Chapter 3

Macro Cell Placement and Routing

3.1 Introduction

In this chapter, we present algorithms for the placement and routing of rectilinearly shaped macro cells. Large complex logic functions such as datapaths, arithmetic logic units (ALUs), and random access memory (RAM) are most efficiently implemented using the macro cell design style. The macro design style closely resembles the hand placed custom design style; like custom design, macro cells are very area efficient. However, global and detail routing are more difficult than row-based styles.

We present the TimberWolfMC (TimberWolf Macro Cell) program which performs floorplanning, placement, and global routing of macro cells. Both hard macros (cells with fixed geometries including pin locations) and soft macros (cells which have undetermined characteristics such as aspect ratio, shape, and pin placement) are supported. In addition, the user may specify several possible versions for a cell. For example, a 1k x 16 bit RAM may be implemented using different multiplexers and decoders. Each implementation yields a different aspect ratio and shape. Yet, all are functionally equivalent. The TimberWolfMC package is the only program to simultaneously address all constraints during placement.

3.2 Previous Work

There have been numerous algorithms proposed to solve the placement problem. Placement algorithms may be broadly divided into four categories: constructive, iterative, analytical, and esoteric algorithms. Constructive algorithms selectively add one cell at a time to the placement based on an evaluation or *score* function. The score function measures the degree of connectivity to the cells which have been placed. The cell with the best score is placed at the position which minimizes total wire length. This process continues until all cells are placed. Speed is the constructive algorithm's main advantage (time complexity $O(n)$). However, constructive algorithms often are *stuck* in local minima which are quite far from the global minimum and hence they usually yield rather poor results. Their speed does make it useful as an initial placement for an iterative algorithm. Hanan first applied constructive algorithms in 1972 [83]. Recently, Sutanthavibul and Shragowitz have augmented the constructive placement algorithm to handle timing constraints [217][218].

Iterative algorithms improve an initial placement by modifying the configuration. Iterative algorithms constitute the largest class of placement algorithms. This class can be further subdivided into deterministic and stochastic algorithms. Among the deterministic class are Steinberg's algorithm, pairwise exchange algorithms, force-directed placement and its variants, and the family of mincut bipartitioning algorithms. The stochastic algorithms include simulated annealing, simulated evolution, and the genetic placement algorithms.

Steinberg's algorithm partitions the netlist into *independent* sets [213]. The independent sets contain cells without common nets. The algorithm iteratively selects a set of independent cells, removes them from the placement, and computes the cost of replacing them in each of the available slots. Because these costs are independent of the location of the other cells in the independent set, an optimum assignment of cells to slots can be found solving the resultant linear assignment problem[1]. Although each independent set may be assigned optimally, it is not guaranteed that the total placement will be optimal.

1. Assuming equal width cells.

The pairwise interchange method begins with an initial cell placement. During each iteration, two cells are selected and trial interchanged. If the interchange does not increase the wire length, the exchange is accepted; otherwise, the move is rejected and the cells restored to their original positions. If there are n cells in the layout, $O(n^2)$ exchanges are attempted in a *cycle*. Interchanges continue until no improvement in wire length is made in a cycle. The closely related neighborhood interchange algorithm only exchanges cells in the vicinity of each other.

In force-directed algorithms, a force vector $\vec{F}_c$ is computed for each cell c, and

$$\vec{F}_c = \sum_i a_{ci} \cdot \vec{d}_{ki} \tag{3.1}$$

where a_{ki} is the edge weight connecting cells c and i, and $\vec{d}_{ki}$ is the vector distance from c to i. The force vector is used to calculate the target position for the cell c. The target position is the location where the sum of all forces on the cell equals zero. Cells may be *relaxed* to zero force target position in many ways: force-directed pairwise interchange (FDPI), force-directed pairwise relaxation (FDPR), and generalized force-directed pairwise relaxation (GFDR) [77].

In a force-directed pairwise interchange algorithm, the force on a cell is computed and trial interchanged with each of its three neighbors in the direction of the force vector. If the wire length is reduced, the interchange is accepted.

In the force-directed pairwise relaxation method, a cell a is selected and its target location calculated. Another cell b is selected with the following criteria: it is within ε distance of a's target slot, and its own target is within ε of a's original location. Cells a and b are trial interchanged. If the wire length is reduced, the interchange is accepted.

Generalized force-directed pairwise relaxation attempts to interchange more than two cells at the same time. If the interchanges are performed on random subsets of cells, the improvement in solution quality will not compensate for the large increase in computation time. Instead, the GFDR method selects trial interchanges for subsets of cells which have a high probability for success.

The mincut family of algorithms are used extensively in industry. They are based on the partitioning algorithm presented by Kernighan and Lin [108]. In this algorithm, the goal is to divide a netlist into two parts such that the number of nets connecting the two parts is minimum. The number of nets interconnecting the partitions is known as the *cut* value of the partition. The Kernighan and Lin algorithm assumes every cell is the same size, and every net connects to exactly two cells.

The algorithm starts with an initial arbitrary partition with half the cells in partition *A* and half in partition *B*. The *gain g* is calculated for all pairs of cells between partitions. The gain is the increase in the cut value when two cells are exchanged between partitions. One cell from each partition is selected such that the interchange of the cells results in the minimum gain (or maximum reduction) in cut value. Once interchanged, the cells are *locked* or forced to remain in their new partition for the remainder of a *pass*. A pass is a total exchange of all cells. During a single step in a pass, the two unlocked cells with minimum gain are chosen for the exchange. At the end of a pass, all cells initially in *A* have been moved to *B*, and those originally in *B* have been transferred to *A* resulting in a net gain of zero, or

$$\sum_{i=1}^{N} g_i = 0. \tag{3.2}$$

From Equation 3.2, we see that either some of the g_i are negative, or all terms are zero. In order to maximize the total decrease in cut value, the Kernighan and Lin algorithm selects the *k* such that

$$\sum_{i=1}^{k} g_i,\ 1 \le k \le N \tag{3.3}$$

is a minimum. Cells $a_1, a_2, ..., a_k$ are transferred to partition *B*, and cells $b_1, b_2, ..., b_k$ are transferred to partition *A* ending a pass. Passes are performed until

$$\sum_{i=1}^{k} g_i \ge 0,\ \forall k \in \{1, ..., N\}\ . \tag{3.4}$$

Although some of the individual exchanges may have increased the cut value, the total decrease in cut value is a maximum. This was the first partitioning or placement algorithm which allowed uphill moves in an attempt to escape from a local minimum.

The partitioning algorithm may be used as a placement algorithm by recursively partitioning the cells into two equal halves until only one cell is in each partition. Breuer proposed a wide variety of partitioning strategies based on recursive partitioning [16].

Schweikert and Kernighan extended the partitioning algorithm to handle nets with more than two connections [194]. In 1982, Fiduccia and Mattheyses reduced the time complexity from $O(N^2 \log N)$, where *N* is the number of cells, to $O(P)$, where *P* is the number of signal pins [64]. Their method exchanges only single cells. In addition, they added constraints to control differences in cell sizes. Although Dunlop and Kernighan found that the quality of the Fiduccia and Mattheyses algorithm does not always equal that of the Kernighan and Lin method, the linear time complexity makes it attractive for large designs [56].

The major drawback with recursive bipartitioning as a placement procedure is the partitioning of a set of cells at one level of the hierarchy ignores connections to cells in other levels. In Figure 3.1, we see that recursive partitioning without regard to other levels of hierarchy often leads to nonoptimal placements.

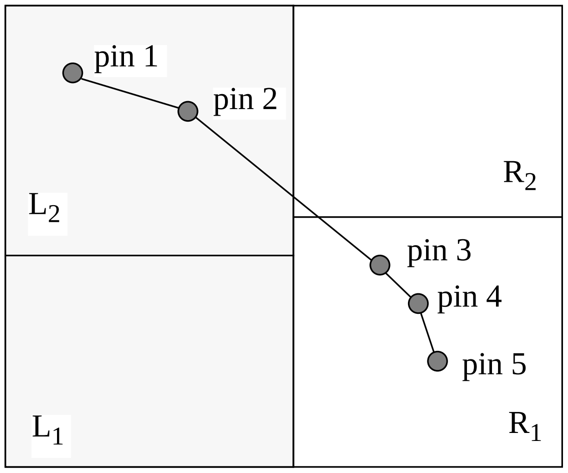


Figure 3.1 Recursive partitioning ignores connections in other levels. At the first level, the design is partitioned into L and R. During the partitioning of L into L_1 and L_2, pins 1 and 2 are assigned to partition L_2. It is possible for pins 3, 4, and 5 to end up in partition R_1

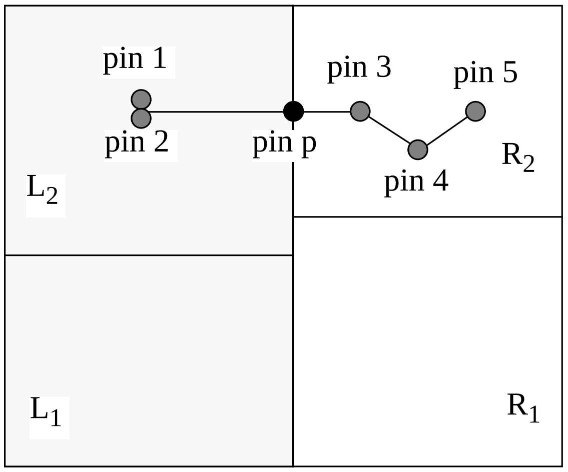


Figure 3.2 Recursive partitioning using terminal propagation. The addition of pin *p* biases the placement of pins 3, 4, and 5.

In response to this problem, Dunlop and Kernighan proposed *terminal propagation* to improve mincut placement. In order to properly bias cells in future partitioning steps, a propagation pin *p* is added to account for cells outside the partition. The propagation pin is centered on the boundary of the partition. Dunlop and Kernighan reported that terminal propagation reduced the chip area by an average of 30 percent, a substantial savings [56].

Suaris and Kedem extended the concept of terminal propagation to *quadrisection* [214][215][216]. Mincut quadrisection involves simultaneous partitioning along horizontal and vertical lines. The number of net crossings across all divisions is minimized.

Using a polar graph representation for macro blocks, Lauther was able to adapt the mincut algorithm to place blocks of any rectilinear size [127]. Later, Mayrhofer and Lauther extended the quadrisection concept to multiple partitions and added a congestion metric using Steiner trees [146]. Many systems such as the BEAR building-block system use variants of recursive mincut in macro floorplanning tools [47].

Stochastic algorithms fill the other important subclass of iterative algorithms. Stochastic algorithms *randomly* perturb the current state searching to discover a better solution. Simulated annealing, genetic, and simulated evolution placement algorithms are the most important members of the class. Simulated annealing was first applied to macro cell placement by Jepsen and Gelatt [101]. Wong and Liu introduced Polish notation to speed the search, but their method does not allow overlap during the annealing process[238][239]. In our experience with row based circuits, we found overlap during annealing leads to substantial area savings[201]. Siarry *et al*. analyzed simulated annealing for the idealized macro cell case in which all nets interconnect only two pins[209]. There have been several reports of parallel versions of simulated annealing for macro cells [26][178]. None of these systems dynamically estimated the routing area needed around the cells, included a placement refinement stage, or had the range of features offered by TimberWolfMC.

In genetic algorithms, a *population* of solutions to the problem is maintained, and successive generations are produced by manipulating the solutions in the current population. Each solution has a *fitness* which measures its competence. New solutions may be created from the merging of two current solutions using the *crossover* operator. Other solutions are formed by randomly *mutating* current solutions. Successive generations are produced with new solutions replacing some of the older ones based on the relative fitness values. A heuristic terminates the algorithm, and the best solution is reported. Cohoon *et al*. applied the genetic algorithm to floorplan design [37]; Shahookar and Mazumder applied it to standard cell placement [205].

Simulated evolution is similar to genetic placement algorithms. However, there are important differences. While the genetic placement method maintains many total solutions, the simulated evolution method generates only one child from each parent during each generation. In addition, the methods for selection are quite different. In genetic placement, a random set is chosen for mutation or crossover, and its fitness calculated. In simulated evolution, each individual cell's *goodness* or fitness determines whether it survives in its current location. This method has been applied to the standard cell placement problem [114][115][116].

Evolution occurs at the individual level; each individual struggles to propagate its gene pool. There is no regard for what is best for a society. By analogy, simulated evolution mimics biological evolution more realistically than genetic placement algorithms. Simulated evolution is claimed to have better convergence properties than genetic placement algorithms.

Analytical algorithms mathematically calculate the positions of the cells from the network description. Results are optimal with respect to the cost function, an important advantage of mathematical algorithms. However, no one has been able to formulate a mathematical model which accurately predicts chip area. Most analytical models use the squared wire length model (rather than the more realistic Manhattan model) since it has continuous first and second derivatives [79],

$$L = \sum_{i,j=1}^{n} c_{ij} \left((x_i - x_j)^2 + (y_i - y_j)^2 \right) \tag{3.5}$$

where c_{ij} is the number of connections between cells *i* and *j*. This can also be rewritten as

$$= x^T B x + y^T B y \tag{3.6}$$

where

$$B = D - C, \ C = [c_{ij}] \tag{3.7}$$

and D is a diagonal matrix with

$$d_{ii} = \sum_{j=1}^{n} c_{ij}. \tag{3.8}$$

This cost formulation is also associated with the quadratic assignment problem. Hanan and Kurtzberg used the quadratic assignment method to solve the placement problem [83]. Although quadratic assignment uses Equation 3.5, it does not exploit the convexity of the function. Mathematical optimization techniques work best when both the domain and the cost function are convex. In this case, only a single global minimum of the cost function exists, and it is possible to find the global minimum by using gradient techniques. The justification for using the quadratic metric for placement was given by Fukunaga *et al*. [69]. Using a state space approach, the placement is determined by the eigenvectors corresponding to the two smallest eigenvalues.

However, it is difficult to map the solution eigenvectors back into the physical domain and to account for finite size components which may not overlap. Cheng and Kuh modeled the cells and the interconnect as a resistive electrical circuit and used relaxation techniques to solve the circuit [30]. They needed to add additional slot constraints to avoid the trivial solution of all cells on top of each other. The PROUD and RITUAL placement systems solved Equation 3.5 using Successive Over-Relaxation and residual iterative update of Lagrange multipliers, respectively [227][212]. The method has also been augmented to handle timing constraints using nonlinear programming techniques [100]. Dynamic programming has also been proposed as a method to map the components [90]. Blanks used a two-step procedure to map the ideal global placement onto the layout surface without violating any constraints [15]. Sha *et al*. encoded all of the constraints into the objective function eliminating the need for mapping [203][204]. Frankle and Karp used a multidimensional method which combined eigenvector probes and linear assignment to direct the mapping [67]. Gordian used partitioning stages to untangle the ideal placement [113]. A later Gordian version used a modified form of the quadric metric to represent a linear metric [210]. They found that the *pseudo-linear* metric yielded area improvements of up to 20% over the quadratic method.

Other mathematical methods that use linear programming (LP) have also been proposed. Mogaki *et al*. presented a LP algorithm for macro cells which incorporated constraints on block size, relative position, and width of the inter-block routing space [155]. Linear programming techniques are CPU intensive and limited by the size supported by the LP solver.

Closely related to the analytic methods is branch-and-bound placement. This method is guaranteed to find the optimal solution. The branch-and-bound method *prunes* the decision tree containing all placements. If an accurate bound is known for the placement cost, placements exceeding the bound may be pruned from the search space resulting in large reductions in computation time. However, accurate bounds require length computations. This method is extremely time consuming because it requires determining accurate bounds or exploring large number of branches in the decision tree. It has only been applied successfully to small problems. Onadera applied a mixed mincut / branch-and-bound algorithm to macro cell placement [165].

Esoteric algorithms are derived from recent advances in other related fields. Placement algorithms based on neural networks and fuzzy logic have been proposed. Libeskind-Hadas and Liu solved a macro cell orientation and rotation problem using the Hopfield and Tank model of neurological networks [139]. In this model, a neuron computes a monotonically increasing sigmoidal output function from a weighted set of inputs. The input weights are used to model the number of interconnections between cells. The placement problem is solved by simulating the neural network.

Lin and Shragowitz applied fuzzy logic to a constructive placement algorithm [140]. Fuzzy logic assigns a probability for an object belonging to a set. This algorithm used the mathematics of fuzzy sets to avoid local minimums obtained with strict constructive algorithms.

Although many claims have been made to the contrary, none of the algorithms above have been shown to outperform simulated annealing in terms of final chip area. The TimberWolf system has continued to produce the best results on the MCNC benchmark set of placements. None of the other algorithms have been found to be as robust over the many design methodologies. Most of the placement algorithms ignore timing performance. No other algorithm has been proposed which minimizes area and guarantees timing performance. For these reasons, simulated annealing is the basis for the TimberWolf macro cell placement algorithm.

3.3 Macro Cell Placement and Routing Algorithm

Our placement algorithm is based on simulated annealing and proceeds in two distinct stages. During the first stage, the interconnect area around each of the macro cells is dynamically estimated. That is, the cell area is modified as a function of position [195]. In the second or placement refinement stage, the routing area estimates are replaced by the density information obtained from global routing. The placement refinement stage consists of several executions of the following three steps: (1) channel definition, (2) global routing, and (3) macro cell spacing. The information obtained in the second step is used to compute the density of all of the channels, which then allows accurate spacing of the macro cells. Detailed routing is then performed on all channels simultaneously using the information from the global routing step to guide the router into the correct regions.

TimberWolfMC, an implementation of the simulated annealing algorithm for macro cell placement was reported in 1988 [195]. Its main objective was to achieve the best results while possibly sacrificing computation time. Another objective was to develop a flexible and extensible placement and routing package which would be applicable to state-of-the-art industrial circuits. TimberWolfMC version 1.0 yielded area savings ranging from 8 to 49 percent versus numerous automatic and manual layout methods. However, large computation times were required in order to achieve reproducible high quality results. In addition, version 1.0 did not include a detailed routing algorithm.

While the performance of TimberWolfMC was encouraging, the need for a much faster placement algorithm was apparent. We therefore, completely reexamined the implementation of the simulated annealing algorithm. In particular, we improved the generation of new configurations function, the cost function, and the annealing schedule. The new approach focuses on: (1) selecting those potential new configurations which have a higher probability of acceptance, (2) minimizing the impact of the penalty functions, (3) normalizing problems of various grid sizes and cell counts, and (4) utilizing the results of a theoretically derived statistical annealing schedule developed by J. Lam [125][126]. The new placement algorithm, part of TimberWolfMC version 2.0, requires 10 to 20 times less CPU time than version 1.0, while achieving placements of the same quality. In fact, it is now possible to achieve the best possible results from TimberWolfMC for the *ami33* benchmark circuit in about 5 minutes on a SUN 4/260.

New features have been added to increase functionality and usability but decrease execution time. TimberWolfMC now features timing driven placement. Cells may be grouped hierarchically. Cells and/or cell groups may be restricted to subregions within the core region. Also, through the use of the X11R3-X11R5 graphics interface, the user may interrupt the automatic execution at any time to add region restrictions or to place macros at specific locations. TimberWolfMC also features a sophisticated I/O placement algorithm which places the pad cells to minimize wirelength subject to side, spacing and pad grouping constraints.

3.3.1 Cost function C

The TimberWolfMC cost function C consists of three terms. The first term is the total wire length, represented by W. The second term is the overlap penalty function represented by P_O. The third and final term is the timing path penalty function P_p. The complete expression for the cost function is given by:

$$C = W + \mu P_O + \lambda P_p \tag{3.9}$$

where the factors μ and λ determine the relative weighting of the three terms in the cost function. The major changes to C include an improved P_O function, the inclusion of the timing path penalty, and an improved method for determining the values of μ and λ.

3.3.1.1 Total Wire Length

Since the final wire length of each net cannot be determined until detailed routing is complete, the wire length of each net is *estimated* in the placement stage. The wire length of a net is estimated as the half perimeter of the minimum rectangle that encompasses the net [76]. $S_x(n)$, $S_y(n)$ are the length and height of the minimum rectangle, respectively. A minimum rectangle of a 3-pin net is shown in Figure 3.3. Such a minimum rectangle shall be referred to as the *min-rectangle* of a net. The total wire length can then be expressed as

$$W = \sum_{n}^{N_N} S_x(n) + S_y(n) \tag{3.10}$$

where n is the number of nets.

To reduce the CPU time necessary to update *W* for large nets, we added an incremental net-span updating scheme to TimberWolfMC.

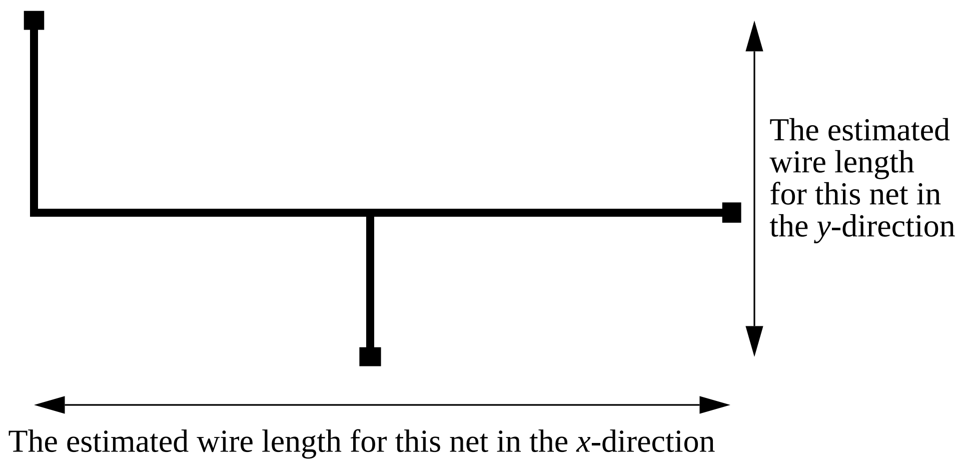

The estimated wire length for this net in the *y*-direction
The estimated wire length for this net in the *x*-direction

Figure 3.3 The minimum rectangle that encompasses this three-pin net.

3.3.1.2 The Overlap Penalty

In TimberWolfMC version 1.0, the CPU time required to update P_O was reduced by a bin scheme in which the core area was divided into a set of two-dimensional nonintersecting bins. When it had to be determined which cells overlapped with a given cell, the search could have been restricted to a local region. But for large circuits, the CPU time necessary to update P_O was larger than the time required to update the remaining terms of the cost function [196].

The new strategy again uses a bin scheme. However, in this case the bins are used to keep track of the total cell area intersecting each bin. The total bin area, where the area of bin *b* is represented by *A(b)*, is set equal to the core region area A_T. The core region area is determined by summing the areas of each dynamically expanded cell (which accounts for the estimated routing area [195]),

$$ {}_{T} = \sum_{c=1}^{N_c} A(c) = \sum_{b=1}^{N_b} A(b) \tag{3.11}$$

where N_c is the number of cells, $A(c)$ is the area of cell *c*, N_b is the number of bins, and $A(b)$ is the area of bin *b*.

The penalty applied to each bin *b* is the absolute difference between the bin area $A(b)$ and the total cell area intersecting the bin $A_c(b)$. The total overlap penalty is just the square root of the sum of the overlap penalties for all bins:

$$P_O = \sqrt{\sum_{b=1}^{N_b} \left|A_c(b) - A(b)\right|} \tag{3.12}$$

Note that the units of P_O are linear with respect to the specified grid size of the cell data. Our experiments have shown that in order to effectively optimize the choice of μ and λ in Equation 3.9, it is important that all terms in the cost function scale uniformly with respect to grid size.

If the bin size chosen is too large, cells may have significant overlap within a bin with zero or near zero penalty. This problem can be eliminated by choosing a very small bin size. However, this increases the CPU time since the penalty for many bins has to be updated whenever a cell moves. Our experiments indicated that the best compromise was to select the bin width and height to be one-half of the average of the shortest dimension of all the cells (including the estimated routing area). In this manner, cells will typically cover at least four bins. More formally, the average of the shorter sides $\bar{S}_S$ is expressed as follows, where $w(c)$ is the width of the bounding box of the cell *c* and $h(c)$ is the height:

$$\bar{S}_S = \overline{\underset{\forall c}{\min} \{ w(c), h(c) \}} \tag{3.13}$$

Therefore, the area of the uniformly sized bins is given by:

$$A(b) = \frac{\bar{S}_S^{\,2}}{4} \tag{3.14}$$

This choice of bin area statistically minimizes the inaccuracies which occur if two cells are placed on top of one another in the same bin without incurring large execution time penalties.

An example of an overlap calculation is shown in Figure 3.4.

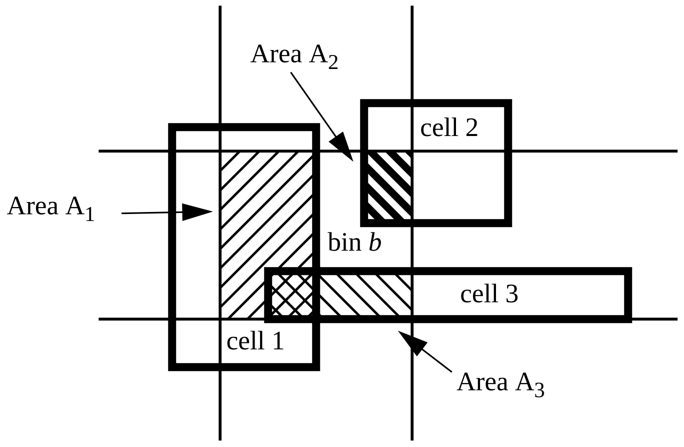


Figure 3.4 The bins for calculating the overlap penalty. The overlap penalty for bin *b* is $|A_1 + A_2 + A_3 - A(b)|$ **.**

Note that this function can be updated by adding or subtracting the cell area from only the bins that the cell spans. The CPU time necessary to update P_O is now a negligible portion (< 2%) of the total time necessary to update C for each new configuration.

The value of μ (in Equation 3.9) strongly impacts the quality of the final configuration obtained by the simulated annealing algorithm. We found that the optimum value of μ varied from circuit to circuit. Attempts to find a relationship for μ as a function of circuit size, grid size, and/or the average changes in W versus the average changes in P_O were not successful.

It is apparent that at the end of the execution of simulated annealing we desire the minimum value of P_O (or $P_{O_{min}}$) to be zero. However, for real circuits with widely varying cell areas, even a value of P_O equal to zero does not guarantee the absence of cell overlapping. For example, a bin might contain two cells whose aggregate area equals the bin area, but the positions of the cells might be such that they overlap significantly. Hence, any residual overlap penalty at the end of the simulated annealing run is removed by spacing the macro cells using a compaction algorithm. This final clean-up step results in some increase (or *jump*) in the value of W as shown in Figure 3.5.

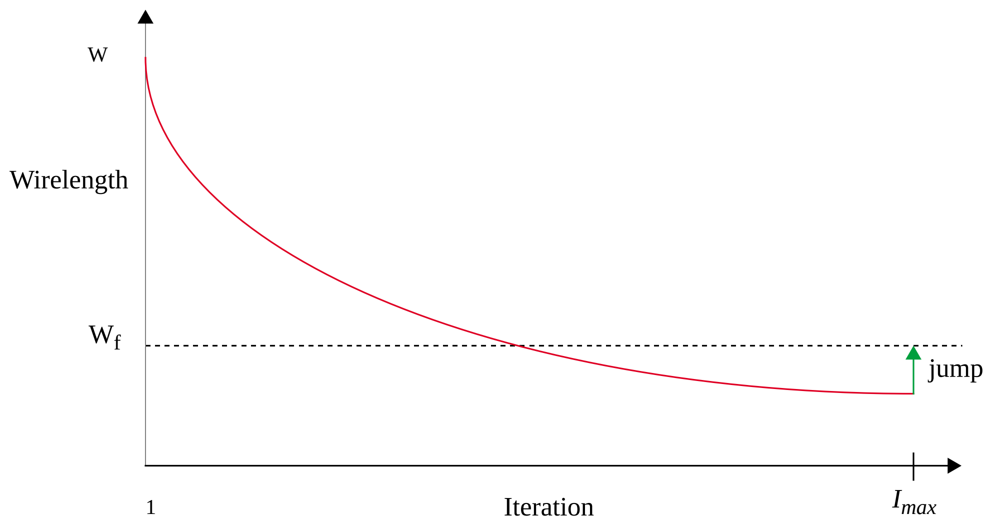


Figure 3.5 The jump in total wire length after spacing (or compaction).

We have observed that there is an optimum amount of this jump in W after spacing the cells. If the overlap is penalized strongly (using a large value of μ), then the amount of the jump is very small. However, we observed that the final values of W (or W_f) are usually quite far from the lowest obtainable. On the other hand, if the overlap is very lightly penalized (using a small value of μ), then the value of W approaching the last iteration is very small comparatively. However, after the clean-up step, the amount of the jump is quite large leading to a poor value of W_f. In an effort to find the optimum value of μ, we measured many industrial circuits and obtained the normalized results shown in Figure 3.6. The value of W_f in this figure was obtained after the clean-up step, that is, after the jump. It is interesting to note that a nonzero (positive) value of $P_{O_{min}}$ which causes an appreciable jump in W_f yields the lowest possible value of W_f. We consistently observed the best performance when the final value of P_O (or $P_{O_{min}}$) was approximately $0.4\sqrt{A_T}$. Therefore, at the last iteration (I_{max}),we desire the target value of the overlap penalty to be

${}^{T}_{O}(I_{max}) = 0.4\sqrt{A_T}$.

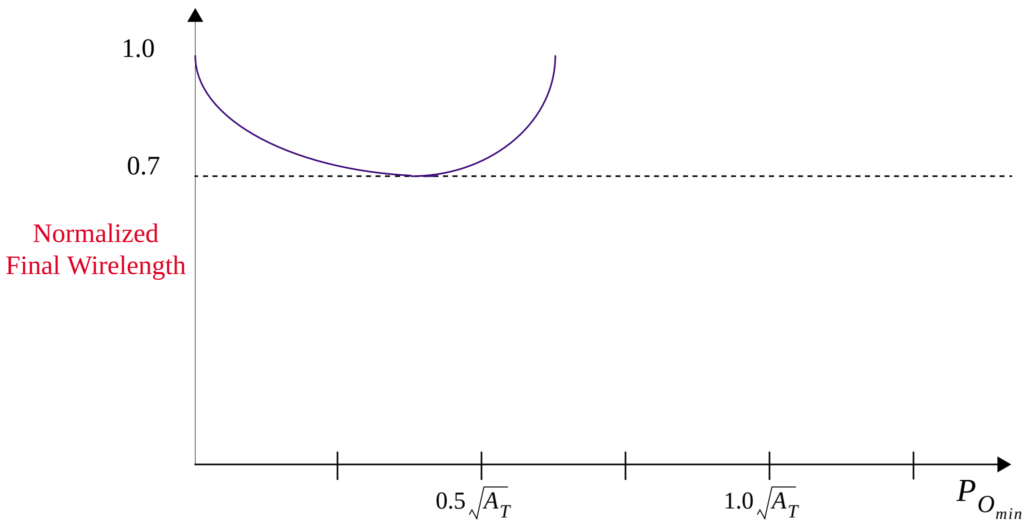


Figure 3.6 Normalized total wire length versus final overlap penalty.

The functional form for $P^T_O(I)$ was derived empirically. First, we determined the constant value of µ which yielded the best solution for a given circuit while monitoring the overlap penalty as a function of time. The form of $P^T_O(I)$ was then obtained by least-squares fit of the best data over 10 industrial circuits. The best fit over all the circuits was found to be:

$$P^T_O(I) = (A + BI + CI^2 + DI^3)\sqrt{A_T} \quad (3.15)$$

where

$$A = 0.85$$
$$B = -1.6\times10^{-4}$$
$$C = 4.3\times10^{-6}$$
$$D = -1.2\times10^{-7}$$

We then devised a negative feedback control function which seeks to adjust the value of μ dynamically throughout the simulated annealing run so as to achieve the desired functional form for all runs of all circuits. The negative feedback control function yielding the value of μ at iteration *I*, or μ_I, is given by

$$\mu_{I+1} = \mu_I + \frac{P_O(I) - P_O^T(I)}{K} \tag{3.16}$$

where *K* is a damping factor used to stabilize the control of the values of μ_I (in our implementation, a very suitable value of *K* is $^{T}_{O}(I)$). At the end of iteration *I*, if $P_O(I) > P_O^T(I)$, then μ is increased somewhat for the next iteration forcing the simulated annealing algorithm to put more emphasis on reducing the overlap penalty. Conversely, at the end of iteration *I*, if $P_O(I) < P_O^T(I)$, then μ is decreased somewhat for the next iteration, thereby allowing the simulated annealing algorithm to put more emphasis on decreasing the total wire length term *W*.

Figure 3.7 shows the results of a typical run of the MCNC benchmark circuit *ami33* versus the target penalty. The graph clearly shows the negative feedback control correcting the emphasis paid to the overlap penalty in order to achieve optimal results.

Equation 3.15 has been tested on more than 30 industrial circuits of widely varying sizes. We have found that $^{T}_{O} \approx 0.4\sqrt{A_T}$ at $I = I_{max}$ for every trial.

3.3.1.3 The Timing Penalty

The third term in the cost function is the timing path penalty function P_T which insures timing correct placement. The user supplies an upper and lower bound on the time delay for either a primary-input primary-output pin pair or specified critical path. Chapter 6 describes the timing penalty in detail. The total penalty is just the sum over all critical paths:

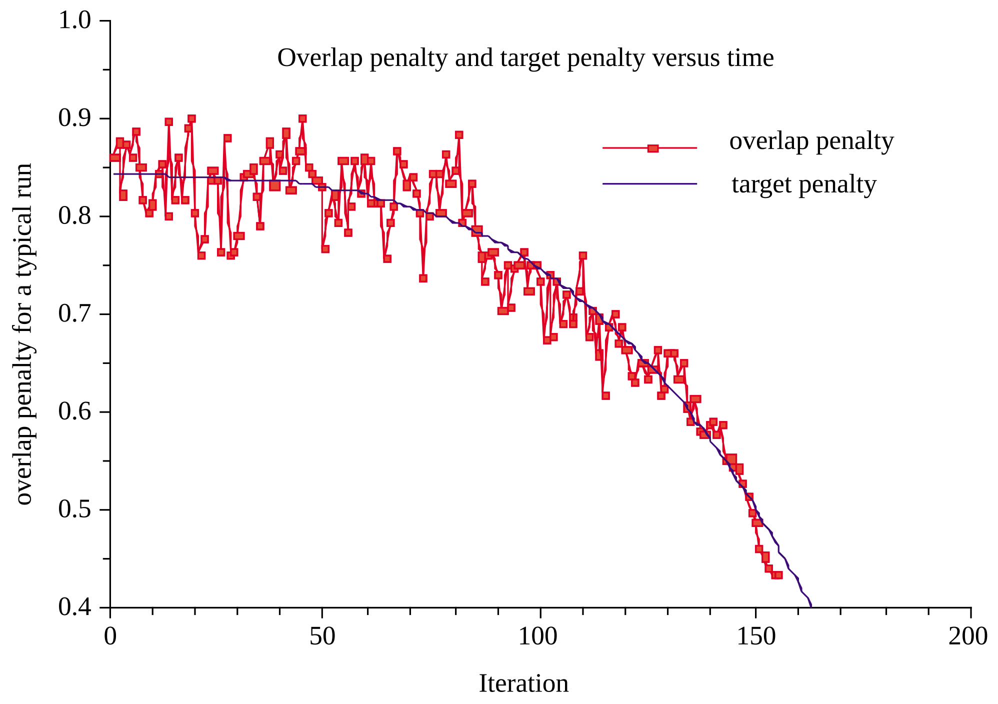


Figure 3.7 Overlap penalty as function of time for the *ami33* benchmark circuit.

$$P_T = \sum_{p=1}^{N_p} P_p \qquad (3.17)$$

Since TimberWolfMC insures that the sum of the time delays of the individual gates and nets in the critical path meet the given bounds, there is no need for the user to partition the path length between the individual signals of the path. Previously reported systems have used net weights on individual nets in an attempt to achieve timing driven placement. However, this is the first macro cell placement algorithm which features critical-path driven placement. This method is superior to net weighting techniques because it overcomes the partitioning problem and reflects more accurately the true timing constraints to be satisfied.

A very large value of λ will surely satisfy all path length bounds (if they can be satisfied) but will result in poor values of W_f. Conversely, too small a value of λ will yield very good values of W_f but with many of the path-length bounds remaining unsatisfied. We therefore, experimented to find a relationship for λ which did not degrade the chip area and W_f, but which nearly satisfies all the path constraints. Since the ratio of paths specified to the total number of nets will vary from zero to a large number from circuit to circuit and run to run, we had to ensure that the timing penalty term was *felt*, regardless of its absolute value. The best results were obtained when we assigned

$$\lambda = 3 \cdot \frac{\overline{\Delta W}}{\overline{\Delta P_p}} \tag{3.18}$$

where $\overline{\Delta W}$ is the average change in wirelength and $\overline{\Delta P_p}$ is the average change in the timing penalty measured during the first iteration.

This implies that the changes in the timing penalty are (in some sense) three times as important as the changes in the wirelength.

Below are the results for a large industrial circuit with 1535 paths. The value of W_f and the chip area were the same as that obtained without the critical path specifications.

Table 3.1 Timing results for a large industrial circuit with 1535 paths.

Criteria	Number of paths	Percentage of total paths
within specification	1359	88
0 - 10% above spec.	112	96
0 - 20% above spec.	64	100
above 20%	0	-

As one can see, the algorithm successfully satisfies the timing constraints without degradation of chip area.

3.3.2 The Improved New State Selection Procedure

Our improved scheme of selecting new configurations makes use of the bin mechanism described in Section 3.3.1.2. The center of each cell is hashed to a particular bin. The new procedure is as follows: Cell *a* is randomly selected from the set of all cells. We then randomly select a new location within the *range limiter window.* (The range limiter window size is designed to increase the acceptance rate at a given temperature. The changes in wire lengths are on the order of the move distance. Therefore, reducing the move distance yields smaller wirelength changes resulting and thus an elevated acceptance rate.) The bin which includes this position is noted. If the bin is empty, then the proposed new configuration is obtained by performing a single-cell move to the new location. On the other hand, if the bin is not empty, a cell *b* is randomly selected from among those cells belonging to this bin. A proposed new configuration is obtained by interchanging cells *a* and *b*.

This new method never places a cell directly on top of another cell generating far fewer new configurations which could otherwise increase the overlap penalty function P_O. In turn, this savings yields a lower value of μ (in Equation 3.9) and allows the simulated annealing algorithm to put more emphasis on decreasing the total wire length term *W*. Therefore, improved placement results are typical.

3.3.3 Stage 2 Placement Refinement

Until now, we have focused on the first stage of the placement algorithm. Now, we turn our attention to the second or placement refinement stage where the routing area estimates are replaced by the density information obtained from global routing. In Figure 3.8, we see a small example at the end of stage one. At this stage, the wirelength has been minimized but there is still some residual overlap. In the example, cell C3 overlaps C1. Note that cell C5 is an eight-sided rectilinearly shaped cell.

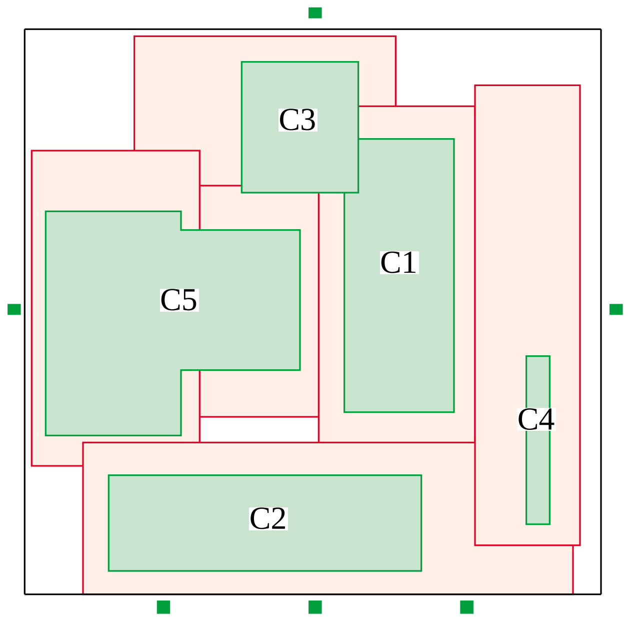


Figure 3.8 Results at the conclusion of stage 1 of the placement algorithm. The darkest regions are cells and the lighter shaded regions are estimated wiring areas.

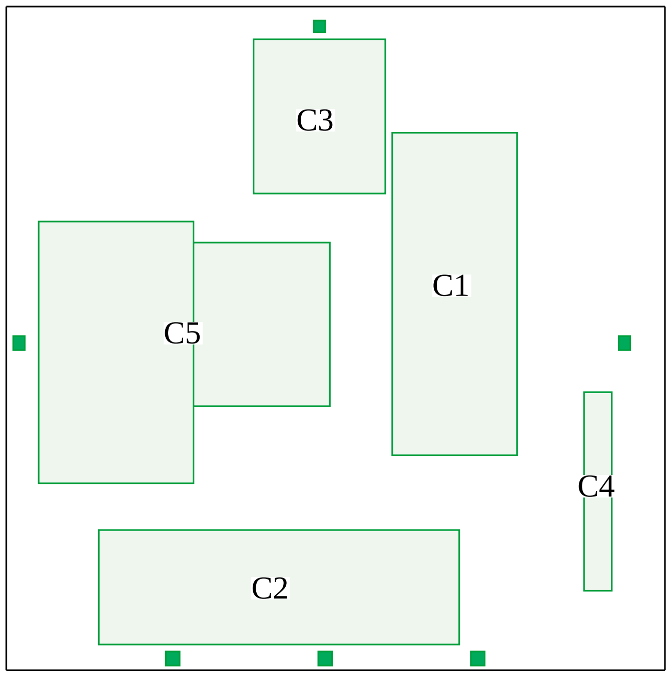


Figure 3.9 Placement after removal of overlap.

In this second stage, we remove any residual overlap between the macro cells while seeking to minimize the disturbance to the rest of the placement. The purpose of this cleanup step is to make it possible to form a channel graph, but it does not insure that the placement is capable of being detail routed. Our small example would now appear as in Figure 3.9. Cells C3 and C1 are spaced at the minimum design rule spacing. Any connections occurring in the small channel between these two cells would be infeasible.

We are now ready to begin placement refinement. The placement refinement stage consists of several executions of the following three steps: (1) channel definition, (2) global routing, and (3) macro cell spacing. The information obtained in step 2 is used to compute the density of all of the channels, which then allows accurate spacing of the macro cells.

3.3.3.1 Channel Graph Generation

Given a placement of macro cells, a *critical region* (Chapter 7 of [196] has a thorough discussion on the subject), either horizontal or vertical, is defined as the common region between pairs of cell edges or between a cell edge and a boundary of the chip. Figure 3.10 shows some examples of critical regions. Nets are routed through these regions. It should be clear that critical regions determine the spacing needed between two cell edges or between a cell edge and a chip boundary.

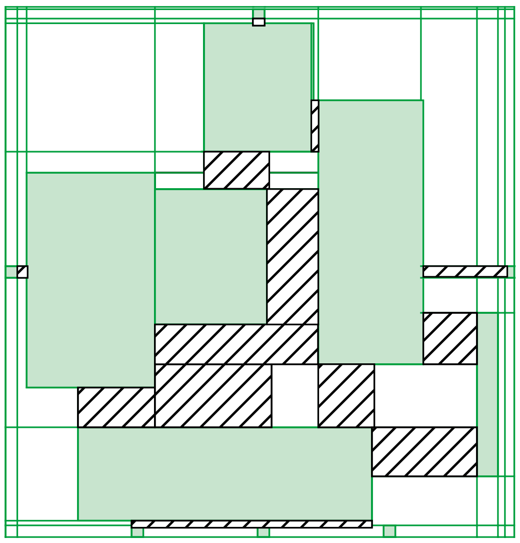


Figure 3.10 The critical regions for the example circuit are shown as hatched rectangles for clarity.

A *rectilinear channel graph* is formed by passing edges (actually channels) through critical regions and forming nodes at their intersections, as shown in Figure 3.11. Routes of all the nets will be generated along the edges of the channel graph. Therefore, pins of a net are mapped perpendicularly from the cells to edges of the channel graph.

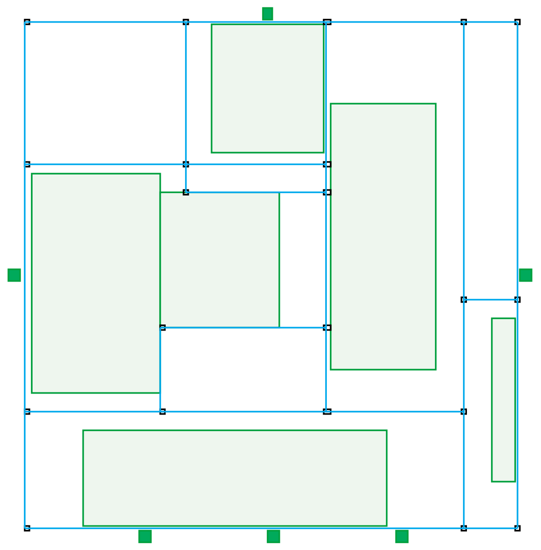


Figure 3.11 The channel graph for the previous example.

3.3.3.2 Global Routing

Next we perform global routing on the channel graph. The new graph-based macro cell global router *Mickey* [29] is net routing-order independent and allows two cost functions: the minimization of the chip area and the minimization of the total wire length under channel (or edge) capacity constraints. This global router minimizes the total wire length implicitly, and calculates channel densities precisely. In addition, the execution time is small relative to the stage 1 placement time, allowing many iterations of refinement without penalty. Figure 3.12 shows the routing of a single multi-terminal net on the channel graph.

3.3.3.3 Compaction

From the global routing, we calculate the routing density d_i in each channel. In order to space the cells correctly, we add routing area to each cell edge on a channel by channel

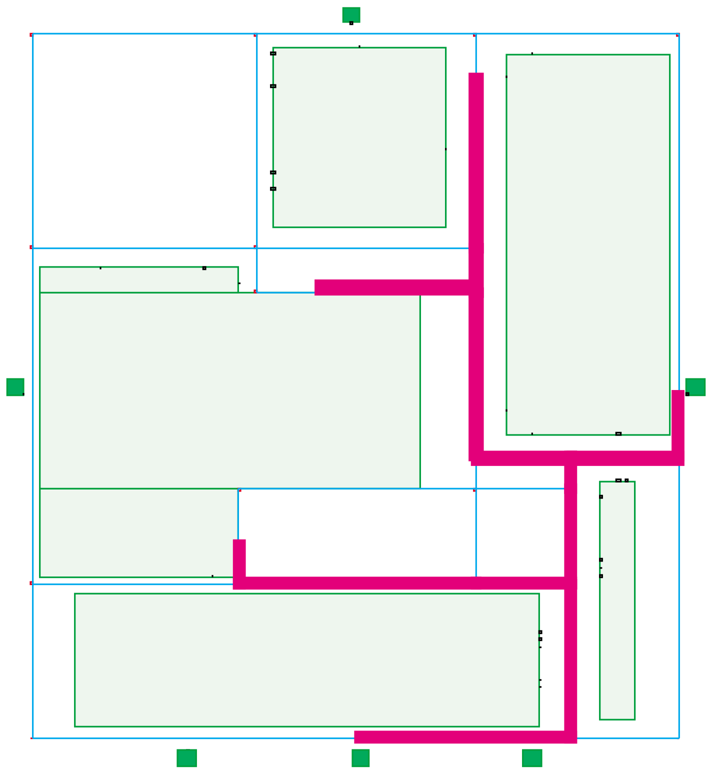


Figure 3.12 An example of global routing on a channel graph. The thick line denotes the global routing tree for a single net.

basis. For each of the two cell edges adjacent to channel i, routing space of width w_i is added where $w_i = \frac{(d_i + 1)}{2} \cdot t_s$ and t_s is the minimum spacing between routing tracks. That is, one half of the routing area of a channel is assigned to each of the two adjacent cell edges. Each cell along with its routing area becomes a fairly complex rectilinearly shaped cell. An example is shown in Figure 3.13.

The next phase is compaction of these cells to remove unnecessary space. By compacting these cells, the placement takes into account the necessary routing area. An example after compaction is shown in Figure 3.14.

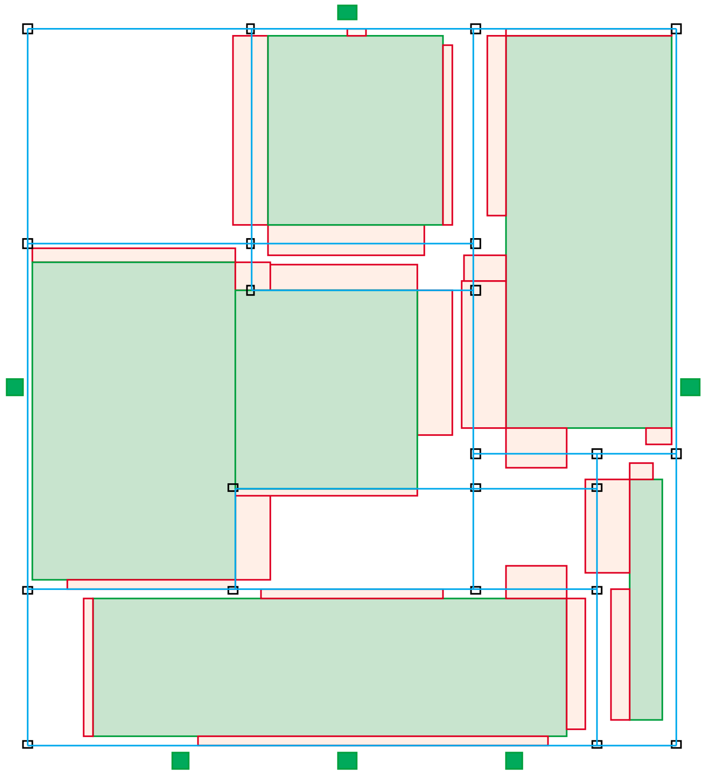


Figure 3.13 The addition of routing tiles to the cells after global routing.

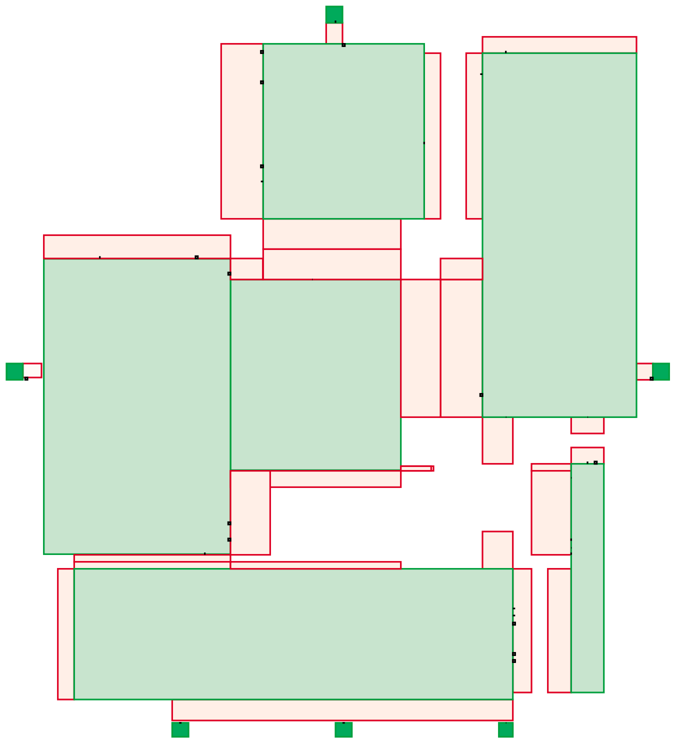


Figure 3.14 The example in Figure 3.13 after compaction. New critical regions are shown.

3.3.3.4 Topology Preservation During Compaction

Since the placement may be modified by the compaction process, the channel graph, and hence the global routing, may become invalid. Therefore, we repeat the placement refinement process until convergence is achieved. After convergence, a global routing step will be executed using the final channel capacity constraints. In this last step, the global router is constrained by the placement of the macro cells and therefore, achieves a feasible final routing.

However, this placement refinement method is not guaranteed to converge. It is possible that the channel graphs of two successive iterations of the refinement method will never be isomorphic. In general, two nonisomorphic channel graphs will yield different global routing areas, and therefore, different placements after compaction. Figure 3.15 and

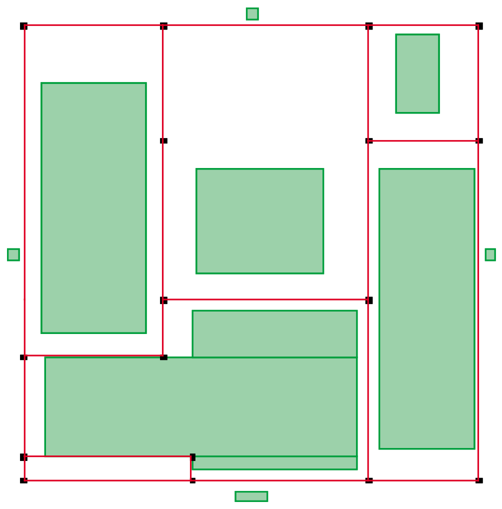


Figure 3.15 Original channel graph (before compaction).

Figure 3.16 show a small example before and after compaction. A comparison of the channel graphs in Figure 3.17 reveals that the topology of the placement has changed. Convergence has not been achieved.

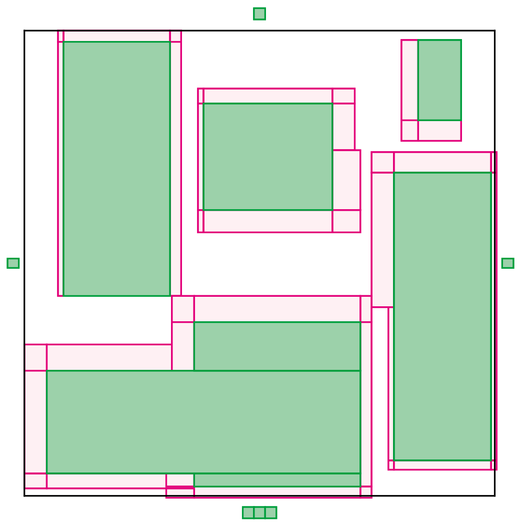


Figure 3.16 Compaction results without channel graph constraints.

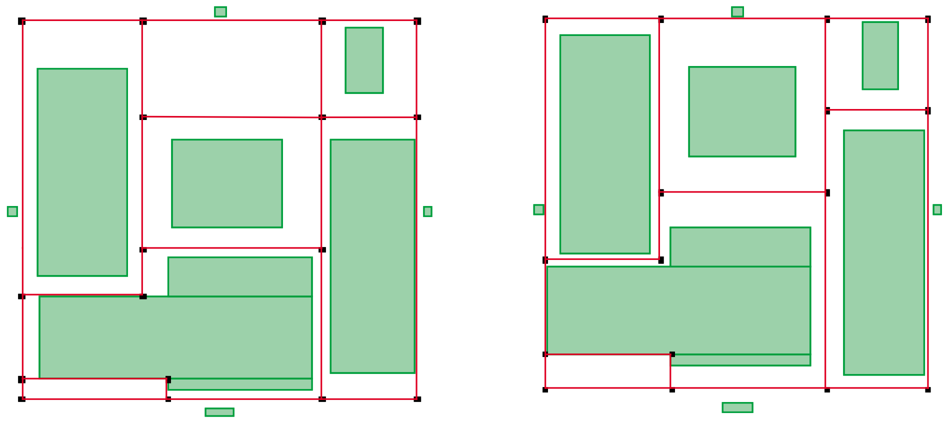


Figure 3.17 The topology changes. The left channel graph is the original topology. The channel graph on right is the topology after compaction.

The solution is to use the channel graph during compaction to preserve the placement topology. We use the following compaction strategy: For objects on the critical path, use constraints derived from design rules to insure minimum size with respect to 1-D compaction. For objects not on the critical path, we first determine the compaction constraints. From the longest path calculation, we determine the *valid* window for an object. The lower bound for the object window is a by-product of the longest path algorithm. The upper bound may be determined by a longest path search starting at the sink and searching in the opposite edge directions [154]. If all components are within their respective valid windows, all design rule constraints will be satisfied for objects off the critical path. Next, determine the *constrained* placement window using the channel graph. The constrained window is calculated by stretching or shrinking the channel graph in the direction of compaction to equal the length of the longest path. The stretching/shrinking operations leave the channel graph isomorphic to its initial state since edges or nodes are not deleted or added. If all components remain in their constrained windows, an isomorphic channel

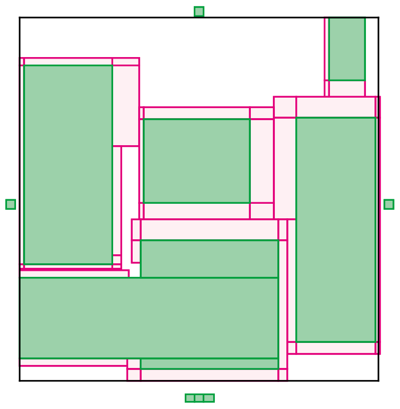


Figure 3.18 Compaction results using channel graph constraints.

graph will be generated in the next iteration. In addition, we calculate the object position which minimizes total wire length. Finally, place noncritical objects at the positions which minimize deviations from the desired positions yet remain in the valid window. Since the deviations are minimized at each iteration, convergence is achieved once the channel graph has been stretched to accommodate the longest path calculations. For this reason, the channel graph constraints are used after the second iteration. Figure 3.18 shows the same circuit using channel graph constraints during compaction. As seen in Figure 3.19, the channel graphs remain isomorphic during placement refinement, and convergence is achieved.

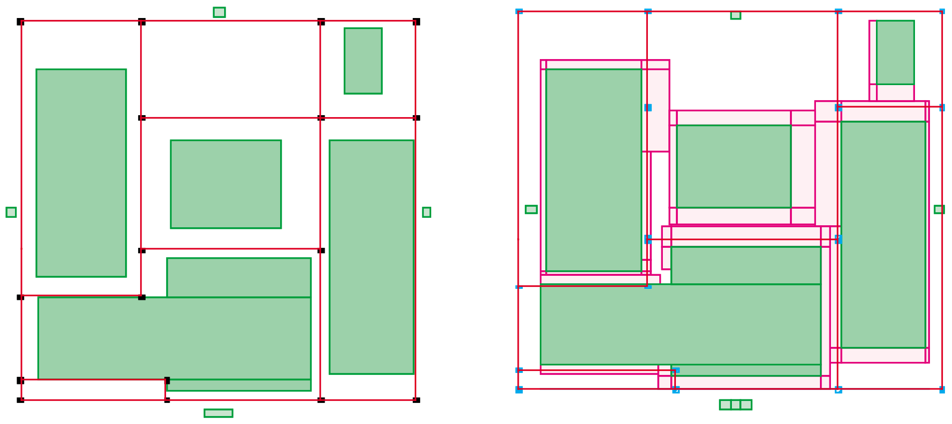


Figure 3.19 The placement topology is preserved. The left and right channel graphs are isomorphic.

3.3.3.5 Adaptive Dynamic Wire Length Estimation

TimberWolfMC dynamically estimates the interconnect area required for each macro cell. In TimberWolfMC version 1.0, the area estimate varies according to core coordinates and pin density,

$$a^i = \frac{1}{2}\alpha C_W f_x(x_i) f_y(y_i) f_p(i) \tag{3.19}$$

where a^i is the estimated interconnect area for cell edge i, α is a constant, C_W is the average channel width, $f_x(x_i)$ is the variation of area as a function of edge x position, $f_y(y_i)$ is the variation of area as a function of edge y position, and $f_p(i)$ is the variation in area as a function of relative pin density [195]. Other wire estimation algorithms have been proposed as well [81][168]. These estimation methods are based on theoretical models. However, they are inaccurate if the design style violates any of the assumptions of the theoretical model. For example, the model must be changed if another routing layer is added. Figure 3.20 and Figure 3.21 show an instance where the theoretical model grossly overestimates the routing area needed for cell C1. This overestimation leads to a suboptimal final placement. In this case, the large number of pins on one edge yields a large area estimate. However, many pins on this cell are connected to the same signal and may be routed using only one track. Although this model could be modified to handle such cases,

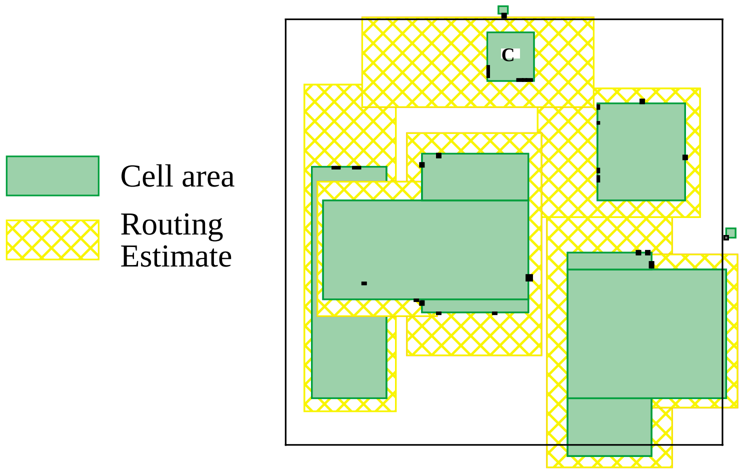


Figure 3.20 Inaccurate modeling. TimberWolfMC version 1 overestimates the routing area needed for cell C1.

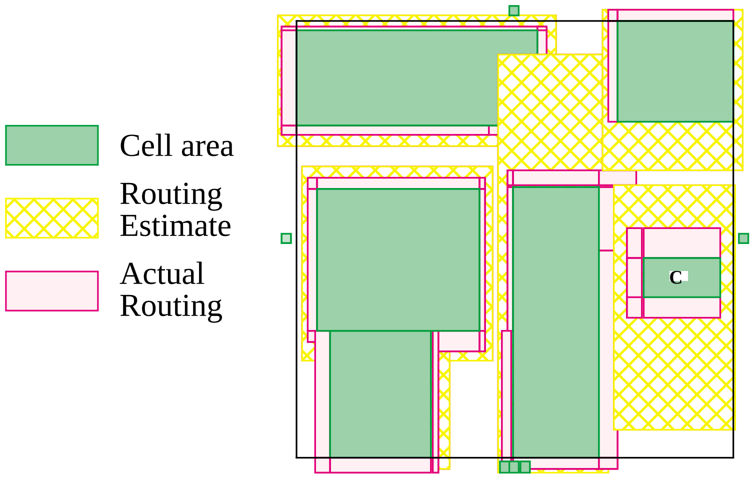


Figure 3.21 Placement after global routing using original wire estimator. Notice the gross inaccuracy in the estimation of the routing area for cell C1.

no theoretical model can predict all the possible routing scenarios accurately.

The solution is to use a general statistical model which adapts for every circuit. We define the estimated interconnect area for cell edge *i* to be

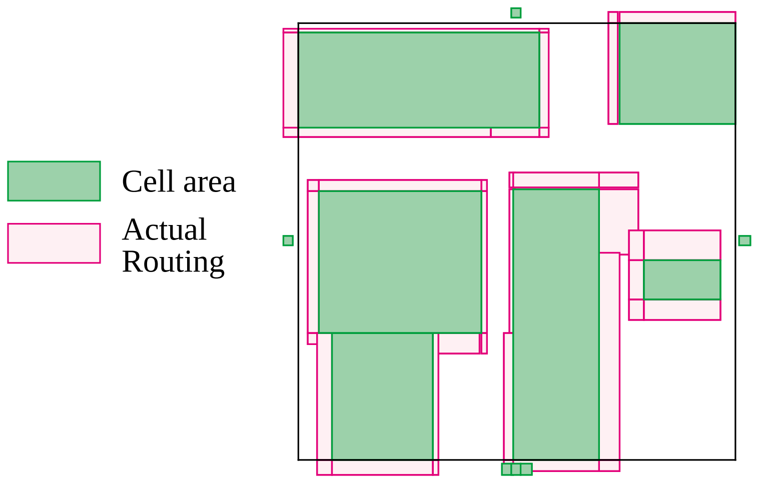


Figure 3.22 Inaccurate wiring estimation leads to poor area efficiency. White space around cells is unused area.

$$^{i} = c_0 + c_1 \cdot x + c_2 \cdot x^2 + c_3 \cdot y + c_4 \cdot y^2 + c_5 \cdot p \qquad (3.20)$$

where $c_0 ... c_5$ are constants, x and y are the normalized chip coordinates, i. e. $x, y \in [0.0, 1.0]$, and p is the number of pins in the channel. To obtain the model constants: place the circuit using a 10x annealing schedule and the theoretical estimation model. This will give a realistic upper bound on the routing area. Perform global routing and/or detail routing to calculate routing areas. Use a least squares method to fit the data to Equation 3.20 [173]. Subsequent placements are performed using the statistical model. After each TimberWolf run, the statistical model is refit allowing TimberWolf to *learn* from the previous executions. In addition, the statistical model adapts to any design style and routing technology. Figure 3.23 and Figure 3.24 show the same example using the statistical wire estimator. In this case, the statistical model accurately estimates the routing area required. The resulting final placement is the minimum area topology. In Timber-

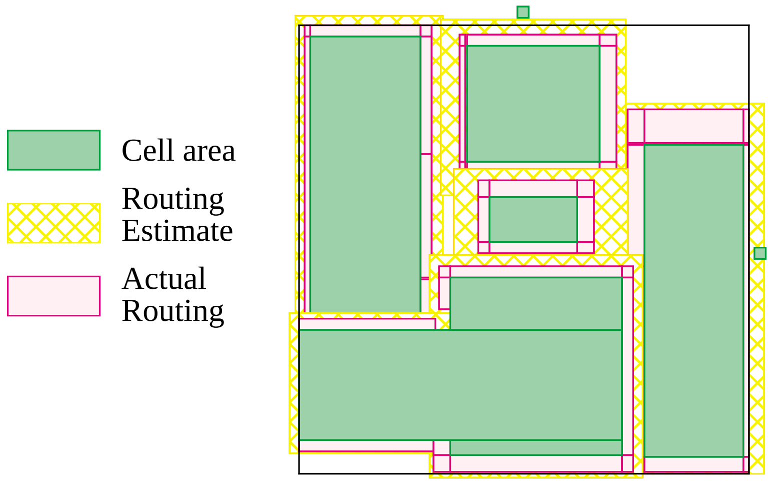


Figure 3.23 Placement after global routing using the statistical wire estimator.

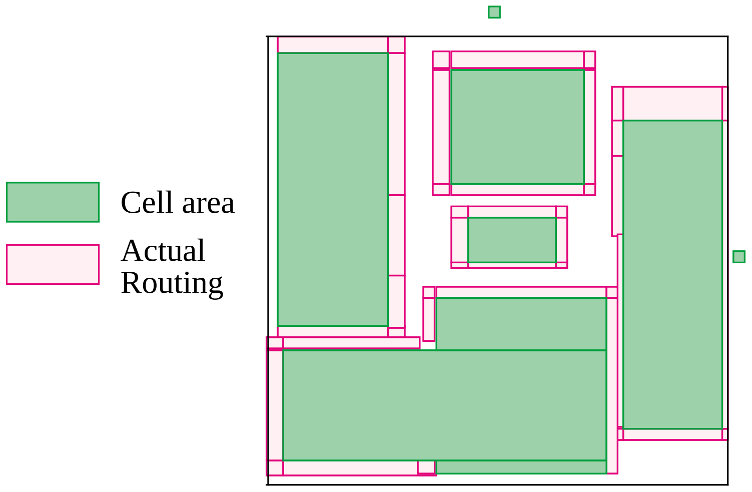


Figure 3.24 Accurate wiring estimation reduces the amount of unused area. This is the minimum area placement for this example. The remaining white space does not impact chip area.

WolfMC, the 10*x* run is transparent to the user. That is, the user will notice that the second and subsequent runs are slightly faster than the first.

3.3.3.6 Detailed Routing

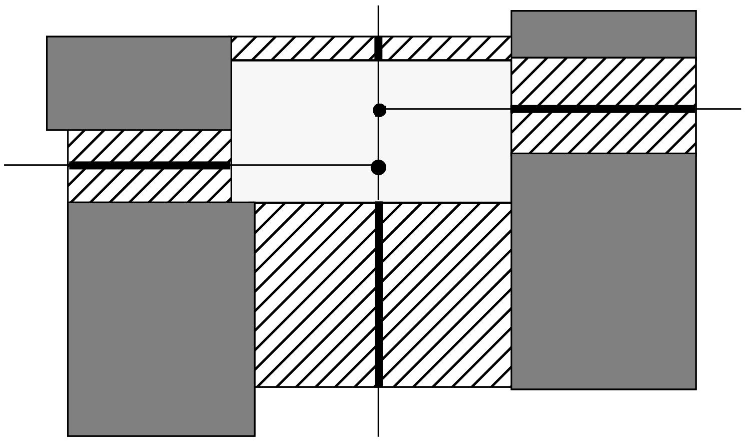
Next, we perform detailed routing using *TimberWolfDR*, our derivative of the *Mighty* maze router[207]. *TimberWolfDR* has been modified to allow routing to cells within a switchbox by using *macro objects* to model the rectilinear cells. The macro objects act as “keep out” regions but allow connections to pins on their boundaries. Each “keep out” region is specified on a per layer basis.

At this point we depart from the classical divide-and-conquer methods of routing macro cell layouts where routing is done channel by channel. Since placement refinement utilizes accurate density information, the cells can be spaced to accommodate the routing area avoiding the need for post-routing compaction. Since we route the macro cell design as obstacles (macro objects) within a switchbox, we eliminate the classically difficult problem of defining channels for detailed routing. In addition, we also avoid the equally difficult problem of determining a routing order for the defined channels.

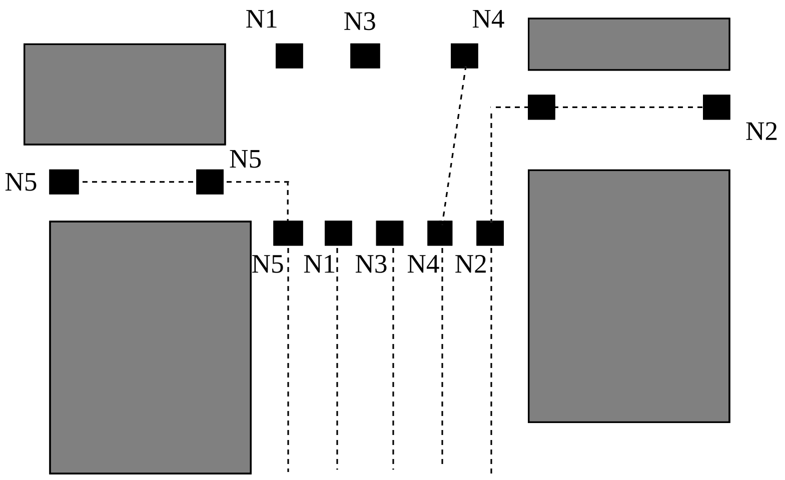
In order to coerce the routing segments into the regions determined by the global router, we introduce *pseudopins* at the channel junctions as determined by the global router. Like signal pins, pseudopins are ports which participate as sources for wave propagation in the TimberWolfDR maze router, but unlike signal pins, they are not required to exist in the final interconnection of pins. Since pseudopins are sources for the wave propagation, regions containing pseudopins are favored in the final route.

For each net, pseudopins are added at every channel junction in the net's global routing tree. Figure 3.25 shows an example of pseudo pin placement. The pseudopins are ordered across the ends of each of the channels using Groeneveld's algorithm [78] to minimize unnecessary twisting of the wires. Our extensions to this algorithm perform a topological sort on the global routing trees to find the ordering of the pseudopins which results in the most planar routing. The ordering shown is planar. If pseudopins *N5* and *N2* were to be reversed at the top of the lower vertical channel, significant area would be wasted due to wire crossings.

Since the detailed router uses these pseudopins as a starting point for its search, and the pseudopins are placed at the junctions of the global routing tree for each net, the detail



(a)



(b)

Figure 3.25 An example of pseudo pin placement. (a) Dark shaded areas are the macro cells. The hatched rectangles are the critical regions. The thick solid lines are channels and the dots denote a channel junction. (b) The dotted lines denote the path the global router has determined for the nets. The dark shaded areas are the macro cells.

router will follow the map laid out by the global router. In addition, the placement of the pseudopins insures the minimum amount of wire crossovers.

Since TimberWolfDR is a maze router, the memory requirements are proportional to the product of the horizontal and vertical grids. We therefore use a *local grid* approach to minimize storage requirements. In particular, grid lines appear only at pin positions, and in addition, the number of grid lines added per channel equals the channel density. The relatively sparse grid allows maze-type routers such as TimberWolfDR to be able to route an entire chip as a single switchbox. Figure 3.26 shows the results of applying the local grid approach to a placement of macro cells.

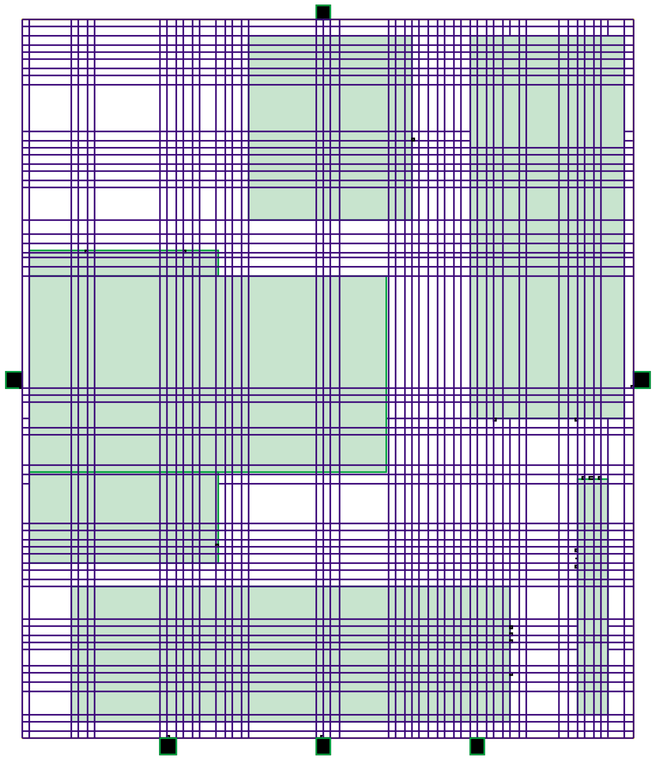


Figure 3.26 The local grid lines for a placement of macro cells

The final detailed routing for the small example appears in Figure 3.27. Since all cells have been spaced at density, subsequent compaction is not needed. In the classical methods based on channel definition, there is no easy way to avoid routing several very complex channels and switchboxes for this example. This routing methodology avoids this problem entirely.

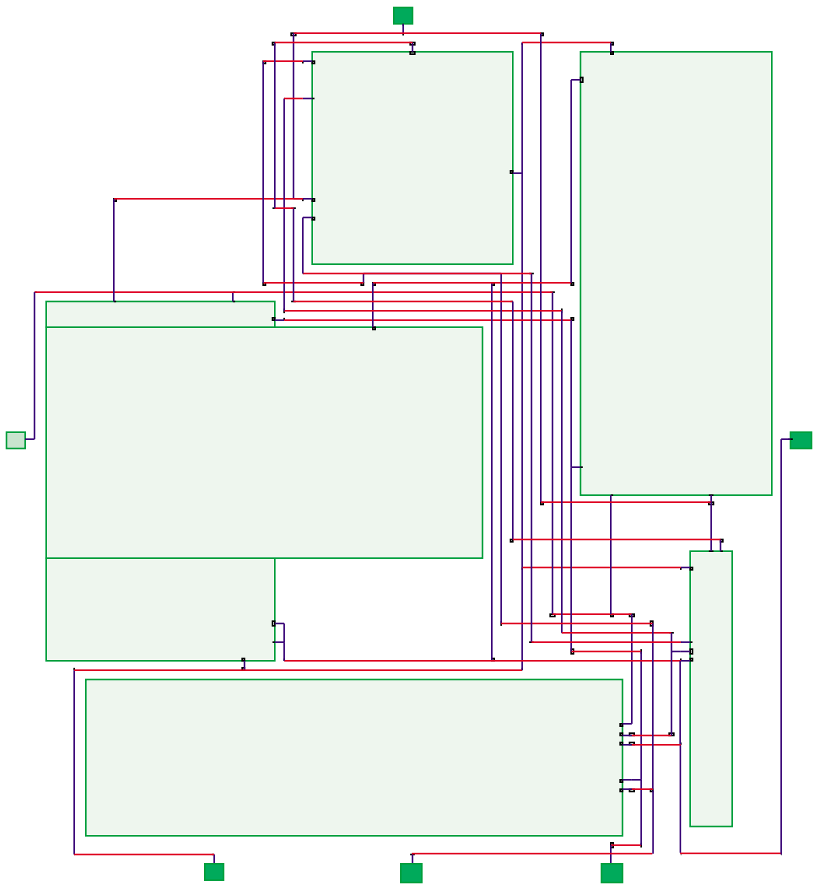


Figure 3.27 An example of the final detailed routing.

3.4 Results

Table 3.2 shows our result for the *ami33* macro cell circuit from the MCNC benchmark set. It is important to note that both the Delft P&R and the Mosaico systems used interactive manual placement to achieve their results, and Mosaico also used interactive global routing modifications. Nonetheless, the fully automatic TimberWolf result is better in terms of chip area and the total wire length (after detailed routing). Figure 3.28 shows the final result for the *ami33* macro cell circuit.

Table 3.2 Previously published results for the *ami33* benchmark circuit [20].

System	Chip Area	Total Wire Length
TimberWolf	2.57	105
Delft P&R	2.60	152
Mosaico	2.71	118
Bear	2.83	131
Industrial 1	2.94	125
Industrial 2	3.12	135

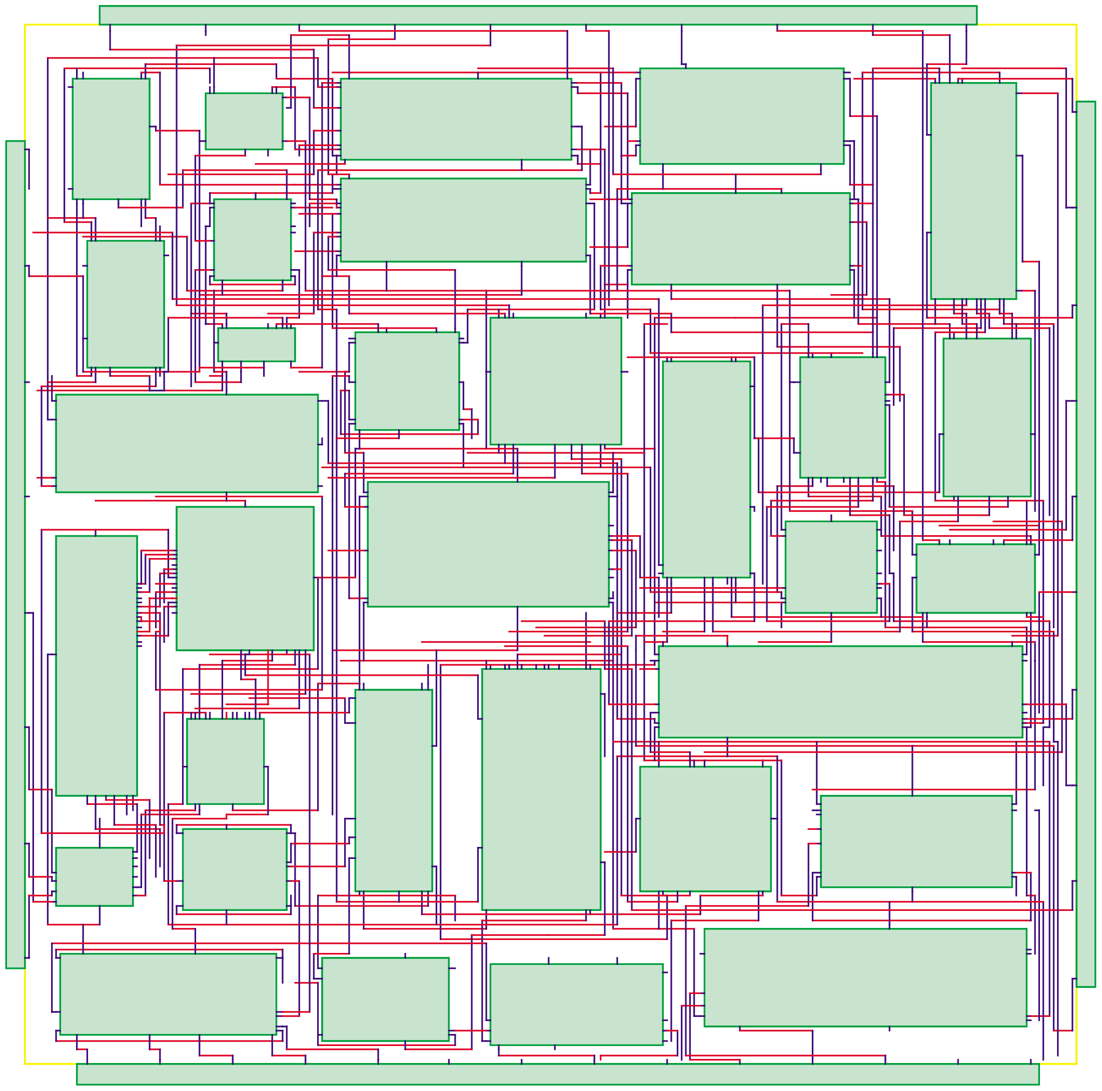


Figure 3.28 Final placement and routing of *ami33* benchmark circuit.

3.5 Conclusions

The keys to the consistent performance displayed by TimberWolfMC version 2.0 are new advances in the implementation of simulated annealing, a new method for placement refinement, and a new method for detailed routing. In particular, we have attempted to optimize the relative weighting between the primary objective term and the penalty function terms in the cost function. Furthermore, we have utilized the results of a theoretically derived statistical annealing schedule. We have placed emphasis on selecting new configurations which have a reasonable chance of acceptance. A new statistical wire estimation algorithm has been developed which reduces the amount of unused routing area. A new placement refinement method has been developed for rectilinear cells which spaces the cells at density avoiding the need for post-routing compaction. In addition, the placement refinement method uses previously generated constraints to maintain the topology. A new detailed routing method has been developed which avoids the classically difficult problem of defining channels for detailed routing, and in addition, avoids the equally difficult problem of defining a routing order for the defined channels. We have obtained the best result for the *ami33* benchmark circuit. Furthermore, our result is better than the best of the previously reported manual placements.