A "GREEDY" CHANNEL ROUTER *

by Ronald L. Rivest and Charles M. Fiduccia

MIT Laboratory for Computer Science, Cambridge, Mass. 02139, and
GE Research and Development Center, Schenectady, New York 12301

March 1981

Abstract

We present a new, "greedy", channel-router that is quick, simple, and highly effective. It *always succeeds*, usually using no more than one track more than required by channel density. (It may be forced in rare cases to make a few connections "off the end" of the channel, in order to succeed.) It assumes that all pins and wiring lie on a common grid, and that vertical wires are on one layer, horizontal on another.

The greedy router wires up the channel in a left-to-right, column-by-column manner, wiring each column completely before starting the next. Within each column the router tries to maximize the utility of the wiring produced, using simple, "greedy" heuristics. It may place a net on more than one track for a few columns, and "collapse" the net to a single track later on, using a vertical jog. It may also use a jog to move a net to a track closer to its pin in some future column. The router may occasionally add a new track to the channel, to avoid "getting stuck".

Introduction

Introduced in 1971 [Ha71], "channel routing" has become a very popular method of routing integrated circuits. (See [KSP73], [Hi74], [De76], [AKT76], [PDS77], [KK79], [Ri82].) Typically, the wiring area is first divided into disjoint rectangular "channels". A "global router" then determines which channels each net traverses. Finally a "channel router" computes a detailed routing for each channel. This approach is effective because it decomposes the overall problem into a number of simpler problems and simultaneously considers all nets traversing each channel.

The general channel-routing problem has been proven NP-Complete ([GJ79], [La80], [Sz81], [SB80]), although algorithms exist for highly-restricted cases ([DKSSU81], [LP81], [Pi81], [To80], [La80]). A slightly different wiring model permits one to come within a factor of 2 of channel density ([RBM81]). Useful methods also exist for computing lower bounds on channel widths ([BR81], [Le81]). These results highlight the need for good practical heuristics.

The algorithm presented here exploits a novel control structure: a left-to-right column-by-column scan of the channel, where the router completes the routing for one column before proceeding to the next. In each column the router acts in a "greedy" manner trying to maximize the utility of the wiring produced.

Our work is an extension of Alford's [Al80]; who also considered a left-to-right scan of the channel. His router did not guarantee success (because it did not allow nets to occupy more than one track in any column), ran quite slowly, and produced noticeably poorer results than our "greedy" algorithm.

Kawamoto and Kajitani [KK79] use a similar column-by-column approach, but not in left-to-right order. They also assume (as we do not) that between adjacent columns there is enough room to wire an arbitrary permutation.

The following paragraphs define what we mean by a "channel routing problem" and its solution.

A channel-routing problem is specified by giving:

(1) A "channel-length" λ. Most of the routing will lie within the channel whose "left end" is at $x = 0$, and "right end" is at $x = \lambda + 1$, on the vertical *columns* at x-coordinates $1, \ldots, \lambda$, although columns outside the channel may also be used.

(2) Top and bottom connection lists $T = (T_1, \ldots, T_\lambda)$ and $B = (B_1, \ldots, B_\lambda)$. T_i (resp. B_i) is the net number for the pin at the top (resp. bottom) of the i-th column (at $x = i$), or is 0 if no such pin exists.

(3) The left and right connection sets, L and R, specifying which nets must connect to the right and left ends of the channel. (They are *sets* since we assume that a net need connect at most once to an end of the channel, and that the relative ordering of such connections may be chosen by the channel router.)

A *solution* to a channel-routing problem specifies:

(1) The channel width w – the number of horizontal "tracks" used. These tracks are at y-coordinates $1, \ldots, w$. A channel router tries to minimize w.

(2) For each net n, a set of connected horizontal and vertical "wire segments" whose endpoints are grid points (x, y) with $1 \leq y \leq w$, except that segments with endpoints $(i, 0)$ or $(i, w + 1)$ must be included if $T_i = n$ or $B_i = n$. Endpoints with $x < 1$ or $x > w$ are legal but should be avoided. A net in L (resp. R) must have a segment touching the line $x = 0$ (resp. $x = \lambda + 1$). Two segments in the same direction are on the same layer, so they may not touch if they are for different nets. Two segments for the same net in different directions that touch at a grid point are said to be connected by a "contact" or "via" at that point. If the segments were for different nets we would have a "crossover".

* This research was supported by the General Electric Corporation, DARPA grant N00014-80-C0622, Air Force grant AFOSR-F49620-81-0054, and NSF grant MCS-8006938.

0420-0098/82/0000/0418$00.75

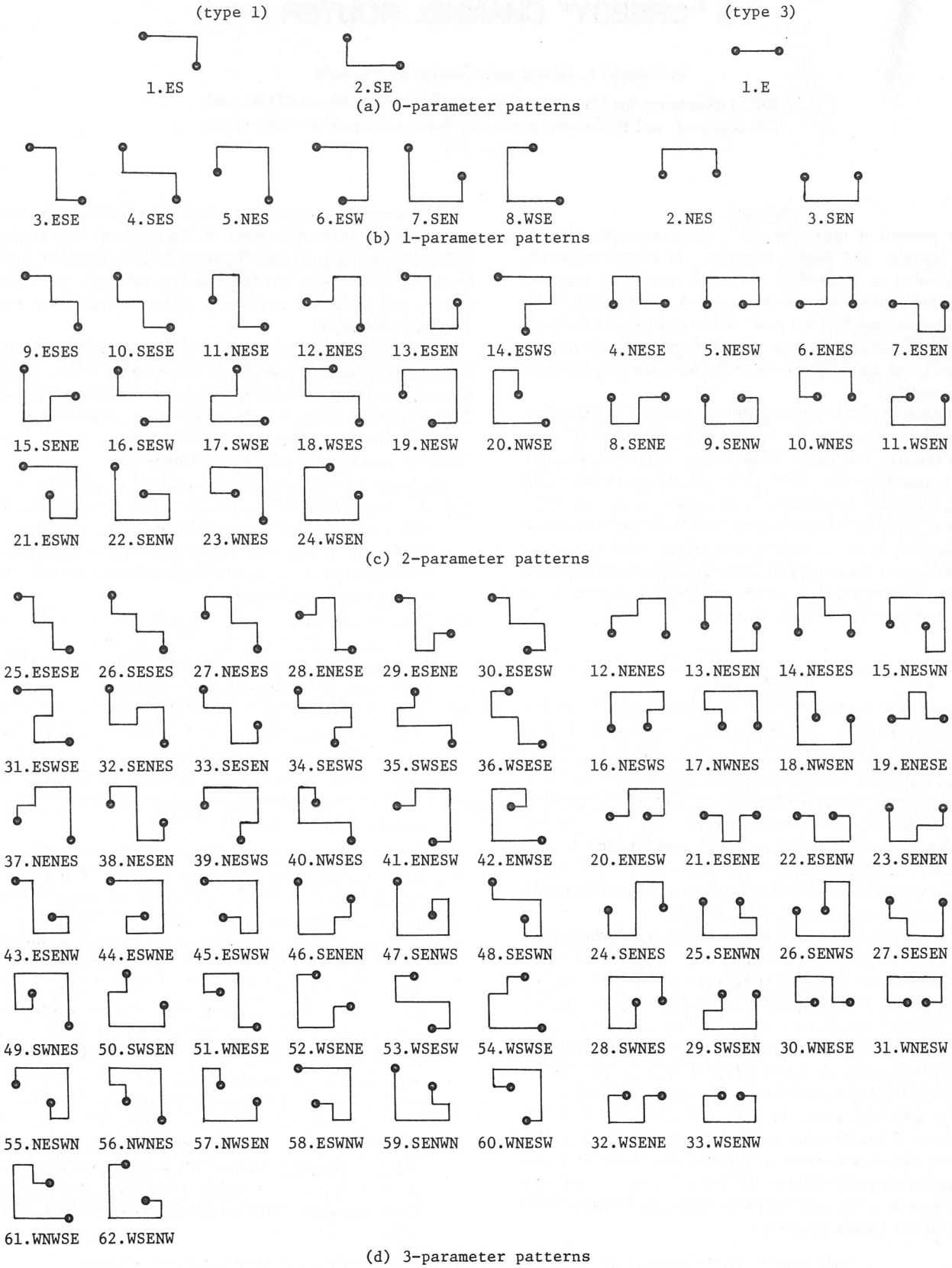


Figure 5 All valid patterns for types 1 and 3.

The channel density of a particular channel routing problem is defined to be the maximum number of nets which have pins on both sides of the line $x = \alpha$, for any α. (We don't count nets all of whose pins lie on a single vertical line.) The channel density is a lower bound on the width of any solution to that channel-routing problem.

If its "conflict graph" ([HS71]) contains cycles, a channel-routing problem may be unsolvable within the channel, for any w (e.g. $\lambda = 2$, $T = (1, 2)$ and $B = (2, 1)$.) Such problems can always be solved by using columns "outside" the channel.

The following factors are often used to evaluate the quality of a successful solution (in a typical order of priority): its width w, the number of columns "off the end" it uses, its total wire-length, and the number of vias it uses.

The Routing Algorithm

The greedy router scans the channel in a left-to-right, column-by-column manner, completing the wiring within a given column before proceeding to the next. In each column the router tries to maximize the utility of the wiring produced, in a simple "greedy" manner.

Its first step in a column is to make connections to any pins at the top and bottom of the column. These connections are *minimal*; no more vertical wiring is used than is needed to bring these nets safely into the channel, to the first track which is either empty or contains the desired net.

The second step in a column tries to free up as many tracks as possible by making vertical connecting jogs that "collapse" nets that currently occupy more than one track. This step may complete the job of bringing a connection from a pin over to a track that its net currently occupies (step 1 might have stopped at an intermediate empty track).

The third step tries to shrink the range of tracks occupied by nets still occupying more than one track, so collapsing these nets later will be less of a problem. Since freeing up tracks has high priority, jogs made here have priority over jogs made in the next step.

The fourth step makes "preference" jogs that move a net up if its next pin is on the top of the channel, and down if its next pin is on the bottom. The router chooses longer jogs over shorter ones if there is a conflict. This tends to maximize the amount of "useful" vertical wiring created. These jogs are effective at resolving upcoming "conflicts", even though no explicit consideration of these conflicts is made.

The fifth step is only needed if a pin could not be connected up in step one because the channel is "full". Then the router "adds a new track" to the channel between existing tracks, and connects the pin up to this track. (The old tracks are renumbered.)

When the processing for a column is complete, the router extends the wiring into the next column and repeats the same procedure. The following paragraphs make precise the algorithm just sketched.

The input for the greedy router consists of (1) a specification of a channel-routing problem, (2) three non-negative integer parameters: *initial-channel-width*, *minimum-jog-length*, and *steady-net-constant*.

The greedy router begins with the *initial-channel-width* given. A new track is added whenever the current channel-width becomes unworkable. The router does *not* begin over when a new track is added, so different initial widths may give different results. Good results are usually obtained with *initial-channel-width* just less than the best final channel width. One can run the router several times, with *initial-channel-width* set initially to the channel density and increased by one each time.

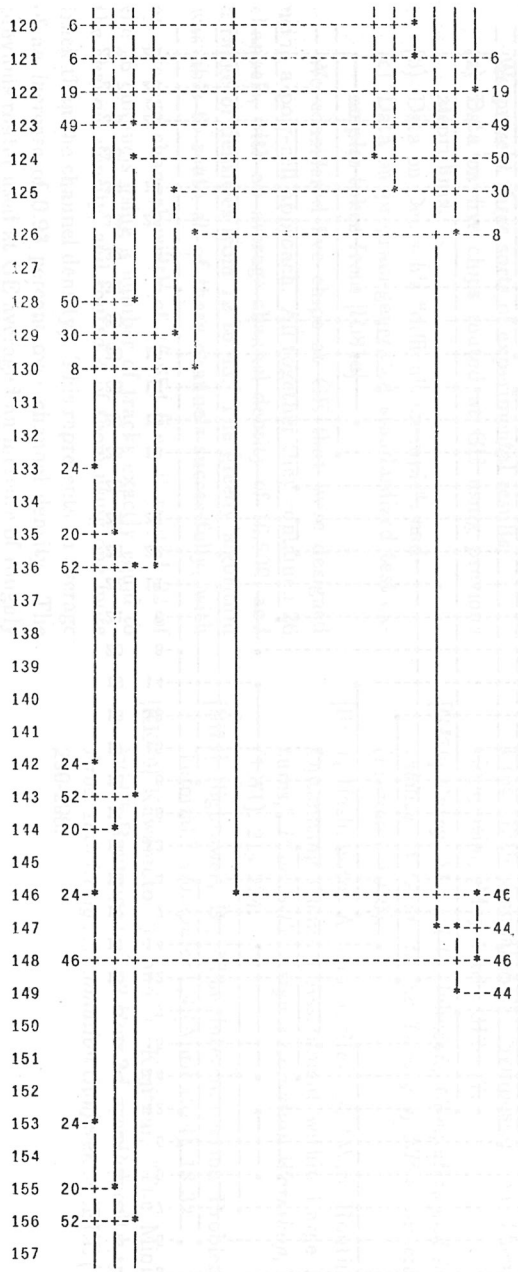
The router will make no "jogs" shorter than *minimum-jog-length*. A higher setting reduces the number of vias and thus produces more acceptable solutions, while a lower setting tends to reduce the number of tracks used. The best results are obtained with a setting of about $w/4$, where w is the best channel width obtainable. By running the router 2-4 times with different initial parameter settings we quickly determined the best solution obtainable.

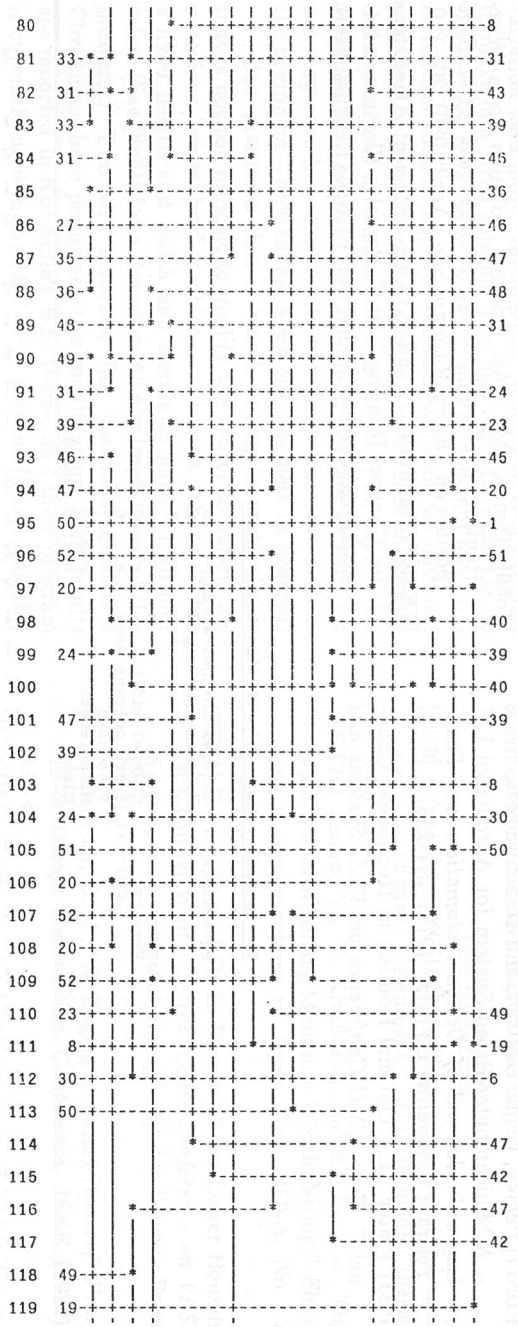
Let $H(n)$ denote the highest column k for which $T_k = n$ or $B_k = n$ (except that $H(n) = \lambda + 1$ if $n \in R$). We say a net n "has its last pin in column k" if $H(n) = k$ and that it "has its last pin by column k" if $H(n) \leq k$.

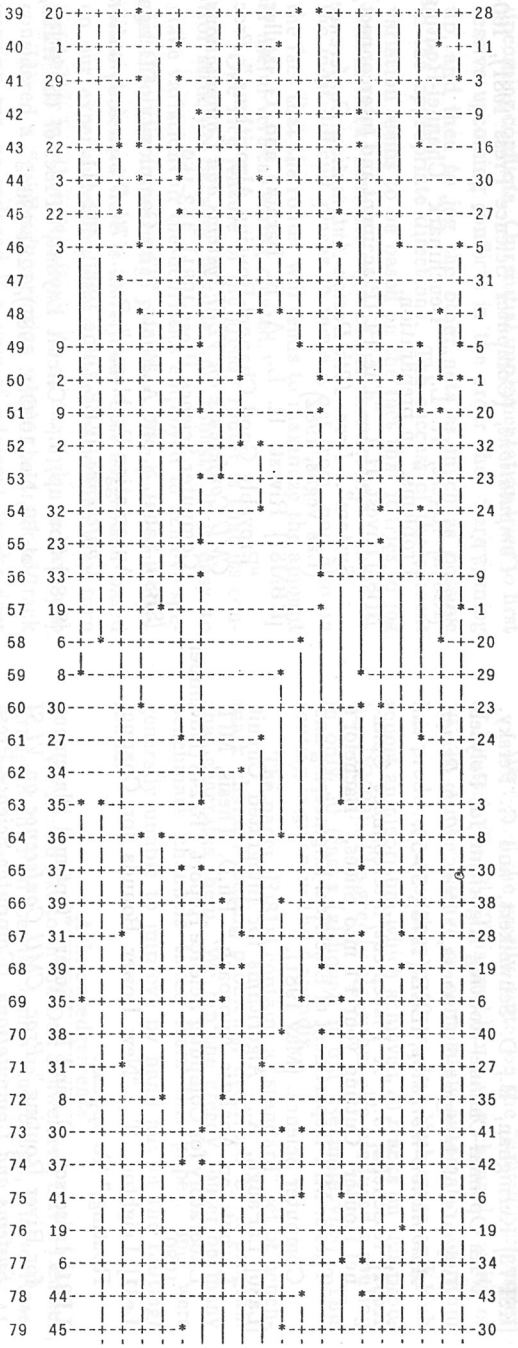
When routing a given column, the greedy router classifies each net which has a pin to the right as either *rising*, *falling*, or *steady*. A net is *rising* if its next pin after the current column will be on the top of the channel (say in column k), and the net has no pin on the bottom of the channel before column $k +$ *steady-net-constant*. *Falling* nets are defined similarly. *Steady* nets are the remaining nets. We typically use a value of 10 for *steady-net-constant*. A larger value reduces the number of times a multi-pin net changes tracks.

The fundamental data structure for this router is the set $Y(n)$ for each net n of "tracks currently occupied" by net n. Each track is denoted by its y-coordinate, so $Y(n)$ is a subset of $\{1, \ldots, w\}$ for each n. If $Y(n) = \phi$ (the empty set), the net is not currently being routed (i.e. we have not yet reached the first column in which net n has a pin, or we have passed the last column in which net n has a pin and have completed all the routing for net n). Otherwise, suppose $Y(n) = \{y_1, \ldots, y_k\}$ when the router is working on column i. Then each point $(i, y_1), \ldots, (i, y_k)$ is a "dangling end" of some wiring already placed for net n. Exactly one such "dangling end" is listed in $Y(n)$ for each connected piece of wiring already placed for net n. The router is obligated to eventually connect together these "dangling ends" so that each net is finally implemented by a single connected piece of wire. When extending the routing from column i to column $i + 1$, horizontal wiring will used in every track y for which $y \in Y(n)$ for some n and either $|Y(n)| > 1$ (the dangling ends have yet to be connected together) or the last pin for net n occurs after column i.

We define a net to be *split* at any time that $|Y(n)| > 1$. We also call a split net "collapsible", since we may be





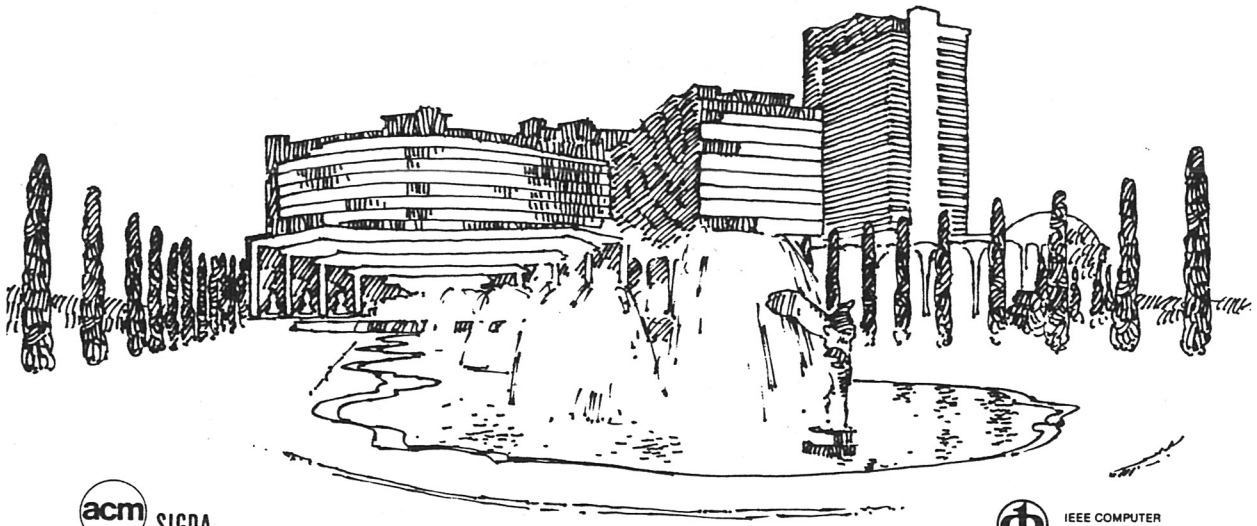


ISSN 0146-7123
IEEE Catalog No. 82CH1759-0
Library of Congress No. 76-150348
IEEE Computer Society Order No. 416
ACM Order No. 477820

ACM IEEE Nineteenth Design Automation Conference Proceedings

Caesars Palace
June 14-16, 1982

Las Vegas, Nevada



acm SIGDA

IEEE COMPUTER SOCIETY-DATC

ACM IEEE Nineteenth Design Automation Conference Proceedings

Caesars Palace
June 14-16, 1982

Las Vegas, Nevada