Minimization of the Number of Layers for Single Row Routing with Fixed Street Capacity

TEOFILO F. GONZALEZ AND SHASHISHEKHAR KURKI-GOWDARA

Abstract—**We present a set of algorithms to solve single row routing problems with a fixed street capacity using the least number of layers. The main difference between these algorithms is in the strategy used to search for an optimal solution. Varying this strategy greatly affects the performance of our algorithms. At the extreme points of our strategy we have algorithms Q and S. The worst case time complexity for algorithm Q is linear and the one for algorithm S is exponential. The best case time complexity of all our algorithms is linear. The main disadvantage of algorithm Q is that the constant associated with its time complexity bounds is large. On the other hand, the constant associated with the best case time complexity bound for algorithm S is small. We also present an experimental evaluation of the performance of our algorithms.**

Keywords—**MPCB routing, layering problem, dynamic programming, algorithms, minimum number of layer.**

I. Introduction

LET $V = \{v_1, v_2, \cdots, v_n\}$ be a set of nodes (points) lying on a straight line (row) with node v_i located to the left of node v_{i+1}, for $1 \leq i < n$. Let $N = \{N_1, N_2, \cdots N_m\}$ be a partition of V. Each set N_i is called a net. The *single row single layer routing problem* consists of introducing a set of wires to make electrically common all the nodes in each net. The wires must follow a path consisting of horizontal and veritical line segments. All the horizontal wires must run along one of the k upper horizontal tracks or the k lower horizontal tracks, and the vertical wires must run along any of the k vertical tracks that appear between every pair of adjacent nodes. Of course wires that carry different signal nets cannot intersect. A feasible solution for the single row single layer problem does not allow *backing*, i.e., any vertical line cannot cross two or more wires that carry the same signal net. *Switching* is allowed, i.e., a wire carrying some signal net assigned to an upper horizontal track can be joined to a wire assigned to a vertical track which can be connected to a wire assigned to a lower horizontal track. The reverse operation is also allowed. There is no limit on the number of such switchings.

Net N_j is said to *include* node v_i if the leftmost node in N_j is not located to the right of node v_i and the rightmost node in N_j is not located to the left of node v_i. The *density of node* v_i, denoted by d_i, is the number of nets that include node v_i. The *density*, d, *for a problem instance* is defined as $\max \{d_i\}$. In certain problem instances, the number of horizontal tracks are not enough to route all the nets. An obvious case is when d is greater than $2k$. Necessary and sufficient conditions for a routing to exist were introduced in [5]. Efficient algorithms to construct a routing when such a routing exists were introduced in [5], [6] and [4]. The algorithms in [4] seem to be the fastest for all values of k.

Problem instances that cannot be routed in one layer, can be routed by introducing additional layers. A direct generalization of the single row single layer routing problem is the *multilayer single row routing problem*, which is also known as the *layering problem*. This problem is defined as the single row single layer routing problem, except that we are required to find a minimum layer routing. In this problem, a net may be routed in several layers, however, no vertical line segment can cross (in all layers) two or more wires carrying the same signal net and the only place where a wire can be switched to another layer is at a node. Hueristic algorithms for the layering problem were studied in [3] and [9]. When $k = 2$, the algorithm given in [9] generates solutions with a number of layers, f, which is not more than $1.33 f^*$, where f^* is the number of layers in an optimal solution. In practice, k is never greater than two. That is why we restrict our attention to the case when $k = 2$. As pointed out in [9] when $k = 2$, the multipoint routing problem can be reduced to a two-point layering problem by increasing the number of nodes. The resulting problem will never have more than twice the number of nodes in the original problem. Therefore, we restrict our attention to the solution of the layering problem when $k = 2$ and all nets are two-point nets. In Section IV we show how to modify our algorithms to deal with the case when $k > 2$. A *single row single layer routing problem* which can be routed using 2 upper and 2 lower tracks is called a *(2, 2)-single row single layer routing problem*. Because of our restriction on k we can define the *layering problem* as the problem of partitioning a single row routing problem into the least number of *(2, 2)-single row single layer routing problems*.

The layering problem is a fundamental problem in MPCB routing. The classical approach, introduced by [8], to solve MPCB routing problems consists of introducing the least number of via columns and assigning vias to the

Manuscript received October 10, 1986; revised September 18, 1987. This work was supported in part by the National Science Foundation under Grant DCR-8503163. A preliminary version of this paper was presented at the IEEE Physical Design Conference, Houston, TX, March 1986. The review of this paper was arranged by Associate Editor R. H. J. M. Otten.

The authors are with the Department of Computer Science, University of California, Santa Barbara, CA 93106.

IEEE Log Number 8718403.

different nets (via assignment problem) in such a way that a solution to the original problem can be obtained by solving a set of single row routing problems. An approximation algorithm for the via assignment problem and references to previous work for this problem appear in [1]. As we mentioned before, single row single layer routing problems can be solved efficiently. In this paper we present algorithms for the layering problem.

In Section II we present our algorithms for the *layering problem*, and in Section III, we evaluate their performance. We discuss extensions of our results in Section IV.

II. The Algorithms

In this section we present our algorithms for the layering problem. Instead of dealing directly with the minimization problem, it is convenient to begin by solving a *decision problem* closely related to the minimization problem. We call this problem the *l-layering problem*. This problem is identical to the layering problem with the exception that we are not required to partition the set of nets into the least number of (2, 2)-single row single layer routing problems, it is only required to determine whether or not it is possible to partition it into l (2, 2)-single row single layer routing problems. Later on we show how to solve the original problem by solving a set of decision problems closely related to the l-layering problem.

Our procedure to solve the l-layering problem proceeds by scanning the set of vertices from left to right. When considering node v_i, we have a set of feasible solutions from node v_1 to node v_{i-1} (in set S_{i-1}) and we construct the set of feasible solutions from node v_1 to node v_i from S_{i-1}. When considering node v_i we say that *net* N_j *is active* if net N_j includes node v_i. The left-most node in a net is called a *begin node*, and the rightmost node is called the *end node*. Instead of keeping feasible solutions separately, we shall combine and represent as a group sets of these feasible solutions. This is important since it greatly decreases both the time and space required by our algorithms. Before introducing our algorithms, it is important to explain our internal representation for sets of feasible solutions.

Since at this point we are only interested in determining whether or not there exists a partition of the set of nets into l (2, 2)-single row single layer routing problems, when representing a feasible solution we shall only keep the layer assignment for active nets, i.e., we do not keep track of the layer assignment of nets whose end nodes appear to the left of v_i. Omitting this information decreases both the time and space required to solve the l-layering problem. Note that two feasible solutions with different past assignments but the same assignment of active nets can be merged into one feasible solution. This is because the solution of the remaining subproblem depends only on the current state. When representing a feasible solution we shall not specify track assignments, we only care about the relative ordering of the nets (wires) at each layer. Note that once we have the relative ordering, any feasible solution with track assignment consistent with this ordering can be easily obtained. Note that this simplification decreases the time and space required by our algorithms. Instead of representing each feasible solution by an l-tuple that defines the ordering of the active nets at each level, we shall represent a set of feasible solutions by an l-tuple. A set of feasible solutions is represented by the l-tuple, $(SL_1, SL_2, \cdots, SL_l)$, where each SL_i represents a set of orderings in layer i. The l-tuple $(SL_1, SL_2, \cdots, SL_l)$ represents all feasible solutions whose ordering in layer 1 is one of the orderings in SL_1, the ordering in layer 2 is one of the orderings in SL_2, and so forth. When we refer to a set of orderings we use the following coding scheme. An asterisk "*" in a tuple, means that any of the nets not yet assigned can occupy this position, and any other symbol at the i'th position indicates that the i'th element in the ordering is the net represented by that symbol. For example, if the active nets in a layer are nets a and b, and a set of orderings is given by (*, *), then the valid orderings are (a, b) and (b, a). Our notation is similar to the one in [4]. The reason for this is that it allows us to solve the single row single layer routing problem on each of the layers quickly, and it enables us to group together sets of feasible solutions. The state of a layer is given by any of the following symbols.

$\varnothing$	No net is active in the layer.
$\{a\}$	Net a is active in the layer and the set of orderings is given by the expression (*).
$\{a, b\}$	Nets a and b are active in the layer and the set of orderings is given by the expression (*, *).
$\{a, b, c\}$	Nets a, b, and c are active in the layer and the set of orderings is given by the expression (*, *, *).
$\langle a, b, c \rangle$	Nets a, b, and c are active in the layer, and the ordering is given by either (a, b, c) or (c, b, a).
$[a, b, c, d]$	Nets a, b, c, and d are active in the layer, and the ordering of the nets is given by either (*, *, c, *) or (*, c, *, *). This is because net c was the last net that became active in this layer.
$\langle a, b, c, d \rangle$	Nets a, b, c, and d are active in this layer, and the ordering is given by either (*, b, c, *) or (*, c, b, *).

Let us now explain the above notation as well as the transition function for these states that appears in Fig. 1. At this point we shall concentrate only on the set of nets assigned to layer h. We traverse the set of nodes from left to right. Let us now consider node $v_i \in N_j$. Let $SL_{h,i-1}$ represent the state of layer h at this point. The next event is either finding out that node v_i belongs to a net which has not been assigned to layer h or that v_i belongs to a net which is assigned to layer h. In the former case $SL_{h,i}$ is just $SL_{h,i-1}$, but in the latter case computing $SL_{h,i}$ is more

complex. Let us now concentrate on this event. Node v_i could be either a begin or an end node of a net assigned to this layer. Clearly, if $SL_{h,i-1}$ contains four nets, it is impossible for the next event to be the begin node of a net assigned to this layer as otherwise we would have five active nets. If $SL_{h,i-1}$ contains zero nets the next event cannot be the end node of an active net in this layer. Let us now consider all possible states $SL_{h,i-1}$. There are seven cases:

Case 1: $SL_{h,i-1} = \varnothing$.

Clearly, the next event must be finding the begin node for net N_j. In this case the set of valid orderings is given by the expression (*). Therefore the next state is $SL_{h,i} = \{N_j\}$.

Case 2: $SL_{h,i-1} = \{a\}$.

If the next event is finding the end node for net a, then $SL_{h,i} = \varnothing$. On the other hand if the next event is finding the begin node for net N_j, then the set of valid orderings is given by the expression (*, *). Therefore, $SL_{h,i} = \{a, N_j\}$.

Case 3: $SL_{h,i-1} = \{a, b\}$.

If the next event is finding the end node for net b (a similar case would arise if net a is the one to end first), then $SL_{h,i} = \{a\}$. On the other hand if the next event is finding the begin node for net N_j, then the set of valid orderings is given by the expression (*, *, *). Therefore, $SL_{h,i} = \{a, N_j, b\}$.

Case 4: $SL_{h,i-1} = \{a, b, c\}$.

If the next event is finding the end node for net c (a similar case would arise if net a or net b is the one to end first), then $SL_{h,i} = \{a, b\}$. On the other hand, if the next event is finding the begin node for net N_j, determining the next state is more complex. The reason for this is that we look ahead to check all the possible states that would arise from this state. If the next event is finding the end node of N_j, then the set of orderings is given by either the expression (*, N_j, *, *) or (*, *, N_j, *) and therefore the next state $SL_{h,i}$ is $[a, b, N_j, c]$. When we find the end node of N_j, the set of orderings is given by (*, *, *) (the resulting state will be $\{a, b, c\}$). However, if the next event is finding the end node of net b (a similar situation would arise if net a or net c is the next net to end), the set of orderings is given by either the expression (*, b, N_j, *) or (*, N_j, b, *) and $SL_{h,i}$ is $\langle a, b, N_j, c \rangle$. When we find the end node of net b, the set of valid orderings is given by the expression (*, N_j, *) and the resulting state will be $\langle a, N_j, c \rangle$.

Case 5: $SL_{h,i-1} = [a, b, c, d]$.

Discussion of this case is omitted since it is included in case 4.

Case 6: $SL_{h,i-1} = \langle a, b, c, d \rangle$.

Discussion of this case is omitted since it is included in Case 4.

Case 7: $SL_{h,i-1} = \langle a, b, c \rangle$.

If the next event is finding the end node for net c (a similar case would arise if net a or net b is the one to end first), then $SL_{h,i} = \{a, b\}$. On the other hand if the next event is finding the begin node for net N_j, determining the next state is more complex. The reason for this is that we shall look ahead to check if this event would produce feasible solutions. If it does not generate even one feasible solution, we shall eliminate this state from further consideration. If the next event is finding the end node of net N_j, or net b (the net in the center), then we know that the set of orderings is given by either (*, b, N_j, *) or (*, N_j, b, *) and, therefore $SL_{h,i} = \langle a, b, N_j, c \rangle$. However, if this is not the case (the next net to finish is either net a or net c), it is simple to see that there exists no feasible routing for the set of nets assigned to this layer. Therefore, we can eliminate this set of feasible solutions.

The transitions explained above are summarized Fig. 1. Now we are ready to present algorithm Q. Let S_i denote the set of feasible solutions when considering nodes 1 through node i. Clearly S_0 is $\varnothing$. Algorithm Q finds all possible states S_i from S_{i-1}. When computing S_i from S_{i-1}, if v_i is a begin node, we assign this begin node to each of the layers in each of the l-tuples following the transition function given in Fig. 1. If it is an end node, we delete such a net from all feasible states and merge identical states. It is simple to see from the way we defined our transition function that a termination of a net is always feasible because such a net, if in the company of three other nets, will always occupy one of the central tracks. We proceed computing these S's until we find a state $S_i = \varnothing$ and there are active nets in which case the solution to the decision problem is false, or until we reach S_n in which case the answer to the decision problem is true. The algorithm is given below.

```
algorithm Q
  S0 ← ∅
  for i = 1 to n do
    case
      :node vi is the begin node for net Nj:
          construct Si from Si-1 by assigning net Nj to
          each of the layers in each of the l-tuples in
          Si-1 and following the transitions given in
          Fig. 1.
      :node vi is the end node for net Nj:
          Si is computed from Si-1 by deleting net Nj
          from all the l-tuples in Si-1 and eliminating
          equivalent states.
    endcase
    if Si = ∅ and there are active nets then re-
    turn(false);
  endfor
  return(true);
end of algorithm Q;
```

The Tl-layering problem can be stated as follows: Given a single row routing problem, and a partition T of the leftmost c nets (c is any integer constant ≥ 0), is there a partition P into l equivalence classes such that each set of nets in each equivalence class is a (2, 2)-single row routing problem and $T \subseteq P$? We say that $T \subseteq P$ if deleting some elements from P generates partition T. This new de-

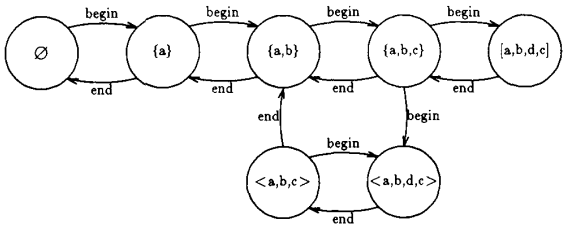


Fig. 1. Transition function.

cision problem can be solved by using an algorithm similar to algorithm Q. The main difference is that instead of starting with $S_0 = \varnothing$, the algorithm starts with $S_c = X$, where X is the set of feasible solutions obtained from partition T. We shall refer to this procedure as algorithm R.

Let us now explain algorithm $S(c)$, where c is a nonnegative integer constant. Algorithm $S(c)$ proceeds in a depth first search order for the first c levels (first c nodes) and for each set of feasible solutions it encounters at level c it solves the remaining subproblem by invoking algorithm R, i.e., it proceeds in a breath first manner from that point on. The procedure stops with an affirmative answer the first time algorithm R returns an affirmative answer. The procedure will stop with a negative answer if all the calls to algorithm R return a negative answer. Note that algorithm $S(0)$ is identical to algorithm Q. An important feature of algorithm $S(c)$ is that none of the identical solutions that appear in the first c levels will be merged together. Also, identical solutions at some level greater than c will be merged together only if they are generated during the same invocation of algorithm R. Later on we explain how these decisions affect the performance of the algorithm.

In practical situations, we are interested in finding a routing with the least number of layers. So far the only problem that we can solve are decisions problems related to the layering problem. Let us now show how an optimal routing can be obtained. Let us assume that we begin with a problem instance with density d. Clearly, a lower bound for the number of layers in an optimal solution is $LB = \lceil d/4 \rceil$. From the result in [5] we know that an upper bound for the number of layers in an optimal solution is $UB = \lceil d/3 \rceil$. To find the number of layers in an optimal solution, we can perform a binary search on the set of integers in the interval $[LB, UB]$. In each iteration we need to solve an l-layering problem. After $\lceil \log_2 (UB - LB + 1) \rceil$ iterations we will find the number of layers in an optimal solution. Let h be this number. Before finding an optimal routing we first must find an optimal partition. There are two methods for finding an optimal partition. The first method consists of modifying the data structure to keep track of all the nets assigned to each layer. Finding an optimal partition with this additional information becomes a trivial matter. Let us now consider the second method for finding an optimal partition. First of all we explain how this works with algorithm Q. To find an optimal partition we will invoke algorithm R several times. Let us assume that the T_{i-1} l-layering problem has a solution. We now show how to find a partition T_i such that the T_i l-layering problem has an answer yes. If node v_i is an end node, then from the transition function we know that for $T_i = T_{i-1}$ the T_i l-layering problem has a yes answer. On the other hand, if it is a begin node, this new net could be assigned to any of the l different layers. We try each of these assignments one at a time by executing algorithm R until one returns the answer yes (note that since the T_{i-1} l-layering problem has an answer yes, at least one of the assignments of the new net to the layers will produce a partition with a yes answer). The partition T_i is just T_{i-1} together with this last assignment. To obtain a net partition using the second method and running algorithm $S(c)$, we proceed exactly as when using algorithm Q after level c. Before level c, we can easily keep the net partition at each point. Note that in this case this additional information can be kept in c records.

III. Complexity Issues and Experimental Results

In most practical situations we are only interested in problems that can be solved in fewer than 16 layers. Therefore, when analyzing the complexity of our algorithms we assume that l is a fixed constant. As noted before, k is also a fixed constant.

The maximum number of different active net partitions at any given level is at most $(2kl)! = (4l)!$. Since all the orderings in each layer can be coded into a single tuple and there are l layers, the maximum number of states at any level is at most $(4l)!/24l$. Therefore, $|S_i| \leq (4l)!/24l$ in each iteration performed by algorithm Q. Since l is a constant, the (worst and best case) time complexity of algorithm Q is $O(n)$. Clearly, the constant associated with these time complexity bounds is large. The space requirements for this algorithm are also large. The same bounds also applies to algorithm $S(0)$ since it is identical to algorithm Q. However, algorithm $S(n)$ behaves quite differently. The best case time complexity arises when the first depth first path leads to an answer of yes. This is $O(n)$ with a very small constant. The only problem with this algorithm is the worst-case time complexity. This is $O(l^n)$ since we cannot cancel identical partial solutions and at each step a net can be assigned to the l different layers. With respect to the space bound, it is small since we need to keep at each level only a small set of feasible solutions if we generate the feasible solutions in a lexicographic order.

Note that the above analysis only considers the extreme cases of our strategy. Different values of c, generate different types of algorithms. As c increases the worst case time complexity bound increases, the space requirements decrease but the constant associated with the best case time complexity bound decreases.

Constructing the actual routing takes $O(n)$ time if we are using the first method (the one with large space requirements). If we use the second method then the worst-case time complexity for algorithm Q increases to $O(n^2)$.

For algorithm $S(n)$ the worst-case time complexity remains exponential.

An experimental evaluation of our algorithms was conducted by solving 50 randomly generated problems on a SUN-2 with 1 megabyte of memory, and on a VAX-750 with 4 megabytes of memory. A description of the problems generated is given in Table I.

For 4 out of the 50 problems, our algorithm was stopped after 100 hours of CPU time. However, for these problems we obtained a solution with no more than $f^* + 1$ layers, where f^* is the number of layers in an optimal solution.

The yes answers were generated by algorithm $S(c)$ with $c = 0.5\ n$ in a matter of few seconds to minutes of CPU time. Algorithm Q had to spend about 30 hours on a few problems to generate a no answer. It is simple to see that if the answer is yes to a problem instance than algorithm $S(c)$ should be used, whereas if the answer is no we are better off using algorithm Q. The performance of algorithm $S(c)$ greatly depends on the value of c. Let us now consider problems with a yes answer. When c/n is 0.65, the algorithm usually takes a few seconds. For $c/n = 0.40$ the algorithm takes about 10 min. However for c/n at about 0.20 the algorithm takes about 25 min. For many of these cases it would take a very long time for $S(0)$ to terminate. Problems with an answer yes usually can be solved quickly. Problems with an answer no usually take a long time.

IV. Discussion

We have shown that there is an "efficient" dynamic programming algorithm to solve the *single row routing problem*, for the case when there is a fixed number of layers and a fixed number of tracks per layer. The worst case time complexity for this algorithm is linear. Our algorithms proceed by solving a set of decision problems. Decision problems with a yes answer can be solved quickly, however, decision problems with a no answer are very hard to solve. Consequently, sequential search is better than binary search in our search for the minimum number of layers. From our experimentation, we conjecture it is always possible to generate solutions with f^* layers quickly.

The best way to solve the layering problem is by executing algorithm $S(c)$ for different values of c in parallel. Parallel computers with about 10 processing elements would be perfect for the job.

There are several heuristics used by our algorithms to speed up the solution process. One of these heuristics uses the results in [5] which assert that a solution can be easily obtained if the remaining problem has density less than or equal to 3 * the number of layers. Other speed ups for the algorithm involve the use of efficient data structures and fast sorting algorithms.

Our algorithms can be easily adapted to solve problems when $k > 2$. The main changes deal with the coding of feasible solutions. The time required to solve these problems is larger because the set of feasible partitions increases with k. Our algorithm can be easily adapted to handle the case when a subset of the nets have been assigned to specific layers. The time and space requirements of our algorithms decreases with these additional constraints.

TABLE I

Number of Problems Generated	Number of Nets	Density
10	20	12
10	30	16
10	40	22
10	50	29
10	60	33

References

[1] T. Gonzalez, "An approximation for the via assignment problem," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 257-264,

[2] T. Gonzalez and K. G. Shashishekhar, "An efficient algorithm to minimize the number of layers in single row routing problems with fixed street capacity," presented at 1986 Physical Design Conf., Houston, TX, Mar. 1986.

[3] S. Y. Han and S. Sahni, "Layering algorithms for single row routing," in *Proc. 22nd Design Automation Conf.*, pp. 516-522.

[4] S. Han, and S. Sahni, "Single row routing on narrow streets," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 235-241, July 1984.

[5] E. Kuh, T. Kashiwabara, and T. Fujisawa, "On optimal single-row routing," *IEEE Trans. Circuits Syst.*, vol. CAS-26, June 1979.

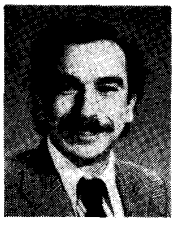
[6] R. Raghavan and S. Sahni, "Optimal single row router," *IEEE Trans. Computers*, vol. C-32, pp. 209-220, Mar. 1983.

[7] S. Sahni, A. Bhatt, and R. Raghavan, "The complexity of design automation problems," in *Proc. 19th Design Automation Conf.*, June 1981.

[8] H. So, "Some theoretical results on the routing of multilayer printed-wiring boards," *IEEE Symp. Circuits and Systems*, pp. 296-303, 1974.

[9] S. Tsukiyama, E. S. Kub, and I. Shirakawa, "On the layering problem on multilayer PWB wiring," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, pp. 30-38, Jan. 1983.

*

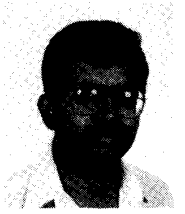


Teofilo F. Gonzalez was born in Monterrey, Mexico, in 1948. He received the B.S. degree in computer science from the Instituto Technologico de Monterrey and the Ph.D. degree in Computer Science from the University of Minnesota, Minneapolis, in 1972 and 1975, respectively.

He has been a member of the Computer Science Faculty at the University of Oklahoma, the Pennsylvania State University, the Instituto Tecnologico de Monterrey and the University of Texas at Dallas. He is now Professor of Computer Science at the University of California, Santa Barbara. His major research interests are in the design and analysis of algorithms, computational aspects of computer-aided design, and scheduling theory.

Dr. Gonzalez is a member of the Association for Computing Machinery, IEEE Computer Society, and the Operations Research Society of America.

*



Shashishekhar Kurki-Gowdara received the B.E. degree in electrical and electronics engineering from Bangalore University, India, in 1980, and the M.S. degree in computer science, from the University of California, Santa Barbara, in 1980 and 1982, respectively. He is currently studying for the Ph.D. degree in computer Science at the University of California, Santa Barbara.

His research interests include design and analysis of algorithms, VLSI routing and placement algorithms, and computability theory.

Mr. Kurki-Gowdara is a member of the ACM.