A Simple Yet Effective Technique for Global Wiring

RAVI NAIR, MEMBER, IEEE

Abstract—**A simple algorithm to perform global wiring is described. Repeated iterations of the algorithm tend to improve the quality of wiring by rerouting around congested areas. Various parameters can be set to give preference to short routes or to reduce the time taken by the algorithm. The algorithm has been tried out for several master-slice chips containing upto 3500 cells with good results. The technique is easily extended to standard cell chip design. An implementation for global wiring of a structured custom chip design style is described along with results. The technique is adaptable to higher level packaging such as chips on modules or modules on a board.**

I. Introduction

GLOBAL WIRING [1]–[3] is the name given to the phase of wiring in which wires are allocated to channels in a chip without specifically assigning tracks within the channels. The latter assignment is accomplished subsequently by an *exact embedding* algorithm. Just as the quality of placement influences the quality of wiring in a chip, the quality of global wiring influences the ability to wire successfully at the interconnections in the exact embedding phase. Clearly, the more information one provides a global wiring algorithm about various available routes through the chip, the better the result that an algorithm may be expected to produce. However, in order to restrict the execution time of the algorithm, one must be able to glean just the right amount of information from details of the chip topology so that the algorithm may provide a reasonable guideline for successful detailed wiring at the exact embedding phase.

A global wiring phase has been shown to be useful also for custom chips [4] and for various levels of packaging [5]. This report will discuss a technique which was developed in connection with the layout of a *master-slice* or *gate array* chip, and its extension to a *structured custom* design.

II. Wiring Model for a Master-Slice Chip

Shown in Fig. 1 is an image of a typical IBM master-slice chip. The rectangular boxes in the center of the chip represent *logic cells* or *circuits*. The rectangles on the periphery of the chip may include I/O cells. Typically, two planes are available for metallization. The first level of wiring is partially occupied by the wiring within the cells themselves. The area left for interconnections on this level form long rectangular horizontal channels. The second level is usually completely free for laying down interconnection metal. It is fairly typical, though not necessary, to divide the two layers so as to have predominantly horizontal wires in the first level and predominantly vertical wires in the second. This permits one to perform some variation of a stick-packing algorithm for assigning vertical tracks followed by a channel-routing algorithm for the horizontal layer. Various effective channel-routing algorithms have been described in the literature.

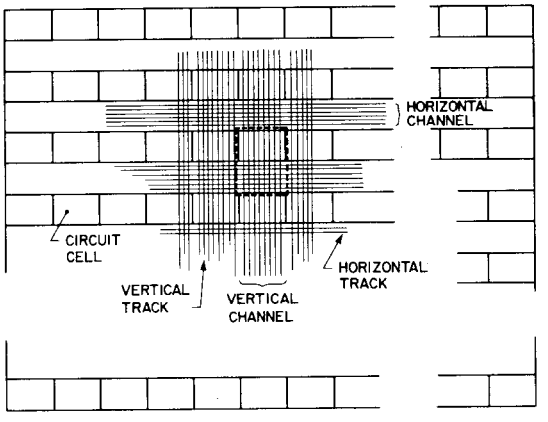


Fig. 1. A typical master-slice chip.

For the global wiring phase, it is necessary to first partition the chip into a rectangular grid of cells. The dotted line shows the outline of a cell for global wiring. Each cell of the global wiring model need not correspond to a cell in the master-slice. In fact, the partitioning of a typical IBM master-slice chip currently contains two cells, one each from either side of a horizontal channel, as shown in Fig. 2.

For each cell (henceforth a cell will refer to a cell in the global wiring of the chip), an estimate is made of the number of wires that may be allowed to pass through each of its four boundaries. This information is provided in a *channel capacity* file. The list of interconnections to be made between the cells, also called a *net list*, is provided as a set of *nets*, where each net is represented by sets of points which are to be made electrically common.

The channel capacity estimate is somewhat conservative in that a track is said to be available at a certain cell boundary only if there is no blockage along that track from the center of the cell on one side of the boundary to the center of the cell on the other side. A special case occurs when a *pin* blocks some track. A separate list (called a *fence list*) is provided which lists all such tracks along with the net to which that pin belongs. This allows the

Manuscript received January 1, 1985; revised October 3, 1986.
The author is with the IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.
IEEE Log Number 8612426.

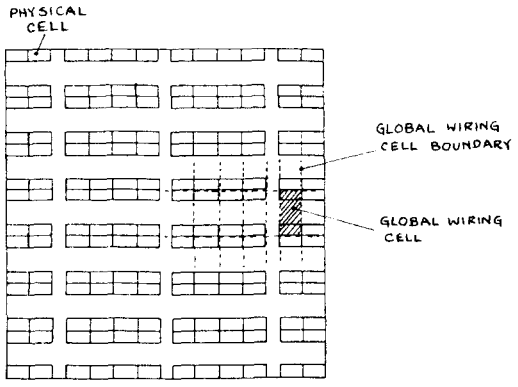


Fig. 2. Partitioning a chip into global wiring cells. (a) Grid with original boundary costs. (b) Scores after Step 5. (c) Scores after Step 6. (d) Scores at the end of forward propagation. (e) The backtrace route.

global wiring routine to provide an unrestricted passage through that boundary for that net.

III. Outline of Algorithm

The algorithm is based on an earlier approach [6] which in turn is a variation of the *Lee–Moore algorithm* [7], [8]. The major problem with the Lee–Moore algorithm is that, in the absence of knowledge of the areas of potential congestion, the results are dependent on the order in which nets are wired and are often unsatisfactory. To get around the order dependence, the paper [6] proposed a way in which the demand due to nets at a given cell boundary could be estimated. The demand was one of the contributions to the cost associated with the traversal of a cell boundary in the forward pass of the algorithm. Good results were obtained using this variant of the Lee–Moore algorithm.

Clearly, the more accurate the estimate of the demand at the cell boundary, the better one would expect the results of the global wiring to be. This suggests that the best results can be obtained if one took, not an *estimate* of the demand, but the *demand* itself. Thus, one could take the actual wiring produced by some means and reroute the nets, now knowing the *actual* remaining supply at the various cell boundaries. The new solution can be guaranteed to be no worse than the previous solution if one net is ripped out and rerouted at a time, the new route being taken only if the resulting solution is measurably better than the old.

While no claim can be made on the dependence of the result to the initial solution, experiments so far have indicated that the results of this algorithm after less than five iterations are approximately the same irrespective of the starting point. In fact, the same algorithm is currently used to provide a starting solution (a rather poor one, of course) with satisfactory eventual results.

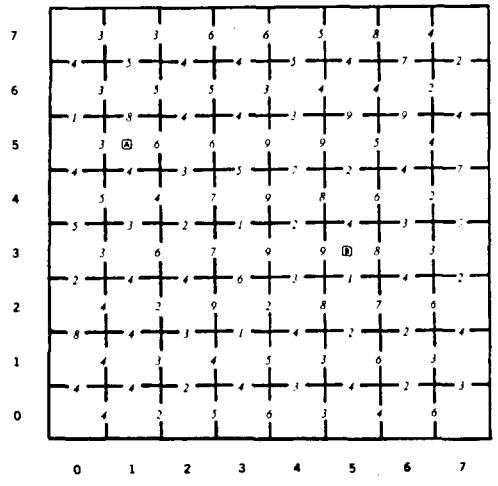
Three aspects of this algorithm distinguish it from other rerouting techniques [5], [9]. First, *every* net is ripped up and rerouted, irrespective of whether it passed through an overflowed cell boundary or not. This was motivated by the observation that quite often nets passing through noncongested areas could be diverted to pass through even less congested areas to make room for nets in adjacent congested areas. In addition, this tends to maximize the weighted average of tracks remaining at cell boundaries, facilitating the task of the exact embedding routine. Second, only one net is ripped out at a time. This ensures that cyclic problems do not arise while ensuring a monotonically improving behavior for the algorithm. Third, nets are rerouted in the same order in every iteration. The rationale behind this can be understood by observing what happens in the first two iterations. In the first pass, each net has absolutely no indication of demands due to the following nets, but has a knowledge of the approximate routes for the previous nets. In particular, the first net was probably wired the worst while the last net was wired the best under the circumstances. By ripping out the first net in the beginning of the second pass, an attempt is made to rectify this situation. (One could think of this approach to be an analog to the bubble sort algorithm.)

IV. Details of the Inner Loop of the Algorithm

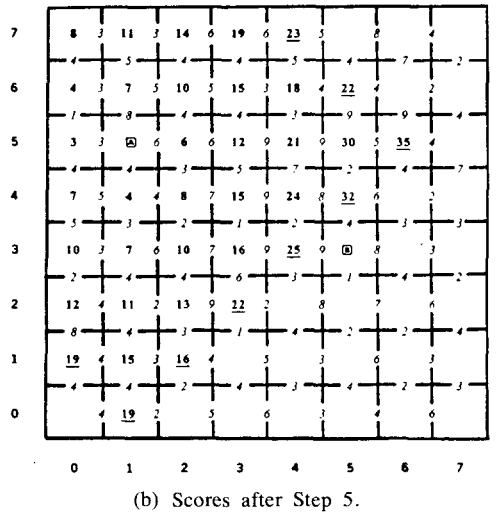
This section will describe the steps in a single iteration of the algorithm. A variation of the traditional Lee–Moore algorithm, which allows the assignment of arbitrary costs to the traversal of a cell in a grid, was described by Akers [10]. In general, the cost of traversing the cell is dependent on the specific *boundaries* of the cell that the path traverses. It is more appropriate to associate costs with each of the boundaries of a cell. In addition, one may wish to associate a cost with the cell itself to reflect the complexity of that cell. The algorithm is illustrated in Fig. 3. A grid with the current costs assigned to the cell boundaries is shown in Fig. 3(a).

Each cell is given a score which represents the sum of the costs of all the cell boundaries traversed by the *cheapest* path from the source to that cell. At any given step, there is a list of cells that will be processed at that step. For each cell, the directional score is computed for each of its four directions as the score of the neighbor in that direction, plus the cost of crossing that cell boundary. If the minimum of these four directional scores is less than the score currently assigned to the cell, the score is updated to the minimum value and the four neighboring cells are put into the list of cells to be processed at the next step. The list of cells initially contains the neighbors of the *source* cell, which is a cell chosen from the set of cells to be wired together. The score for the source cell is set to zero. Since only nonnegative numbers are assigned as costs, the process terminates either when the list is empty or when all the updated scores at the end of some step are equal to or higher than the score assigned to one of the destination or *sink* cells.

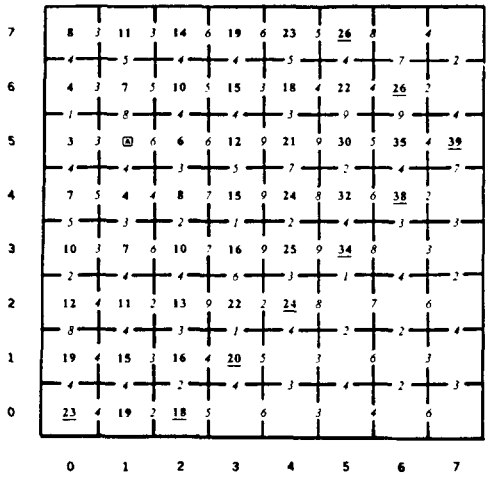
The scores at the end of the fifth step are shown in Fig. 3(b). Cells with underlined scores changed values at that step. (The empty cells are ones which have never entered the list. For programming convenience, these cells may be assigned some arbitrary large score at the beginning.) The updated scores and the cells changing scores at the sixth step are shown in Fig. 3(c). Fig. 3(d) shows the



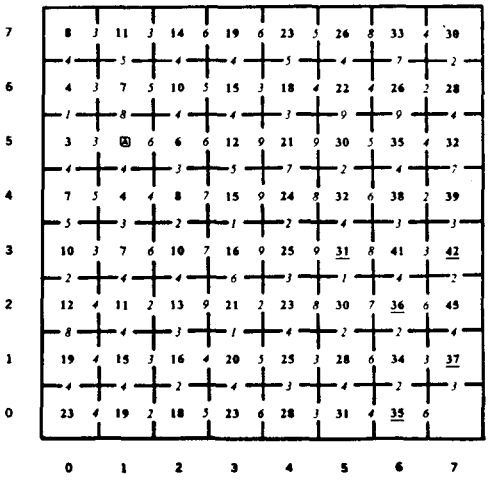
(a) Grid with original boundary costs.



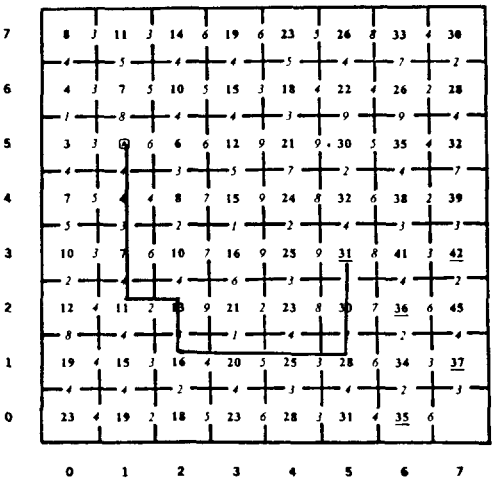
(b) Scores after Step 5.



(c) Scores after Step 6.



(d) Scores at the end of forward propagation.



(e) The backtrace route.

Fig. 3. Illustration of algorithm with nonuniform cell-boundary costs.

situation when the forward process terminates. The backtrace procedure is as described by Akers; the result is shown in Fig. 3(e).

When there are more than two points to be interconnected in a net, the source cell, the destination cell with the least score, and all cells which are in the backtrace path from this destination become additional source cells for the next forward propagation iteration.

The main differences between this algorithm and that described by Akers are the incorporation of costs on the

boundaries of cells in addition to the cells themselves, and the inclusion of via costs which affect the algorithm in a manner described in a later section.

V. Reduction of Execution Time of the Algorithm

The algorithm as described above has a complexity (measured in terms of the number of cells labeled) which grows quadratically as the length of the connection if all boundaries have identical costs. With distinct costs, the complexity does not behave predictably. There could be instances when the number of cells visited is no more than the number of cells in the backtrace path; there could also be instances when all the cells in the chip would have to be visited. Experimental results show that the average number of cells visited per connection grows as d^c, where d is the distance between the source and destination and c lies between 1 and 2.

Even this complexity is formidable when the average interconnection length increases. Fortunately, the average interconnection length is only a slowly growing function of the number of cells on a chip [11] when the placement is good and interconnections are largely local. Nonetheless, a fair speedup in the run time can be obtained by limiting the search for a path to a *window* which just encloses the rectangle bounding the points to be connected, often referred to as the *minimum rectangle* of the net. As observed in [12], nets seldom stray outside a window which exceeds the minimum rectangle by two cell widths. This limitation to the search could result in a path which is not the cheapest; however, a simple scheme can be employed to ensure that the channel usage of a boundary does not exceed the supply when an alternative acceptable path exists. This is done by maintaining two priority queues of cells to be processed. The first is a regular work queue and contains only those cells within a selected window. When a cell at a window boundary is processed, those of its neighbors that fall outside the boundary are placed in a second queue. Candidates for wavefront expansion are taken only from the first queue. This queue becomes empty only if an acceptable route within the window is not found. In such a case, the window is expanded by one unit on all sides. Since this makes the cells in the second queue valid candidates for propagation, the queues are swapped, allowing the second queue to empty its cells on to the work queue. The forward propagation is now resumed.

These speedup features were implemented in the algorithm, the results of which are quoted in Section VIII.

VI. Cost Computation

The effectiveness of the algorithm just described depends a lot on the accuracy of estimates for the cost at the various boundaries. The incremental cost of traversing a cell boundary is broken down into two components: 1) remaining supply measure, and 2) cost of cell traversal.

The currently implemented cost as a function of the remaining channel capacity is as follows: If the remaining number of tracks at a boundary is less than zero, the cost assigned to that boundary is infinity. (Infinity is implemented as the largest machine representable number. Other implementations which mark a node as visited/not visited are also possible.) If the number is greater than 3, then the cost assigned is 1. For intermediate values, the cost increases as a power of 8, i.e., 8, 64, 512, 4096 for remaining capacities of 3, 2, 1, and 0, respectively.

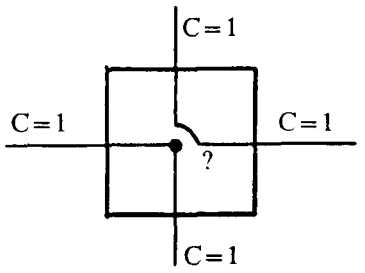


Fig. 4. Illustration of limit on the number of vias.

The cost of cell traversal itself plays a different role. If this cost is a constant, then the cumulative effect of such a cost would favor paths which are of minimal length. By setting this cost to a high value, the shortest path is found; by setting this cost to zero, the cheapest path within the window is found. Further, one could take into account factors such as the physical dimensions of the cell, or the *complexity* of wiring the cell by having different cost penalties for different cells and for different directions. This happens to be the case when the number of vias in a cell must be limited.

The fact that one can do a global wiring without exceeding the supply at any boundary does not guarantee the existence of a detailed embedding at all cells. Fig. 4 shows the simple case of a cell which has only one track available in each direction. The capacity of each channel is 1. However, if a net is globally wired passing through the west and south boundaries, it requires a via to be placed at the intersection of the two tracks. Since the supply at the north and east boundaries remains undiminished, a global path may be taken by some future net occupying these boundaries. During the exact embedding phase, no via can be placed and an overflow will result.

Often, technology rules also prohibit the placement of vias in adjacent intersections. As a result of these restrictions, one finds that there is a certain physical limit on the number of vias that can be placed in a cell [13]. The algorithm as implemented currently assigns a cost of infinity to a cell in which the number of vias placed exceeds the number of maximum vias (computed from the channel capacities). The cost decreases as a factor of 4 as the number of available via positions increases.

An acceptable path is one which has *total length* less than the assigned value for infinity. If no acceptable path is found, the cost at each boundary is divided by 3 (temporarily) and the forward propagation repeated. This is necessitated by the use of fixed width integer representation for the costs. It could be avoided at the cost of program speed by using a floating-point representation.

TABLE I
SUMMARY AND COMPARISON OF RESULTS ON FIVE IBM MASTER-SLICE CHIPS

				Old algorithm				New algorithm				
Part	Ckts.	Nets	Steiner length	Final length	Over-flows	New zeros	3081 secs.	Iter.	Final length	Over-flows	New zeros	3081 secs.
L	253	312	1938	2030	42	9	19	3	2010	33	7	30
I	1084	1292	9608	11861	135	272	98	3	11832	115	256	170
								4	11874	105	254	248
R	2057	1726	23824	26118	1144	336	547	2	25786	804	337	572
								3	25938	790	291	1007
M	2896	3041	34463	38865	37	664	1230	3	37015	44	472	678
								4	37610	10	438	974
								5	37727	4	408	1264
S	3460	3105	24433	25041	0	0	192	1	24728	0	0	135
								2	24706	0	0	313

TABLE II
PROGRAM PARAMETERS USED TO OBTAIN THE RESULTS OF TABLE I

Part	Iteration	Window Frame	Distance penalty	Excess capacity	3081 secs
L	1	2	100	2	7.3
	2	4	100	1	9.9
	3	100	100	0	13.2
I	1	1	100	2	35
	2	3	100	1	52
	3	5	100	0	83
	4	4	100	0	78
R	1	2	100	2	126
	2	4	100	0	446
	3	4	50	0	435
M	1	2	100	2	196
	2	2	100	1	239
	3	2	100	0	243
	4	4	100	0	296
	5	4	30	0	290
S	1	2	100	0	135
	2	4	50	0	178

VII. ALGORITHM MODIFICATION TO ACCOUNT FOR VIA COSTS

As a result of the introduction of a via cost into the calculation of the global route, a unique score can no longer be associated with a cell during forward propagation. A score is associated with the horizontal and the vertical directions in each cell. A backtrace pointer must also be associated with each of the directions. The direction of the backtrace path through a cell can be determined by referring to the pointer stored for the direction from which the backtrace originated.

VIII. EXPERIMENTAL RESULTS FOR MASTER-SLICE CHIPS

The program was run on five IBM master-slice chips. The results are shown in Table I. The old algorithm statistics for each chip corresponds to the results of running the chip through EDS routines (EDS is the standard Engineering Design System of IBM). The other statistics correspond to runs using the algorithm of this paper. Each line indicates the results obtained after the specified number of iterations starting from an unwired configuration.

It can be seen that the number of overflows is better than that obtained by the old program. The consistent improvement in quality with further iterations can also be noticed. Further, except for one case, the lengths of paths produced are also shorter than before. This is somewhat surprising considering that detouring (and, hence, increasing the length of paths) is one of the ways to reduce congestion.

In general, when chips are hard to wire, it is a good idea to start initially with channel supply in excess of the actually available channel capacities. This, accompanied by a small window, ensures that the first pass produces short wires. An excess capacity of 2 was found to be large enough to enable this. Too large a capacity tends to produce unreasonable initial routes. (In fact, a very large supply was used to determine how well the algorithm does in estimating the Steiner length. Assuming that the original estimate of Steiner lengths for the nets is accurate, the above algorithm produced routes which were on an average less than 1 percent longer. The final routes with proper capacity varied from 1 to 22 percent over the Steiner lengths.) Also, it appears to be a good idea to keep opening the window in later iterations so as to allow the wires to detour to avoid congestion. This must be balanced, however, by the fact that opening the window could cause the run times to be prohibitively high. Summarized in Table II are the various parameters used for the above runs. It should be emphasized that these were the *only* parameters tried out for the experiments. The above figures do not represent the result of numerous attempts with various parameters. In fact, it is quite possible that a different set of parameters could produce better results or reduce run times.

IX. WIRING MODEL FOR A STRUCTURED CUSTOM CHIP DESIGN STYLE

Fig. 5 shows a design style where a chip is partitioned into regions which abut each other. Each region contains circuits belonging to some macro, interconnections among those circuits, interconnections from these circuits to other macros, and interconnections between other macros that pass through this region. The task of global wiring is to determine the routes of wires interconnecting the various regions, such that the boundary capacities of each of the regions are not exceeded, while keeping interconnection lengths as small as possible. In the system described in [14], these routes are used to place *perimeter pins* around each region, so that the placement of components and interconnections within each macro in a region can be treated as smaller self-contained problems.

An undirected graph is used to model the problem. A region containing a macro is represented by a *node*. An *edge* connects one node to another if the corresponding regions have some common boundary. (We restrict ourselves to regions which are convex. However, the model

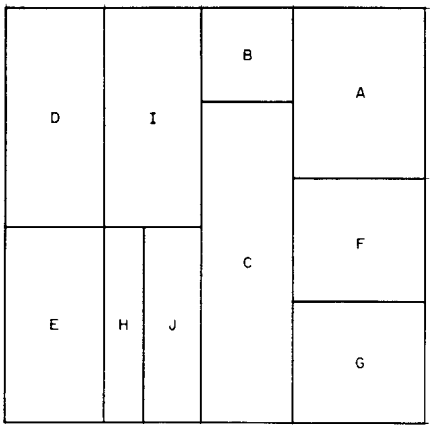


Fig. 5. Illustration of a floorplan in a custom design style (from [14]).

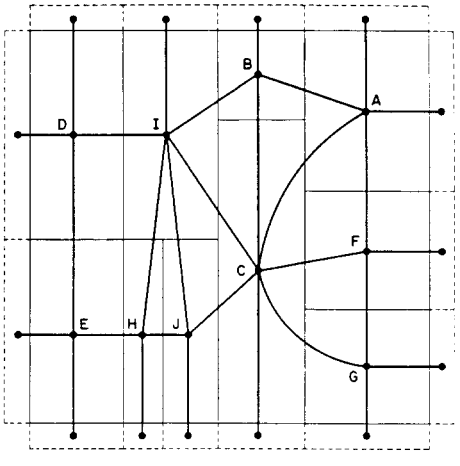


Fig. 6. Graph model for the floorplan in Fig. 5 (from [14]).

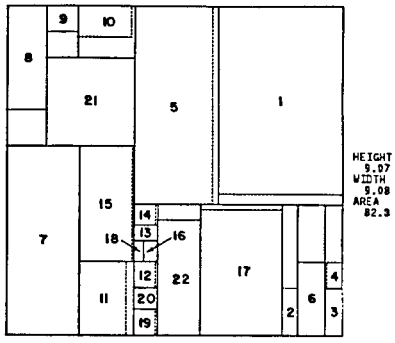
could be extended easily to nonconvex regions which may imply more than one edge between the same pair of nodes.) Associated with each edge are two quantities, the *supply* and the *length*. The supply indicates the number of perimeter pin positions available along the common boundary between two adjacent regions. The length of an edge is an approximation to the length of a wire segment which crosses the boundary represented by the edge. In the absence of knowledge about the placement of components within a region, it is convenient to set the length of an edge to the center-to-center distance between macros placed within the corresponding adjacent regions. Fig. 6 shows the graph for the example of Fig. 5.

The general graph global wiring is performed in an identical manner to the grid global wiring for master-slice chips. In fact, it is similar to the shortest path labeling algorithm mentioned in [15]. One of the nodes to be connected is chosen as a source. It is labeled with a value 0 and is inserted in a list of nodes to be processed. All other nodes have their labels set to infinity. Among the nodes in the list, the one having the smallest label is picked. For each edge connected to this node in turn, the label value is added to the cost of crossing the edge. The node at the other end of the edge has its label set to this value and is inserted into the list if the resulting value is smaller than the one it already has. The back pointer is also set appropriately. This process is continued until the smallest label value in the list is at least equal to the value at some sink. By tracing from the sink back to the source, the least cost path between the nodes is found. As before, the process is continued with the source, sink, and all nodes in the least cost path as new source nodes.

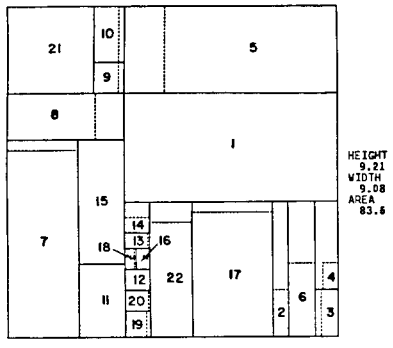
The implementation of this algorithm, called GLB, has several interesting aspects. First, the cost of traversing an edge is computed in a procedure external to the program and is controlled by the user. Parameters passed to this procedure include the length of the edge, the original supply at the edge, the remaining supply at the edge, etc. Often, besides the capacity at the boundaries of the regions and the length of the interconnections, it may be necessary to have other restrictions. For example, it may be necessary from a performance point of view to minimize the length of a small subset of nets rather than minimize the total length of all nets. Or, it may be necessary to route certain classes of nets, e.g., busses, identically. Further, the choice between cost functions may depend on the stage of design. Two cost functions are used in a current design process. The first function has a cost that is exponential with the *fractional supply* available at the edge; the second function has a cost that is exponential with the *supply* itself. In the early stages of design, when the gross validity of a floor plan is to be evaluated, use of the first cost function helps in determining wiring bottlenecks in the floor plan. Alternative floor plans can be quickly evaluated to determine the best one in terms of wireability and wire length. In the final stages of design, it is more appropriate to use the second function which helps in maximizing the absolute number of pin positions remaining at cell boundaries. By allowing the user to control the cost function, the program can be used in a greater variety of situations.

The second novel aspect of the GLB is in the control of iteration parameters. In addition to the number of iterations, and the choice of cost function, the user may specify a *capacity factor* for each iteration of the algorithm. The capacity factor is multiplied by the available capacity before the cost is computed for an edge. A capacity factor greater than one, used in the early iterations, results in short wires, but a lot of overflows. A capacity factor less than one, used in the final iterations, attempts to maximize the remaining capacity at the edges. A larger number of remaining pin positions at a boundary helps in providing a larger degree of freedom to the pin assignment phase which follows the global wiring.

Fig. 7 demonstrates the use of GLB for a real floorplanning problem. An automatic floorplanning tool [16] generated the floorplan shown in Fig. 7(a). GLB was used for determining the wireability of the resulting layout. The results indicated that the floorplan would result in an unacceptable number of overflows when wired. The distribution of the overflows was studied to determine a series of interactive moves (see [16]) on the floorplan. The resulting floorplan, shown in Fig. 7(b), was analyzed using



(a)



(b)

Fig. 7. Two sample floorplans for same chip (from [16]). (a) Original floorplan. (b) Floorplan after interactive modifications.

TABLE III
SUMMARY OF RESULTS FOR THE FLOORPLANS OF FIG. 7

	Floorplan (a)	Floorplan (b)
Area (square units)	82.3	83.6
Worst remaining capacity at an edge	−16	2
Number of overflowed edges	48	0
Total number of overflows	491	0
Total net length (units)	731.5	547.8
Total net length (edges)	2883	1950
Run time (IBM 3090 secs.)	9.28	5.45
Number of macros	22	
Number of graph edges	82	
Number of nets	887	

GLB. No overflows resulted. The routes of wires determined by GLB could now be easily translated to perimeter pin positions for the macros. Table III summarizes the results. The increase in area for the floorplan of Fig. 7(b) is insignificant, yet there is clearly a tremendous improvement in the wiring characteristics in comparison to the floorplan in Fig. 7(a).

Normally, three iterations are performed for a macro before performing the final perimeter pin assignment. The first iteration assumes the capacities at the edges to be larger than what they actually are. This leads to short wires but often a lot of overflows. The second iteration uses the results of the first iteration but brings the capacities down to their actual values. The last iteration crunches the capacities down to below their original values so that the remaining capacity at the worst edge is maximized. Variations of this scenario, or sometimes more iterations, may be needed in special cases.

X. Concluding Remarks

A simple iterative maze-running technique which rips up and reroutes every net in every iteration has been shown to be very effective in reducing the number of overflows in the global wiring of master-slice chips. Global wiring has been performed on chips having upto 3500 gates in reasonable time using mainframe CPU's. For chips that are considerably larger, it appears to be reasonable to use the technique hierarchically with a few hundred blocks in each level of the hierarchy.

The adaptation of the algorithm to a structured custom chip design environment has also been described. The data structures required to model the general problem make the algorithm somewhat slower than in the master-slice case. However, examples in this environment tend to have far fewer blocks (typically less than 100), making the run time of the algorithm acceptable. The simplicity of the algorithm and the ability to easily model technology-dependent constraints within the algorithm make it powerful and more useful than rigorous mathematical approaches taking less computation time.

The adaptability of the algorithm makes it a strong candidate for the solution of wiring problems in other levels of the packaging hierarchy, e.g., chips on modules or modules on boards. Since the grid of available wire tracks is typically large, a hierarchical global wiring would be appropriate here. For example, a grid of 1500 × 1500 wiring tracks could be subdivided into a higher level of 50 × 50 squares, each square containing a grid of 30 × 30 tracks.

Acknowledgment

Discussions with S. J. Hong, M. Burstein, D. Mehta, and E. Schanzenbach, and the encouragement provided by J. Darringer and L. Berman are greatly appreciated. Thanks are also due to P. Hauge, J. Hutt, G. Sorkin, L. Woo, and E. Yoffa for their help, feedback, and comments.

References

[1] K. A. Chen, M. Feuer, K. H. Khokhani, N. Nan, and S. Schmidt, "The chip layout problem: An automatic wiring procedure," in *Proc. 14th Design Automat. Conf.* (New Orleans, LA), 1977, pp. 298-302.

[2] H. Shiraishi and F. Hirose, "Efficient placement and routing techniques for master-slice LSI," in *Proc. 17th Design Automat. Conf.* (San Diego, CA), 1980, pp. 458-464.

[3] J. Soukup and J. C. Royle, "On hierarchical routing," *J. Digital Systems*, vol. V, no. 3, pp. 265-289, 1981.

[4] Z. Syed, A. El Gamal, and M. A. Breuer, "On routing for custom integrated circuits," in *Proc. 19th Design Automat. Conf.* (Las Vegas, NV), 1982, pp. 887-893.

[5] R. Linsker, "An iterative-improvement penalty-function driven wire routing system," *IBM J. Res. Develop.*, vol. 28, no. 5, pp. 613-624, 1984.

[6] R. Nair, S. J. Hong, S. Liles, and R. Villani, "Global wiring on a wire-routing machine," in *Proc. 19th Design Automat. Conf.* (Las Vegas, NV), 1982, pp. 224-231.

[7] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 346-365, 1961.

[8] E. F. Moore, "Shortest path through a maze," in *Annals of the Computation Laboratory*. Cambridge, MA: Harvard Univ. Press, 1959, pp. 285-292.

[9] W. A. Dees and R. J. Smith, "Performance of interconnection rip-

up and reroute strategies," in *Proc. 18th Design Automat. Conf.* (Nashville, TN), 1981, pp. 382-390.
[10] S. Akers, "Routing," in *Design Automation of Digital Systems: Theory and Techniques*, vol. 1, M. A. Breuer, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1972, pp. 283-333.
[11] W. E. Donath, "Wire length distributions for placements of computer logic," *IBM J. Res. Develop.*, vol. 25, no. 3, pp. 152-155, 1981.
[12] D. Wallace and L. Hemachandra, "Some properties of a probabilistic model for global wiring," in *Proc. 18th Design Automat. Conf.* (Nashville, TN), 1981, pp. 660-667.
[13] D. T. Lee, S. J. Hong, and C. K. Wong, "Number of vias: A control parameter for global wiring of high density chips," *IBM J. Res. Develop.*, vol. 25, no. 4, pp. 261-271, 1981.
[14] P. S. Hauge and E. J. Yoffa, "Vanguard: A chip physical design system," in *Proc. 23rd Design Automat. Conf.* (Las Vegas, NV), 1986, pp. 440-446.
[15] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematik*, vol. 1, pp. 269-271, 1959.
[16] L. S. Woo, C. K. Wong, and D. T. Tang, "PIONEER: A macro-based floor-planning design system," *VLSI Design*, vol. VII, no. 8, pp. 32-43, 1986.

*

Ravi Nair (M'82) received the B.Tech. degree in electronics and electrical communications engineering from the Indian Institute of Technology, Kharagpur, India, in 1974. He received the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana, in 1976 and 1978, respectively.

Since 1978, he has been with IBM at the T. J. Watson Research Center in Yorktown Heights, NY. He has worked on computer synthesis of VLSI layout, parallel machines for physical design, fault-tolerant systems, and testing. He is interested in algorithms and systems for the design of digital integrated circuits.