A NEW ITERATIVE SUPPLY/DEMAND ROUTER WITH RIP-UP CAPABILITY FOR PRINTED CIRCUIT BOARDS

Eric Rosenberg
AT&T Bell Laboratories
Holmdel, NJ 07733

Abstract

In this router, an iterative, non-sequential "supply/demand" router attempts to find conflict-free (non-crossing or overlapping) paths for each net. If 100% interconnection is not achieved with supply/demand routing, the rip-up procedure removes enough nets to eliminate all conflicts. Nets that are ripped-up are returned to the supply/demand router for re-routing. The new router was compared to a Lee-type router on six double-sided boards, and on average obtained 0.8% higher completion.

1. Introduction

This paper documents and reports computational experience with a new router for PCB interconnection. The new router is quite different from existing routers. Most current routers are sequential and non-iterative, which means that segments (connections between two given points in a net) are routed one by one, and the path chosen for one segment is never modified to make room for a later segment (channel routers do modify paths in a channel, but we are concerned here with global Lee-type routers). The segments are typically ordered (e.g., by increasing length) to increase the chances of achieving 100%. Unfortunately, there may not exist any ordering such that a sequential router can obtain 100%. This is not surprising, for a sequential router is essentially a "greedy" router, which allocates routing space based on segment priority - and greedy methods cannot in general be expected to solve difficult optimization problems.

The new router has two major pieces (see Figure 1): an *iterative, non-sequential "supply/demand" router* and a *rip-up procedure*. The supply/demand router attempts to find conflict-free (i.e., non-overlapping or crossing) paths for each net. If the supply/demand router fails to achieve 100% conflict-free routing, the rip-up procedure removes enough nets to eliminate all conflicts. Nets that are ripped-up are returned to the supply/demand router for re-routing.

In the supply/demand router, all nets compete equally for "valuable" routing resources: nodes in the grid graph. (A node in this graph corresponds to a "cell" in a cell-based router.) No net ordering is done. In each iteration, each net is routed *independently* of the others (and thus all nets can in principle be routed simultaneously with parallel processing). The net routings are cost driven: "arc costs" (the nominal costs of moving from one node to an adjacent node) are combined with "node costs" (costs assigned to the nodes) to obtain the costs actually used to route the nets. Each net is routed for minimum cost. If, at the end of an iteration, no node is used by more than one net then we have succeeded in obtaining a 100% routing. Otherwise, if two or more nets desire a node, the node is made more expensive. New routing costs are calculated with the revised node costs, and the nets are then routed again, using the new routing costs. If a node desired by multiple nets is made sufficiently expensive, then hopefully at most one net will be willing to pay the price for using that node - hence the name "supply/demand router".

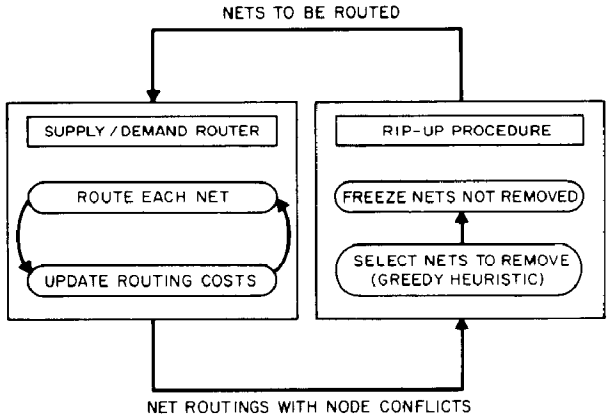

FIGURE 1 ROUTER OVERVIEW

If, after a specified number of supply/demand iterations, the supply/demand router does not succeed in resolving competition for nodes, enough nets are ripped-up (i.e., removed) to eliminate all conflicts. The rip-up problem is modelled as a special type of integer program (a generalized set covering problem [13]) and a heuristic solution method, based upon an effective heuristic for the set covering problem, was devised. The rip-up procedure uses information about those nodes used by multiple nets to decide the best set of nets to remove. The nets *not removed* are "frozen" (i.e., their current routings are now considered permanent and can no longer be modified). The nets removed are then routed on whatever space on the board is not occupied by the frozen nets. The actual routing of the removed nets is accomplished by again using the supply/demand router. If, after several iterations, the removed nets cannot be routed in the remaining space without conflicts, then the rip-up procedure is again applied to remove enough of the nets just routed to eliminate all conflicts. The new set of nets removed is again given to the supply/demand router, and this process continues until all nets are routed without conflicts or no further progress is possible.

2. Survey of Related Literature

In work closely related to our new router, Feo and Hochbaum [3] use Lagrangian duality [5] to determine the feasibility of the routing problem where all nets are two point nets. Their algorithm is not designed to find a routing yielding 100% completion, but rather seeks to determine if it is theoretically possible to achieve 100% completion. Hu and Shing [6] apply large-scale linear programming techniques, together with problem reducing heuristics, to solve the PWB routing problem. No computational results are presented.

Iterative routers have been proposed by Rubin [12], Moore and Ravitz [10], and Linsker [9]. In these methods, at each iteration every net is routed so as to minimize a penalty function which is the sum of path length and crossover penalties. After each iteration, the crossover penalty constant is increased. Unlike our new router, these routers are still "sequential": at each iteration the crossover penalties incurred by some net depend on the nets *already routed* in that iteration. In the simulated annealing method of Vecchi and Kirpatrick [16], a "random" route is generated for each net from a set of simple alternatives. After a random route is selected for every net, a penalty function sums, over all arcs in the grid, the number N of nets using the arc. Comparing the penalty function in the current and previous iterations, the current routes are selected according to probabilistic rules. In Soukup's Lee-based method [15], if two or more nets both want to use some cell then the cell is allocated to one of the competing nets, depending on the net priorities. Net (and subnet) priorities are in general changed frequently throughout the routing, so individual cells are reassigned, as the routing progresses, to different nets.

We now review the literature on rip-up. Existing rip-up methodology is experimental and heuristic, and is much less developed than constructive routing. In Rubin [12], if the iterative routing procedure fails to route all nets without crossovers, then the nets with the highest penalties are removed and routed on other layers or with discrete wire. A similar strategy is used by Linsker [9]. Dees and Karger [2] consider various reroute metrics, and consider estimating the value of ripping-up some net (in order to make room for others) and also estimating the success of re-routing some net that was ripped-up.

One striking feature of current rip-up work is the lack of any mathematical modeling. One of the contributions of this work is the recognition that rip-up can be modeled as an optimization problem for which we propose an effective heuristic solution method.

3. Mathematical Formulation of PWB Routing

We formulate the PWB routing problem on a grid graph with two or more layers. A node of this graph is defined by an (i,j,k) triple, where k designates the layer. An arc is a link between some node (i,j,k) and an adjacent node (i.e., a node (i',j',k') such that $|(i-i')|+|(j-j')|+|(k-k')|=1$, where $|x|$ denotes the absolute value of x). If $|k-k'|=1$, the arc is a via, otherwise it is an arc on a layer.

Each net is defined by a set of terminal nodes (i.e., nodes corresponding to the terminal pins) to be interconnected by a Steiner tree. The optimization problem we consider is to interconnect each net with a Steiner tree, such that the sum of the arc costs (defined below) is minimal and no two Steiner trees share a common node.

To formulate this mathematically, let y_{as} be the binary (0 or 1) variable such that $y_{as}=1$ if the arc a is used in the Steiner tree for net s, and $y_{as}=0$ otherwise. Let x_{ns} be the binary (0 or 1) variable such that $x_{ns}=1$ if the node n is used in the Steiner tree for net s, and $x_{ns}=0$ otherwise. Let c_a be the user-supplied cost of using arc a (for any net). For each a, c_a is given one of three values: the cost $C1$ ($C2$) if a is an arc on the preferred (non-preferred) direction on a given layer, or the cost CV if a is a via.

For each arc a, let C_a be the set of nodes n such that, if a is used to route net s, then node n also becomes dedicated to net s. For example, let a be any arc on a layer (i.e., a is not a via). Then C_a is the set consisting of the two end-nodes of arc a. If a is a large via (whose diameter is much larger than the path width and thus overlaps neighboring nodes) then C_a contains in general more nodes than the two end-nodes of arc a. We also define B_n to be the "inverse operation" of C_a:

$$B_n=\left\{a \mid n\in C_a\right\}.$$

For example, on a double-sided board if all vias are microvias (vias whose diameter is roughly equal to the path width) and node n is not on the boundary of the grid then $|B_n|=5$, since there are four arcs incident to n on the layer containing n and one via incident to n. Let $|B_n|$ denote the cardinality of (number of elements in) the set B_n.

We may now formulate the PWB routing problem as follows:

$$\textit{minimize} \sum_a c_a \sum_s y_{as} \tag{1.1}$$

$$\textit{subject to: } \{y_{as}\} \textit{ forms a Steiner tree for net } s, \textit{ all } s, \tag{1.2}$$

$$\sum_{a\in B_n} y_{as} \le |B_n|\, x_{ns}, \textit{ all } n \text{ and } s, \tag{1.3}$$

$$\sum_s x_{ns} \le 1, \textit{ all } n, \tag{1.4}$$

$$y_{as}=0 \textit{ or } 1, \text{ and } x_{ns}=0 \textit{ or } 1, \textit{ all } n, a, s. \tag{1.5}$$

The objective function is to find the lowest total cost interconnection routing. The first set of constraints require that an interconnection routing be found for each net. The second set says that, for each node n and each net s, if some arc a in B_n is used to route net s (i.e., if $y_{as}=1$), then node n must be dedicated to net s (i.e., $x_{ns}=1$). The third set of constraints states that each node n can be dedicated to at most one net.

4. Evolution of the Supply/Demand Router: A Lagrangian Relaxation Approach

The supply/demand router is the product of our computational experience with a combinatorial optimization technique called Lagrangian relaxation [4], [14]. This technique simplifies hard problems by incorporating difficult constraints into the objective function. In problem (1) (i.e., the problem defined by (1.1)-(1.5) in Section 3), the difficult constraints are the constraints (1.3), since they involve both the y_{as} and x_{ns} variables. We associate a nonnegative "Lagrange multiplier" or "price" α_{ns} with each constraint of type (1.3). We then "dualize" the problem by adding to the objective function the sum of the "weighted constraints" to arrive at the following "relaxed problem":

$$\textit{minimize} \sum_a c_a \sum_s y_{as} + \sum_{n,s} \alpha_{ns} \left(\sum_{a\in B_n} y_{as} - |B_n|\, x_{ns} \right) \tag{2.1}$$

$$\textit{subject to: } \{y_{as}\} \textit{ forms a Steiner tree for net } s, \textit{ all } s, \tag{2.2}$$

$$\sum_s x_{ns} \le 1, \textit{ all } n, \tag{2.3}$$

$$y_{as}=0 \textit{ or } 1, \text{ and } x_{ns}=0 \textit{ or } 1, \textit{ all } n, a, \text{ and } s. \tag{2.4}$$

Notice that problem (2) completely decomposes into subproblems depending on y_{as} and subproblems depending on x_{ns}. In particular, define

$$\bar{c}_{as} = c_a + \sum_{n\in C_a} \alpha_{ns}. \tag{3}$$

Then for each net s we obtain the following routing problem, involving no other nets:

$$\textit{minimize } \sum_a \bar{c}_{as}\, y_{as} \tag{4.1}$$

$$\textit{subject to: } \{y_{as}\} \textit{ defines a Steiner tree for net } s, \tag{4.2}$$

$$y_{as}=0 \textit{ or } 1, \textit{ all } a \text{ and } s. \tag{4.3}$$

Also, for each node n we obtain the subproblem

$$\textit{maximize } |B_n| \sum_s \alpha_{ns}\, x_{ns} \tag{5.1}$$

$$\textit{subject to } \sum_s x_{ns} \le 1, \tag{5.2}$$

$$x_{ns}=0 \textit{ or } 1, \textit{ all } n \text{ and } s. \tag{5.3}$$

Problem (5) is simply interpreted as assigning each node to the highest bidder (i.e., highest α_{ns}). Of much greater interest is

problem (4): Interconnect net s using special "routing costs" defined by (3). Interpreting α_{ns} as the "price" that net s must pay for using node n, then, to use arc a to route net s, we must pay the original cost c_a plus the price of each node that is "influenced" by arc a ("influenced" means $n \in C_a$). For example, if all vias are micro vias, then each arc routing cost in (4) is the original cost plus the price of each endpoint of the arc.

Suppose now that for some fixed choice of multipliers α_{ns} we solved each of the subproblems (4) and (5). Then we will obtain a Steiner tree for each net (from (4)), and each node will be assigned a net (from (5)). If no two Steiner trees share a common node, then we have obtained conflict-free routings for all nets; under certain easily verified conditions these routings will be optimal for (1). However, in general we may expect that the Steiner trees for two nets, say nets p and q, may share a common node, say node n. If also $\alpha_{np} > \alpha_{nq}$ then $x_{np} = 1$ and $x_{nq} = 0$, so the constraint (1.3) for node n and net q is not satisfied (the constraint (1.3) for node n and net p *is* satisfied).

The theory of Lagrangian relaxation specifies rules [5], utilizing the solutions of subproblems (4) and (5), for updating the multipliers α_{ns} in the hopes of achieving routings that do satisfy all the constraints of (1). If, for some n and s the corresponding constraint in (1.3) is violated, then α_{ns} is increased. If there is slack in the constraint, α_{ns} is decreased. If the constraint is satisfied exactly, α_{ns} is unchanged.

A code implementing the Lagrangian relaxation technique was written, solving subproblems (4) using the Steiner tree heuristic in [7]. Using small problems constructed by the author (e.g., 10 nets on a square grid 30 nodes on a side), many experiments were made with various updating rules. We found that the updating rules changed the multipliers α_{ns} in undesirable ways, sometimes causing the multipliers to oscillate without converging, and sometimes causing the multipliers to move too slowly to accomplish the required effects. In particular, we noted that, for a given node, multiple nets would often be in contention for the node, with no net becoming the clear winner as the iterations progressed. Thus, not only did we fail to achieve a feasible solution to (1), but the routes generated in step (4) often varied greatly from iteration to iteration. Fortunately, a way was discovered to rectify these problems while still retaining the attractive economic interpretation afforded by Lagrangian relaxation.

5. The Supply/Demand Router

The supply/demand router is an iterative procedure, inspired by the Lagrangian relaxation method described above. In each iteration of the Supply/Demand Router, each net is routed, using special "routing costs", independently of the other nets. Then the routing costs are updated and the next iteration begins. Two major departures from the Lagrangian relaxation approach were taken to produce the supply/demand router: (i) using a single node cost (independent of s) at each node, and (ii) ignoring the variables x_{ns} and the associated problems (3). In addition, price updating has been simplified. To explain this, we introduce the new concepts of node color and usage. We associate with each node n a node "color" K_n. The color represents the number of the net that is currently assigned to the node. For example, $K_n = s$ means that net s is currently assigned to node n. We allow the node color to be either positive, negative, or zero. If $K_n = 0$, then no net is currently assigned to the node. If $K_n = s$ and $s < 0$, then net s is *temporarily assigned* to the node. If $K_n = s$ and $s > 0$, then net s is *permanently assigned* to the node in the *current iteration* of the supply/demand router. Note the emphasis on "current iteration": a permanent assignment may become a temporary assignment at the end of an iteration.

Node Costs. The supply/demand router uses a set of node costs $\{\beta_n\}$. The price β_n is the price charged to any net that wants to use node n in its routing. It has the same function as the prices $\{\alpha_{ns}\}$ arising from Lagrangian relaxation. However, our costs are simpler (one subscript instead of two) and lead to a more stable router.

Routing Costs. The supply/demand router in each iteration calculates a set of "arc prices" $\{d_a\}$. The arc prices are defined by substituting β_n for α_{ns} in formula (3):

$$\bar{d}_a = c_a + \sum_{n \in C_a} \beta_n .$$

The arc prices are used in the following way. For each arc a and each net s let d^0_{as} be the routing cost, in the current iteration, to net s of using arc a in the Steiner tree. Then the following simple rule is used to define d^0_{as}: $d^0_{as} = BIG$ if $K_n > 0$ and $s \neq K_n$ for any $n \in C_a$; $d^0_{as} = \bar{d}_a$, otherwise. Thus, the routing cost for net s to use arc a is BIG if some node in C_a was permanently assigned to another net at the end of some previous iteration (see Section 7). Otherwise, the routing cost is set to $\bar{d}_a$, a value independent of s. Using the same $\bar{d}_a$ for each net makes good economic sense: the price of an item should not depend on the identity of the prospective buyer. At the end of the iteration, the node prices $\{\beta_n\}$ are updated by the rules detailed in Section 7.

6. Rip-up

Rip-up is executed after a specified number of supply/demand iterations (or until a stopping criterion is reached). The rip-up problem can be formulated as an integer program as follows. Let

$N =$ *the set of all nodes,*
$M =$ *the set of nodes used by more than one net,*
$S =$ *the set of nets whose routing uses nodes in* M,

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1q} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ a_{p1} & \cdots & a_{pq} \end{pmatrix}$$

where

$p = |N|$, *the cardinality of* N,
$q = |S|$, *the cardinality of* S,

$$a_{ns} = \begin{cases} 1 & \text{if } net\ s \text{ uses } node\ n \\ 0 & otherwise, \end{cases}$$

$r_s =$ *the penalty cost for removing net* s,
$b_n =$ *the number of nets using node* n.

Note that $b_n > 1$ if $n \in M$. We wish to determine the decision variables

$$y_s = \begin{cases} 1 & \text{if } net\ s \text{ is removed} \\ 0 & \text{otherwise.} \end{cases}$$

The rip-up problem may now be stated as follows:

$$\text{minimize} \quad \sum_{s \in S} r_s y_s$$

$$\text{subject} \quad \sum_{s \in S} a_{ns} y_s \geq b_n - 1, \quad \text{all } n \in M,$$

$$y_s = 0 \text{ or } 1, \quad \text{all } s \in S.$$

The objective is to minimize the sum of the penalty costs for removed nets, subject to the constraint that enough nets must be removed so that each node previously in conflict now has at most one net using it. (We say "at most" since, in the process of removing the nets, a node which was in conflict may now not be used by any net.)

The Greedy Algorithm.
The integer program defined above is nearly identical to the classical set covering problem [13] (it would be exactly a set covering problem if each right-hand-side $b_n - 1$ were replaced by 1). Since this is a

specially structured integer program, we will employ a heuristic solution procedure rather than an optimum-finding integer programming code. The following greedy algorithm generalizes the heuristic of Chvatal [1] for set covering problems.

Start: $i=0$, $M_0 \leftarrow M$, $y_s \leftarrow 0$ for *all* $s \in S$
At iteration i:

1. Find that net s^* for which $y_s = 0$ and
$$r_s / \sum_{n \in C_s} a_{ns} \text{ is } minimal.$$
2. Set $y_{s^*} \leftarrow 1$.
3. Set $b_n \leftarrow b_n - a_{ns^*}$, for all $n \in N$.
4. Set $M_{i+1} \leftarrow \{n \in M_i \mid b_n > 1\}$.
5. STOP if $M_{i+1} = \emptyset$, otherwise set $i = i+1$ and go to 1.

The greedy procedure, being a heuristic, cannot in general be expected to actually determine the optimal solution of the above integer program. The effect of non-optimality of our heuristic is to possibly remove more nets than necessary to eliminate conflicts. However, since removed nets are re-routed, the penalty for non-optimality can be expected to be small.

The rip-up heuristic requires a choice of objective functions. Possible objective functions are (i) remove the fewest possible nets, (ii) remove a set (call it S) of nets such that the sum of the number of segments of the nets in S is minimal, (iii) remove a set S of nets such that the sum of the lengths of the nets in S (as measured by the number of arcs in the current routing) is minimal, and (iv) remove a set S of nets such that the sum of the number of vias (in the current routing) of the nets in S is minimal. All of our experiments to date use objective (ii).

7. Required Updates

This section details the updating rules for the entire supply-demand/rip-up router. In particular, we update the node prices β_n, which are updated to reflect the competition for the node by the nets, and the node colors K_n, which was used when defining the routing costs. We also introduce "node counters" and a scheme called "node reserving".

There are six types of updates; Figure 2 illustrates their inter-relationships.

Type 1: Updates Made Before the First Supply/Demand Router Execution.
1. Specify which nets are to be routed (e.g., signal nets only, or all nets).

Type 2: Updates Made at the Start of the Supply/Demand Router.
The following assignments are made at the start of each execution of the supply/demand router.
1. Initialize the node costs β_n.

Type 3: Updates Made at the Start of Each Inner Iteration.
1. Initialize the node colors K_n for all permanently assigned nodes (e.g., nodes covered by a terminal land for net s are permanently assigned to net s).
2. Define and initialize a new array $\{U_n\}$ where U_n is the number of nets using node n.

Type 4: Updates Made After a Net has been Routed.
The following updates are made in each supply/demand inner iteration after any net is routed (thus we do these updates in each iteration after net 1 is routed, after net 2 is routed, etc.) These updates involve only that single net just routed (call it net s). Recall that node n is used by net s if $n \in C_a$ for some arc a used in the Steiner tree for net s.
1. Update K_n and U_n for each node n used in the net s routing.

Type 5: Updates Made at the End of an Iteration.
The following updates are made at the conclusion of each iteration of the supply/demand router. They have the functions of adjusting the node costs to reflect competition, and implementing a "node reservation policy".
1. Update each node costs β_n, based on the node counter U_n.
2. For each node, if the node is used by exactly one net, then we permanently assign it to (reserve it for) that net. If the node is permanently assigned to (reserved for) some net that has not been frozen, but the node is unused in the current iteration, we un-reserve it so that some other nets have the chance to use this node in the next iteration.

Type 6: Updates Made After Rip-up.
The node updates made in rip-up have the job of telling the supply/demand router which nodes are now occupied by the frozen nets. Note that by a frozen net we mean below any net frozen in the current application of rip-up or frozen in a prior application. By currently routed nets we mean those nets routed by the supply/demand router execution immediately preceding the current rip-up execution. The updates are as follows.
1. Update the set of nets to be routed.
2. Update the node colors K_n and node counters U_n for all nodes used in the current routings.

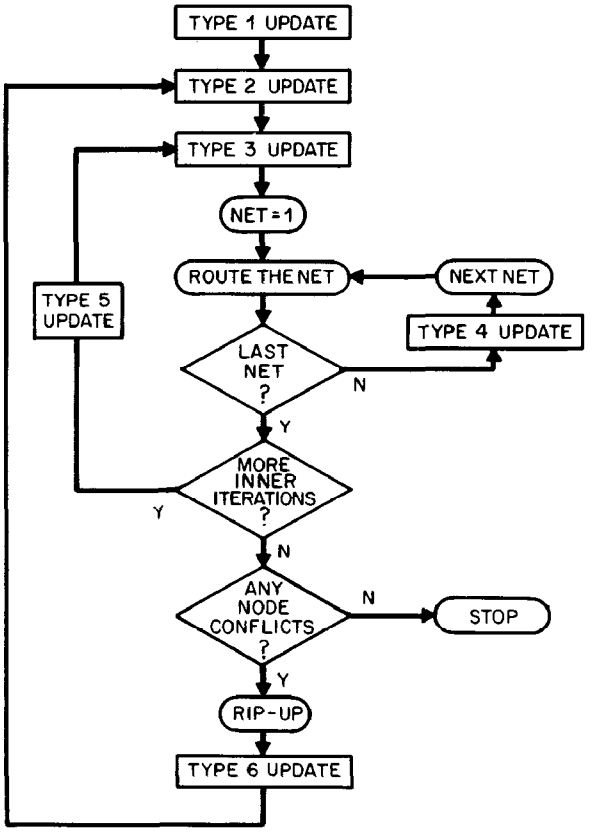

FIGURE 2 FLOW OF UPDATES

8. Computational Results and Comparisons

In this section we present some results, for six double sided boards (no multi-layer boards were routed), comparing our router and a sequential, non-iterative Lee-type router. Referring to Table 1, B1 and B2 were constructed to test the router; the others are real boards. The routing areas (in inches) are 6.5x4.5 (B3), 10x7.6 (B4),

9.8x3.5 (B5), and 5.5x9 (B6). Only signal nets were routed (not power and ground). All boards were routed on a 25 mil grid. Signal path width and clearance are 10 mils. Vias are 50 mils in diameter. To make valid comparisons, the same assumptions were made for the Lee router. We used the heuristic in [7] to compute Steiner trees. For the boards B1, B2, and B3, seven iterations were performed for each execution of the supply/demand router; six iterations were used for the other three boards. Results with the new Supply-Demand/Rip-Up (SDRU) and Lee routers are presented in Table 1. For each board, the table lists (i) the number of signal nets, (ii) the number of signal segments (a net with N terminal nodes has $N-1$ segments), (iii) the percentage of segments routed (conflict-free) by the Lee router, and (iv) the percentage of segments routed (conflict-free) by the SDRU router.

Table 2 gives information, for each board, on the number of nets and segments removed after each execution of the rip-up procedure. An entry of zero nets (or segments) removed indicates that the current execution of the supply/demand router routed the current nets with no conflicts (and thus 100% conflict-free routing was achieved for the entire board). The CPU time on an IBM 3033 for these runs (for the prototype Fortran code) was 14 minutes for B1, 56 minutes for B2, 5.2 hours for B3, 2.5 hours for B5, and about 18 hours for B4 and B6. (It is evident from Table 2 that, for B4 and B6, considerable effort was expended, with little success, in the latter iterations; the router should terminate when it is observed that little progress is being made.) Relatively, very little time was spent in rip-up; e.g., for board B3 the total CPU time for rip-up was under 3 minutes. Finally, early results with the C language production implementation of the code (currently being developed by D.B. Mellen and W.J. Li) show a great speedup: CPU times on a 2.8 mips computer are 6 minutes for B2, 4.4 hours for B4, 1 hour for B5, and 3.7 hours for B6. The use of parallel processing for net routings could provide substantial additional speedup.

Table 3 gives information about the progress made by the first execution of the supply/demand router. Here "nodes" is the number of nodes in conflict (i.e., used by more than one net), and "count" means the sum, over all nodes in conflict, of the number of nets using the node. In the notation of Section 7,

$$count = \sum_{\{n \mid U_n > 1\}} U_n .$$

Table 1: Router Comparisons

Board	Nets	Segments	Lee%	SDRU%
B1	40	72	88.9	100
B2	57	81	93.0	100
B3	104	167	93.8	100
B4	136	228	94.8	99.1
B5	149	213	93.3	100
B6	195	405	89.1	94.5

Table 2: Nets and Segments Removed in Each Rip-up

Rip-up No.	1	2	3	4	5	6	7	8
B1								
nets	4	0	-	-	-	-	-	-
segments	11	0	-	-	-	-	-	-
B2								
nets	17	4	0	-	-	-	-	-
segments	25	5	0	-	-	-	-	-
B3								
nets	57	31	14	3	0	-	-	-
segments	90	46	19	4	0	-	-	-
B4								
nets	81	49	27	14	8	4	3	2
segments	132	74	42	18	8	4	3	2
B5								
nets	70	33	15	2	0	-	-	-
segments	101	41	20	3	0	-	-	-
B6								
nets	98	60	41	32	21	17	16	14
segments	216	130	89	62	38	29	24	22

Table 3: Progress of Supply/Demand Routing Prior to First Rip-up

Iteration	1	2	3	4	5	6	7
B1							
nodes*	607	422	331	193	199	22	34
count**	1519	1051	828	465	412	45	68
B2							
nodes	761	658	474	463	332	411	291
count	1818	1919	1309	1206	972	964	905
B3							
nodes	2623	2609	2504	2415	2431	1999	2068
count	7216	7087	7378	7063	7988	7119	7151
B4							
nodes	5044	5246	4971	4631	4288	4286	
count	15471	18005	17119	15793	16216	14338	
B5							
nodes	3382	3500	2683	2671	2243	2121	
count	8841	10211	8081	8278	7319	6668	
B6							
nodes	5937	4504	4172	4097	4130	3668	
count	16091	15619	13787	13997	13143	12408	

nodes: number of nodes in conflict (i.e., used by >1 net)
count: sum (over all nodes in conflict) of the number of nets using the node

9. Current Status and Future Work

Work is underway to implement a production version of the new router. Extensive testing will be used to determine the best strategy for terminating the supply/demand iterations and choosing the best rip-up objective function.

Acknowledgments

The author wishes to thank G.A. Kochman, M.E. Meth, G.L. Miller, C.W. Rosenthal, and J.P. White for many helpful suggestions, and J.P. Grossmann for developing very useful graphics capabilities. The rip-up procedure was coded by J.S. Harrison.

REFERENCES

[1] V. Chvatal, "A Greedy Heuristic for the Set-Covering Problem", *Mathematics of Operations Research, 4,* (1979) 233-235.

[2] W. A. Dees and P. G. Karger, "Automated Rip-up and Reroute Technique", *Proc. 19 Design Automation Conference,* (1982) 432-439.

[3] T. A. Feo and D. S. Hochbaum, "A Lagrangian Relaxation Method for Testing the Infeasibility of Certain VLSI Routing Problems", unpublished manuscript, November, 1985.

[4] M. L. Fisher, "The Lagrangian Relaxation Method for Solving Integer Programming Problems", *Management Science, 27,* (1981) 1-18.

[5] M. Held, P. Wolfe, and H. Crowder, "Validation of Subgradient Optimization", *Mathematical Programming, 6,* (1974) 62-88.

[6] T. C. Hu and M. T. Shing, "A Decomposition Algorithm for Circuit Routing", in *Mathematical Programming Study 24* (R. W. Cottle, ed., North Holland, Amsterdam, 1985), pp. 87-103.

[7] L. Kou, G. Markowsky, and L. Berman, "A Fast Algorithm for Steiner Trees", *Acta Informatica, 15,* (1981) 141-145.

[8] C. Y. Lee, "An Algorithm for Path Connections and its Applications", *IRE Trans. Electron. Comput., EC-10,* (1961) 346-365.

[9] R. Linsker, "An Iterative Improvement Penalty-Function-Driven Wire Routing System", *IBM J. Res. Develop., 28,* (1984) 613-624.

[10] A. Moore and C. Ravitz, "Weighted and Iterative Multi-Wire Routing", *IBM Technical Disclosure Bulletin, 25,* (1982) 3619-3628.

[11] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity,* (Prentice-Hall, Englewood Cliffs, NJ 1982).

[12] F. Rubin, "An Iterative Technique for Printed Wire Routing", *Proc. 11 Design Automation Workshop,* (1974) 308-313.

[13] H. M. Salkin, *Integer Programming,* (Addison-Wesley, Reading, MA, 1975).

[14] J. F. Shapiro, "A Survey of Lagrangian Techniques for Discrete Optimization", *Annals of Discrete Mathematics, 5,* (1979) 113-138.

[15] J. Soukup, "Global Router", *Proc. 16 Design Automation Conference,* (1979) 481-484.
[16] M. P. Vecchi and S. Kirpatrick, "Global Wiring by Simulated Annealing", *IEEE Trans. Computer-Aided Design CAD-2,* (1983) 215-222.