Computers and Artificial Intelligence, Vol. 6, 1987, No. 3, 229—242

EXPERIMENTAL RESULTS FOR A LINEAR PROGRAM GLOBAL ROUTER

Antony P-C NG[1], Prabhakar RAGHAVAN[2], Clark D. THOMPSON[3]

Computer Science Division, 573 Evans Hall,
U. C. Berkeley, CA 94720, USA

Abstract. Classical approaches to global routing in gate-arrays involve maze-running or hierarchical decomposition. We present a completely different approach here based on an integer programming formulation of the problem. Furthermore, we describe an efficient procedure for approximately solving such integer programs. Our approximation is provably good in that we always find a solution close to the optimal one. Nets are routed simultaneously rather than sequentially, with no backtrack. Experiments with our method were conducted on two industrial examples; in both cases our router took less than one minute to find a feasible routing.

Экспериментальные результаты для глобального рутера линейной программы
А. П.-К. Нг, П. Раджаван, К. Д. Томпсон

Резюме. Классические подходы к глобальной маршрутизации в массивах вентилей включают прогон лабиринта или иерархическую декомпозицию. Нами предлагается иной подход, основанный на формулировке данной проблемы в виде задачи дискретного программирования. Более того, дается описание эффективной процедуры для приближенного решения таких дискретных программ. Доказано, что предлагаемая аппроксимация является хорошей в том смысле, что всегда находится решение, близкое к оптимальному. Сети трассируются скорее одновременно, чем последовательно, без бэктрекинга. Эксперименты, основанные на нашем методе, проводились в рамках двух промышленных примеров, в обоих случаях нашему рутеру потребовалось менее 1 минуты для нахождения допустимой маршрутизации.

[1] Supported by California State MICRO (Mentor Graphics) and the National Computer Board of Singapore.

[2] Supported by an IBM Doctoral Fellowship and Semiconductor Research Corp. grant 82-11-008.

[3] Supported by Semiconductor Research Corp. grant SRC 82-11-008.

1. INTRODUCTION

Gate-arrays are an important and popular framework for semicustom VLSI design. Logic gates or functional blocks are arranged in a rectangular array on a chip manufactured in volume. The gates in a customer's circuit are first mapped onto the gates on the array. This is known as the placement process, and it is followed by routing. In the routing phase, the gates on the array are to be interconnected electrically in such a manner as to realize the customer's circuit for the given placement. This is done by running wires through on the array joining the gates to be connected. A detailed description of the routing problem is deferred to the next section.

Classical methods for routing fall into two major classes: the "Lee-type" [8] or maze-running routers and the hierarchical routers [1]. In the former, connections between gates are processed sequentially and routed whenever possible. When it is no longer possible to route a connection, a "rip-up-and-reroute" phase is entered in which some of the connections already laid out are removed and routed again along different paths. The performance of such routers is highly dependent on the order in which the connections are processed, which routes are "ripped-up", and other such heuristic factors. Such routers will either take exponential time to find a difficult solution or, if rip-ups are limited, there is no guarantee of finding a good solution.

Hierarchical routers employ a divide-and-conquer strategy in which the array is recursively partitioned into smaller sub-arrays; the smaller sub-problems are solved independently, followed by a "merge" step in which the routings for the sub-problems are joined at the interface. The problems here arise mainly from the divide and the conquer phases — there is no obvious way of deciding the best partition or the best sequence for merging the sub-problems. Hu and Shing's hierarchical LP router [3] offers no performance guarantee. The hierarchical routers of BURSTEIN [1] and of KARP et al. [5] do offer performance guarantees, but these guarantees are far too pessimistic to be of practical value. The guarantee offered by our method is considerably stronger.

Simulated annealing [14] is a recent development in the field, and has stochastic guarantees of finding an optimal solution. There is, however, no useful bound on the time taken to find the optimum.

We present here a completely novel approach to the routing problem. Our method consists of casting the problem as an integer linear program. An immediate advantage of such a formulation is that the problem instance is viewed in its entirety, and does not depend on any sequential ordering of the connections. Furthermore, we have been able to develop an efficient procedure for approximately solving such integer programs. The approximation produced by our method is provably good in that it finds a solution guaranteed to be close to the best possible. Finally, our procedure runs in polynomial time. There is no "rip-up-and-reroute" phase where some nets are removed and routed again. Our method permits us to control the routes chosen by the algorithm, so that we can avoid routes that are undesirable from a practical standpoint (routes that are unduly circuitous or that have many bends).

The theoretical bases for the algorithm and the performance guarantees are developed elsewhere [11, 12, 13]. For the sake of completeness, we include a brief but selfcontained description of the algorithm in Section 2; proofs are omitted. In Section 3 we present an implementation of our method, followed in Section 4 by experimental results on two gate-arrays from industrial sources. These experiments are extremely encouraging from several standpoints, as we show in Section 4. Our method could route both arrays successfully, with surprisingly short runtimes.

2. THE LINEAR PROGRAMMING ALGORITHM FOR GLOBAL ROUTING

In this section we provide an outline of our algorithm for global routing in gate-arrays. We begin by defining the terminology we will use for the remainder of the paper. Our description of gate-arrays below concentrates on their topological rather than functional aspects.

2.1. The routing model

A *gate-array* is a two-dimensional rectangular array of rectangular *gates*. Each gate has a *left-channel* and a *bottom-channel*, corresponding to its left and bottom boundaries. These channels are used for running wires to realize electrical interconnections between the gates in the array. A channel contains a number of *tracks*, where each track can accommodate one wire. The number of tracks in a channel is called its *capacity*, for obvious reasons. Each gate also has a number of *terminals* to which wires are to be connected.

An instance of the *global routing problem* is specified by a list of *nets*, where a net is a collection of gates to be connected electrically by means of wires. Throughout this paper, we will be dealing with *two-point decompositions* of nets; i.e. we decompose each net into several pairs of gates to be connected (there is one exception to this — when there are four gates at the corners of a rectangle that are to be connected, we treat them as a *box* connection rather than decompose them into three two-point connections; more on this in Section 3). We call a two-point pair or box of gates a connection; when all the connections belonging to a net are realized, the interconnection pattern for that net is realized. The manner in which we decompose nets into connections is described in Section 3.

A route is a "Manhattan" path between the gates to be connected; it may traverse several other gates on the way, passing through their channels (a Manhattan path is one whose segments are all parallel to the sides of the gates). The number of routes passing through any channel may not exceed its capacity. A solution to the global routing problem consists of specifying a route for each connection, such that no channel-capacity is violated.

2.2. Formulation as an integer linear program

We now describe our formulation of the global routing problem as a 0-1 linear program (LP). Assume that we have decomposed the nets in a problem instance into a set of two-point connections (using guidelines similar to the ones in the next section). For each connection i in the problem, we pick a set of allowable routes. We wish to realize the connection by means of one of these routes. The choice of this set of allowable routes affects both the quality of the routing and the running time of the router. In practice, it is undesirable to route a connection by means of an excessively circuitous route as this degrades circuit performance. Also, in many fabrication technologies, routes with a large number of bends decrease the reliability and performance of the circuit. On the other hand, complicated routes may sometimes be the only way of circumventing "bottleneck" regions on a gate-array (those with several congested channels). Our experiments described in Section 4 discuss several choices of allowable routes for a connection, and the resulting routings.

We assign one indicator variable for each of the possible routes. (Indicator variables are integer variables that assume values of either 0 or 1). Thus, x_{ij} would represent the j-th route of connection i; $x_{ij} = 1$ would indicate route j was used to realize connection i and $x_{ij} = 0$ would indicate that it was not. Figure 1 gives a small example of such

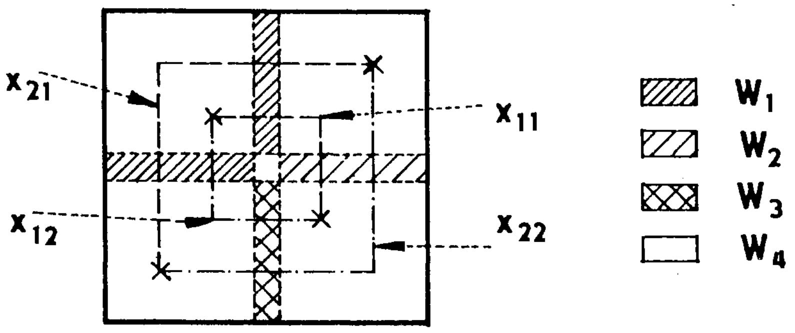


Fig. 1. A 2×2 example array with two 2-point nets.

indicator variables for two 2-point nets in a 2×2 gate-array. Net 1 connects the gate in the upper-left corner to the gate in the lower right corner, while net 2 connects th other two gates. Channel L_{22} denotes the left-channel of the gate at position 2,2. Channel B_{11} denotes the bottom-channel of the gate at position 1,1. Note that in this case, the channels on the bottom and left edges of the gate-array are not used. Each net has two allowed L-shaped routes in this example. To ensure that each connection gets realized with exactly one possible route, we write the following constraints (for our example):

$$x_{i1} + x_{i2} = 1, \qquad i = 1, 2; \quad x_{ij} = 0 \text{ or } 1. \tag{1}$$

Given the positions of the gates to be connected and the topology of the route corresponding to any variable, it is easy to specify constraints that ensure that the number of routes through a channel does not exceed its capacity. In our example, if the capacities of channels L_{21}, B_{12}, L_{22}, and B_{22} are respectively W_1, W_2, W_3, and W_4, then our LP becomes:

Minimize E subject to equations (1) and

$$\begin{aligned} x_{12} + x_{22} - W_1 &\le E \\ x_{12} + x_{21} - W_2 &\le E \\ x_{11} + x_{21} - W_3 &\le E \\ x_{11} + x_{22} - W_4 &\le E. \end{aligned} \tag{2}$$

Note that we have introduced a new variable E, which can be thought of as the excess over any channel's capacity. In other words, E is the maximum, over all channels, of the number of wires through a channel minus the capacity of that channel. The LP attempts to minimize E. In particular, if we can solve the above LP resulting in a value $E \leq 0$ (meaning no channel capacity is violated), we have a feasible solution to the routing problem. Notice that our formulation allows for E to have negative values, corresponding to routings where every channel has at least one unused track.

A number of algorithms are available for solving LP problems, but none of them can efficiently find solutions in which every variable x_{ij} has an integer value (0 or 1). (What we have here is actually an integer linear program, for which there is no known efficient algorithm). We therefore solve a *relaxation* of the above integer program — one in which each of the variables x_{ij} is allowed to take on any *fractional value* in the interval [0, 1]. Although not physically meaningful, a fractional route is not without significance. For instance, if in the example above $W_1 = W_2 = W_3 = W_4 = 1$, the optimal LP solution would be $E = 0$ (no excess over capacity) and $x_{11} = x_{12} = x_{21} = x_{22} = 0.5$. This may be thought of as routing one-half of each connection along one of the two possible routes, so that the "flux" through any channel does not exceed its capacity. This intuitively appealing concept also illustrates why the LP optimum value for E is a lower bound on the best integer solution. By restricting the variables to 0-1 values, we can do no better than the LP relaxation (a proof of this statement is straightforward). Also, if the fractional LP optimum contains the values $x_{i1} = 0.9$ and $x_{i2} = 0.1$, this suggests that connection i should probably be routed by route 1 rather than by route 2.

Since a fractional solution does not correspond to a physically meaningful routing, it is necessary to "round" the fractional values delivered by the LP to integer solutions corresponding to routings. While there is no general procedure for rounding the fractional values of a linear program to integer solutions, it is possible to use a technique which we call *randomized rounding* for this purpose. Before describing this technique, we note that although the LP solution is not guaranteed to be integral, several of the variables in a solution may be assigned integer values (thus, some of the connections may be assigned fixed routes in the LP solution). This does happen quite often in practice, as our experimental results in Section 4 will demonstrate. We thus have to round only those variables x_{ij} that are non-integral in the LP solution.

2.3. Randomized rounding

For each variable x_{ij}, let x_{ij} be the optimum value determined by the LP. Define random variables x_{ij}^{R} as follows:

$$\begin{aligned} &\text{Probability} \quad x_{ij}^{\mathrm{R}} = 1 = x_{ij} \\ &\text{Probability} \quad x_{ij}^{\mathrm{R}} = 0 = 1 - x_{ij}. \end{aligned} \tag{4}$$

We solve our routing problem by setting $x_{ij} = x_{ij}^{\mathrm{R}}$ independently for all i, and mutually exclusively for all j for any i. Thus exactly one of the allowed routes is chosen for each connection. The probabilistic choice is readily implemented using a random-number generator (see Section 3.3). This rule has the intuitively appealing property that if an LP variable has a fractional value close to 1, it is more likely to be set to 1. We have shown that the above rule produces results that are provably close to the *best possible* routing. We reproduce here the main theorem from [13].

Let X_i be the sum of the x_{jk} appearing in the ith line of equations (2). Thus X_i counts the number of wires passing through one channel. Let X_i and X_i^R be the analogous sums of LP-optimal x_{jk}^* and random variables x_{jk}^R, respectively. Note that if E^* is the optimum value of E, then $X_i - W_i \leq E^*$, for all i. Let C be the total number of channels in the array. Finally, let ε be any positive constant less than unity. The following theorem places an upper bound on the number of wires our algorithm will place in the ith channel.

Theorem 1. If $W_i \geq \ln(C/\varepsilon)$ for all i, $1 \leq i \leq C$, then randomized rounding results in a solution in which all

$$X_i \leq X_i^* + (e-1)\left(X_i^* \ln \frac{C}{\varepsilon}\right)^{\frac{1}{2}} \leq W_i + E^* + (e-1)\left(X_i^* \ln \frac{C}{\varepsilon}\right)^{\frac{1}{2}} \tag{5}$$

with probability at least $1 - \varepsilon$.

Proof sketch. The X_i are sums of independent Bernoulli trials. Applying Chernoff's bound to the right tail of such a distribution, we find that each X_i exceeds the bound (5) with probability at most ε/C.

A complete proof appears elsewhere [12, 13], along with some extensions.

The significance of the theorem is that not many more than X_i^* wires are likely to pass through channel i. We are thus guaranteed at least a 50-50 chance of finding a feasible routing ($E \leq 0$) if

$$E^* \leq -(e-1)(X_i \,\text{In} 2C)^{\frac{1}{2}}.$$

In practice, we do much better than this guarantee. We routinely find feasible routings with E^* at or near 0. See Section 4.

If randomized rounding fails to find a feasible routing, we can repeat the rounding procedure hundreds or even thousands of times. This is important, since solving the LP is the most computationally intensive portion of our method; in comparison, the time taken for a rounding is very small.

Of course, there is no guarantee that we will obtain a feasible solution from a given LP optimum, after any fixed number of roundings. When we are unable to obtain a feasible solution (as in example A of Table 3), we can add more routes to the LP in an effort to decrease E^*. Normally this should increase our chance of finding a feasible solution, and it certainly cannot increase E^*. It is important to note, however, that the chance of finding a feasible solution depends on both the value of E^* and on the number of integral x_{ij}^*. Thus adding too many routes to the LP can actually make it more difficult to find a feasible rounding, if one's LP solver produces more non-integral x_{ij}^* for a more degenerate problem (one with more routes). Example B3 of Tables 2 and 3 will illustrate this behaviour.

The reader may wonder, why not use a deterministic rounding procedure? Our answer is that we know of none that will do better than our randomized rounding procedure (however, a deterministic procedure can do as well as randomized rounding [12]). In particular, routing net i by choosing the largest x_{ij}^* is a poor idea. If each net can be routed in four different ways, the resulting integer solution can have $4X_i^*$ wires running through channel i.

3. DETAILS OF ALGORITHM IMPLEMENTATION

The implementation of the algorithm can be broken down into three parts — (1) the decomposition of nets into connections, (2) the generation of the LP from the decomposition, and (3) the rounding of the LP solution.

3.1. Decomposition of nets

Each net in the gate-array is decomposed into connections using the following heuristics due to HANAN [2].

There are four different types of connections, dependent on the relative position of the gates to be connected. If the gates lie on the same row, they are connected with a *row* connection. Similarly, if they lie on the same column, they are connected with a *column* connection. If the gates do not share a common row or column, then they are connected by a *two-bend* connection. In the special event that there are four gates on the corners of a rectangle to be connected, a *box* connection is used.

3.1.1. Decomposition heuristics

The decomposition phase reduces all nets to four types of connections (see Fig. 2).

Nets with two terminals are trivially decomposed, since a two-point net is either a row, a column, or a two-bend connection.

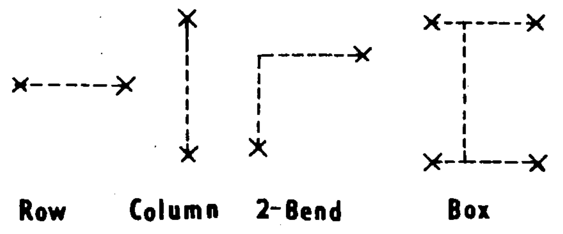


Fig. 2. Types of connections.

Nets with three terminals are differentiated into two separate cases (see Fig. 3). The *median-point* is the point whose coordinates are the median values of the three row and three column coordinates. If a terminal exists at the median-point, then the median-point is routed to the other two terminals by two-bend connections. Otherwise, a Steiner point is introduced at the median-point and the three terminals are connected to the Steiner point by two-point connections.

Four terminal nets are decomposed by considering the 4-corner rectangles (see Fig. 4).

Each corner rectangle (labelled C) is decomposed by connecting all terminals in it to the nearest corner of the inner rectangle (labelled I). The inner rectangle is then decomposed as either a two-bend or a box connection.

Nets with five or more terminals are decomposed by finding a minimum spanning tree [7]. Each edge of the tree connects two terminals and is a row, column, or two-bend connection.

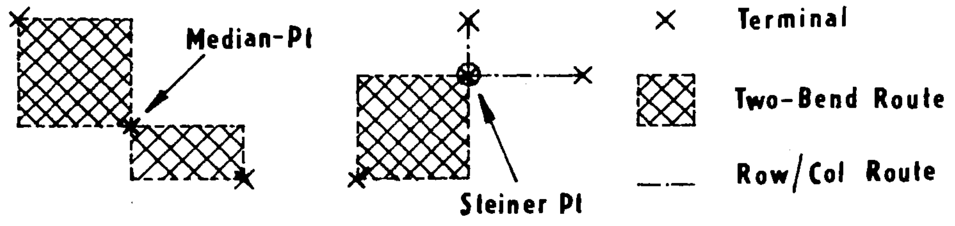


Fig. 3. Decomposition of three-point nets.

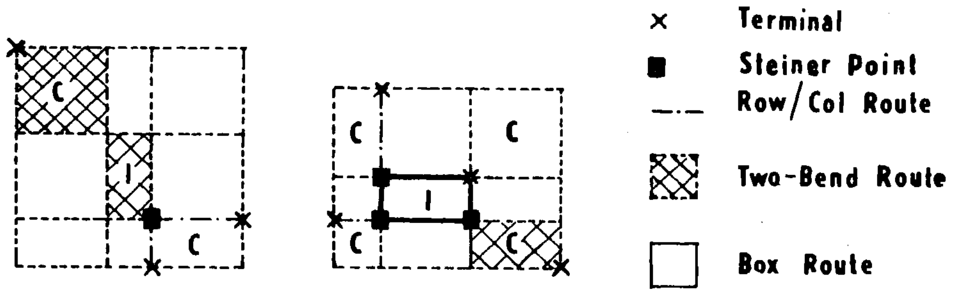


Fig. 4. Decomposition of four-point nets.

There is no restriction on the locations of the terminals of a net; in particular, terminals can lie on the same row or column. Two cases should be mentioned. The first is when a supposed two-bend routing is actually a row or column route. It is handled by converting all such two-bend routes into the corresponding row or column routes. The second is when the two terminals to be routed are coincident, in which case no routing is emitted.

3.2. Linear program generation

The LP is generated from the connection list produced by decomposing the nets. Each connection is realized in a number of different physical routing configurations. We introduce a SPAN parameter which governs the size of the LP. A large value of SPAN produces a large number of configurations for row and column connections. These connections can be displaced on either side from the minimum distance connection by up to SPAN channels, yielding 2(SPAN) + 1 configurations for each row or column connection.

Two-bend connections are realized by considering all the possible minimum distance two-bend routings. Box connections are realized by joining adjacent corners of the rectangle with a pair of parallel routes. A third route then joins the first two (see Fig. 5).

3.3. Rounding of LP solution

The LP returns fractional solutions for the configurations of each connection. Since the fractional solutions for each connection sum to 1, they can be viewed as intervals

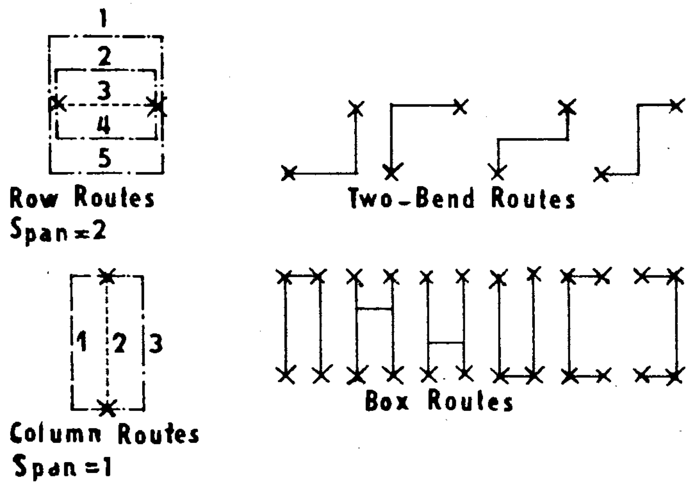
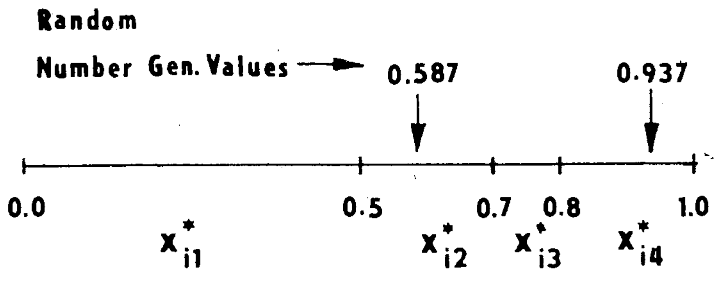


Fig. 5. Configurations for various connections.



Variable	Value
X^*_{i1}	0.5
X^*_{i2}	0.2
X^*_{i3}	0.1
X^*_{i4}	0.2

Fig. 6. Randomized rounding of net *i*.

along the [0, 1] number line. The relative positions of the intervals are not important. Independently, for each connection, we generate a random number in [0, 1] and determine which interval it lies in. The configuration corresponding to this interval is then chosen to realize the connection. For instance, if the pseudo-random number generator returned 0.587, configuration 2 would be selected ($x_{i2} = 1$ and $x_{i1} = x_{i2} = x_{i4} = 0$). If 0.937 were returned instead, configuration 4 would be selected (see Fig. 6). The gate array is routed when all the connections have been rounded and realized.

3.4. Problems arising from net decomposition

Two problems arise from the decomposition style of routing. Both problems occur because it is not possible to specify within the context of the LP which net the connections belong to.

When two connections pass through the same channel, they occupy two tracks. If they belong to the same net, then the physical route will only need to occupy one track. This phenomenon is referred to as *track sharing* (see Fig. 7). Since the LP attempts to minimize the excess E, if track sharing occurs in a channel that is "tight" (i.e. a channel with $X_i = W_i + E$), the counting of the extra connection will cause E^* to assume a higher value. There may thus be channels in the gate array that seem crowded only because of track sharing. In reality, these may be easily realized without exceeding the channel capacities.

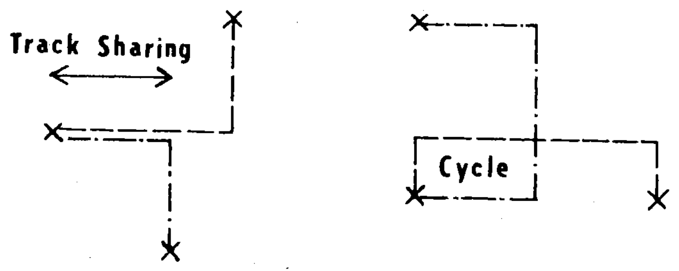


Fig. 7. Track sharing and cycles.

A *cycle* occurs when, in a physical realization of a net, two of its routed connections overlap (see Fig. 7). Again this means counting unnecessary routes which by the same argument will cause the excess E to be above the actual value.

3.5. Coding

A collection of programs for decomposition, LP generation, and rounding were written in the C programming language and run on a VAX 11/785 running Berkeley 4.3 BSD UNIX. Processing time on the VAX is short enough to allow interactive usage of the various modules. In particular, once the LP is solved, one rounding operation on all the nets can be solved in about 1 VAX CPU second. The LP is run using MPSX [9], an implementation of the simplex algorithm for linear programming, on an IBM 3081 running IBM/CMS.

4. EXPERIMENTAL RESULTS

This section describes the performance of the LP router on two gate-arrays from industrial sources; we call them example A and example B. Table 1a gives basic data on the two examples, such as array size, number of nets and statistics on the net sizes. Example B has over twice as many gates as example A. While B does not have twice as many nets as A, it contains several very large nets (including a 21-point net) and thus a larger average net size. Larger nets lead to more 2-point connections, non-minimal distance routings, and problems with track-sharing and cycles (see Section 3.4). Table 1b contains information about the examples after their nets have been decomposed into

Table 1a. Input data for the two examples.

Example	Size (gates)	Number of nets	No. of 2-pt. nets	No. of 3-pt. nets	No. of 4-pt. nets	No. of ⩾ 5-pt. nets	No. pts. largest net	Average no. of pts. in a net
A	15 × 12	285	111	152	19	3	5	2.6
B	17 × 23	449	257	88	29	75	21	3.64

Table 1b. Decomposition statistics.

Example	No. of connections	No. of row connections	No. of column connections	No. of 2-bend connections	No. of box connections
A	506	154	238	109	5
B	1266	481	440	341	4

2-point connections as described in the previous section. Notice that although B did not have twice as many nets as A, it has over twice as many connections — this is, as we noted above, due to the more complex nets in B.

For example A, four LPs were generated for four values of the variable SPAN — corresponding to varying extents of freedom for the row and column routes. Three values of SPAN were tried with example B — these sufficed to give us satisfactory results. Table 2 gives information on the LP — number of constraints (rows), variables (columns), the number of simplex iterations to feasibility and optimality, the IBM 3081 runtime in seconds and the optimal value of the objective function E^*.

The last column of Table 2 shows a surprising phenomenon; it was found that many of the variables in the LP solution had already been assigned integer values (0 or 1), and thus did not have to be rounded. Integral-valued variables correspond to routed connections, and the last column in Table 2 indicates the percentage of such connections in each

Table 2. LP statistics for four cases for each example.

LP solution	SPAN	Number of LP rows	Number of LP columns	Iterations to feasibility	Iterations to optimality	Runtime (sec.)	Optimal value of E*	% of integer solutions
A1	1	840	1774	497	514	25.10	0.0	77 %
A2	2	840	2435	402	474	32.93	0.17	75 %
A3	3	840	3050	0	145	15.63	0.17	92 %
A4	4	840	3600	0	146	19.74	0.17	92 %
B1	1	2009	5570	126	203	38.23	0.00	100 %
B2	2	2009	7270	133	187	42.98	0.00	100 %
B3	3	2009	8875	135	179	47.05	0.00	99 %

Table 3. Results of 51 roundings of LP solutions.

Example	No. of violations		Frequency of best solution
	average	best	
A1	27.61	19	2 %
A2	22.92	14	4 %
A3	2.73	0	4 %
A4	3.53	0	8 %
B1	0	0	100 %
B2	0	0	100 %
B3	0.76	0	38 %

of the cases. The most interesting result here is that in example B, for the cases B1 and B2, all 1266 connections were routed deterministically (100% integer solutions). Furthermore, this was with an objective function $E^* = 0$; in other words, the LP had found a perfect routing! For example B, increasing SPAN to 3 produced no reduction in the objective function, although some of the variables now took on fractional solutions.

The LPs for example A did not directly yield a routing with all the solutions integral, so that randomized rounding was necessary. However, E^* did attain negative values for A2, A3 and A4. Interestingly, the versions with SPAN = 3 and 4 (more freedom allowed) reached the optimum with fewer simplex iterations and thus decreased runtime. Thus, increased flexibility in the routes means an increase in the size of the LP but not necessarily an increase in runtime. Also, the versions A3 and A4 had sufficient freedom that MPSX [9] found a feasible basis without any iterations.

Where necessary, the fractional variables from the LP were converted to integer (0-1) solutions using randomized rounding, to produce physically meaningful routings. For each of the cases A1, A2, A3, A4 and B3, fifty-one independent randomized roundings were performed (no rounding was necessary for B1 and B2 since the LP solution in these cases was perfectly integral). Table 3 summarizes the results of rounding.

In each case, we list the number of channel capacities violated by the rounded solution (the routing); a value of zero corresponds to a feasible routing to the problem. Note that feasible solutions were found in the cases A3, A4, and B1 through B3 (the column "Best" under "No. of violations" gives the minimum number of channel-capacity violations among the 51 roundings). Notice also that the trend for the cases A1 through A4 under "average number of violated channels" shows that as the freedom (SPAN) is increased, we proceed closer to a feasible routing. This is also confirmed by the frequency with which the best solution occurs — A3 produced a feasible routing only 2 times out of 51 roundings (4%), while A4 yielded a routing 4 times out of 51 (8%). Our results suggest that SPAN = 3 is necessary to successfully route example A (SPAN = 2 could not even come close to a routing). Notice also that B3, with more freedom than B1 and B2, gave a routing only 19 times out of 51 (due to its fractional routes that required rounding — recall that B1 and B2 had perfectly integral LP solutions).

5. CONCLUSIONS AND DIRECTIONS FOR FURTHER WORK

The algorithm that we have presented differs from classical ones in several respects. The nets in a problem instance are not processed sequentially as in most other routers. Instead the problem is viewed in its entirety. Therefore, unlike sequential routers, there is no need to backtrack ("rip up and re-route"). Heuristics involving backtrack search rarely yield performance guarantees — our algorithm, on the other hand, is provably good in light of Theorem 1. Moreover, our experience to date is encouraging — we have successfully routed two industrial examples. We are currently investigating several other data sets.

Perhaps the most remarkable results of our experiments are the runtimes; it takes less than one minute to solve the linear programs for each of the problem instances. Since linear program optimization is the most computationally intensive part of our algorithm, our technique compares very favourably with other routers.

Although we have been using the simplex method for linear programming in our experiments so far, it would be interesting to apply recent developments in the field. A polynomial-time algorithm for linear programming [4, 6] could be used to ensure that our algorithm runs in polynomial time. This is in marked contrast to classical maze-style routers which may take exponential time.

The nature of our algorithm allows us to restrict the choice of routes. This permits us to control the quality of the routing — in particular we can avoid routes that are long or have many bends. This capability is not easily achievable in other routers.

Our solution to the routing problem does not make allowances for subsequent detailed routing. In the event that a detailed router cannot complete the routing of a channel, the rounding process can be rerun. Failing this, the entire global routing can be redone with larger numbers of choices of routes for each connection.

The linear programming technique does not restrict us to 2-point decompositions of nets. We could specify a set of possible routings for an entire net without decomposing it. Each such routing would be a Steiner tree spanning the terminals in the net. We are currently developing a language that generates Steiner tree configurations in a systematic manner [10]. We are now implementing this style of router.

REFERENCES

[1] Burstein, M.—Pelavin, R.: Hierarchical wire routing. IEEE Transactions on Computer--Aided Design, Vol. CAD-2, October 1983. No. 4, pp. 223—234.

[2] Hanan, M.: On Steiner's problem with rectilinear distance. SIAM Journal of Applied Math., Vol. 14, 1966, No. 3, pp. 255—265.

[3] Hu, T. C.—Shing, M. T.: A decomposition algorithm for circuit routing. Mathematical Programming Study, Vol. 24, 1985, pp. 87—103.

[4] Karmarkar, N.: A new polynomial-time algorithm for linear programming. Proceedings of the Sixteenth ACM Symposium on Theory of Computing, ACM, New York 1984.

[5] Karp, R. M.—Leighton, F. T.—Rivest, R. L.—Thompson, C. D.—Vazirani, U.—Vazira-

NI, V.: Global wire routing in two-dimensional arrays. Proc. 24th Annual Symp. on Foundations of Computer Science, October 1983, pp. 453—459.

[6] KHACHIAN, L. G.: A polynomial algorithm for linear programming. Soviet Math. Doklady, Vol. 20, 1979, pp. 191—194.

[7] KRUSKAL, J. B.: On the shortest spanning subtree of a graph. Proc. Amer. Math. Soc., Vol. 7, 1956, pp. 48—50.

[8] LEE, C. Y.: An algorithm for path connections and its applications. IRS Transactions on Electronic Computers, Vol. EC-10, 1961, pp. 346—365.

[9] International Business Machines, IBM Mathematical Programming System Extended/370 Program Reference Manual, White Plains, New York, Dec. 1979.

[10] P-C NG, A.—RAGHAVAN, P.—THOMPSON, C. D.: A language for describing rectilinear Steiner tree configurations. Proc. 1986 ACM Design Automation Conference, July, 1986, pp. 659—662.

[11] RAGHAVAN, P.—THOMPSON, D.: Provably good routing in graphs: regular arrays. Proceedings of the Seventeenth ACM Symposium on Theory of Computing, ACM, New York, May 1985.

[12] RAGHAVAN, P.: Randomized rounding and discrete ham-sandwich theorems: provably good algorithms for routing and packing problems. Ph. D. Dissertation, Computer Science Division, U. C. Berkeley, August 1986.

[13] RAGHAVAN, P.—THOMPSON, C.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. To appear, Combinatorica, 1986.

[14] VECCHI, M. P.—KIRKPATRICK, S.: Global wiring by simulated annealing. IEEE Transactions and Computer-Aided Design, Vol. CAD-2, October 1983, No. 4, pp. 215—222.

Received April 22, 1986 — revised version December 19, 1986