Principles and Preliminary Results for Force-Directed Floorplanning with Malleable Blocks

Joanna A. Ellis-Monaghan
Department of Mathematics
St. Michael's College
Colchester, VT 05439

Paul Gutwin
Cadence Design Systems
San Jose, CA 95134

Jamey Lewis
St. Michael's College
Colchester, VT 05439

Greta M. Pangborn
Department of Computer Science
St. Michael's College
Colchester, VT 05439

Abstract

We apply force-directed graph drawing techniques to the floorplanning process of computer chip design, which is essentially a problem of fitting interconnected rectangles into a prescribed region without overlap. We adapt the force-directed graph drawing techniques to accommodate the rectangular vertices representing chip components by developing a physical model that allows the components to 'pass through' each other and to adjust their aspect ratios as needed while approaching a solution. We provide the underlying mathematics and some preliminary output from a prototype program based on our heuristics.

Key words: Floorplanning, netlist layout, spring embedding, force-directed, graph drawing, geometric graph theory, geometric visualization, routing, wiring, placement.

Introduction

A major component of computer chip design consists of generating a netlist layout, i.e. determining where to place the gates (functional elements) and how to route the wires (connections between gates) when manufacturing a chip. We present a physically motivated graph theoretical model for floorplanning, an early step in this process. During the physical design of the chip, the floorplanning step determines a rough high-level grouping and placement of related gates within the chip area. We assume the first part, grouping related gates into larger blocks with fixed area, has been accomplished, and consider the question of arranging the interconnected blocks into a fixed rectangular region (the chip area).

A netlist is the logical description of the function(s) of a computer chip. The netlist describes functional elements (often basic Boolean operators like AND and NOT) and the interconnections between them. It specifies the pins, the partition of the pins into operating units of fixed size called gates, a second partition of the pins corresponding to interconnections or signals (which translate physically to wires), and lastly, for each pair of pins a maximum delay, represented by limits on the lengths of the wires.

Given a netlist, the goal is to design a layout that provides a geometric realization of the logical circuit from which the chip is actually manufactured. For large chips, the layout process currently consists of three separate but closely related problems. The first is floorplanning, where a floorplan is a rough high-level grouping and placement of related gates within the chip area. The floorplan can be viewed as a rough map of where major subsections of the netlist should be placed relative to the chip edge for connection to the "outside world", and relative to each other, for performance reasons. The second is the specific placement of the gates on the placement (bottommost) layer so that the gates do not overlap. The gates are placed in as compact a fashion as possible, and the placement minimizes the distance between gates connected together by wires. The third is a rectilinear routing of the wires in the wiring space (successive layers above the gate placement layer providing alternating horizontal and vertical channels in which to place wires), so that the distances between every pair of pins in the same signal are less than their delays and so that the set of wires is pair-wise disjoint (topologically). See Figures 1 and 2. Since many aspects of these processes are known to be NP-Hard, the overall problem realistically becomes finding reasonable heuristics for generating good layouts. A comprehensive introduction to chip design is given in [Len90], and an overview of graph theoretic techniques in network design can be found in [Cac89].

The placement and wiring problems are very closely connected, as a poor placement can dramatically affect the difficulty in finding a satisfactory routing solution. The floorplanning step provides global guidance for the placement and routing steps, anticipating the challenges facing the placement and wiring algorithms. Thus, a high quality floorplan can be very effective in reducing the difficulty of the placement and wiring challenges. However, as the floorplan must anticipate many aspects of the detailed implementation, generating a floorplan (especially by hand) can be very time consuming.

Due to the highly competitive nature of the microelectronics industry, there is strong interest in heuristics that may shorten the chip design cycle by effectively automating the floorplannning process, and improving the floorplan quality. Heuristics that better integrate the three processes of floorplanning, placement, and routing, have recently led to improvements in the layout process. This approach has been taken in [CCPY00], one of the trailblazers in the relatively new idea of considering the signal delays during the floorplanning

process. By moving optimization techniques usually not implemented until the routing stage into the floorplanning process, they were able to achieve significant improvement in delay reduction for their test case. Similarly, [VC04], [MTB00], [CCPY00] and [EJ98] have achieved notable improvements in automated layout by using force-directed methods.

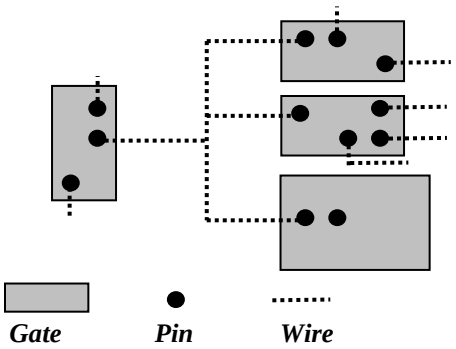

Figure 1. Expanded view of chip components (gates actually abut, with wires on metal layers above the gates).

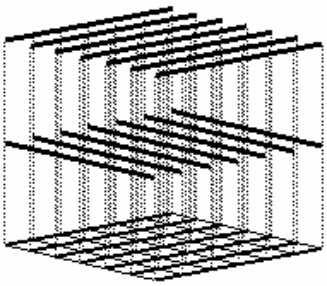

Figure 2. The wiring space, showing the placement layer on the bottom and two wiring layers above.

However, one of the critical considerations involved in using force-directed methods is preventing overlap of the blocks. The novelty of our approach lies in our handling of preventing block overlap and reshaping blocks dynamically as the system nears a solution. As in other recent force-directed layout algorithms, we model signal delays with spring forces along edges, thus simultaneously addressing both routing and placement in the floorplanning process. However, we considerably modify force-directed graph drawing techniques by developing

a physical model that allows the vertex blocks to 'pass through' each other and to adjust aspect ratios as needed while approaching a solution.

A Force-Directed Floorplanning Approach

The floorplanning phase uses the logical design of the chip to grossly partition the netlist into about ten to fifty blocks (see Figure 3). Here we see a tree denoting the major sub-blocks of the total netlist (CHIP_TOP). Each major sub-block may contain simply gate level elements (e.g. MAC), or other sub-blocks (e.g. DSP) or both sub-blocks and gates (not shown). Each of these sub-blocks will have fixed area, since each contains a given number of individual gates (transiently), each with fixed physical dimension. However, the aspect ratios of these blocks are malleable as long as the area stays constant and the rectangle aspect ratios stay within given bounds. The connections among the blocks derive from the gates they contain, and thus the blocks are typically highly interconnected. Figure 4 shows a potential floorplan of the netlist in Figure 3. Note that not every level of the hierarchy in the netlist tree is represented in the floorplan – the choice of which sub-block in the hierarchy to plan for in the floorplan depends on many variables, and is outside the scope of our discussion. We assume that a good selection of the sub-blocks has been provided for the floorplan.

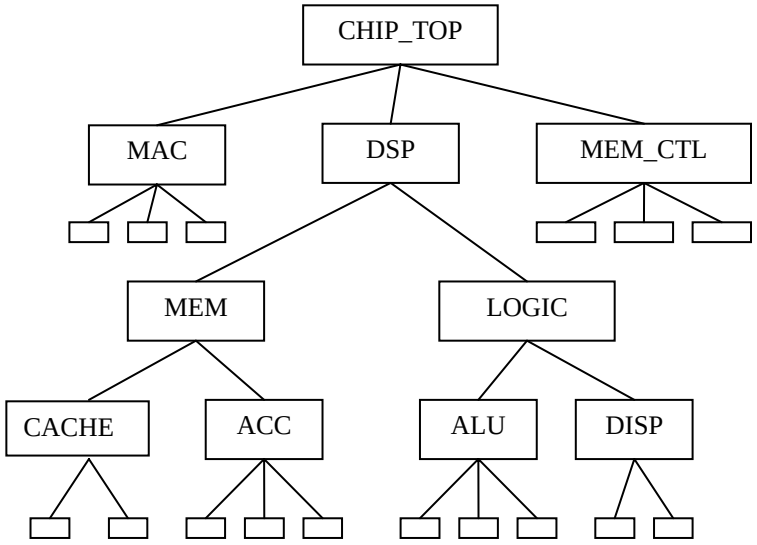

Figure 3. Grouping of components in the logic hierarchy of the netlist.

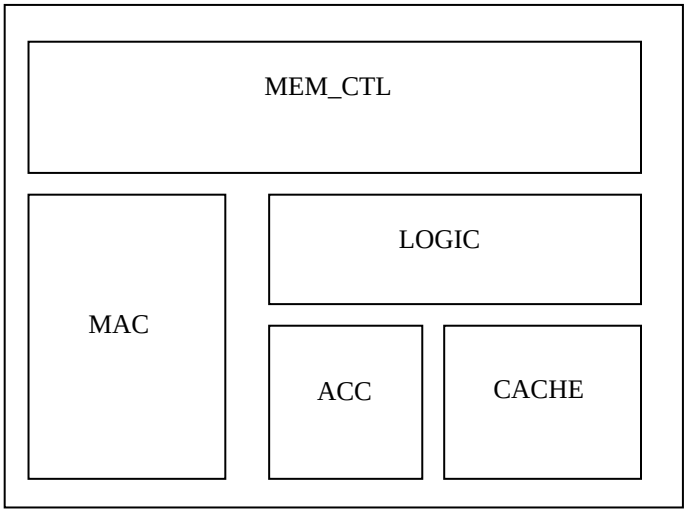

Figure 4. A floorplan (interconnections not shown).

Graphs and hypergraphs are natural mathematical objects for modeling netlists during the different stages of the layout process. During the floorplanning stage, the nodes of the graph represent the blocks and the edges the connections between blocks implied by the wires. Eades [E84] first suggested a spring embedding algorithm in which the vertices and edges are veiwed as masses and springs and thus subjected to physical force laws modeled on Hooke's law and electrical repulsion equations. The algorithm calculates the forces acting on each mass and adjusts each node's position accordingly, then repeats this process iteratively. The system as a whole eventually stabilizes in a configuration that minimizes the energy of the springs (at least locally).

This approach has been explored by [VC04], [MTB00], [CCPY00] and [EJ98] for various aspects of netlist layout. However, the difficulty comes from the fact that for floorplanning, as in other layout processes, the nodes have physical dimensions and may not overlap. This usually prevents the nodes from being able to occupy the position determined by a simple spring embedding algorithm. Various approaches, from bin packing in [MTB00] to congestion-based repulsive forces in [EJ98], have been used to address the problem of preventing overlapping. We have chosen to work directly with the blocks as physical objects, in part because of the need for a floorplanning model to incorporate the malleability of these floorplan blocks.

In addition to the blocks, which have an area, our model includes two other types of vertices, both of which are dimensionless. One type represents the

signals among the blocks. These interconnections among the blocks are actually hyperedges, involving sets of blocks rather than simply pairs. As do, for example [MBT00] and [VC04], we model these hyperedges by claws ($K_{1,n}$ graphs), creating an artificial vertex for each claw, as in Figure 5. Like [VC04], we use cliques instead of claws for hyperedges of size two or three to reduce the number of computations. The other kind of vertex occurs along the perimeter of the layout area and represents a connection between a block and the edge of the chip.

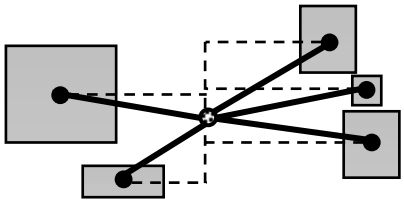

Figure 5. A signal (dotted lines) modeled by a claw on an artificial vertex.

As in general spring embedding algorithms, at the core of our algorithm is an iterative process that repeatedly computes attractive and repulsive forces. Our use of both local and global temperatures to control oscillation and force the system to converge to a solution, as well as our spring force computations, are fairly standard. However, we modify the repulsive forces between blocks in three ways. First, the blocks repel from a rectangular perimeter rather than a dimensionless point, and only repel to this perimeter rather than simply using an inverse square law. Second, this repelling perimeter grows outward from the center of the block as the algorithm progresses, beginning very small, and then eventually encompassing the given area of the block. This allows the blocks to pass through one another other early in the algorithm as the system grossly positions the components, but prevents blocks from overlapping as the system nears stability. We refer to this perimeter as the *effective* width and height of the block. Thirdly, we compute boundary pressures (depth of penetration by neighboring blocks) for each block, and then adjust the aspect ratio of the block to equalize these pressures, thus allowing blocks to find good shapes as well as positions in the final floorplan.

Since we assume the gates have previously been grouped into blocks, our input consists of:

- the rectangular chip layout area,
- the given blocks of related chip components,
- the areas of these blocks,
- information about how the blocks are connected among themselves and to the perimeter of the layout area,
- delay constraints on the interconnections.

The principal components of our algorithm, discussed in detail below, are:

- the temperature,
- the spring equations on the signal edges,
- the repulsion equations on the blocks,
- the pressure equalization equations to reshape overlapping blocks.

The output is a floorplan, that is an arrangement of the blocks in the layout area so that the blocks do not overlap and so that connected blocks with short delays are placed near one another.

Temperature

Temperature works both locally and globally to ensure termination of the algorithm. We increase or decrease a temperature variable associated to each individual block or vertex based on its prior movement. We detect oscillation by comparing the direction of the vertex's movement with its last recorded movement direction. If the difference is greater than 90 degrees, we decrease the temperature of the vertex and slow its movement to damp any oscillation. Otherwise, we assume the vertex is moving toward a better position, so we increase its temperature and hence speed its movement toward that position. We also globally decrement the temperature variables throughout a run, slowly cooling the system as a whole, and thus forcing a final stable state.

Spring Forces

We apply attractive forces modeled on spring equations along the edges representing connections between components, where a component may be a block or one of the two types of special vertices. Given an edge (*u*, *v*) between a component *u* with center (x_u, y_u) and a component *v* with center (x_v, y_v), equation (1.1) describes the impulse of x_u (the *x*-coordinate of the center point of component *u*) to change due to the influence of edge (*u*,*v*):

$$S \cdot W_{uv} \left(d_{uv} - l_{uv}\right) \cdot \frac{x_v - x_u}{d_{uv}} \cdot \frac{T_u}{T} \text{, where} \tag{1.1}$$

- *S* is a global spring stiffness parameter,
- W_{uv} is an edge springiness parameter determined by the timing constraints in the netlist,
- d_{uv} is the Euclidean distance between the centers,

- l_{uv} is the ideal length of the spring between u and v, determined by the timing constraints in the netlist and the effective dimensions of the blocks,
- T_u is the local temperature of u,
- T is a global maximum temperature.

We then sum over all edges incident with u to find the new value of x_u and compute the new value of y_u analogously, thus setting the new position of component u for the next iteration. We handle the special case when u and v are both blocks a little differently. In the case that the blocks are offset more horizontally, we replace d_{uv} by the difference in the x coordinates, and l_{uv} by the average effective width, and simply increment vertically by half the difference in the y coordinates, treating the case of greater vertical offset analogously.

Determining reasonable values of W_{uv} and l_{uv} for each edge from the raw netlist data is a separate and very challenging problem in its own right. Therefore, following common practice, we currently set these values globally, with SW_{uv} always equal to one and with l_{uv} equal one of the average effective height or width of u and v, or else very near zero. We then simply seek to minimize the total wire lengths in the final solution. We retain the parameter S as a control variable for future implementation combining the current work with a genetic algorithm.

Repulsion Forces Among Blocks

The purpose of the (electrical) repulsion equations in many traditional force-directed graph drawing algorithms is to spread the vertices and thus provide a good visualization. Since our block objects have a prescribed height and width, they require a different repulsion model. In the final solution blocks cannot overlap. However, we do not require (or want) excess space between blocks. Thus, there should be very little repulsion between non-overlapping blocks. Additionally, we need to allow the blocks flexibility to "move through" each other to better positions initially, so at the start of a run the repulsion between overlapping blocks should allow overlaps, but at the end of the run prohibit them. We achieve this by means of the effective widths and heights of the blocks, multiplying each dimension of each block by $\frac{B-R}{B}$, where B is a global constant and R increments from B to 0 throughout the run..

The following equation describes the impulse of x_u and y_u (the x and y coordinates, respectively, of the center point of component u) to change due to the influence of a block v whose effective perimeter overlaps that of u, in the case that $|x_u - x_v| > |y_u - y_v|$:

(1.2)

$$\Delta_v(x_u) = \left(\frac{AEW(u,v) - |x_u - x_v|}{2}\right)\frac{(x_u - x_v)}{|x_u - x_v|}$$

$$\Delta_v(y_u) = \frac{|y_u - y_v|}{2|x_u - x_v|}\left(\frac{AEW(u,v) - |x_u - x_v|}{2}\right)\frac{(y_u - y_v)}{|y_u - y_v|}$$

where $AEW(u,v)$ is the average of the effective widths of blocks *u* and *v*. Analogous equations apply if $|x_u - x_v| < |y_u - y_v|$.

We sum over all blocks *v* whose effective perimeter overlaps that of *u*, to get the total changes Δx and Δy, and then set $x_{new} = x_{old} + \Delta x$ and $y_{new} = y_{old} + \Delta y$.

The motivation for the non-symmetric equations for $\Delta_v(x_u)$ and $\Delta_v(y_u)$ comes from wanting to move overlapping blocks so that their effective perimeters just abut, but to do so in the most effective of the vertical or horizontal directions, while still moving a proportionally lesser amount in the other direction. I.e., we favor one direction over the other, but not completely. The repulsion is still proportional to the amount of overlap between the two blocks, so the greater the overlap, the greater the repulsion.

In some runs we set the size of a bounding rectangle representing the fixed chip area in which we want the components to reside. If part or all of a component is outside the bounding box we apply a repulsion force to push it in towards the center.

Pressure Responsive Blocks

A unique aspect of the floorplanning problem is that the blocks may be reshaped as long as their areas remain constant and their aspect ratios stay within given bounds. Intuitively, this flexibility should lead to better floorplanning solutions than might be possible with rigid blocks. The challenge is finding a way to leverage this advantage. In our model, we view the blocks as being compressed both horizontally and vertically by overlapping neighboring blocks as the system converges toward a solution. Within the physical system model, it seems reasonable that a block would adjust its shape to equalize these pressures, and this is the basis of our approach.

For a block *u* we determine the maximum horizontal and the maximum vertical penetrations by any blocks that overlap it. If the maximum horizontal and vertical penetrations come from distinct overlapping neighbors, we reshape the block to equalize these penetrations, as in Figure 6. In the case that the maximum penetrations come from a single neighbor, as in Figure 7, we compress the block only in that direction. In any case, we do not want to

decrease the width (height) if the horizontal (vertical) penetration is greater than half the width (height), since such an overlap is likely to be better handled by either changing the shape of the overlapping block or by the action of the repulsion forces. Thus, we only consider penetrations into less than half the height or width of a block.

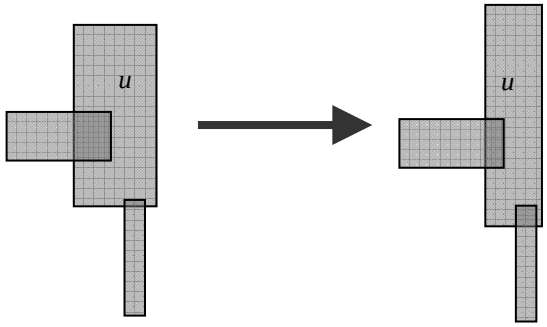

Figure 6. Block *u* reshaping to equalize pressures.

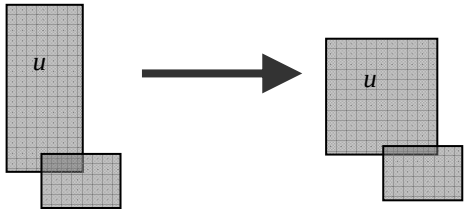

Figure 7. Block *u* responding to maximum penetrations by a single neighbor. The neighboring block will also resize, eliminating the remaining half of the penetration, leaving no overlap.

When the maximum horizontal and vertical overlaps are contributed by two different blocks, *v* and *w*, then our goal is to equalize the horizontal and vertical penetration pressures in equations (1.3) and (1.4).

(1.3) $$horizontal\ penetration = \left(\frac{width_u + width_v}{2} \right) - | x_u - x_v |$$

(1.4) $$vertical\ penetration = \left(\frac{height_u + height_w}{2} \right) - | y_u - y_w |.$$

Here, (x_i, y_i) are the coordinates of the center point of block *i*.

We set these penetration values equal and use of the fixed area of *u* with *area = width*height* to solve for the new width of *u*.

(1.5)

$$-\frac{width_u^{\,2}}{2}+\left(\frac{width_v}{2}-|x_u-x_v|-\frac{height_w}{2}+|y_u-y_w|\right)*width_u-\frac{Area_u}{2}$$

This equation has exactly one positive root, corresponding to the width of a rectangle shaped to equalize penetration. If this value lies between the minimum and maximum allowed value of the width, we set the equilibrium width of *u* to this value. Otherwise, we set the width to whichever of the minimum or maximum allowed width is closest to this value.

In the case that the maximum penetrations come from a single neighbor, we react to the smaller of the penetration directions to minimize the resulting change in shape. If the smaller penetration is horizontal, we subtract half the penetration distance from the old width of *u* to determine the equilibrium width, and use the fixed area to determine the corresponding height. Again, this is only if the resulting equilibrium width is within the allowed range, otherwise we set the equilibrium width to the minimum allowed width. We follow the analogous process if the smaller penetration is vertical. We adjust the block by only half the overlap amount since the neighboring block will also be adjusted by at least as much when we compute its pressure equations.

As with the spring and repulsion equations, we incorporate the temperature into this scheme in order to prevent oscillation and ensure the termination of the algorithm. Thus, we set the new width of the block to be a convex combination of the old width and the equilibrium width we have just computed. As the temperature of the vertex goes to zero as the algorithm progresses, the weight placed on the old width will increase.

(1.6) $$newWidth=\left(1-\frac{T_u}{T}\right)*oldWidth+\left(\frac{T_u}{T}\right)*equilibriumWidth\,,$$

where as before, T_u is the temperature of component *u*, and *T* is a global maximum temperature.

Experimental Results

Our first example is a four by four grid of uniformly sized rectangles with randomized initial positions. Figures 8-11 show the progression of the algorithm. Figure 8 shows the initial random positions of the blocks. Figure 9 demonstrates how the spring forces dominate in the early part of a run. Figure 10 illustrates the progressive effect of the repulsion forces and pressure equalizations. Figure 11 gives the final result.

Note that the rectangles in Figure 11 are narrower than the initial rectangles in Figure 8. This results from the algorithm striking a balance between the conflicting objectives of minimizing the total sum of edge lengths and respecting equal spring tensions on the individual edges (shorter overall edge length is possible, but at the expense of one or more disproportionately long edges).

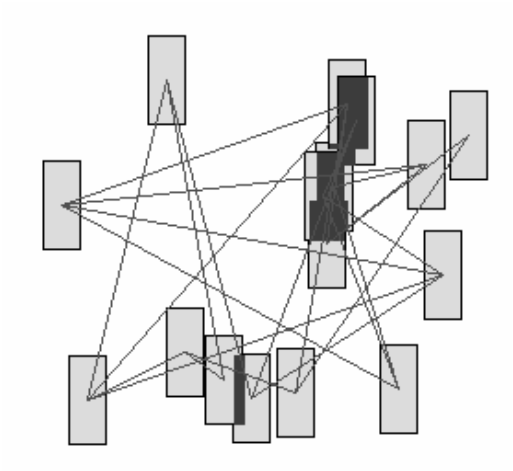

Figure 8. 4x4 grid with randomized block positions.

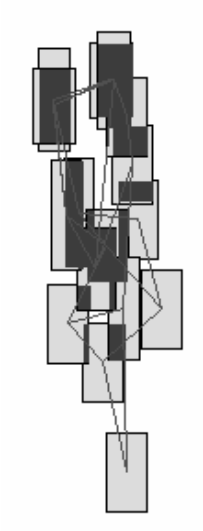

Figure 9. Spring Forces dominate early in the run, and blocks pass through one another.

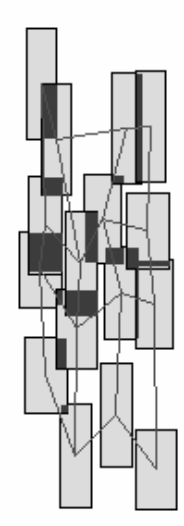

Figure 10. Effect of repulsion and penetration pressures.

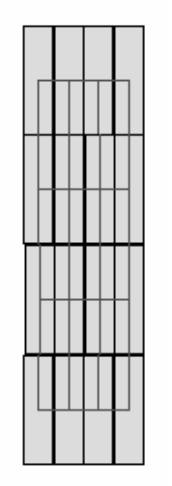

Figure 11. Final result for the 4x4 grid.

Figures 12 and 13 show the initial and final positions for a representative floorplanning example that incorporates disparately sized blocks, bounding box, perimeter pins, and signal claws**.** Future work includes modifying pressure equalization equations for the bounding box.

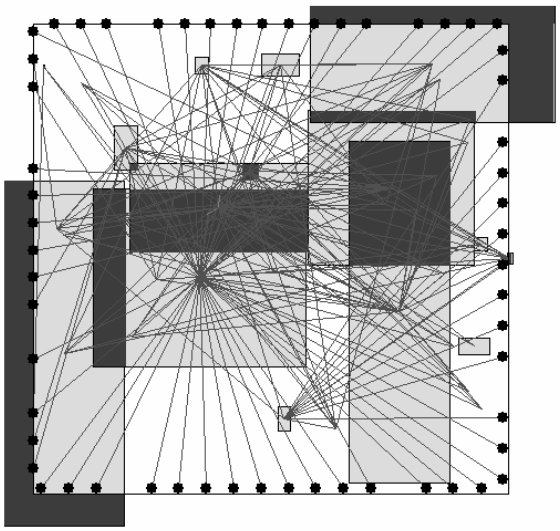

Figure 12. Floorplan blocks in initial (randomized) positions.

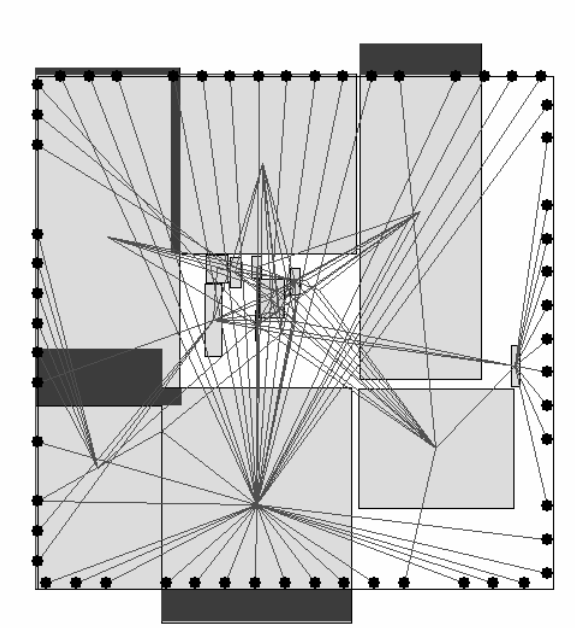

Figure 13. Floorplanning final position.

Acknowledgements

Portions of this work were supported by VT EPSCoR under grant NSF EPS 0236976 and by NASA under Training Grant NGT5-40110 to the Vermont Space Grant Consortium.

Bibliography

[Cac89] L. CACCETAA, Graph theory in network design and analysis, *Recent Studies in Graph Theory*, V.R. Kulli, ed. Vishwa International Publications (1989) pp 29-63.

[CCPY00] C.-C. CHANG, J. CONG, D. Z. PAN, X. YUAN, Interconnect-driven floorplanning with fast global wiring planning and optimization, *Proceedings SRC TechCon Conference, September 21-23*, Phoenix, 2000.

[E84] P. Eades, "A Heuristic for Graph Drawing," Congressus Numerantium, 42, 149-160, 1984.

[EJ98] H. EISENMANN, F.M. JOHANNES, Generic global placement and floorplanning, *Proc. 35th Annual Conference on Design Automation Conference*, 1998.

[Len90] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, 1990.

[MTB00] F. MO, A. TABBARA, R.K. BRAYTON, A force-directed macro-cell placer, *IEEE/ACM International Conference on Computer Aided Design*, November 2000, ICCADOO, Santa Clara.

[VC04] N. VISWANATHAN, C. CHU, FastPlace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net, *ISPD*, April 18-21, Phoenix, AZ pp. 26-33.