitions *P* for *N* blocks. This empirically derived formula implies that for less than 12 blocks, only one partition should be used.

Equation 1. Optimal number of partitions for N blocks.

$$P = \frac{\sqrt{N}}{1.25} - 2$$

We do not find the one-time execution time of LPP to be burdensome—partitioning a 3000 block circuit takes under 10 seconds on our test platform. We also find that this time is completely offset by the speedup of the physical layout phase of Traffic. Since that aspect's runtime is roughly $O(n^2)$ on number of blocks in the partition being worked on, executing on many small partitions is faster than running on one large partition.

3.2 Local Grouping

For shorter wires, Traffic binds *highly connected pairs* together in the final layout. Highly connected pairs are two blocks within the same partition which have significantly more inter-block nets than average for that partition. For instance, in the GSRC benchmark *n300a*, blocks 57 and 76 are connected by 14 different nets. These are the most highly connected blocks in the circuit, and their distance apart in the final floorplan will make a noticeable impact on the total wiring estimate.

Instead of complicating the physical phase with a cost function to address connectivity, we choose to use a side-effect of the algorithm. As will be explained in more detail in the next section, blocks of the same height will have high spatial locality in each partition. To persuade these highly connected blocks to be adjacent to each other, we expand the shorter block to the height of the taller. Then we lock these highly connected blocks so that they may not rotate. Thus they will remain identical height until the termination of the physical algorithm. As the spatial locality of these blocks will be high in the final layout, these highly-connected pairs will not contribute significantly to the wiring estimate.

Connectivity phase	1:	partitions[ ] = global_grouping(*blocks*);
	2:	for *part* from 0 to num_partitions :
	3:	local_grouping(*part*);
Physical phase	4:	for *run* from 0 to num_runs :
	5:	for *part* from 0 to num_partitions :
	6:	*rows* = initial_placement(*part*);
	7:	for *step* from 0 to 25 :
	8:	mutate(*part.rows*);
	9:	if (no_change) break;
	10:	squeeze(*part.rows*);
	11:	is_local_best(*part.rows*);
	12:	*all_rows* = merge_partitions();
	13:	is_global_best(*all_rows*);

Figure 1. Traffic algorithm pseudocode.

The downside to this approach is the increased area used by the expanded blocks. To mitigate this issue, we only bind block pairs whose connectivity is above the 95th percentile. We also rotate these blocks before expansion so as to minimize the amount of padding that will be added. Experiments have shown the expanded size of these blocks does not appreciably affect the final area. Experiments have also shown that binding more than two blocks together degrades results, so a block can only be bound to one other. Execution time of binding blocks is minimal even for large circuits—this completes in under a second for 3000 blocks on our test platform. As with the global grouping, this is a one-time cost as all runs will utilize the same bindings.

4. PHYSICAL PHASE

After the blocks have been grouped both globally and locally, Traffic begins it's physical layout phase (lines 4-12 in Figure 1). Though wire-minimization is the ultimate goal of Traffic, a floorplan that is packed more tightly together will tend to have a lower wire estimate. This phase of the algorithm ensures a final layout with as little white-space as possible. We start our explanation of the physical algorithm with a high-level overview of the algorithm, then elaborate on the details.

4.1 Overview

Figure 2 is a graphical representation of what Traffic attempts to do for each partition. In (a), each block is first randomly rotated and then placed into one of many buckets depending on height. For instance, heights 1-15 might go in bucket 1, heights 16-31 in bucket 2, and so on. However, blocks that have been bound by the local wire optimization are not rotated (indicated by the two connected blocks in the upper-right). This is the initial placement for Traffic. Buckets are then sorted by height (even buckets ascending and odd buckets descending) and lined up in contiguous rows in (b). Thus each bucket is now a single row and should resemble a trapezoid. Since there is a one-to-one correspondence between buckets and rows, we use the terms interchangeably from here onward. In (c), we move blocks between rows to even their lengths out, and flip the even rows over so that we form rowpairs (i.e., 1-2 and 3-4 in the figure). These rowpairs are then squeezed together tightly, leaving only

small gaps between. This is repeated for all partitions independently. Finally, the partition floorplans are placed atop each other to form a global floorplan in (d). There is no guarantee the partition floorplans will be of the same width, so the bounding box becomes the total chip area

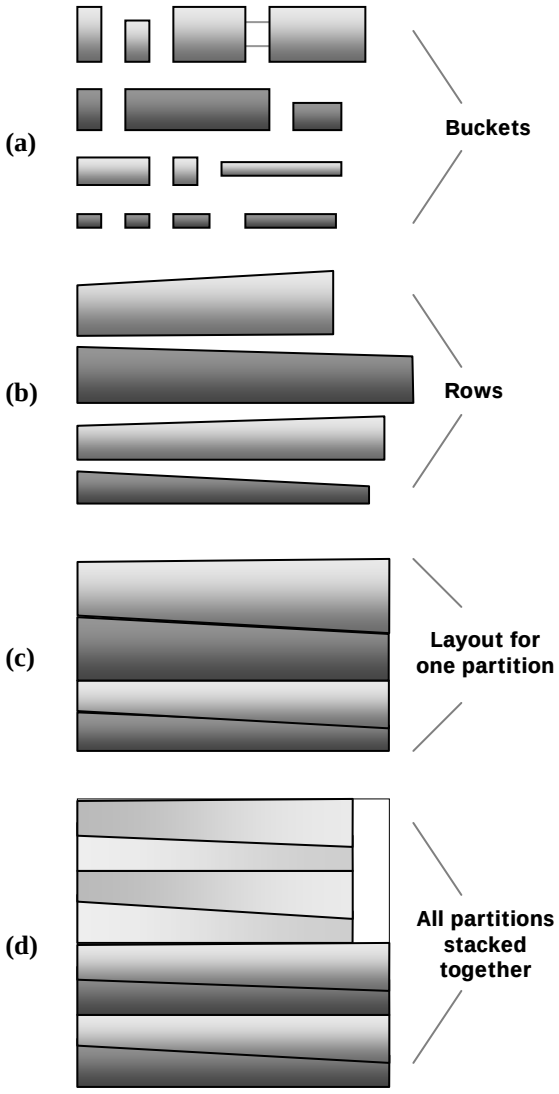

Figure 2. Traffic physical phase illustration.

In the pseudocode of Figure 1, line 6 is the random rotation of the blocks and the initial bucket setup. Evening of the rows' lengths is done on line 8, and line 10 does the flipping of the even-numbered rows. The evening and squeezing of the rows is how Traffic explores the local solution space, so it is done iteratively. This is done for each partition, so line 12 merges all partition floorplans into a global floorplan.

The evening of the rows, also called mutating, usually achieves even row balance within 3 or 4 steps. This leads to very tight layouts with very little white-space. However, it is advantageous to let the mutations continue many more times. As blocks are moved around, the trapezoids change shape, possibly creating tighter fits between row-pairs or better wire estimates. Empirically, we found that the best results are usually found within 25 steps, thus the constant in the inner loop of the pseudocode.

One run is fairly insignificant in its exploration of the solution space, so we also do runs iteratively (each with a random initial rotation) to explore the global solution space. We evaluate each run based on wire-length or area, and save the best global floorplan in a data structure.

The remainder of this section explains the three most important steps in this algorithm—the initial placement, the mutations, and the squeezing. These are done to each partition individually, and once completed, all partitions are stacked to form the global layout. The remainder of the algorithm is mostly file I/O, half-perimeter wire-length estimation, and bookkeeping for saving the best floorplans.

4.2 Initial Placement

Line 6 in the pseudocode encapsulates the initial work to create the buckets/rows: set up of data structures, creation of buckets, and placing blocks into buckets. We explain each in turn.

The data structures for Traffic are very simple and, more importantly, static. There are no lists, vectors, graphs, or other dynamic structures present. Instead, all of the work is mainly done on a 2-D array representing the rows and a 2-D adjacency matrix representing the nets. Both contain pointers to a 1-D array of blocks. Each block is a C structure holding very basic information such as width, height, and two lock bits (explained in Section 4.3).

The block array is filled once upon reading in the data files and never modified, but the row array must be recreated at the beginning of each Traffic run. To create this double array, we must first know the number of buckets to be used for the current partition. The Traffic algorithm determines this with Equation 2.

Equation 2. Number of Traffic buckets for a partition.

$$Buckets_p = \left\lceil \frac{\sqrt{\sum Area_b}}{Height_{avg}} \cdot \frac{N_p}{N_{total}} \right\rceil_2$$

To derive this formula, we start with an ideal Traffic layout and work backwards to the number of rows needed to make it. This perfect floorplan is one-partition, square, row-based, and without white-space (similar to Figure 2c). The square's area is simply the sum of the individual blocks' areas since it is completely filled by non-overlapping blocks. Consequently, the height of the this square would be equal to the square-root of this block area sum. If block heights were uniformly distributed, the number of rows would be roughly equal to the height of the square divided by the average height of a block. We then multiply this number of buckets by the ratio of blocks in this partition versus the whole circuit. This will reduce the rows and flatten the partition so that, when later stacked, the global floorplan will be relatively square. Finally, we wish to have an even number of rows to form pairs with, so we round up to the next even number.

Once Traffic determines the number of buckets, it assigns each an equal range of heights. For instance, if there were 10 buckets and blocks ranged in dimensions from 50 to 100, then the first bucket would be assigned heights 50-54, second bucket assigned

heights 55-59, and so on. Blocks are then randomly rotated 0 or 90° (except the highly-connected blocks, which have been locked) and sorted into these buckets. As we padded the heights of highly-connected blocks to be equal and they did not get rotated, they are guaranteed to be in the same bucket.

There is no guarantee, in fact it is quite unlikely, that the buckets have an even distribution of blocks. Figure 3 shows a typical result of this crude bucket sorting for a partition. This is not a concern since we will be evening out these buckets/rows in the next step.

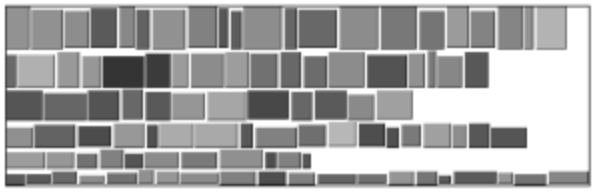

Figure 3. Initial placement of a Traffic partition.

Generally, these jagged initial layouts like Figure 3 have 30-50% white-space. Yet, the overhead to set up the data structures and create the buckets is just a few milliseconds on our test platform. Since our algorithm relies on doing tens, if not hundreds or thousands, of runs to explore the solution space, this setup speed is important. This is the overhead of a Traffic run—the real progress is made in the mutations and squeezing.

As mentioned earlier, all non-bound blocks are randomly rotated before being sorted into buckets. This produces up to 2^N possible starting arrangements of N blocks, making each initial state probably unique in a tractable number of runs. Though starting configurations may converge to the same result after our deterministic mutations, it is safe to say that each Traffic run produces a different layout.

4.3 Mutations

After initial bucket creation, we start calling them rows to illustrate how the buckets will appear in the layout. The focus is now on evening their widths, which we do through mutations (line 8 in the pseudocode). There are four types of mutations which even out rows by moving blocks in different ways:

- ***Shrink widest row***: moves blocks from the widest row to *adjacent* rows.
- ***Grow narrowest row***: moves blocks into the narrowest row from *adjacent* rows.
- ***Shrink widest row via rotation***: takes blocks from the widest row, rotates them 90°, and place them in the rows matching the height ranges of these blocks.
- ***Grow narrowest row via rotation***: takes blocks from wider-than-average rows that, when rotated 90°, belong in the narrowest row.

Traffic does one shrink and one grow mutation per step. The non-rotating functions are preferred so they are called first. If the adjacent rows were already too wide to accept blocks or too narrow to remove blocks from, we call the rotate versions instead to get blocks to/from non-adjacent rows:

```
if ( shrink_widest_row() == no change )
        shrink_widest_row_rotate();
if ( grow_narrowest_row() == no change )
        grow_narrowest_row_rotate();
```

If all four mutation functions return "no change", all of the blocks must both their locks set. Thus there is no work to do and this run is completed early (line 9 in the pseudocode).

To prevent live-lock, each block is assigned two locks – a move-lock and a rotate-lock. Once a block is moved to another row without rotation, its move-lock is set, and once it is rotated and moved its rotate-lock is set. These locks prevent further movement between rows or rotation, respectively. Highly connected blocks which were bound by the local wire grouping have had both their move and rotate locks preset, so they are guaranteed not to participate in any mutation.

It is important that mutations are deterministic and blind to the floorplan they will create. Whereas Simulated Annealing uses complicated cost functions and probabilistic swaps, Traffic's mutations only strive to even row widths regardless of how this might affect the layout. It is not coincidence, however, that floorplans with even row widths tend to produce less white-space in Traffic. This simpler cost-function allows mutations to be extremely fast and productive to the layout.

4.4 Squeezing Rowpairs

The final step (line 10 of the pseudocode) encompasses sorting even rows ascending and odd rows descending, rotating the even rows around, and squeezing the rowpair trapezoids together. Squeezing time is negligible as it only involves small quicksorts and simple arithmetic.

Since highly-connected blocks have the same height, sorting will leave them adjacent. This is why Section 3.2 states that the side-effect of Traffic is sufficient to bind these pairs. They started out in the same row and never moved between rows, and since these blocks did not rotate, they will be sorted adjacently.

Figure 4 shows a sample partition after mutations have equalized the rows and squeezing has compressed the trapezoids together. The rows infringe on each other yet do not overlap, and the trapezoids tile together very tightly.

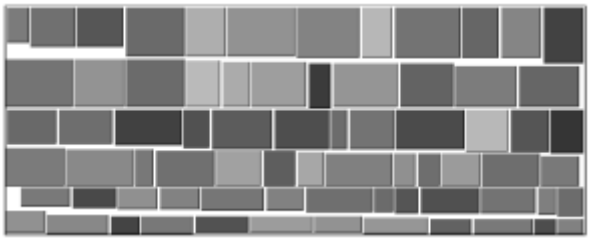

Figure 4. Sample Traffic partition after squeezing.

After squeezing is complete, statistics are gathered on this layout and the Traffic step is completed. If it is the best of the 25 layouts produced at each step, it is considered the *local best* for this partition. After each partition completes this run, each local best layout will be stacked to form a global floorplan for that run. If it is the best of all global floorplans, it is the *global best*.

5. EXPERIMENTAL SETUP

All performance evaluations are done on a Sun Ultra 10 workstation with a 440 MHz SPARC processor and 512 MB of memory running Solaris 8. All timing results are user-level time as measured by the Unix *time* command which has a resolution of 10ms. We use the half-perimeter method of wire estimation.

The Traffic code is written in ANSI C. To avoid comparing pad placement algorithms, wiring results do not include nets going to pads. Thus, a net that connects blocks A, B, and a pad will only look like a connection between A and B. We also assume that all connections emanate from the center of a block. Traffic is compiled with GNU gcc 2.95.3 with –O3 and –funroll_loops, and no parameters are tuned between executions.

Table 1. Block and net counts for GSRC circuits.

circuit	blocks	nets
n10a	10	118
n30a	30	349
n50a	50	485
n100a	100	885
n200a	200	1585
n300a	300	1893

Table 2. Statistics for HGR circuits before and after partitioning into 300 and 1000 blocks.

	original		300B	1000B
circuit	cells	nets	nets	nets
avq_large	25178	25384	2600	5229
avq_small	21918	22124	2686	5194
golem3	103048	144949	10935	21485
ibm01	12752	14111	4524	6558
ibm02	19601	19584	9106	10812
ibm03	23136	27401	8670	11942
ibm04	27507	31970	11264	14814
ibm05	29347	28446	12859	15808
ibm06	32498	34826	10501	14633
ibm07	45926	48117	14489	19239
ibm08	51309	50513	14642	19484
ibm09	53395	60902	14703	22382
ibm10	69429	75196	22040	25000
ibm11	70558	81454	19609	28271
ibm12	71076	77240	22142	34589
ibm13	84199	99666	23341	33607
ibm14	147605	152772	35802	48745
ibm15	161570	186608	39300	56453
ibm16	183484	190048	41974	58893
ibm17	185495	189581	54937	71395
ibm18	210613	201920	38785	55169
industry2	12637	13419	3633	5495
industry3	15406	21923	7254	10763
s13207P	8772	8651	958	1898
s15850P	10470	10383	1092	2041
s35932	18148	17828	1505	2748
s38417	23949	23843	1587	3037
s38584	20995	20717	2056	3671

For comparison, we chose the Parquet Simulated Annealing floorplanner [1] dated 4/11/2002 which is written in C++. The choice of this annealer over others was due to source-code availability and good performance. We also modified the code to similarly ignore connections to pads and connect all wires to the center of the block. All other code is left the same, and the default cooling schedule and –compact (which was found to appreciably improve results without a significant time penalty) are used. Unless specified, we do not force fixed-outlines—output floorplans may be of any aspect ratio. Per the Parquet Makefile, it is compiled under GNU g++ 2.95.3 with –O3. The –funroll_loops optimization was not used for Parquet since it made the execution run more slowly. As with Traffic, we do not tune any parameters between executions.

Subsections 6.1, 6.2, and 6.3 will evaluate results for the GSRC benchmarks. Table 1 lists statistics for each, showing the number of blocks and nets in the circuit. Subsection 6.4 will use the ISPD and MCNC circuits in Table 2 which have been partitioned from standard cells into 300 and 1000 blocks using Flare [3]. The table shows the net and cell count for the original circuit and the nets after partitioning. Not all MCNC circuits are used since some contain too few cells to produce sufficient blocks for our experiments.

6. EVALUATION

As stated earlier, Traffic produces high quality floorplans in only a fraction of the time of Simulated Annealing. We first compare the performance of these two algorithms when focusing only on white-space minimization, then with fixed-outline constraints, and finally the more relevant cases of wire and large-scale wire minimization.

6.1 Area Minimization

Table 3 shows area-optimization results with data for execution time (in seconds) and white-space (as a percentage of floorplan). For this experiment, we execute the Traffic algorithm for 100 runs and the Parquet Simulated Annealing algorithm for 5 runs, and choose the best area results. Runtime includes all aspects of execution including file I/O, and all experimental parameters are as described in Section 5. Since wire-optimization is not needed, Traffic does not perform any block grouping—only the second phase of the algorithm is executed.

It is evident from Table 3 that Traffic produces floorplans with significantly less white-space than SA in far less time, especially for larger numbers of blocks. These numbers are flexible since we could have only done one SA run instead of 5, or done 1000 Traffic runs instead of 100. However, these run-counts were found to be roughly the point of diminishing marginal returns.

Table 3. Area minimization comparison. Data for run-time (in seconds) and white-space (as floorplan percent) is given.

	Traffic		SA	
circuit	t(s)	ws(%)	t(s)	ws(%)
n10a	0.0	5.3	0.2	9.1
n30a	0.2	4.7	1.9	9.7
n50a	0.4	5.0	5.3	9.1
n100a	1.5	3.5	26.0	10.8
n200a	4.7	3.4	127.2	12.9
n300a	7.0	3.8	337.3	12.7

Since modern circuit floorplanning places more emphasis on wire minimization than area minimization, these numbers are no longer relevant to VLSI design. This experiment shows, however, that Traffic is exceptional at solving the 2-D block packing problem.

6.2 Fixed Outline Floorplanning

A somewhat more pertinent floorplanning case is enforcing a bounding-box constraint. Top level designs of large chips may include outlines of modules which have yet to be laid out. Though these outlines usually have a reasonable amount of built-in white-space, a fixed-outline floorplanner must be flexible enough to fit any aspect ratio.

The Traffic algorithm is easily adaptable to fixed-outlines if the number of buckets (rows) are computed with Equation 3. This formula is similar to Equation 2 except we don't need to calculate the height of the ideal floorplan since it is given. We

also don't round up to the nearest even number. Though this may leave us with an unmatched odd row, the flexibility is needed to achieve any outline size.

Equation 3. Number of buckets for a fixed outline.

$$Buckets_p = \frac{Height_{fixed}}{Height_{avg}} \cdot \frac{N_p}{N_{total}}$$

Table 4 shows the average amount of time the Traffic and SA floorplanners take to satisfy the given aspect ratio with 10% white-space. For Traffic, we execute the algorithm 100 times (each of which would involve multiple runs) and average the amount of time it takes fit the given bounding box. For Parquet we specify the aspect ratio option (-AR), the maximum white-space option (-maxWS), and since it does not stop when it has found a matching solution, 100 runs. Thus the average time to a satisfactory solution is then given as 100 runs divided by the number of successful runs multiplied by the time per run. For example, if there were 20 successful runs out of 100, that would mean an average of 100/20 = 5 runs was needed to achieve the outline, and at 10 seconds per run would give an average time of 50 seconds. If an algorithm could not fit the bounding box within an hour, the table entry is marked with a dash. All other experimental parameters are as given in Section 5.

Table 4. Fixed outline comparison. Data for average time to achieve various aspect ratios with 10% white-space is given.

	Traffic			SA		
circuit	2:1	3:1	4:1	2:1	3:1	4:1
n10a	0.1	0.0	-	0.4	-	0.2
n30a	0.0	0.1	-	1.8	1.8	3.2
n50a	0.1	0.0	1.3	4.4	4.6	6.8
n100a	0.2	0.1	0.0	17.2	21.9	26.5
n200a	0.1	0.1	0.1	86.4	102.7	152.0
n300a	0.1	0.3	0.2	205.4	267.2	351.6

Though smaller circuits are sometimes difficult for both algorithms, results show that Traffic can quickly adapt its geometry to different aspect ratios by only changing the bucket formula. This means that Traffic continues to provide excellent run-times in fixed outline situations.

6.3 Wire Minimization

Table 5 presents results for the more relevant case of wire minimization. Data for run-time (in seconds) and the half-perimeter wire estimate (in mm.) are shown. Setup is as described in Section 5, except we add the "–minWL 1" switch to Parquet and the analogous switch to Traffic to move the focus to wire reduction rather than white-space. As with the boundless area experiment in Section 6.1, we run both algorithms until the approximate point of diminishing returns – 100 Traffic runs and 5 runs of Parquet. Results are once again slightly mutable by doing more or less runs. No fixed outlines were used, though both algorithms support wire minimization within an outline.

In this experiment, runtime improvements are unanimous and the quality improvement of Traffic over SA is clear with more than 50-block floorplans. Given the growing complexity of VLSI circuits, it is reasonable to place more emphasis on these high block-count designs as smaller circuits are tractable enough to be hand-optimized.

Table 5. Wire minimization comparison. Data for run-time (in seconds) and HP wire estimation (in mm.) is given.

	Traffic		SA	
circuit	t (s)	wires	t (s)	wires
n10a	0.0	14.4	1.7	6.2
n30a	0.2	39.3	15.1	27.4
n50a	0.5	83.1	41.5	71.0
n100a	1.4	120.8	163.6	122.5
n200a	6.0	255.4	722.6	270.6
n300a	13.6	384.7	1538.4	448.4

6.4 Large Scale Wire Minimization

The experimental setup of 6.3 becomes increasingly impractical as chip designs grow larger. For evaluating a large design space or small architectural change, expending over five minutes per Simulated Annealing run for 300 blocks is burdensome. For circuits with even more blocks, run-times easily last hours. Thus practical time limits must be placed on the amount of time expended to achieve a solution.

Table 6. Large scale wire minimization comparison with run-time limit set at one second per block. Data for HP wire estimation (in mm.) is given.

	Traffic		SA	
circuit	300 blocks	1000 blocks	300 blocks	1000 blocks
avq_large	183	273	199	729
avq_small	161	256	181	686
golem3	1742	355	2048	807
ibm01	226	308	250	814
ibm02	626	736	752	1792
ibm03	686	884	917	1969
ibm04	883	1076	1109	2547
ibm05	1375	1573	1524	3035
ibm06	993	1229	1275	2876
ibm07	1552	1884	1926	4319
ibm08	1741	2078	2179	4672
ibm09	1495	2137	2037	5311
ibm10	2589	2911	3821	6110
ibm11	2369	3026	3136	7074
ibm12	3101	4143	3929	9298
ibm13	2970	4060	3951	9890
ibm14	5813	7496	9334	18182
ibm15	7043	9314	10814	21409
ibm16	7856	10386	12553	24600
ibm17	10803	13322	19698	30496
ibm18	7706	10633	12011	25281
industry2	188	261	223	664
industry3	378	513	451	1343
s13207P	37	83	31	243
s15850P	46	75	37	210
s35932	75	127	75	347
s38417	100	164	86	186
s38584	115	189	98	219

Table 6 shows the results for very large HGR circuits with 300 and 1000 blocks. Here we implement a one-second-per-block time limit (i.e., 300 seconds for 300 blocks) and analyze the results. Since this limit is less than SA's normal runtime, the execution is sped up. Parquet, like many SA implementations when given a time limit less than a run, reduces its move time to fit one full run exactly within the limit. Thus, the time limit does not cut-off the SA algorithm, but rather hurries it along. For Traffic, the time limit is more than generous, allowing about 1000 runs on 300 blocks and about 600 runs on 1000 blocks. This gives a reasonable exploration of floorplan solutions.

It is important to note that results are not comparable between 300- and 1000-block circuits. The wire estimate given is for only inter-block wires – wires contained within a single block are not included. Thus with fewer inter-block nets in the 300-block versions, there will naturally be a lower wire estimate.

Given the trend in Table 5 these results are predictable and show the growing impracticality of Simulated Annealing for modern designs. Predictors can also give us these numbers quickly, but Traffic produces valid floorplans in the same order of time.

7. CONCLUSION

Traffic derives its advantages from its fast, efficient algorithm. A modified partitioner and binding of well-connected blocks performs global and local grouping. This prepares the circuit for the physical phase in which Traffic uses small, static data structures to provide quick initial layouts. The iterative mutations, based on a simple heuristic, are rapid and productive. The final step of squeezing uses basic geometry to produce layouts with little white-space, which is correlated to reducing global wire-length. The end-result is an algorithm capable of besting Simulating Annealing results in much less time.

Moreover, we believe the importance of floorplanning speed and quality will only increase. As transistor integration continues to grow, rapid feedback on design changes is needed at all levels of design. At the highest level, architects can no longer assume that the physical design is an independent stage, separate from their concerns. Thus these designers require a way of physically evaluating small changes to large chips, such as changes in buffer sizes and bus width. For this application, Traffic can produce viable floorplans for very large circuits in seconds rather than hours, giving immediate feedback to the architect in similar time as a predictor.

At the block level, engineers must also make educated decisions concerning layouts. Choosing 10 large or 1000 small blocks will make a significant impact on routability, timing, and similar concerns. Traffic gives layout designers a way to generate and evaluate layouts for all points within this solution space in a practical amount of time and thus improve their design choices.

At the final stage of design, run-time is less critical. Hours are a reasonable investment of time for floorplanning a completed chip, thus SA is a reasonable choice here. In these situations, however, we believe that Traffic can greatly accelerate the speed and improve the quality of the annealing process through the creation of initial solutions. Previous work [12] has shown that by using a different algorithm for initial placements, Simulated Annealing can produce better results in significantly less time.

Current Traffic work is focused on quantifying this speedup of SA with our initial solutions. Other ongoing investigations include the addition of rectilinear shapes and fixed blocks to our algorithm, as well as soft-blocks and timing. We are also investigating the use of Traffic in areas other than VLSI design, such as OS process scheduling. Indeed, many common problems such as truck loading and wrapping-paper slicing can be solved with floorplanning techniques. Given the priority on wire rather than area minimization, VLSI floorplanning presents a unique challenge. Traffic, however, successfully addresses this complex problem with scalability in mind.

8. REFERENCES

[1] S. Adya and I. Markov, "Fixed-Outline Floorplanning Through Better Local Search," *Proceedings of the International Conference on Computer Design*, pages 328-334, 2001.

[2] Y. Chang, Y. Chang, G. Wu, and S. Wu, "B -Trees: A new representation for nonslicing floorplans," *Proceedings of the Design Automation Conference*, pages 458-463, 2000.

[3] J. Cong and S. K. Lim, "Physical Planning with Retiming," *Proceedings of the International Conference on Computer-Aided Design*, pages 2-7, 2000.

[4] W. E. Donath, "Placement and Average Interconnection Lengths of Computer Logic," *IEEE Transactions on Circuits and Systems.*, vol. 26, pages 272-277, 1979.

[5] A. A. El Gamal, "Two-Dimensional Stochastic Model for Interconnections in Master Slice Integrated Circuits," *IEEE Transactions on. Circuit and Systems*, vol. 28, pages 127-138, 1981.

[6] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-tree Representation of Nonslicing Floorplan and Its Applications," *Proceedings of the Design Automation Conference*, pages 268-273, 1999.

[7] T. Hamada, C. K. Cheng, and P. M. Chau, "A Wire Length Estimation Technique Utilizing Neighborhood Density Equations," *IEEE Transactions on Computer-Aided Design*, vol. 15, pages 912-922, 1996.

[8] S. Kang. "Linear Ordering and Application to Placement," *Proceedings of the Design Automation Conference*, pages 457-464, 1983.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, pages 671-680, 1983.

[10] H. Murata, K. Fujiyoushi, S. Nakatake, and Y. Kajitani, "Rectangle-Packing-Based Module Placement," *Proceedings of the International Conference on Computer-Aided Design*, pages 472-479, 1995.

[11] M. Pedram and B. Preas, "Interconnection Length Estimation for Optimized Standard Cell Layouts," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 390-393, 1989.

[12] A. Ranjan, K. Bazargan, S. Ogrenci and M. Sarrafzadeh, "Fast Floorplanning for Effective Prediction and Construction," *IEEE Transactions on VLSI Systems*, Vol. 9, pages 341-351, 2001.

[13] C. Sechen, "Average Interconnection Length Estimation for Random and Optimized Placements," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 190-193, 1987.

[14] L. Stockmeyer. , "Optimal Orientations of Cells in Slicing Floorplan Designs," *Information Control*, vol. 59, no. 2, pages 91-101, May 1983.

[15] T.-C. Wang and D. Wong, "An Optimal Algorithm for Floorplan Area Optimization," *Proceedings of the Design Automation Conference*, pages 180-186, 1990.

[16] D. Wong and C. Liu, "A New Algorithm for Floorplan Design," *Proceedings of the Design Automation Conference*, pages 101-107, 1986.

[17] F. Young and D. Wong, "Slicing Floorplans with Pre-placed Modules," *Proceedings of the International Conference on Computer-Aided Design*, pages 252-258, 1998.