A Genetic Algorithm Approach to Compaction, Bin Packing, and Nesting Problems

Erik D. Goodman, Professor and Director
A. H. Case Center for Computer-Aided
Engineering and Manufacturing
Director, MSU Manufacturing Research Consortium
Professor, Electrical Engineering, Mechanical Engineering
College of Engineering
Michigan State University

Alexander Y. Tetelbaum, President and Professor
International Solomon University (Kiev, Ukraine)
Adjunct Professor, Electrical Engineering
College of Engineering
Michigan State University

Victor M. Kureichik, Professor
Taganrog State University of Radio-Engineering (Russia)
Visiting Professor, Electrical Engineering
College of Engineering
Michigan State University

TECHNICAL REPORT # 940702
CASE CENTER FOR COMPUTER-AIDED
ENGINEERING AND MANUFACTURING
MICHIGAN STATE UNIVERSITY

TABLE OF CONTENTS

1. INTRODUCTION

This technical report is prepared to record the preliminary work carried out in beginning a research project on the solution of a group of related problems by means of Genetic Algorithms (GA's). These problems include integrated circuit layout compaction, bin packing, and nesting. The principal problem is compaction, with the others serving to illuminate and guide the compaction work, as well as possibly receiving benefits from new tools developed. Efficient and near-optimal solution of the compaction problem will enhance the linkage between symbolic level modeling of the layout and the mask level design of hierarchical (fragment-based), ultra-fast, low-power analog and logic circuits. The research goals are to develop new methodologies, abstractions, and theory that lead to more efficiently defined and utilized Computer-Aided Design (CAD) algorithms and tools for compaction, bin packing and nesting. These new methodologies will incorporate important improvements in the handling of the physical, technological, and optimization aspects of compaction. More effective solution of the bin packing and nesting problems can help to solve the compaction problem, as well as being valuable in their own right for many practical problems. This will be accomplished by defining a new approach to the use of *Genetic Algorithms* (GAs)--for the compaction, bin packing, and nesting problems.

The research will concentrate mostly on the **application area** of high-speed, low-power analog and digital circuits with clock rates up to 20 GHz. In many applications in this high performance area, dramatically reduced chip area and signal delays are important design requirements. The **technology area** of interest is multi-chip modules (MCM) based on current or advanced silicon and GaAs technologies [1]. These technologies have the potential to deliver improvements in power management, low voltage operation, low crosstalk interconnects, and clock frequencies. The **CAD area** of concentration deals with the compaction, bin packing, and nesting problems and the solving of them based on GAs. These algorithms, using simple encoding and recombination mechanisms, display complicated behavior, and they turned out to be useful in solving some extremely difficult problems.

1.1. Report Overview

The rapid design of VLSI systems plays an important role in the progress of science and technology. In general, several important problems can be described in VLSI CAD systems. They are typically design specification, functional design, logical design, circuit design, physical design, and fabrication (technology design). To link these problems, we must perform functional and logical simulation, circuit analysis, extraction and verification. Physical design automation of VLSI systems consists of six major problems. They are partitioning, placement, assignment and floorplanning, routing, symbolic layout and compaction, and verification.

Traditional solution of most of the VLSI physical design problems is carried out by iterative methods [2,3]. These methods assume successive improvement of an initial variant, which is obtained by one of the constructive algorithms through local replacement of elements. However, these methods have a serious fault -- their use of a single variant of the initial solution.

Besides, when a satisfactory approximate solution is obtained by the constructive algorithms, the iterative algorithms stop at a local optimal solutions which could not be improved by further iterative steps of the same process. One way to improve the quality of solution is to incorporate new methodologies into CAD systems. In this work, we will develop and apply *genetic algorithms* (GA's), which have recently received widespread attention.

In 1975, J. Holland [4] described a methodology for studying natural adaptive systems and designing artificial adaptive systems. It is now frequently used as an optimization method, based on analogy to the process of natural selection in biology. The biological basis for the adaptation process is evolution from one generation to the next, based on elimination of weak elements and retention of stronger elements ("*survival the fittest*" -- those with the best performance *in the current environment*). Of course, over the longer term, it is not the strong individuals themselves which ultimately survive, but rather offspring related to them genetically in proportion to their *reproductive fitness* or success at reproduction. The searching of this representation space is performed using so-called "genetic algorithms" [4-6].

GAs are now widely recognized as an effective search paradigm in artificial intelligence, image processing, job scheduling, pattern recognition and many other areas [4-6], particularly as a method for gaining relatively good solutions to NP-hard optimization problems of high dimensionality. A GA maintains a population of strings (chromosomes) that encode candidate solutions to a problem. These strings are the analog of chromosomes in natural evolution. A fitness function defines the quality of each solution. The GA chooses parent organisms from the population in such a way that the more fit organisms are more likely to be chosen. It applies operators to generate offspring from the parents. The operators are often loosely modeled on biological genetic processes, however they are only very crude abstractions of the processes known and understood to be significant in natural evolution. Most commonly used operators are *mutation*, which randomly makes a local modification in a chromosome, and *crossover*, which combines genetic material from two parents. There are many other operators such as *inversion, translocation, segregation, duplication, etc,* which are sometimes, but not usually, used in GA's applied to optimization tasks. After performing these operations on individuals selected for reproduction, the GA selects current population members, often those with lower fitness values, to be replaced by the new offspring. The more fit members of the population thus propagate and combine to generate a population with generally increased fitness.

There are many design choices in the construction of a genetic algorithm for a particular problem. On the basis of our previous experience [7-19] and the results of many other investigations, we have found that GA's can evolve good designs and produce good solutions to many combinatorial problems. In this work, we are going to develop a new construction that is appropriate and effective for solving of the *compaction problem* in VLSI CAD systems, and bin packing and nesting problems[1] which are similar in many respects to compaction. Compaction is the translation of a layout designed from a generic technology (such as CMOS) to a particular fabrication technology's design rules. It translates the output of the detailed-routing phase into

---

1

mask data and has to convert the circuit elements into the appropriate mask elements and minimize the chip area. The goal is to minimize the area of the layout, while preserving the design rules and not altering the function of the circuit. This goal gives this process the name compaction. Compaction changes the geometry of the topological design to produce as small a layout as possible while enforcing the design rules. In this work, we are going to develop a new design methodology which incorporates some new genetic algorithm ideas that have been developed recently for other problems, plus efficient encoding/decoding methods and useful heuristic layout strategies. This new methodology will contain significant developments in the genetic algorithm approach and provide the basis for efficient suboptimal solution of compaction problems and the related bin packing and nesting problems. Our approach is to develop a hierarchical chromosome representation (HCR) and appropriate genetic operators incorporating various heuristics, and combine them with our "injection island" GA architecture (iiGA).

In this work we investigate the HCR for the layout compaction and bin packing problems. This representation will be designed to preserve building blocks in chromosomes, and the encoding scheme will make genes represent groups. The rationale is that in the problem being considered, "it is the groups of elements and their location which are the meaningful building blocks, i.e. the smallest piece of a solution which can convey information on the expected quality of the solution they are part of. This is crucial; indeed, the very idea behind the GA paradigm is to perform an exploration of the search space, so that promising regions are identified, together with an exploitation of the information thus gathered, by an increased search effort in those regions. If, on the contrary, the encoding scheme does not allow the building blocks to be exploited (i.e. transmitted from parents to offspring, thus allowing a continuous search in their surroundings) and simultaneously to serve as estimators of quality of the regions of the search space they occupy, then the GA strategy inevitably fails and the algorithm performs in fact little more than a random search or naive evolution" [20].

Each group of elements (subgroups) is a piece of a layout -- elements and connections in some area (or bin, for the bin packing problem). At the beginning, when we are given with some initial layout to be compact, each group contains only one element. During the GA process some groups are merged in a new more large group (of a more higher hierarchical level). It is important that all the GA operations now should work not with elements but rather with groups. For the proposed representation, we can use as the decoding mechanism one of the existing silicon compilers [2], or at least the philosophy of this approach. The silicon compiler uses a similar representation of the circuit and finds coordinates for each element (group) and each connection between elements (groups). During this compilation, at each step, two or more groups are merged into one new group, with simultaneous determination of relative coordinates of elements and connections within the group formed. Finally, all groups are merged into one, which represents the whole layout.

The formulation of the compaction problem to be solved is also to be refined, to take into account an additional very important criterion -- circuit performance. The standard compaction problems are one-dimensional (1-D) and two-dimensional (2-D) compaction. The bin packing problems include one-dimensional (1-D), two-dimensional (2-D), and three-dimensional (3-D) bin

packing. Nesting problems are usually 2-D (blank nesting, optimal nesting, or optimal cutting problems).

In the 1-D bin packing problem, the goal is to minimize the number of bins that contain a given set of weights, subject to a limitation of the total weight each bin can contain [21-24]. In the 2-D bin packing problem, the goal is to minimize the area of a single orthogonal bin by packing arbitrary-dimensional rectangular boxes into it [25]. Overlapping of features is not allowed, but there are no constraints on diversity or grouping. In the 3-D bin packing problem, the goal is to minimize the volume of a single orthogonal 3-D bin containing arbitrary dimensional 3-D boxes into it [22,26]. In the 2-D nesting problem, the goal is to maximize the number of arbitrary-shaped elements embedded into a single orthogonal bin[2] [27-30]. It is often allowed to turn the elements during the process.

Our methodology is expected to produce good solutions more efficiently than other techniques. This is very important, because the problems are NP-hard. For example, reasonable approximation algorithms for the bin packing problem can only guarantee to be within 22% of optimal [25]. Our algorithm will first initialize a population of chromosomes in which each chromosome represents the layout (compaction, bin packing, or nesting) as a HCR. For this representation, it is possible to develop a modified form of crossover, inversion, translocation, and some other operators not previously applied to this class of problems. The decoding algorithm, based on a silicon compiler, takes any chromosome and forms a legal layout (packing). We expect that our algorithm will advance the state of the art in making genetic operators robust with regard to quantity of data, variation in dimensions of boxes, and variation in the aspect ratio of the bin.

In the 2- and 3-D problems, we initialize a population of chromosomes (solutions) that can be induced by the initial symbolic layout and in this way decrease incredibly the number of possible initial chromosomes. Our algorithm evolves these populations of individuals. Each individual is a string of genes representing some group of elements or subgroups and the corresponding layout that can be obtained after the decoding procedure. New individuals are produced by a stochastic mix of the modification of classic genetic operators: crossover and mutation, specialized for solving hierarchical problems, and some other genetic operations (inversion, translocation, etc.). We propose to explore several alternative heuristic operator/selection strategies, including ones we call "random with pressure" and "random-direct" selection. This will introduce less selection-induced variability. These algorithms will be run using the parallel architectures described below, as already developed by our group.

1.2. Research Tools and Validation

These ideas and objectives will be investigated in the framework of existing commercially available and public domain tools, specifically, Mentor Graphics tools for design, and the GALOPPS ("Genetic ALgorithm Optimized for Portability and Parallelism" System) (a highly

---

[2]

a very flexible and portable GA system with roots in Goldberg's SGA, but totally rewritten) already developed by one of the authors and distributed to collaborators in GA research in 10 universities around the world. The use of the Mentor Graphics package will be a useful tie into current CAD technology and is viewed as an essential tool to study the compaction.

Benchmark problems will be assembled for validation of the methodology developed. This set of benchmark problems will come from the literature, from random generation according to realistic characteristics, and from real chips, and will be used for comparison of our methods with existing compaction techniques. Several possible validation paradigms are referenced in Section 2. Specific details of the tasks and methodology which will be used to achieve the goals of this research are provided in Sections 2 and 3 of this proposal.

2. BACKGROUND

2.1. Introduction to Layout Compaction

The layout designer is driven by two conflicting demands. The first is to find the layout with an area as small as possible, in order to minimize manufacturing cost and maximize circuit performance. The second is to create the layout quickly in order to minimize design time and cost. The designer usually develops the layout during two major layout phases [2,31,32]:

1. *Topological design*, after which the relative placement of components and wires are defined.
2. *Geometrical design*, after which the geometrical (physical) positions of all the layout elements are defined.

The designer uses different design methodologies to solve the problems of these two phases. *Symbolic layout* and *compaction* are two closely related design methodologies that encourage the separation of topological design and geometrical design and help to automate geometrical design [2].

Symbolic layout allows the topological designer to work with the transistors, wires, and cells as primitives, rather than manipulating the individual polygons used in fabrication. A symbolic layout can be drawn as a *stick diagram*, which uses line segments and components as symbols, or as a layout display with wires and devices drawn as rectangles similar or identical to those used in the layout. The stick diagram may clarify the cell topology, while the layout display gives the designer a feel for the relative sizes of layout elements.

The translation of the output of the detailed-routing phase into mask data must convert the circuit elements into the appropriate *mask elements* . It should ensure that all design rules are met, while simultaneously minimizing the layout area. This last goal gives this process the name *compaction* [2,3]. Compaction changes the geometry of the topological design to produce a small layout while enforcing the design rules.

Compactors speed layout design by automating geometric design. The designer gives the compactor a preliminary layout. The compactor moves components and wires in the plane to optimize the layout (the first goal) and to correct it in accordance with the design rules (the second goal). The compactor usually moves subcells only in the plane, preserving the designer's topology for the cell. The designer can therefore have a great deal of control over the layout without performing the work required to turn a sketch of a layout into a correct, space-optimized design [33].

Compaction performs a translation from the graph domain into mask geometry. Compaction is more than just an *optimization problem.* Compaction is quite a difficult problem not only from a combinatorial, but also from a systems point of view. Advanced compactors aim at separating combinatorial and technological issues as much as possible. Compaction algorithms use the information from the design-rule database to compute the mask features. The structure of information that the compaction algorithm has to extract from the design-rule database can be quite complicated. It involves not only simple numerical parameters, such as minimum distances between features on different masks, but also connectivity information and electrical information.

Most of the steps in the compaction process are made much simpler by working with a symbolic description of the layout. Combining compaction with symbolic layout creates advantages for the *computer-aided design* developer as well.

2.2. Basic Definitions and Taxonomy of Compaction

Compaction algorithms may be classified along two basic axes, A and B [3,31,32]:

A describes how components move during compaction;

B covers the algorithms used to position the components.

Algorithms from A can be divided into the following groups.

A.1. One-dimensional (1-D) compaction. In 1-D compaction, only the, say, x coordinates of the mask features are changed. This kind of compaction is also called *x compaction* (y compaction is defined analogously). The goal is to minimize the width of the layout, while preserving the design rules and not altering the function of the circuit.

So, in 1-D compaction, components are moved only in the x direction or only in the y direction. Most steps of 1-D compaction can be done efficiently. In fact, they can be done with almost linear-time algorithms [3,34]. A few versions of 1-D compaction are NP-hard [2,31,35].

A.2. Two-dimensional (2-D) compaction. In 2-D compaction, both x and y coordinates can be changed simultaneously in order to minimize area, i.e. in 2-D compaction, a single step can move a component in both x and y. In 1-D compaction, the cell is alternately compacted in x any y, while in 2-D compaction components are selected to move as required to improve the layout.

Most versions of 2-D compaction are NP-hard [34,36]. The difficulty of 2-D compaction lies in determining how the two dimensions of the layout must interact to minimize the area. To circumvent the intrinsic complexity of this question, some heuristic are used to decide locally how the interaction is to take place [31-33]. These heuristics are sometimes referred to as *1.5-dimensional compaction* [3,35].

Algorithms from B can be divided into the following two major types (groups) [2,3,31,32].

B.1.Constraint-graph algorithms. The constraint graph algorithm describes the required connections and separation rules as linear inequalities, which can in turn be modeled as a weighted, directed graph [36]. The constraint graph is used to find new positions for the components, and the result is applied back to the layout.

B.2. Virtual grid algorithms. The virtual grid algorithm finds the subcell positions by considering the layout to be drawn on a grid; it moves all components on a grid line together so that adjacent virtual grid lines are as close as possible while satisfying all required separations between the symbols on the grid.

Modern designs are modularized hierarchically. An example is the bottom level of the hierarchy. Optimizing leaf cell layout requires awareness of many interacting constraints and complex cost functions. Traditionally, this area of design has been left to human experts. *Hierarchical compaction* works on cells that are constructed from other cells as well as primitive layout symbols [37,38]. Any of these techniques can be made hierarchical by applying the compaction algorithm to a leaf cell and then using the compacted cell in larger cell. Hierarchical compaction is also called *cell assembly*. A variety of hierarchical compaction algorithms have been developed for both constraint-graph algorithm and virtual grid algorithm compaction [39-41].

2.3. Algorithms for Compaction.

The general procedure in compacting a cell is shown in Fig. 1 [2,3,31-35]. Each iteration through the loop is a compaction step. First, the layout is analyzed to determine the spacing rules that must be obeyed. Then the layout is compacted to satisfy those constraints. Finally, a wire length minimization algorithm is applied to the compactor's solution to adjust the positions of non-critical wires in the layout.

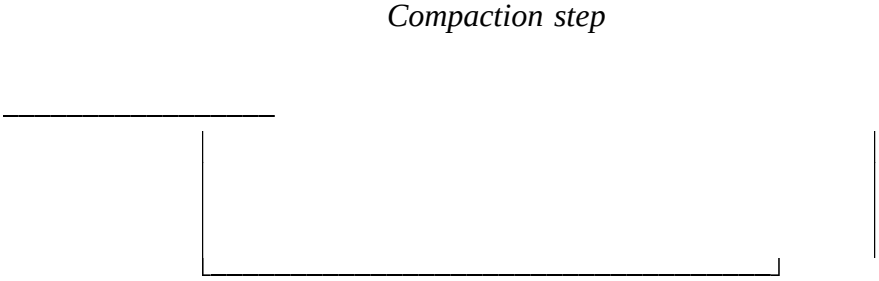


Fig. 1. The compaction process.

2.3.1. Constraint-graph 1-D Compaction Algorithms

There are two basic approaches to the constraint-graph 1-D compaction algorithms: compression ridges and graph-based compaction.

Compression-ridge method. This method was pioneered by Akers [42]. In this

approach, a region of empty space is identified in the geometric layout, which separates the layout into a left and a right part. Such a region is subsequently removed by translation of the right part of the layout to the left by as much as the spacing constraints will allow. This step is repeated until no more compression ridges are found.

Disadvantages: This method progresses from the top to the bottom of the layout. This progression amounts to a search through the layout with backtracking (if the compression-ridge method cannot progress further) and thus is computationally complex. Furthermore, it finds only horizontally convex compression ridges.

The key to finding compression ridges is the representation of the empty space in the layout by a directed edge-weighted graph, the *tile graph* [3,42]. The tile graph is constructed as follows. First the horizontal segments of the contours of the features are extended through the adjacent regions of empty space. This process divides the empty space between the features into rectangular tiles. The tile graph G=(V,E) has a vertex for each tile. Special vertices s and t represent the top and bottom sides of the layout. Vertices representing vertically adjacent tiles are connected with anti-paralleled edges. Thus each anti-paralleled edge pair corresponds to a horizontal segment separating two tiles. The edges are weighted as follows: An edge e pointing downward receives the weight c(e), which is the width of the corresponding tile. An edge pointing upward receives the weight c(e)=inf.

The ordered 1-D compaction can be solved as follows [3,32]:

Step 1: Find a maximum flow from s to t in G.

Step 2: For all edges e from E that point downward, compress the tile corresponding to e by the amount indicated by the flow along e.

It was shown [3] that the preceding algorithm solves the ordered 1-D compaction with d(1,1)=0, and p=id and without grouping constraints, as long as the initial layout is legal. In [3] they restrict their attention to compression ridges that correspond to paths in the tile graph. They give an O(n log n) time algorithm for finding such a ridge that allows for the largest decrease in layout width. However, before compaction is complete, many compression ridge may have to be found. The advantages of the compression-ridge method are that the compaction can be broken up into small steps and that it is possible to interleave compaction in both dimensions. This feature is especially important in an environment in which compaction is done interactively, step by step. The Akers compression-ridge method can be applied in a virtual grid setting.

Graph-based compaction. This method turns compaction into a system of linear inequalities that is subsequently solved using shortest-path methods [31,32,43,44]. The compactor's job is to recognize and enforce spacing rules while minimizing area. Spacing rules can be written as constraints (the *Constraint-Graph Model*).

The positions of the components, represented as variables in the inequalities, are represented by vertices in the graph. Negative weights on the graph edges indicate that a component is allowed to be to the left of another: C can be at least -d units to the right of W (or d units to the left), and vice versa. The edges representing these constraints form a cycle in which the sum of the weights around the cycle is non-positive.

The graph represents a 1-D compaction because all the variables in the constraints represent positions in the same dimension. The graph includes two artificial vertices, L and U, which represent the lowest and highest positions in the cell, respectively. L is a source of the constraint graph, and U is a sink of the graph. Values assigned to the vertices represent positions of the cells in the dimension of compaction. The minimum $L \geq U$ distance is determined by the longest path of constraints from L to U. One can translate the longest path problem into a shortest path problem by negating the weights of the edges. One can view 1-D compaction as a *linear programming problem*:

MINIMIZE	U - L
subject to	$- L + A \geq 0$
	$- L + B \geq 0$
	$- A + B \geq 4$
	etc.

The objective function of the linear program is U-L, the distance between the lower and upper edges of the graph. The simplex algorithm can perform such a search. The average number of iterations required for the simplex method is very nearly linear in the number of constraints.

Constraint-graph leaf cell compaction

THE SHADOWING ALGORITHM [2,31-33,44,45]. The shadowing algorithm examines the positions of cells to determine what constraints are redundant. As shown in Fig.7, one can imagine a region that contains cells B,C,D,... that must be constrained against cell A in x as falling under a shadow cast from A.

The shadow's maximum height is the height of A extended by the distance in each direction. If the shadow falls on a cell, that cell must be constrained against A; thus, we must generate a constraint from B, C and D to A.

The shadowing algorithm describes how to generate the constraints on a cell from all the cells below it or to its left. The algorithm keeps the bounds of the shadow's left edge in a shadow "front". The shadow is a list of edges, each of which has a position in the constraint generation dimension and an upper and lower bound in the opposite dimension. The shadowing algorithm starts with the shadow extending from A to the cell's left edge, then searches from right to left cells that block portions of the shadow. The function IN SHADOW returns true if a cell's right edge is in the shadow, which implies that it must both overlap at least one edge in shadow-front in y and also be above that edge in x. If the cell is in the shadow, the algorithm adds a constraint from it to the sink and calls the update shadow procedure to modify the shadow front so that one edge of the shadow coincides with the left edge of the cell that required it. The shadowing algorithm moves from left to right.

There are some difficulties in creating a robust implementation of shadowing. Some components may move through each other during compaction, and the shadow must not be

blocked until all cells that may enter the shadow have been considered.

THE INTERVENING GROUP METHOD [45-47]. This method uses a simple heuristic to eliminate many, but not all, redundant constraints, by remembering a path through the constraint graph that provides a lower bound on the size of relevant constraints.

The first function in intervening group method, CONSTRAINED, returns TRUE if the constraint graph includes a path of constraints between two cells that satisfies the spacing rule between them. The second procedure in the method, GENERATE CONSTRAINT GRAPH, considers each cell from left to right, generating constraints to the i cell from all previously considered cells 1,...,i-1. The speed with which the set of all constraint paths to a cell can be tested and updated allows the algorithm to eliminate redundant constraints very quickly.

PERPENDICULAR-PLANE-SWEEP ALGORITHM (PP shadowing algorithm) [35,37,48-53]. The PP shadowing algorithm uses a scan line to perform a shadowing analysis. X-dimension constraints are generated by a vertical line that is swept from left to right across the layout. In design rule checking, because polygons do not move, only polygons within a fixed distance of the scan line need be checked, but for compaction the scan line must collect arbitrary distance elements. Constraints are generated from one element's right edge to another's left edge. A right edge is added to the edge data structure and is checked to find left edges in the data structure against which it must be constrained. A left edge, because it blocks the shadow cast by a right edge, causes right edges to be removed from the data structure. The one-layer version of the PP shadowing algorithm has a time complexity of 0(nlog(n)), where n is the number of rectangles in the layout.

Once physical connectivity and separation constraints have been generated, the constraint set must be solved to find the longest path from L to U. An iterative algorithm is usually used on a constraint graph with cycles. An edge can be used to define an ordering on the vertices it connects. All positive-weight edges are forward, as are any negative-weight edges that satisfy the partial ordering defined by the positive-weight edges; any other edges are backward. In an iteration, the method first applies acyclic-longest-path to find vertex positions that satisfy the forward edges, then it updates that solution to also satisfy the backward edges. The algorithm stops when a forward update/backward update iteration produces no change in the solution. This algorithm is of complexity 0(ve), but because constraint graphs have relatively few large strong components, its average behavior is better.

Virtual grid algorithms

VIRTUAL GRID LEAF CELL COMPACTION [34,39,40,54]. The model simplifies both constraint generation and solution. The virtual grid compaction system determines spacing rules and assign coordinates simultaneously, avoiding the need for an intermediate data structure to describe the constraints.

Compaction starts by assigning the left-most virtual grid the mask position 0. The other virtual grid lines are assigned mask positions in order by looking at the components that have

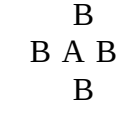
already been placed. The most commonly used method to find all the constraints that must be satisfied is the most recent layers algorithm.

After both x and y compaction, virtual grid compactors use an xy compaction step to reduce cell size. The 1-D compaction model produces a rectangular barrier around a component. The xy compaction step looks for adjacent components that can be moved closer together and adjusts the positions of their virtual grid lines. The xy compaction step is usually performed simultaneously with the second x or y compaction, since it must look at the same constraints. Because the virtual grid lines determine how components move, only one x and one y compaction step are necessary. Further x or y compaction will not reduce the cell size.

SPLIT-GRID COMPACTION [39,41]. This method allows objects on a virtual grid line to move independently, but does not allow connections with steps. Split-grid compaction produces a layout closer in quality to a constraint-graph algorithm compaction, at the cost of more CPU time than required for simple virtual grid algorithms.

2.3.2. Two-dimensional compaction (2-D compaction)

2-D compaction is an extension of 1-D constraint-graph compaction. As shown in Fig. 2, two cells A and B can have 4 relative positions: B can be above, below, left of, or right of A [36,55].



$B(x) \geq (A(x)+c)$ (right),
$A(x) \geq (B(x)+c)$ (left),
$B(y) \geq (A(y)+c)$ (above),
$A(y) \geq (B(y)+c)$ (below).

(a) Layout. (b) Constraints.

Fig. 2. 2-D Compaction constraints.

In each case, some spacing rule determines the minimum distance required between them. The minimum distance is expressed as a linear inequality. Left-of/right-of placement constraints are expressed as functions of x-dimension variables, and above-below constraints are written in terms of y-dimension variables.

This formulation is a special case of *mixed integer linear programming*, which works on linear constraints and 0/1 decision variables. A simple insight helps us to discover algorithms for solving these problems -- for any particular setting of decision variables, the selected constraints form two 1-D constraint graphs, one for x and one for y. Solution algorithms choose settings for the decision variables (in effect selecting the relative locations of cells), solve the

resulting 1-D compaction to get the cell positions, and evaluate the quality of the resulting layout.

Kedem and Watanabe [55] used branch-and-bound search to solve a 2-D constraint set. Schlag et al. [36] allow all four possible placements of component pairs. Their algorithm not only searches through all possible compactions of a given planar topology, but also changes the planar topology -- by moving a component from one side of a wire to the other, for instance. One simple way to use this algorithm is to assemble 2-D compacted leaf cells using the 1-D hierarchical compaction algorithms. The limitation of the algorithm is its high computational complexity.

2.3.3. Constraint-graph hierarchical compaction

There are two approaches to constraint-graph hierarchical compaction [15,32,36,55-62]. The first treats subcells as fixed, whereas the second allows subcells to stretch during compaction.

FIXED-CELL HIERARCHICAL COMPACTION. This compaction is used in SPARCS [2,37]. These programs' model of subcell is the position of its ports and a protection frame used to determine cell-to-cell spacing. The protection frame includes material within the maximum design rule distance on each layer; this material determines the subcell's separation requirements.

PITCHMATCHING [63-65]. Pitchmatching---stretching cells so they can be connected by abutment---is simply recompacting the cells with additional ports to be at the same positions. Hierarchical compaction with stretchable cells required the compactor to build a more complex model of a subcell than that required for fixed-cell assembly. The ideal model would contain information about how the cell's contents stretch during compaction -- how the ports can move during compaction -- and information to determine the spacing required between the cell and other cells.

The port abstraction model fully characterizes the stretching behavior of the cell with a much smaller constraint graph that includes only the ports as vertices and flexible protection frame that describes the cells boundary. A port abstraction consists of two constraint graphs, one for x and one for y. The vertices in each graph represent the ports. The edges describe how the ports can move during compaction. The motion of the ports is determined by the components and wires in the cell's interior. Because cells contain many fewer ports than components and wires, the port abstraction is a greatly simplified model of the cell.

The abstraction's constraints are determined by computing a subset of the transitive closure of the constraint graph. Because the abstraction contains only port vertices, only the longest path between ports need to be computed. The simplest algorithm for calculating transitive closure is Floyd's algorithm [66], which takes $O(v^3)$ steps irrespective the graph's sparseness. The Eic method [67] insures that each pair of components is constrained by either an x or a y constraint. The longest paths between all pairs of components and wires in the cell are computed; if their ranges conflict in one dimension, then a constraint in the other dimension

is introduced. This method is robust, but costs much more to compute.

Finding the spacing required between two subcells during hierarchical compaction is much more complicated than finding the spacing required between primitive symbols, because components in the subcell can move during compaction. The port abstraction does not contain enough information to determine how cell-to-cell spacing changes as cells stretch, so most systems use worst-case spacing between subcells.

VIRTUAL GRID HIERARCHICAL COMPACTION [68,69]. Most virtual grid systems perform a simple two-level hierarchical compaction. First, all leaf cells are compacted, then all cells are assembled into the final design by a pitchmatching algorithm. During pitchmatching, the mask positions of virtual grid lines in the cells are adjusted to stretch the cells for abutment. Stretching subcells during virtual grid hierarchical compaction is best formulated as a simple constraint problem. The virtual grid compaction of the subcells determines the minimum positions of their pins. The minimum distance of each pin from the base of its subcell can be written as a separation constraint, and a connection between pins can be written as an equality constraint. The resulting graph, after merging nodes connected by equality constraints, is acyclic and can be solved with breadth-first search.

Once the port positions have been found, it can update the positions of the other virtual grid lines for the subcells to maintain the required spacings. The virtual grid model simplifies the determination of cell-to-cell spacing -- the virtual grid allows the compactor to predict how components along the boundary will move. The CAD system PANDA [2,68] performs cell-to-cell spacing in two passes: first, using worst-case spacing, then pushing cells closer together based on a design rule analysis near the boundaries of abutted cells. CAD COORDINATOR [2,69] uses constraint-generation algorithms for leaf cell compaction, which make feasible the compaction very large (10,000) cells.

2.3.4. Wire length minimization [40,41]

The simplest algorithm for wire length minimization works directly on the compaction constraint graph. First the compaction constraints are augmented with constraints that maintain the order of wire endpoints. The augmented graph is solved by repeatedly applying the minimization step. A vertex is chosen that, when moved, will reduce the weighted sum of wire lengths in the cell. The wire length minimization algorithm is described as a linear programming formulation of the problem, and solves it using network programming techniques. Because the optimum solution to a linear program occurs at the simultaneous solution of two or more constraints, the wire length need be checked only at the ends of the vertex's range. When no vertex can be moved to reduce total weighted wire length, the cell's wire length is minimal. An algorithm that performs grouping and shearing operations to minimize total wire length has been developed.

INCREMENTAL CELL OPTIMIZATION [32,54,61]. The most incremental cell

optimization methods are jog introduction -- adding a degree of freedom to a connection by changing a wire segment to a z-shaped wire. Let us consider two typical situations for adding a jog. In both cases, allowing the ends of the wire to move separately allows the components to move closer together in y. Automatic jog introduction occurs at the start of a compaction step -- wires are selected for jogging and jogs are added, then constraint generation and solving proceeds normally [70].

Super-compaction [2,31-33] introduced methods for analyzing the critical path to find a small set of jogs sufficient to reduce the cell size. Finding a set of jogs that breaks all critical paths is equivalent finding a cut-set of the critical path graph. Critical path analysis also allows us to consider geometric reorganizations that break the critical path by moving cells to eliminate separation constraints.

2.4. Layout Approaches Based on Genetic Algorithms

In this proposal, we describe various approaches to physical design of VLSI systems based on genetic algorithm methodology, in order to show the utility of the GA approach to layout design.

Partitioning Problem

Let us define the VLSI partitioning problem as follows. We can represent a VLSI system as a hypergraph H=(X,E), where X represents the set of nodes and E represents the set of hyperedges. Let P={P(1),P(2),...,P(l)} be the set of partitions of the H. Let each partition P(i) contain the elements x(i); that is, P(i)={x(1),x(2),...,x(k)} and each x(i) belongs to X. The cost function is the number of hyperedges (connections) between different partitions P(i). Our task is to find a partitioning that minimizes the cost function. Genetic algorithms start with an initial population of partitions (randomly generated or generated by some constructive method). They evaluate this population based on the cost function. Then they select some members of the population and perform genetic operators (crossover, mutation, inversion, translocation and/or others). Applying GA's allows one to find better solutions for partitioning problems [71,72] than those typically found by other methods.

Placement Problem

The placement problem of standard cells on the VLSI chip is to arrange a given set of standard cells of common or variable height and width on the chip surface. The placement must satisfy criteria of routability of the design. This is usually an objective function which minimizes a function of the wire lengths. Genetic algorithms for placement usually use the following steps: initialization of the population, evaluation, selection, and allocation. Some algorithms use so-called "pressure" techniques. After formation of each generation, they perform local optimization to find the best solutions. Such a procedure helps in finding many local optima, and perhaps also

the global optimum [73,74].

Floorplanning

The floorplan design problem is to arrange a given set of "flexible" modules (with undefined shape and pin assignment) in the plane to minimize the chip area and some function of the wire lengths. In [75], the theory of punctuated equilibria and genetic algorithms are used. They propose a new structure and new model for coding of VLSI systems, as well as a novel genetic strategy and some new genetic operators combining local and random search, by means of which they state that "better" results may be obtained.

Routing

Let us consider two optimization problems, the channel routing problem (CRP) and detailed router problem (DRP) [2,3,76,77]. The CRP is specified by a given rectangular channel with two rows of terminals along its top and bottom sides. The objective of the channel router is to assign each net to a horizontal track (or tracks) in order to minimize the number of such horizontal tracks. Construction of a genetic algorithm for the CRP includes five major steps [76].

1. Select a representation to encode the search space.
2. Determine an appropriate evaluation function and scaling scheme.
3. Determine a selection algorithm.
4. Choose genetic operators.
5. Set control parameters.

A new scheme for selection and some heuristic knowledge-oriented genetic operators combining local and random search improve the performance of the GA in this application. After each generation, the algorithm keeps the best of the previous generation and the newly generated offspring. This process improves chances of finding the optimum routing. Some new results for the DRP are also developed in [77].

Symbolic Layout and Compaction

Symbolic layout and compaction are two closely related design methodologies that encourage the separation of topological design and geometrical design and help to automate geometrical design [2,3]. The translation of the output of the detailed-routing phase into mask data must convert the circuit elements into the appropriate mask features. At the same time, it should ensure that all design rules are met, while minimizing the layout area. In [78], a genetic algorithm for performing the compaction is described. A population consisting of lists of constraints is used with chromosomes differing with respect to the order in which these constraints are applied. We will describe this problem and GA’s for it in more details in the next section.

The GA strategy is a powerful methodology for avoiding premature convergence at local optima.

It can be efficiently used for design of VLSI systems. GA's are easily parallelized, for increasing speed in massively parallel or distributed computing environments, including low-bandwidth networks of high-performance computing elements. Parallel GA's hold the promise nearly linear speedup of calculation with the number of processors.

2.5. Bin Packing, Nesting and Compaction Using Genetic Algorithms

2.5.1. 1-D Bin Packing

Problem formulation. Given a finite set of elements $E=\{e_1,...,e_n\}$ with associated weights $W=\{w_1,...w_n\}$ such that $0 \leq w_i \leq w^*$. Partition E into N subsets such that the sum of weights in each partition is at most w* and that N is the minimum.

Let us first consider the classic traditional methods. In [79] they call an algorithm *on-line* if the numbers in list L are available one at a time and the algorithm has to assign each number before the next one becomes available. Let L = (x(1), ... , x(n)) be a given list of real numbers in (0, 1), and let BIN(1), BIN(2), ... be an infinite sequence of bins, each of unit capacity. The BPP is to assign each x(i) to a unique bin, with the sum of numbers in each bin not exceeding one, such that the total number of bins used is minimum. It is shown that any on-line algorithm S must have $r(S) \geq 3/2$, where r(S) = S(L)/L*, S(L) is the number of bins used by S, and L* denotes the minimum number of bins needed.

Commonly Used Heuristics

FIRST-FIT (FF) algorithm (an on-line algorithm).

Given a list L = (x(1), ... , x(n)), the FF assigns x(j) sequentially, for j = 1, 2, ... , n to BIN(i) with the smallest i whose current content does not exceed (1-x(j)). The measure r(FF) = 17/10. The FF algorithm is an O(n log n)-time algorithm. It is shown in [79] that in generalized, d-dimensional bin packing, any O(n log n)-time algorithm S must have $r(s) \geq d$.

FIRST-FIT-DECREASING (FFD) algorithm (NOT on-line).

Given a list L = (x(1), ... , x(n)), the FFD first sorts the x(j)'s into decreasing order, and then performs FF. FFD has a running time O(n log n). r(FFD) = 11/9.

REFINED-FIRST-FIT (RFF) algorithm.

Any element x(j) in a list L will be called an A-piece, B(1)-piece, B(2)-piece, or X-piece if x(j) is in the interval (1/2, 1]-A, (2/5, 1/2]-B(1), (1/3, 2/5]-B(2), or (0, 1/3]-X, respectively. Before packing, they [79] divide the set of all bins into 4 infinite classes. Let m from {6, 7, 8, 9} be a fixed integer. Suppose the first j-1 numbers in list L have been assigned: they process the next number x(j) according to the rules:

a) Put x(j) by FF into a bin in
   class 1, if x(j) is an A-piece,
   class 2, if x(j) is a B(1)-piece,
   class 3, if x(j) is a B(2)-piece, but not the m(i))th B(2)-piece seen so far for any integer $i \geq 1$,
   class 4, if x(j) is an X-piece.
b) If x(j) is the (m(i))th B(2)-piece seen so far for some integer $i \geq 1$ they
   put x(j) into the first-fit bin containing an A-piece in class 1, if possible, or put x(j) in a new bin of class 1 otherwise.

RFF runs in O(n log n) time, as it essentially performs a FF within each class, which takes o(n log n) time for each x(j). r(RFF) = 5/3.

Article [80] describes two additional heuristics, but with the bin capacity w*=1: Given a list L =(a(1),..., a(n)) of real numbers in [0, 1], place the elements of L into a minimum number L* of bins so that none contains numbers whose sum exceeds 1.

BEST-FIT (BF) algorithm.

Let the bins be indexed as B(1), B(2), ... , with each initially empty. The numbers a(1), a(2), ... , a(n) will be placed in that order. To place a(i), find the least j such that B(j) is filled to level $b \leq 1$-a(i) and b is as large as possible, and place a(i) in B(j). B(j) is now filled to level b + a(i).

BEST-FIT-DECREASING (BFD) algorithm.

Arrange L = (a(1), a(2), ... , a(n)) into non-increasing order and apply BF to the derived list.

The FF algorithm places each number, in succession, into the first bin in which it fits. The BF algorithm places each number, in succession, into the most nearly full bin in which it fits. In [80], they show that neither the FF nor the BF algorithms will ever use more than (17/10)L* + 2 bins. They prove that, if L is in decreasing order, then neither algorithm will use more than (11/9)L* + 4 bins.

In [23], the authors describe classical solutions of on-line packing problems, i.e., that pack items as they arrive without any knowledge of future arrivals. Thus, such algorithms will assign the items to bins in order of increasing index, under the single constraint that at each time t there be no bin that contains "currently active" items whose sizes sum to more than w*. First Fit (FF) packing is the central algorithm of interest. The ordering for occupied bins is maintained by this procedure. They generalize the classical one-dimensional bin packing model to include dynamic arrivals and departures of items over time. In [24], the authors describe two classic

approximation algorithms for packing rectangles, called *next-fit* and *first-fit shelf* algorithms. Shelf algorithms can pack the rectangles in the order specified by the given list (queue), without sorting them first. Unlike [23], the shelf algorithms can pack an infinite list of rectangles such that for any finite initial segment of the list, the height of the packing is within a constant times the height of an optimal packing of that segment.

A nature-based stochastic approach as a new approximation algorithm for solving the bin-packing problem and as a solution-improving tool for the FFD in its worst cases is proposed in [21]. This approach is called the Annealing-Genetic Algorithm (AGA), which incorporates a GA into a simulated annealing algorithm (SAA) to improve the performance of the SAA. Both SAA's and GA's are stochastic optimization techniques based on analogy with natural processes, the former based on thermodynamics and the latter based on natural evolution. The major advantages of these nature-based algorithms are their broad applicability, flexibility, ease of implementation, and potential to find the optimal solutions. Their disadvantages include the following: a SAA may require a long computation time in order to converge to the optimal solutions, and the SAA may easily be trapped into a local optimum. GA's are not well suited to rapidly performing finely-tuned local search and may exhibit premature convergence due to the loss of alleles required to reach the global optimum.

The main ideas of this approach in speeding up SAA are as follows. Instead of just one starting point and one ending point of a Markov chain in SAA, there are many starting points and ending points to construct many search paths. The starting points are selected from the old generation P(K) based on their fitness values, and the end points are placed in P'(K+1) to become candidates for the population of the new generation P(K+1). The GA's P'(K+1) is operated upon by SAA to obtain the new ending points placed in P(K+1), and these new ending points have lower average cost than the old ending points in P'(K+1). Finally, the new ending points become the population of the new generation. The important parameters affecting the efficiency of SAA are the initial temperature, the total length of the Markov chain, the move generation strategy, and the frozen condition. Concepts from GA theory are used to control these parameters.

The Annealing-Genetic algorithm (AGA) starts with a randomly generated population P'(0). Next, the Genetic Operators (GO's) are applied to produce a new population P(0) by rejecting the higher cost offspring so that the average cost of P(0) is less than that of P'(0). Then the search paths are generated from the points of P(0) at a small value of temperature T(0) until the first generation P(1) is created. The initial value of the temperature is

$$T(1) = (\text{the highest cost - the lowest cost})/(0.5 \text{ population size}).$$

Starting off with the point of lowest cost in the old generation, a next point is generated from the current point by the move generation strategy. It is accepted by the Metropolis criterion ($Pr=\exp(-\Delta C/T(K))$; ΔC=the highest cost - lowest cost; $T(K)=-\Delta C/\ln Pr$; $Pr=0.6$; $T(K)=-\Delta C/\ln 0.6=2a$), the next point not only becomes the current point but also is a member of the next generation. The process is continued until the new population is generated. Finally, when 80% of the population in a certain generation has the same cost as the solution vector, the

frozen condition is signalled.

Assume, that every bin has capacity γ, and that there are n weights in the list $0<a(i)\leq\gamma$ for $1\leq i \leq n$. The sum of the a(i) packed in a bin B(j) is called the content of B(j) and denoted by W(B(j)). The quantity $[\gamma$-W(B(j))] is called the gap of B(j), and is denoted by Gap(j). Then, $Gap(j)=\gamma$ denotes an empty bin, Gap(j) = 0 denotes a full bin, and Gap(j) < 0 is an illegal packing. Then

$$\text{COST}(B(j)) = \begin{cases} \text{Gap}(j), & \text{if } \text{Gap}(j) \geq 0 \\ \text{Gap}(j)^2 + \text{penalty cost}(T(K)), & \text{if } \text{Gap}(j) < 0. \end{cases}$$

Penalty cost increases as the temperature decreases. The assignment vector A[1:n] denotes all weights to bins

$$\text{COST}(A) = \text{sum}(j=1,m)\ \text{COST}(B(j)).$$

There are two strategies to be used in an algorithm for generating the next point of the search path. The first one is the GREEDY MOVE (GM) and the second is the SWAPPING MOVE (SM). In a GM, a weight a(i) is randomly selected from the bin, say, B(1), which has the largest gap among all bins. Next, this weight is assigned to another bin, say B(2), in an attempt to decrease the current number of bins needed. If B(2) has enough room for a(i), move a(i) from B(1) to B(2). In a SM, we randomly choose two weights a(i), a(j) which have been assigned to bins B(1) and B(2) respectively, and then exchange their roles. In these moves, the change of costs is calculated by

$$\Delta C = (\text{COST B}(1') + \text{COST B}(2')) - (\text{COST B}(1) + \text{COST B}(2)).$$

The three GO's not only modify the structure of the population to create new structures, but also reduce the average cost of the population. However, no illegal packings are allowed. The GOs are performed in the following steps.

Step 1. Two parents are selected from the population based on their fitness values. Then the Crossover Operator is applied to produce their two offspring. The offspring must have lower costs than the average cost of the old generation; otherwise, the parents continue for the following steps.

Step 2. The reordering ("inversion") operator is applied to the parent to reorder its sequence, yielding a new cost. If the new cost is lower than the old one, the offspring is copied to the next generation; otherwise, the parents continue to the next operation.

Step 3. The mutation operator is applied to each parent, and the offspring is retained if its

cost is lower than the cost of the parent. Finally, the parents are copied to the next generation.

These steps are repeated again and again until the population of the next generation is created. The AGA has the best average performance among all the algorithms they reviewed [21]. The AGA can be viewed as a SAA with population-based transition and GO-based quasi-equilibria control, or it can be viewed as a GA with a Boltzman-type selection operator. It can be shown that the AGA has polynomial time complexity.

In [20], it is formulated in transportation terms: given a set of boxes of different sizes, how should one pack them all into containers of a given size in order to use as few containers as possible. Paper [20] present a genetic grouping algorithm for this problem. The problem is defined as follows: Given a finite set O of numbers (the object sizes) and two constants B (the bin size) and N (the number of bins), is it possible to "pack" all the objects into N bins, i.e. does there exist a partition of O into N or fewer subsets, such that the sum of elements in any of the subsets does not exceed B?

It is possible to use the number of bins required directly as the cost function to be minimized, but that is unusual in practice. The authors offered a novel cost function: maximize

$$f=\sum_{i=1}^{N}\frac{(fill(i)/C)^{k}}{N}$$

with N being the number of bins used, fill(i), the sum of sizes of the objects in bin i, C, the bin capacity and k, a constant, k > 1. In other words, the objective function to maximize is the average, over all bins, of the k-th power of "bin efficiency", measuring the exploitation of a bin's capacity. The constant k expresses our concentration on the well-filled, "elite" bins in comparison to the less filled ones; k=2 typically gives good results [20].

Bin packing belongs to the optimization (grouping) problems in which the aim is to group members of a set into a small number of families, in order to optimize a cost function. So bin packing can clearly be seen to be a grouping problem: the aim is to group the objects into families (bins) and the cost function f above indeed grows with the size (fill) and decreases with the number (explicitly via N, and implicitly via fill) of the families. The classical mutation operator would be too destructive once the GA begins to reach a good solution of the grouping problem. The main reason is that the structure of the simple chromosome (which the above operators work with) is much too object oriented, instead of being group (i.e. bin) oriented.

That is why the authors of [20] have chosen the following encoding scheme: the standard chromosome is augmented with a group part, encoding the bins on a one-locus-for-one-bin basis. For example, the chromosomes shown below, which would ordinarily be encoded only as the parts to the left of the colon (i.e., one locus for each OBJECT) would be expanded to the following encodings:

A D B C E B : B E C D A and
A A A B B B : A B

with the "group part" (one locus per BIN) written after the colon, indicating that there are a total of 5 bins in the first example, and two bins in the second.
The additional structure is utilized by a strongly restructured crossover operator, called Bin Packing Crossover (BPCX). The FF heuristic[3] serves well as the initial solution-generator for population initialization. Then crossover proceeds as follows: Consider the following group parts of two chromosomes to cross (recall that there is one locus per bin in the group part):

A B C D E F parent 1
a b c d parent 2

First, copies are made of the two parents (in order not to destroy the parents) and two crossing sites are chosen at random in each of them, yielding, for example

A | B C D | E F
a b | c d |

Next, the bins between the crossing sites in the second chromosome are injected into the first

A c d B C D E F

Now some of the objects appear twice in the solution and must be thus eliminated. Suppose some objects injected with the bins "c" and "d" also appear in bins C, E and F. The next step is to eliminate bins C, E, and F, leaving

A c d B D.

The elimination of those three bins, however, most probably eliminated some objects which were not injected with bins c and d. Those objects are thus currently missing from the solution. To fix this last problem, apply the FFD heuristic to reinsert them, yielding, say

A c d B D x,

---

[3] *First Fit*
Best Fit

where x represents one or more bins containing reinserted objects which did not fit into the previous bins (i.e., those to its left).

The child just constructed indeed inherited important information from both parents, namely bins A, B and D from the first and c, d from the second. Note, however, that bins A, B and D might not be exactly the original ones found in the first parent, because the FFD might have filled them up further with some of the objects reinserted in the last stage of the BPCX. First Fit Descending (FFD) first sorts the objects in order of decreasing size before applying the FF strategy. It works much slower than FF, but performs slightly better.

2.5.2. 2-D Bin Packing and Nesting

2-D Bin-Packing . Given a finite set of rectangular boxes $E=\{e_1,...,e_n\}$ with associated sizes $W=\{(x_1,,y_1,)...(x_n,y_n)\}$ such that $0\leq x_i,y_i\leq L^*$. Place without overlapping all or some of the boxes from E into the rectangular bin with sizes $X>L^*$, $Y>L^*$ such that the relation of the sum of box areas to the bin area is the maximum.

Smith has looked at the problem of bin packing arbitrary-dimensional rectangular boxes into a single orthogonal bin using a GA [25]. The problem is NP-hard. A solution is to represent the bin packing as a list of the boxes plus an algorithm for decoding the list into a bin packing. The list is readily mutatable (flipping boxes), and is amenable to a modified form of crossover.

The decoding algorithm takes any list of boxes and forms a legal packing. There are two decoding algorithms. The first is called Symbolic Layout IDE (SLIDE) PACK. It takes each box in order from the list, places it in one corner of the bin and lets it fall to the farthest corner away, as if under the influence of a gravity that only allowed it to move orthogonally. The effect is that a box will zigzag into a stable position in the opposite corner from which it was placed. Slide pack is fast as there is no backtracking, and it is simple to compute. Its time complexity is $O(n^2)$, where n is the number of boxes. (There are n! possible orderings of the list of n boxes.)

The second algorithm is called SKYLINE PACK. For each box in the list, in order, it tries the box in all stable positions, and in all its orientations in the partially packed bin. A stable position is one in which the box is tucked into a corner, or a cave formed by other previously packed boxes. The algorithm takes its name from the fact that it tours the skyline formed by the previously packed boxes to find the position in which the next box fits best. Skyline pack has time complexity $O(n^4)$.

Smith uses a modified crossover which takes the order of the boxes before the splice from the first list and the order of the boxes which remain to be packed from the second list after the splice point (see Fig. 3).

```
string 1:      1 2 | 3 4 5
string 2:      5 4 | 3 2 1
---------------------------
child          1 2   5 4 3
```

Fig. 3. An example of Smith's crossover.

One of the mutation operators Smith has experimented with is SCRAMBLE -- that is, randomly reordering some portion of the list. FLIP mutation to try different orientations of the boxes is necessary if the decoding algorithm does not try the box it is packing in all its orientations. The flip mutation operation may be applied discretely to boxes in the list.

A straightforward evaluation criterion is the ratio of the area of the boxes packed to the area of the bin. Smith [20] suggests a way to measure partial bin packing, which favors boxes filling in caves, especially if they fit tightly into the cave. There is some analogy here to gravitational effects, and indeed such an evaluation allows us to pack in space as if the boxes were attracted to each other. A practical disadvantage is that each time we run the process we will end up with a different packing. Note that paper [81] also has details of a GA on a two-dimensional bin packing problem.

Paper [22] treats 1-D, 2-D and 3-D packing problems. The 2-D BPP is intuitively defined as packing a finite number of 2-D objects, always squares or rectangles, into a 2-D bin of a given height and infinite length, minimizing the total length required. The 3-D BPP is an extension of the 2-D BPP; the objects usually studied are cubes or rectangular solids and the bin is has a square or rectangular base and infinite height or length.

In [22], the authors use the term stochastic optimization as a generic term for optimization heuristics which include such approaches as GA and SA, and their algorithm, which draws upon both of the former. They define a figure as right-angled polygon loosely resembling one of the block letters. In [22] they assume that the length of each edge of a figure is a multiple of this unit length. As a result, each figure occupies an integral number of unit squares. A figure has a default initial orientation, but may be rotated 90, 180, or 270 degrees. A solution is a structure [(f(1),o(1)), ... , (f(N),o(N))]; for a given problem with N figures, each figure is numbered between 0 and (N-1). They use f(i), $0 \le f(i) \le (N-1)$, to represent the i^{th} of N figures; o(j), $0 \le o(j) \le 3$, is the figure's current orientation, and L is the length (cost) of the solution. Their algorithm has 5 components: (1) Initialization, (2) Selection, (3) Solution Generation, (4) Evaluation, and (5) Termination. The algorithm works with a population of solutions.

(1) The population is initialized randomly, i. e. they generate an integer between 0 and (N-1) for each f(i), and an integer between 0 and 3 for each o(i). Evaluation of each solution produces its length L. The population is then sorted on L, from best to worst.

(2) Biased selection of solutions is performed using a linearly biased random generator.

The better the solution, the higher the probability of its being selected. The probability of selecting the best solution of the population is B times greater than the probability of selecting the worst solution, where B is a parameter (a bias) provided by user. Values of B used range from 1.5 through 3.0.

(3). Generating a new solution from one or more solutions already in the population (called perturbation in SA) can be performed in either of two ways. First, a new solution may be "derived" from an existing solution. A solution S is selected from the population using linear bias. Several solutions in the "neighborhood" of S are randomly generated and evaluated. If the best of these neighborhood solutions is at least as good as S (i.e., the length of the best solution is equal to or less than the length of S), then the best solution is inserted into the population. Two solutions are defined to be neighbors if the sequence in which their figures are placed is identical except for exactly two figures. The second method simply generates a new solution randomly, as is done during initialization. The purpose of generating a solution randomly is to introduce new permutations, possibly very different from those present in the population, in order to prevent the population from converging prematurely.

(4). Evaluation of solution S involves assigning a location in the 2-D bin to each figure such that no two figures overlap. The figures are assigned locations in the order defined by the permutation. Evaluation of solutions is deterministic, and there exists some permutation which, when sent to the evaluation function, produces the optimal length.

(5). There are 3 conditions under which the process terminates:

*the population converges,

*no improvement in the best solution has occurred for a pre-specified number of iterations,

*a predefined maximum number of iterations is reached.

The algorithm is designed such that it can take advantage of multiple processors if available. The algorithm is divided into two independent processes: process 0 and process 1. Process 0 manages the population of solution: collecting the initial solutions during initialization, selecting solutions in order to form new ones, inserting new solutions, deciding whether a new solutions will be generated from an existing solution or whether an entirely random solution will be formed, and checking for convergence or termination. Process 0 does everything except evaluate solutions. Process 0 sends a solution to Process 1 for evaluation. Process 0 is a master process which farms out the time-consuming task of solution-evaluation to servant processes, each of which may be executing on a different processor.

Thus, [22] differs from many other approaches in their description of the 2-D packing problem, in the stochastic optimization algorithms they use, and in their use of multiple processors to reduce execution time.

Nesting formulation. Given a finite set of arbitrary-shaped elements $E=\{e_1,...,e_n\}$. Place without overlapping all or some of the elements from E into a rectangular bin with edges X, Y such that the relation of the sum of element areas to the bin area is the maximum.

Nesting in [29] is the process of selecting the optimal arrangement for a combination of 2-D

shapes on raw stocks in order to minimize material wastage while taking into consideration the constraints imposed by the cutting process. The automatic layout process adopted in the Patnest-Ship algorithm can be divided into three distinct steps:

(a) shape processing,
(b) local optimization, and
(c) rectangle packing.

(a) This process serves three main functions: First, it rejects all invalid geometries, and does not allow the boundaries defining the shape to cross. The shape processing routine will return to the user upon detecting any invalid geometries, for the necessary correction. Second, it transforms the input data (i. e. graphics elements described randomly) into a logical data structure that is suitable for use by the program. Third, it extracts information such as slope, length, change of direction from one edge to another, area, and other simple properties from the data structure describing the approximate geometry of the shape. A simple approach is a used to classify the input shapes. Shapes are classified as: floors; brackets-rectangular; brackets-trapezoidal; brackets-triangular, etc.

(b). There are two separate local optimization processes. The first aims at locating the smallest rectangular enclosure for clustering similar shapes. The second aims at squeezing "small" shapes into the void areas of "big" shapes. Some pre-defined arrangements include: identical sides in contact; rotating one part by 180 degrees and laying side-by-side; laying side-by-side. In cases where more than one pair of similar plates needs to be nested, the paired plates are laid side by side until they touch. Hence an analysis of the most efficient arrangement for each class of plates was made and the most efficient arrangement for each class of plates was identified. The second method used to fit the "small" pieces into the void spaces of the "big" pieces is similar to the "rectangle packing" approach with a few added considerations. They include: the irregularity of the boundary imposed by the presence of the "big" plate as a constraint; and the presence of multiple void areas. Note that an additional check using the actual geometry of the small shape is attempted even after the enclosing rectangle has failed.

(c). The steps involved in the "rectangle packing" process are as follows:

* sort the rectangles to be packed according to the criterion selected,
* place the first rectangle at one corner of the stock sheet,
* identify the pivot points created as a result of introducing a new rectangle. Pivot points are corners created by neighboring boundaries of rectangles or the stock sheet,
* select the pivot point which gives the smallest resulting rectangle. Check that the newly placed rectangle does not overlap with the boundary of the stock sheet or with a rectangle placed earlier,
* repeat step 4 for all remaining rectangles until no more new rectangles can be laid on the stock sheet.

Three different criteria were used to sequence the order of the plates to be nested: packed in order of area, packed in order of length of the longer side of the enclosing rectangles, and the

length/breadth ratio of the enclosing rectangles. Patnest-ship runs on a SUN SPARC workstation. The Patnest-AutoCAD pre-processor provides facilities for users to input part geometry by using AutoCAD's drafting commands. It then converts these geometries into a specially formatted file. After the automatic nesting process, the Patnest-AutoCAD post-processor can be used to interactively modify the layout produced. The algorithm is based on the assumption that ship/offshore structural plates can be grouped into several classifications, each with an "optimal" arrangement for multiple similar pieces. By restricting the variety of shapes to be manipulated, the authors managed to reduce the problem size. However, Patnest-ship may provide rather poor solutions if highly irregular shapes are present.

One of the most critical topics in [30] is the actual nesting, or positioning, of the part blanks onto the metal strip or sheet stock. They developed an automated system based on mathematical programming techniques which optimize blank nesting for a continuous strip stamping processes. Efficient nesting of blanks on a coil of metal is essential to reduce the amount of scrap produced. Today, most such nesting is done manually [30]. The authors formulate the problem by first describing the geometry of the part or parts to be nested for stamping and specifying on initial layout. They use a novel integer grid technique to efficiently and accurately compute the overlap between parts and then apply a Simulated Annealing Algorithm (SAA) to determine a new part layout with zero overlap and minimal scrap. They assume no restriction on the shape of the blank. The shape of the blank is provided by an ordered list of points on the boundary that are connected by straight lines. Even for very simple shapes there are two or more nesting arrangements that are locally optimal. A locally optimal configuration has the following property:

there exists a d(0) such that for all $d<d(0)$

$$f(Y(0)+-d)\geq f(Y(0)) \qquad (1)$$

where f is the objective function, Y(0) is the locally optimal configuration, and f(Y(0)) is the local minimum. The global minimum satisfies (1) for all d(0).

The description of the blank is provided by an ordered set of points on the boundary of the blank, which are connected together by straight line segments. They desire to minimize the engineering scrap: SCRAP = l * w - sum(i)AREA(i), where l, w are length and width on a coil. The objective function is thus OBJ = pressure * SCRAP + penalty * OVERLAP. The weight associated with the SCRAP cost in the objective function is analogous to pressure. They take the pressure =1. The neighborhood structure defines a set of configurations from which the next move is picked at random.

They use a SAA (Simulated Annealing Algorithm) technique to allow for an increase in the value of the objective function in a controlled manner, as opposed to a downhill-only iterative improvement technique, which only accepts moves that result in an immediate improvement in the objective function. The following terminology is used to describe algorithm. A feasible configuration, (K), is a point in the allowable region; the controlling parameter, (T), is the quantity analogous to temperature in annealing of solids; the cost C(K) is the quantity to be minimized; the neighborhood of a configuration is a set of predefined feasible points from which

the next configuration is picked randomly; the Metropolis criterion is the acceptance/rejection condition; a move is the process of picking a new configuration and applying a Metropolis criterion; a cooling schedule specifies how the temperature is decremented and how many moves are performed at each temperature T; the cost function is said to be in a state of equilibrium at T when the probability (Pr) of being in configuration K is $Pr(config.=K) = e^{-C(K)/T)/Z(T)}$, where $Z(T) = sum(j)e^{-C(j)/T}$ is a normalization factor.

The Metropolis acceptance criterion can be described as follows. If i is the current configuration with cost C(i), then the probability of accepting j as the next configuration, with $\lambda C = C(j) - C(i)$ is $Pr\{new=j|current=i\}=1$, if $\lambda C \leq 0$; or $=e^{-\lambda C/T}$ otherwise. The SAA is started at a "high" T and a Markov chain of configuration is generated by randomly picking a configuration from the neighborhood set of the current configuration in the chain. The cooling schedule specifies how many moves are attempted at each temperature and how the T is reduced. In summary, acceptance of intermediate configurations with higher costs is the strength of this algorithm. This property allows for "hillclimbing" moves, that is, climbing out of local minima. During computation of the overlap area, the interior is first discretized into triangles and then each triangle in the first body is intersected with each triangle in the second body. They find the boundary of the overlap region and then compute its area. For the nesting of one blank, the objective function is a function of two variables, configuration (C) and length (L). In all the blank nesting problems the T is reduced by a factor of 0.9 -- that is, $T(k+1) = 0.9*T(k)$.

Article [28] describes a typology of cutting and packing (C&P) problems(C&PP). C&PP appear under various names in the literature:

* cutting stock or trim loss problem;
* bin packing,dual bin packing, strip packing, vector packing, knapsack (packing) problems;
* vehicle loading, pallet loading, container loading and car loading problems;
* assortment, depletion, dividing, layout, nesting, and partitioning problems;
* line balancing, memory allocation, scheduling problems, capital budgeting, change making etc.

A typology founded on the basic logical structure of C&PP is developed in [28]. There are two groups of basic data whose elements define geometric bodies of fixed shapes (figures) in a one-or more-dimensional space of real numbers: the stock of the so-called "(large) objects", and the list or order book of the so-called "(small) items". The C&P process realizes patterns being geometric combinations of small items assigned to large objects. C&P can also be considered in an abstract, generalized sense taking place in non-spatial dimensions. Examples: knapsacking and vehicle loading for the weight dimension; assembly line balancing and multiprocessor scheduling for the time dimension; capital budgeting for financial dimension; computer memory allocation for data storage dimensions.

The most important characteristic of C&P is dimensionality. Elementary types are: 1-D, 2-D, 3-D, and multi-D problems. A 4-D problem might be obtained when a 3-D BPP in space has time as the fourth dimension. Another main characteristic is the way of measuring the number of large objects and small items respectively: discrete (or integer) measurement; or continuous (fractional) measurement. The combined type of 1-D problem with continuous

measurement is often called "one-and -a-half-dimensional" (1.5-D). The figure of an object or an item is defined as its geometric representation in the space of relevant dimensions. A figure is determined by its: form, size, and orientation. For multi-dimensional problems, an important question is whether the form of the figures is regular or irregular. The classification is given by the shape(s) and number of permitted figures. The availability characterizes the quantity of objects and items considered. It refers to lower and upper bounds on their quantity; their sequence or order; and the date when an object or item can be or has to be cut or packed. Four important groups of pattern restrictions are identified:

* minimal or maximal distances between small figures or between cuts dividing large figures are often important.
* the orientation of the small figures relative to each other and/or to the large figure may have to be taken into account.
* there may be restrictions with respect to the frequency of small items or figures in a pattern, especially regarding the combination or number of different small figures or the number of small items, be it in total or relative to certain figures.
* the type and the number of permitted "cuts" are essential, particularly if the objects and items are of rectangular or block form.

The objective function of C&P problems often has geometrical as well as combinatorial aspects. Some include both aspects, some none. Distinct kinds of criteria appear, depending on whether they refer to the

* quantities of large objects or small and residual pieces assigned to patterns;
* geometry of the patterns (layout-optimization), or
* sequence, combination or number of pattern.

It is typical for many C&P problems that more than one objective has to be considered. Combined types include:

1. Dimensionality: 1-D, 2-D, 3-D, N-D with N>3.
2. Kind of assignment: all objects and a selection of items; a selection of objects and all items.
3. Assortment of large objects: one object; identical figures; different figures.
4. Assortment of small items: few items (of different figures); many items of many different figures; many items of relatively few different (non-congruent) figures.

By combining the main types, one obtains 4*2*3*4 = 96 different types of C&P problem. It becomes obvious that line balancing, multiprocessor scheduling, and memory allocation belong to the same combined type as the classical bin packing problem. Solution approaches include:

1.Object or item-oriented.
    1.1 Branch and bound, dynamic programming;
    1.2 Approximation algorithms.
2. Pattern-oriented.
    2.1 One-pattern (knapsack algorithms);
    2.2 Several patterns (linear programming (LP)-based and general heuristics).

Approaches of the first type (object or item-oriented) immediately assign items to objects.

Approaches of the second type (pattern-oriented) first construct patterns and then assign large objects as well as small items to some of these patterns. The two main pattern-oriented approaches can be distinguished as: heuristics and those based on LP relaxations. The quality of a heuristic heavily depends on the problem-specific choice of "good" patterns. The LP-based approximation algorithms first solve the LP-relaxation of the pattern-oriented model and then search for an integer solution by more or less sophisticated strategies, usually by simply rounding up.

2.5.3. 3-D Bin Packing

Problem formulation. Given a finite set of rectangular 3-D boxes $E=\{e_1,...,e_n\}$ with associated sizes $W=\{(x_1,,y_1,,z_1)...(x_n,y_n,z_n)\}$ such that $0\leq x_i,y_i,z_i<L^*$. Place without overlapping all or some of the boxes from E into a rectangular 3-D bin with dimensions $X>L^*$, $Y>L^*$, $Z>L^*$ such that the relation of the sum of box volumes to the bin volume is the maximum.

In paper [82], the authors extended the classic Bin Packing problem to three dimensions. They investigated the solutions for the 3-D packing problem using first fit and next fit packing strategies with and without GA's. They studied several existing crossover functions for GA's: PMX, Cycle, and Order CO. They presented a new crossover function, Rand1. The GA was tested using a randomly generated initial population pool and using a seeded initial pool. The seeded pool was generated from a package (small item) ordering produced by rotating and sorting the packages by decreasing height. In [82], it is shown that the seeded GA using Next Fit and PMX produced the best overall results for the data sets tested. The seeded GA using Next Fit and Order CO provided the best results considering both rapid execution time and packing efficiency.

In paper [83], a GA for macro cell placement problem is presented. The algorithm is based on a generalization of the 2-dimensional bin packing problem. The genetic encoding of a macro cell placement and the corresponding genetic operators are described. The algorithm has been tested on a benchmark, and the quality of the placements produced is comparable to other published results for the benchmark.

In [20], the authors present GA's for two NP-hard problems, the bin packing and line balancing problems. They define a cost function suitable for the bin packing problem, and show that the classic GA performs poorly on this problem. They present an improved representation fitted to these problems. Efficient crossover and mutation operators are introduced for bin packing. Results of performance tests on randomly generated data are included.

Paper [26] describes a way to create a multiple-chromosome GA that performs well on a 3-D packing problem involving regular-shaped boxes of different sizes and weights into larger containers. The heuristic solution developed, called SMILE [84], is a layer-by-layer scheme that scheme that finds the appropriate boxes in the next layer. They combine the multiple-chromosome GA and SMILE to solve the 3-D BPP. Let N the total number of boxes to be packed, L the total number of layers, LEN the length of the container, WID the width of

the container, VOL(i) the volume of box i, and MAXH(L) the height of packing when the packing job is done. They assume that LEN is greater or equal to WID for both containers and boxes. Thus the packing cost, objective function, can be expressed as follows:

minimize C=(MAX(L)*LEN*WID)-sum(i=1,N)(VOL(i)),

subject to the constraints:

*Each box must occupy a unique and contiguous space.
*Each space is assigned to at most one box.
*The heavier boxes are placed below the lighter ones.
*All unit cubes assigned to a box must be contiguous and partially sum to the shape of the box.
*A box has a certain side that is the right side up.

A hybrid GA (HGA) has 5 components that must be designed: Representor, Creator, Evaluator, Generator, Decoder.

Representor. In this application, the chromosomes are simply treated as permutations of the list of boxes to be packed. In [26], a multiple-chromosome representation is proposed, instead of a single chromosome, to represent an individual. An individual represents one layout and each chromosome represents one layer. The system selects the high performance individuals for the next generation, but does crossovers and mutations only among homologous chromosomes and without intra-chromosome exchange. In other words, crossovers and mutations are done layer by layer for single or paired individuals.

Creator. The initial population pool is another basic components of GAs. First, they sort the boxes by their weight; if the weight of two boxes is the same, then the larger volume one has higher priority. Second, they search the available space for these boxes according to the sorted list to construct the first feasible layout, which is used as the first population, so they guarantee at least one feasible layout can be found. Third, they do mutation on chromosomes based on the existing individuals, until N individuals, a predefined limit taken to be the same as the problem size, have been created.

Evaluator. Interpretation is a minimization problem. They take the objective function to be the fitness value of an individual so that the preferred solutions have smaller fitness values. When two individuals have the same fitness value, a vertical cross penalty will be considered as the second criterion in terms of the quality of the population.

Generator. The selection of parents to reproduce uses roulette wheel selection. Since this is a minimization problem, individuals with smaller fitness values have higher probability of being selected as the parents for the next generation. The arithmetic inverse is used to adjust the fitness value of population when the roulette wheel selection method is applied. The order of the chromosome is very important in assuring that every box has been assigned to one and only one location. All operations that modify the individuals in the population must be performed in a structure-preserving way. The partially mapped crossover method is used to perform crossover of two selected individuals. This information is used by GAs to select high-quality chromosomes according to their performance and to cross these notions with many other high performance

notions from other strings to produce the next generations. Mutation operations carry out local modifications of chromosomes and are needed because even though selection and crossover effectively search and recombine extant notions, they may become overzealous and lose some potentially useful genetic material. They also use an "inversion" or reordering operator. Inversion acts by partially shuffling the string components. Reproduction is used to generate the next generation after crossover and mutation are performed on the selected individuals. To guarantee that the best individual is not lost during these operations, an elitist policy is adopted. In other words, if no better individuals has been discovered between generations, the elitist policy simply carries forward the most fit individual from the previous generation into the next. Reproduction will maintain the same number of population from generation to generation without losing the best one so far.

Decoder. An ordering list of boxes string does not directly represent a packing layout. It is important to transfer the ordering string into a feasible 3-D layout. Heuristic rules will be used when generating the layout according the order of chromosomes. The best individual, which has the minimum objective function among these individuals from the first generation to last generation, will form the final layout of the packing.

2.5.4. Compaction

Problem formulation. Minimize the area of the layout, while preserving the design rules and not altering the function of the circuit; both x and y coordinates of elements can be changed simultaneously.

In [78], the genetic algorithm evolves populations of strings, the length of which is not fixed. New individuals are produced by a stochastic mix of the classical genetic operators [4-6]: crossover, mutation and inversion. The layout problem may be thought of as a form of 2-D bin-packing [78]. A collection of rectangles is to be placed in the plane to satisfy certain design rules and minimize some cost function. The simplest version of this problem has:

- rectangles of fixed sizes,
- a design rule such that distinct rectangles should not overlap,
- cost given by area of bounding box.

This version of the problem is already intractable. Suppose we satisfy the constraint that the distinct rectangles, p, q should not overlap, by stipulating that one of the 4 elementary constraints: p above q, p below q, p left q, p right q is satisfied. Then for n rectangles, we have $N=n^2-n$ pairs and, a priori, 4^N elements in our search space of layout strategies. This approach considers layout strategies made up of consistent lists of elementary constraints. The rectangles are placed in the first quadrant of the plane as close to the origin as is consistent with the list of elementary constraints.

Populations of consistent lists of constraints are evolved using various orderings for selection [78]. The simplest criterion attempts first to remove design-rule violations and then to reduce the area of the layout. Strategies with fewer violations beat those with more and, for

those with the same number of violations, strategies with smaller bounding boxes win. This simple prioritization of concerns has led to the generation of some unpromising strategies -- while the selection criterion was busy removing design rule violations, for example, any strategy with few such violations (compared to the current population norm) was accepted. Fourman found that the performance of the algorithm was improved by introducing a selection favoring shorter chromosomes. His algorithm selects a criterion randomly each time it has a selection to make. Each time the algorithm is asked to compare two individuals, it non-deterministic chooses one of these criteria and applies it, ignoring the others. The resulting populations show greater variability than when a deterministic selection is used involving all criteria simultaneously.

Then, Fourman considers a symbolic layout of blocks connected by wires. The rectangles (blocks) are of fixed size and may be translated but not rotated. The interconnected lines (wires) are of fixed width but variable length. A surface level deals with tiles of three kinds---blocks, horizontal wires, and vertical wires. In addition to evolving layout constraints dealing with the relative positions of tiles (above, right of etc.), Fourman uses a fixed list of structural constraints, to represent the information in the symbolic layout, and fundamental constraints, which represent the size limitations of tiles.

Structural constraints: v crosses h, Nbv, Sbv, Ebv, Wbv. Here v, h are the vertical and horizontal wires and b is a block. These constraints allow stipulation of which wires cross (and hence are connected) and which wires connect to which edges (North, South, East, or West) of which blocks. At a deeper level, unseen by the user, the problem is represented in terms of the primitive layout elements, north b, south b, east b, west b, left h, right h, y posn h, top v, btm v, and x posn v.

For each tile, Fourman generates a list of fundamental constraints expressing the relationship between the primitive layout elements arising from it. Again, he evolves lists of layout constraints. These are compiled, together with the fixed structural and fundamental constraints representing the symbolic layout, to give graphs of constraints on the primitive layout elements, whose positions are thus determined. The number of design-rule violations and the area of the resulting layout are again used to select between rival strategies.

The algorithm appeared to get stuck for long periods on local minima (in the sense that one (non-optimal) configuration would dominate the population). This lack of variation in the population reduced the usefulness of crossover. When mutation led to a promising new configuration, there would be a period of experimentation leading rapidly to a new local minimum.

The genetic algorithm may be viewed as a (non-deterministic) machine which is programmed by supplying it with a selection criterion---an algorithm for comparing two lists of constraints. Fourman experimented with various selection criteria based on combinations of total intersection area of overlap involved in design-rule violations, and the area of a bounding rectangle. Fourman also implemented the idea of having several weakly interacting populations running in parallel.

In [85], compaction (CMP) is carried out simultaneously in both x and y directions. It exploits the full freedom to place blocks and wires in its search for an optimal solution. The layout CMP problem can be described as follows: Given the description of a layout, either in

symbolic form or as a stick diagram, they want to space the circuit elements and interconnections to minimize the total chip area. Only orthogonal structures are allowed. They proposed a search algorithm based on the technique of simulated annealing (SA). This method examines complete solutions one after another, while in a branch and bound algorithm, partial solutions are examined and expanded. An advantage of examining complete solutions is that it allows termination of search according to any time limit, since one can always keep track of the best solution encountered so far. Given a symbolic layout such as a stick diagram, they first convert it to a set of rectangular elements. The CMP algorithms then rearranges these rectangular elements to obtain a more compact layout satisfies all the design rules and preserves the given connection requirements of the circuit. A given SL consists of a set of elements which are blocks, horizontal connecting wires, and vertical connecting wires.

A placement is a specification of the positions of the rectangles in the plane. The positions of a rectangle can be specified by the coordinates of its lower-left and upper-right corners. Then a layout can be specified by the 4 sets of real numbers (X,Y) (X',Y'). The size of a layout is defined to be the area of the bounding rectangle, namely AREA = max{Z | Z from X or X'}*max{Z | Z from Y or Y'}. A valid layout is a choice of the values X, X'; Y, Y' so that a certain set of constraints is satisfied. There are 4 types of constraints: size constraints, overlap constraints, minimum distance constraints, and user-specified constraints. In [85], the constraints are classified as follows:

1. B, those constraints that MUST be satisfied, which include the size constraints, the overlap constraints, and the user-specified constraints, and
2. D, those constraints that are divided into groups such that at least (or exactly) one of the constraints in each group must be satisfied.

A valid set of constraints is a subset E, of constraints that contains all the constraints in B and at least (or exactly) one constraint in each group in D. The goal of 2-D CMP is to obtain a valid layout of minimum size. The key to obtaining the optimal CMP is to have an efficient procedure to choose the set of E of constraints. They propose a method that first reduces the size of the solution space and then uses the technique of SA to choose the set E of constraints.

In [85] they reduce the size of the solution space using some pruning techniques. The first pruning technique is to reduce the number of variables in the problem. The second pruning technique involves pruning some of the constraints in the groups in D. After the pruning phase, the algorithm proceeds to look for a set E of valid constraints. The method of SA is used for this purpose. It is also well-known that to be able to efficiently apply SA, we need the following key ingredients: a concise solution representation; a good neighborhood definition; a suitable cost function; and an annealing schedule. They represent valid layouts by using the corresponding valid sets E of constraints that they satisfy. Conversely, given a set E of constraints, they can apply the method in [52] to obtain a layout solution J(E) that is minimum with respect to E. For any solution E, they let C(E) = AREA(J(E)) be the cost function. The cost function can be computed using the longest path method [36,66]. Given a solution (B or M), they define a move as the operation of selecting a group in D and exchanging a constraint for one in that group. Two solutions (B or M) and (B or M') are said to be neighbors if M' can

be obtained from M by the interchange of the chosen constraint in one of the groups in D. It is possible to go from a given solution to any other via a sequence of moves.

The SA algorithm can start with any initial solution. To speed up the search, in [85] they have incorporated an initial search to look for a consistent set of constraints to be used as the initial solution E. In [85], they used a fixed ratio temperature schedule T(k) = r*T(k-1), k=1, 2, ... Experimental evidence [85] indicates that setting r to 0.9 produces satisfactory results. The SA algorithm involves many thousands of moves in its search for good solutions. The method used in [85] proceeds in two steps. The first step is an $O(n^2)$ pruning algorithm to reduce the size of the solution space. The second step employs simulated annealing to examine layout solutions one after another in its search for an optimal layout solution.

2.6. Conclusions

Compaction is the last layout subproblem in our phased approach to layout. The main purpose of compaction is to achieve independence from the specific fabrication technology. Today, the most commonly used approach to compaction is graph-based compaction. This framework provides a basis for 1-D and 2-D compaction. Heuristics exist for interrelating both dimensions efficiently during the compaction without solving the 2-D compaction optimally. Graph-based compaction handles only rectangular layout features. A large portion of compaction deals with intricate sets of design rules. Fabrication technology may dictate many complicated design rules. For ultrafast circuits, the die area is no longer the single function to be optimized -- we also need to involve the circuit delay or performance.

The bin packing problem is NP-hard in all of its many formulations. A few versions of 1-D compaction are NP-hard. Most versions of 2-D compaction are NP-hard. The difficulty of 2-D compaction lies in determining how the two dimensions of the layout must interact to minimize the area. To circumvent the intrinsic complexity of this question, some heuristics decide locally how the interaction is to take place. New perspectives for better solving of compaction have been discovered by applying various forms of genetic algorithms to the problem. But these GA approaches are far from perfected, as it usually is with first attempts to use a new approach on a difficult problem. Some of the weaknesses were described in section 2.4. For example, the algorithm in [78] appeared to get stuck for long periods on local minima; a practical disadvantage of [25] is that each time we run the process we will end with a different packing. Now, on the basis of newly published GAs and the new ideas of the authors, as described in section 1.1, we hope to continue the development of compaction, bin packing, and nesting using genetic algorithms.

Compaction and bin packing problems are closely related to each other. For example, 2-D bin packing can be regarded as 2-D compaction without some of its constraints -- all element sizes are fixed and there are no restrictions on connectivity of elements. In some sense, we can say that the bin packing problem is a less restricted compaction problem. So it seems rational to start with bin packing problems and then to extend the investigation to compaction.

In our future work, we are planning to generalize 2-D and 3-D formulations of bin

packing to make these problem closer to compaction. Then we are going to explore some of the many ideas concerning GA's (see, for example, Table 1 - Table 5), which have been found to be efficient for other applications or in nature, to see which may be most useful for solving bin packing problems. Finally, we want to try these useful ideas, selected during the bin packing investigation, for the compaction problem. The above-mentioned problems and our goals are represented in Table 6 - Table 9. We assume that using GA's will allow both layout and logic designers to explore more of the solution space to find the best possible design.

Table 1. Examples of Crossover Operators from Various Applications

Operation	Source	Comments
One-point CO	[4]	A schematic of one-point CO shows the alignment of two strings and the partial exchange of information using a cross site (point) chosen randomly.
Two-point CO	[86]	Two-point CO treats the string (chromosome) as a ring. Two unique points are selected at random, breaking the ring into two segments that are exchanged between the parents to produce two offspring.
Multi-point CO	[86], [87]	Multi-point CO is the extension of two-point CO. Like two-point CO, it treats the string as string which the crossover point cut into segments since the segments of the child must alternate between the two parents, there must be an even number of segments, and hence an even number of crossover points. By increasing the number of pairs of crossover points, it decreases the positional bias and it introduces a distributional bias.
Order CO	[6], [88]	Pass the left segment from parent 1. Construct the right segment by taking the remaining elements from parent 2 in the same order.
Enhanced order CO	[89]	Enhanced order CO proceeds almost the same as order CO. The difference is only that after two cut crossover points are chosen at random in the first parent, the second parent is rotated until the element just before the second cut point is the same as the element just before the second cut point in the first parent.
Partially mapped CO (PMX)	[5], [90]	The right segments of both parents act as a partial mapping of pairwise exchange to be performed on parent 1.
Cycle CO	[91]	Cycle CO performs recombination under the constraint that each locus (gene) comes from the identical position in one parent or the other, and it thus tends to preserves absolute position of each locus to the maximum extent feasible, while sampling features of both parents approximately equally.
Heuristic CO (an example)	[92]	The heuristic CO in [92] constructs an offspring for a traveling salesman problem (TSP). Pick a random city as starting point for the child's tour. Compare the two edges leaving the starting city in the parents and choose the shortest edge. Continue to extend the partial tour. If the shorter parental edge would introduce a cycle into the partial tour, then extend the tour by a random edge. Continue until a complete tour is generated.
Pattern CO	[93]	Pattern CO is achieved by replacing the alleles (genes) in a string from schema (for example) A, with the corresponding genes of schema B and vice versa.
Punctua-ted CO	[94]	To the end of each string, attach another bit string of the same length. The bits in the new section are interpreted as crossover punctuation (1 for yes, and 0 for no). The bits from each parent string are copied one-by-one to one of the offspring from left to right.

Segmented CO	[86]	Segmented CO is a variant of multi-point CO, which allows the number of crossover points to vary. Instead of choosing in advance a fixed number of unique crossover points, a segment switch rate is specified.
Uniform CO	[95], [96]	Uniform CO exchanges bits rather than segments. For each bit position in the string the bits from the two parents are exchanged with fixed probability p. Uniform CO removes the positional bias of traditional one-point CO. Its use, however, renders inversion useless and linkage (epistasis) meaningless.
Shuffle CO	[86]	Shuffle CO is similar to classic CO [4] except that it randomly shuffles the bit positions of the two strings in tandem before crossing them over and then unshuffles the strings after the segments to the right of the crossing point have been exchanged. Shuffle CO is designed to eliminate positional bias by having a schema disruption probability that is independent of schema defining length. Of course, its use renders inversion useless.
Analogous CO	[97]	Analogous CO is a modified CO designed to work with order-dependent production programs. In contrast to classic CO, which determines corresponding crosspoints according to their respective positions in the strings, analogous CO uses the phenotypic function of parameters as the corresponding cross point criterion.
Masked CO	[98]	Masked CO uses binary masks to direct CO. Masked CO is used to preserve schemata identified by the masks.
Position-based CO	[95]	A set of positions is random selected, but in this operator, the positions of elements selected in one parent are imposed on the corresponding elements in the other parent.
Edge recombination CO	[99]	This CO involves building a table of adjacent elements in each parent and then constructing a child using the adjacent information in the table. Edge recombination CO builds a child with elements that are almost always next to each other in one or the other parent.

Table 2. Examples of "Non-Standard" Architectures for GA-related Search.

Search	Source	Comments
GAMAS - migration and artificial selection	[100]	This paper presents an "improved" GA based on migration and artificial selection (GAMAS). GAMAS is an algorithm whose architecture is specifically designed to confront the causes of premature convergence. GAMAS is not concerned with the evolution of a single population, but instead is concerned with macroevolution, or the creation of multiple populations, and the derivation of solutions from the combined evolutionary effect of these populations. GAMAS claims to consistently outperform simple GAs and to alleviate the problem of premature convergence.
Metalevel GA's	[101]	Author attempts to determine the optimal control parameters for GA's. He describes experiments that search a parameterized space of GA's in order to identify efficient GA's. This search is performed by a metalevel GA. Author studies 6 parameters characterizing GA's: population size, crossover rate, mutation rate, generation gap, scaling window, and selection strategy. The metalevel GA uses this information to conduct a search for a high-performance algorithm. The experimental data suggests that while it is possible to optimize GA control parameters, very good performance can be obtained with a range of GA control parameter setting.
	[102]	They use the meta-genetic algorithm [101] to optimize a GA for cell placement. The three parameters optimized are the crossover rate, inversion rate, and mutation rate. They vary crossover rate and mutation rate during the optimization. The meta-genetic algorithm is itself a genetic optimization process, which runs a GA to solve a placement problem, and manipulates its parameters to optimize its fitness.
SIGH - stochastic iterated genetic hill-climbing	[96]	A search strategy called stochastic iterated genetic hillclimbing (SIGH), resembles both simulated annealing and GA. However, in SIGH, the convergence process is reversible. The connectionist implementation makes it possible to diverge the search after it has converged, and to recover coarse-grained information about the space that was suppressed during convergence. SIGH can be viewed as a generalization of a GA and stochastic hillclimbing algorithm, in which genetic search discovers starting points for subsequent hillclimbing, and hillclimbing biases the population for subsequent genetic search.
PE - Punctua-ted Equilibria	[103]	Uses two principles of the paleontological theory of Punctuated Equilibria (PE) - allopathic speciation and stasis. Allopathic speciation involves the rapid evolution of new species after geographical separation. Stasis implies that after equilibrium is reached in an environment, there is little drift in genetic composition. PE stresses that a powerful method for generating new species is to thrust an old species into a new environment -- that is, a new adaptive landscape, in which change is beneficial and rewarded. They chose this method for an optimal linear arrangement problem.

CHC	[104]	A nontraditional GA which combines a conservative selection strategy, that always preserves the best individuals found so far, with a radical (highly disruptive) recombination operator that produces offspring that are maximally different from both parents.

Table 3. Examples of Selection Methods and their Modifications and Properties.

Strategy	Source	Comments[4]
SSwR - Stochastic sample with replacement	[4], [5], [6], [87], [105].	Based on roulette wheel selection. In SSwR, the wheel is composed of the original expected values and remains unchanged between spins. SSwR provides zero bias, unlimited spread ($0 \geq N$), cost O(NlogN) and is not readily parallelizable.
SSwPR - Stochastic sampling with partial replacement	[105]	SSwPR provides medium bias, upper bounded spread, cost O(NlogN) and is not readily parallelizable.
RSSwR - Remainder stochastic sampling with replacement	[105]	In RSSwR, the fractional parts of the expected values are sampled by the roulette wheel method. The individual's fractions remain unaltered between spins, and hence continue to compete for selection. It provides zero bias, a lower bound on the spread, cost O(NlogN) and is not readily parallelizable. Any individual with an expected value fraction > 0 could theoretically obtain all samples selected during the fractional phase.
RSSwoR - Remainder stochastic sampling without replacement	[105]	RSSwoR also uses the roulette wheel for the fractional phase. However, after each spin, the selected individual's expected value is set to zero. Hence, individuals are prevented from having multiple selections during the fractional phase. It provides medium bias, minimum spread, cost O(NlogN) and is not readily parallelizable.
DC - Deterministic Sampling	[105], [106]	DS provides high bias, minimum spread, cost O(NlogN), and is not readily parallelizable. The result of a DS is a minimum sampling error for each generation and a high overall bias. DS is not widely used.

4

RSIS - Remainder stochastic independent sampling	[105]	The fractional phase in RSIS is performed without use of the error-prone roulette wheel. This is accomplished by deterministically assigning offspring according to the integer part of the expected value, and using each fractional expected value as a probability of selection. It provides low bias, minimum spread, cost O(N). However, it requires traversing the population and making a stochastic decision for each individual.
SUS - Stochastic universal sampling	[4], [105]	SUS is a simple, single-phase, O(N) sampling algorithm. It is zero biased, has minimum spread and will achieve all N samples in a single traversal. The algorithm is strictly sequential.
RE - Reproductive evolution	[107]	RE is a heuristic method for improving GA search. RE, when used in conjunction with an exponential bias, focuses the search by biasing the allocation of reproductive trials toward schemata which are the most promising candidates for reproduction.
LR - Linear ranking	[108]	Selection in evolutionary algorithms is defined by selection (reproduction) probabilities p(s)(a(i,t)) for each individual within a population. Here a(i,t) is an individual in population P(t)={a(1,t),...,a(λ,t)}, t from N, λ>1. For LR probabilities p(s)(a(i,t)) = 1/λ(hmax-(hmax -hmin)(1-i)/(λ-1)), where hmin = 2-hmax and 1< hmax $\leq$2.
UR - Uniform ranking	[109]	For UR probabilities p(s)(a(i,t)) = 1/m, if 1$\leq$i$\leq$m or = 0, if m<i$\leq\lambda$.
PS - Proportional selection	[4], [109]	For PS probabilities p(s)(a(i,t)) = f(a(i,t))/sum(j=1,λ)(f(j,t), where f, the fitness function, provides the environmental feedback for selection. Many selection methods are specific implementations of PS.
Extinctive versus preserva-tive selection	[109]	The term preservative describes a selection scheme, which guarantees a non-zero selection probability for each individual; i.e., each individual has a chance to contribute offspring to the next generation. A selection scheme is called preservative if for each t$\geq$0, for each P(t)=(a(i,t), ..., a(λ,t)), for each i from (1, ..., λ) p(s)(a(i,t)) > 0. A selection scheme is called extinctive if for each t $\geq$0, for each P(t), there exists an i such that p(s)(a(i,t)) = 0.
Left- versus right-extinctive selection	[109]	A selection scheme is called left extinctive selection (LES) if for each t>0, for each P(t), there exists L from {1,...,λ-1}, i$\leq$1 => p(s)(a(i,t))= 0. A selection scheme is called right extinctive selection (RES) if for each t>0, for each P(t), there exists L={2,...,λ}, such that i$\geq$L => p(s)(a)i,t)) = 0. Of course, in any condition the sum (i=1,λ)p(s)(a(i,t))=1 must be satisfied.

Dynamic versus static versus proportional selection schemes	[4], [109]	A selection scheme is called static if there does not exist i from $\{1,...,\lambda\}$ such that for all $t \geq 0$, probabilities $p(s)(a(i,t)) = c(i)$, where $c(i)$ are constants. A selection scheme is called dynamic if for each i from $\{1,...,\lambda\}$, and for each $t \geq 0$ probabilities $p(s)(a(i,t)) = c(i)$. PS is a dynamic preservative scheme; LR realizes a static, preservative scheme; UR is a static and extinctive selection scheme.
ES - Elitist selection (the property of elitism)	[5], [87], [109]	In an ES scheme, some or all of the parents are allowed to undergo selection with their offspring. This might result in unlimited lifetimes of super-fit individuals. A selection scheme is called ES or k-elitist if there exists k from $\{1,...,\lambda\}$, such that for each $t > 0$ and for each i from $\{1,...,k\}$, $f(a(i,t)) * f(a(i,t-1))$, where * denotes the $\leq$ relation in case of minimization task and $\geq$ in case of a maximization problem.
Pure selection	[109]	A selection scheme is called pure if there is no k for $\{1,...,\lambda\}$ which satisfies the k-elitist property.
Steady- state versus generational selection	[5], [87], [109], [110]	SSS is a special variant of elitist selection in which the set of parents incorporated into selection is larger than the set of offspring, which is of size 1. In the case of SSS [110], an offspring immediately replaces a parent if it performs better. The set of prospective parents may change for every step of reproduction. In contrast, in generational selection, the set of possible parents remains unchanged until all λ offspring for that generation have been produced.
Incest prevention	[104], [111]	The IP mechanism is a relatively direct approach for preventing similar individuals from mating. Individuals are randomly paired for mating, but are only mated if their Hamming distance is above a certain threshold. The threshold is initially set to the expected average Hamming distance of the initial population, and then is allowed to drop as the population converges.
CHC's selection	[104]	CHC is a GA that combines elitist replacement-selection with an unbiased reproduction-selection strategy. CHC is able to moderate selection pressure. It eliminates the traditional selection bias for reproduction, and relies only upon the fitness-bias of replacement-selection. CHC is said to be able to use mating and recombination strategies that help maintain diversity.
Sharing functions	[5], [112]	Developed and investigated to permit the formation of stable subpopulations of different strings within a GA, thereby permitting the parallel investigation of many peaks (when on-line performance is important). A sharing function is nothing more than a way of causing a degradation of an individual's payoff due to presence in the population of a neighbor at some distance as measured in some similarity space. For multimodal problem spaces in which on-line performance is important, a GA with sharing is able to maintain stable subpopulations of appropriate sizes: the number of points in each cluster is roughly proportional to the peak fitness value in the neighborhood of the cluster.

GS, PhS - Genotypic and phenotypic sharing	[113]	When the proximity of individuals is defined in the decoded parameter space, it is called PhS. The use of sharing based on genotypic proximity is called genotypic sharing or GS. The genetic closeness of two individuals may be taken as the number of different alleles in their chromosomes (the Hamming distance between the strings, for binary representations).
CS - Crowding scheme	[87], [113]	Crowding can be used to modify many selection schemes, altering the strategy for selecting individuals to replace with new offspring. In crowding, separate niches are created by replacing existing strings according to their similarity to other strings in an overlapping population. Two parameters, generation gap (G) and crowding factor CF, are defined. G dictates the fraction of an overlapping population that is permitted to reproduce each generation. When selecting an individual to die, CF individuals are picked at random from the population, and the one which is most similar to the new individual is chosen to be replaced, where similarity is defined in terms of the number of matching alleles. In [87] CF=2 and 3 were found useful, with G=0.1.
FS - Fitness scaling	[5]	A linear scaling modifying "raw" fitness before selection. The raw fitness f and the scaled fitness f' are defined. Linear scaling requires a linear relationship between f' and f: f' = af + b, where a and b are the coefficients; they may be chosen in a number of ways. However, the researcher generally wants the average scaled fitness f'(avg) to be equal to the average raw fitness f(avg). To control the number of offspring given to the population member, one can use maximum fitness f'(max) = c(mult)f(avg), where c(mult) is the number of expected copies desired for the best population member. For typical small populations (n = 50 - 100) a c(mult) in the range [1.2 - 2} has been used successfully.
Preselection	[5], [114]	In this scheme an offspring replaces the inferior parent if the offspring's fitness exceeds that of the inferior parent. In this way, diversity is maintained in the population, because strings tend to replace strings similar to themselves (one of their parents).

Table 4. Mutation Operators (MO).

Operator	Source	Comments
SMO - Simple MO	[4], [5], [6]	SMO is the occasional (with small probability) random alternation of the value of a string position. In the binary coding this simply means changing a 1 to a 0 or vice versa.
CMO's - Cavicchio's MO	[5], [114]	CMO-1 changes a single pixel within a detector. CMO-2 changes all pixels within a detector. CMO-3 changes pixel associations between adjacent detectors.
Bosworth, Foo and Zeigler MO	[5], [115]	1. Fletcher-Reeves (FR) MO. It is a fairly sophisticated hill-climbing algorithm. In FR MO, approximate gradient information (obtained from 2r other function evaluations, where r is the number of real parameters) is used to determine the line of conjugate ascent, which is then explored using golden search. Not widely applied in practice. 2. Uniform random MO. 3. Quadratic gaussian approximation MO. 4. Cubic gaussian approximation MO. 5. Zero MO.
KA - Knowledge-Augmented MO	[5], [115]	Encodes multidimensional parameter optimization problem using real parameters. In [115], developed several MOs incorporating nonpayoff information. KA MO uses FR MO (a conjugate gradient method) and golden search together as a MO. The use of KA MO has not been restricted to MO.
FSMD - Finite-state machine diagram MO	[116]	The mode of MO is determined by the interval within which a number selected from a random number table lies. The intervals are chosen in accordance with a probability distribution over the permitted modes of MO. Additional numbers are then selected in order to determine the specific details of the MO. Thus, the offspring is made to differ from its parent either by an output symbol, a state transition, the number of states, or the initial state.
PB - Position-based	[117]	Two tasks are selected at random, and the second task is placed before the first.
OB - Order-based	[117].	Two tasks are selected at random, and their positions are interchanged (also called "swap" mutation.
SMO - Scramble MO	[117]	Under the assumption that the neighborhood of tasks in a task list is important, it chooses a sublist randomly, and scrambles the order of the tasks within the sublist.
MMO - Mass MO	[118]	It represents an attempt to adapt the GA to dynamically changing problems. A reasonable strategy would be to restart the GA periodically with a newly generated population (often randomly generated) independently of the previous solutions. The alternative that suggests itself is to include in an initial population for the incoming problem some (current or earlier population) of the evolving current solution modified by MMO.

UMWO - Unbiased-mutate-weights operator	[119]	For each entry in the chromosome, UMWO will, with fixed probability (p=0.1, for example), replace it with a random value chosen from the initialization probability distribution.
BMWO - Biased-mutate-weights operator	[119]	For each entry in the chromosome, this operator will, with fixed probability (p=0.1, for example), add to it a random value chosen from the initialization probability distribution.
SMO - Supervised MO	[93]	SMO combines extra features present in schema A with those of the strings in B'=B-A. Thus, it is spreading proven qualities in a controlled way to a larger population. If the superiority of one of the schemata is not evident SMO might be superfluous.
Varying probability of mutation	[115], [120]	4 mutation regimes are used. The first consists of a constant probability MO across all bits and over all generations, as is usually the case. The second decreases the probability of MO exponentially over generations while the third increases it exponentially over the bit representation of each integer. The fourth regime is a combination of the second and the third.
FMO - Flip MO	[25], [117]	FMO is used for solving bin packing problems. FMO tries different orientations of the boxes, which is necessary if the decoding algorithm does not try the box it is packing in all its orientations. FMO is applied discretely to boxes in the list.
MO BPP - A MO for the bin packing	[20]	The MO BPP is follows: Given a chromosome, select at random a few bins and eliminate them. The objects which composed those bins are thus missing from the solution and they are re-inserted in random order using a first fit algorithm.

Table 5. Inversion and Other Operators[5] and Some Descriptive Terminology.

Operation	Source	Comments
SINO - Simple (linear) Inversion Operator	[4],[5], [121], [122]	Two points are chosen along the length of the chromosome. The chromosome is cut at those points, and the order of the cut section is reversed.
L+E INO - Linear + end INO	[5], [121]	The L+E INO performs SINO with a specified probability (for example, 0.75). If SINO was not performed, end INO would be performed with equal probability (for example, 0.125) at either the left or right end of the string.
CINO - Continuous INO	[5], [121]	A CINO is applied with a specified inversion probability p(i) to each new individual as the individual is created.
MINO - Mass INO	[5], [121]	Mass INO's are designed to eliminate the proliferation of noninteracting subpopulations that accompanies strict-homologue mating.
ShINO - Shadow INO	[5], [121]	ShINO is used to retard disruption.
Deletion	[5], [123]	The loss of a chromosome piece is called Deletion. It can happen in several ways: one break near a chromosome tip, two breaks followed by loss of a small interior piece, two breaks followed by loss of both tips and formation of a ring chromosome.
Duplication	[5], [123]	Duplication may result when, following three chromosome breaks, a segment of one chromosome is inserted elsewhere in the homologous chromosome or into a different chromosome.
Translocation	[5], [123]	Breaks in two or more nonhomologous chromosomes, followed by reattachments in new combinations, is called translocation if one or more segments ends up on a different chromosome than it started. If the rearrangement of chromosome parts is complete, with no leftover pieces, the translocation is reciprocal.

5

Hybridization	[7]	An operation resembling a length-increasing crossover between genomes from radically different genotypes, each of which codes for a separate process activated by a distinct set of internal and external cues, which produces a new, composite genome that contains the code for both processes (frequently useful in genetic programming).
Segregation	[5], [123]	To form a gamete, we randomly select one of each of the haploid (a single) chromosomes. This random selection process is known as segregation. It effectively disrupts (i.e., with high probability) any linkage that might exist between genes on different (nonhomologous) chromosomes.

Table 6. 1-D bin packing problem.

Classical formulation	Characteris-tics	Algorithms	Short-comings	Goals
Given a finite set of elements $E=\{e_1,...,e_n\}$ with associated weights $W=\{w_1,...w_n\}$ such that $0 \leq w_i \leq w^*$. Partition E into N subsets such that the sum of weights in each partition is at most w^* and that N is minimum.	Although the most remote formulation from compaction among the problems to be studied, it is the most studied of the problems. Therefore very suitable for initial explorations of various GA techniques.	Classical heuristics [23,24]	Results may be quite far from the minimum.	To investigate new genetic ideas, operations, codings, selection methods, and structures (Table 1 - Table 5).
		Stochastic (simulated annealing + simple GA) [21,22]	Big CPU time, local optima, premature conver-gence.	

Table 7. 2-D bin packing problem.

Classic formulation	Characteris-tics	Algo-rithms	Short-comings	Goals
Given a finite set of rectangular boxes $E=\{e_1,...,e_n\}$ with associated sizes $W=\{(x_1,,y_1,)...(x_n,y_n)\}$ such that $0\leq x_i,y_i\leq L^*$. Place without overlapping all or part boxes from E into the rectangular bin with sizes $X>L^*$, $Y>L^*$ such that the relation of the sum of box areas to the bin area is the maximum.	Closest to compaction of the "classical" problems. A good formulation to start from in exploring GA's for compaction.	Classical heuristics [23,24]	Results are likely quite far from the minimum.	To generalize the formulation: sizes X and Y can be variables; a new criterion -- -the minimum of the bin area; and new kind of boxes -- with adjustable sizes and/or non-rectangular. To investigate new genetic ideas, operations, codings, selections, and structures.
		Simple GA's [22,25,78]	Big CPU time, poorly directed improve-ment.	
Given a finite set of rectangular boxes $E=\{e_1,...,e_n\}$ with associated sizes $W=\{(x_1,,y_1,)...(x_n,y_n)\}$ such that $0\leq x_i,y_i\leq L^*$. Place without overlapping all boxes from E into N rectangular bins with sizes $X>L^*$, $Y>L^*$ such that N is the minimum.	Generaliza-tion of the first formulation.	Classical heuristics [23,105]	Results are likely quite far from the minimum.	
		Simple GA [20]	Nonhomo-geneous represen-tation decreases CPU time.	

Table 8. 3-D bin packing problem.

Classic formulation	Characteris-tics	Algorithms	Short-comings	Goals
Given a finite set of rectangular 3-D boxes $E=\{e_1,...,e_n\}$ with associated sizes $W=\{(x_1,,y_1,,z_1)...(x_n,y_n,z_n)\}$ such that $0\leq x_i,y_i,z_i<L^*$. Place without overlapping all or part boxes from E into the rectangular 3-D bin with sizes $X>L^*$, $Y>L^*$, $Z>L^*$ such that the relation of the sum of box volumes to the bin volume is the maximum.	Closest formulation to compaction in the case of 3-D integrated circuit technology and MCM. Probably a good foundation on which to test construction of GA's for 3-D compaction.	Classical heuristics. [23,24]	Results are likely quite far from the minimum.	To generalize the formulation: sizes X and Y can be variables, and can add a new criterion -- minimization of some function f(X,Y), and a new kind of boxes -- with adjustable sizes and more complex shapes. Could aid in investigating new GA ideas, operations, codings, selection mechanisms, and structures.
		Simple GAs [82,22,26]	Complex crossover. Placement is done in layer-after-layer manner only.	

Table 9. Compaction problem.

Classical formulation	Characteris-tics	Algorithms	Short-comings	Goals
1-D formulation: Minimize the width of the layout, while preserving the design rules and not altering the function of the circuit; only x coordinates of elements can be changed.	See section 2.1.	Traditional approaches [2-40, 66,67,124-127]	Results are only locally minimal.	To generalize the compaction criteria (signal delay minimization) and introduce new operations (90^o turns and mirror reflections of elements). Develop the approach of [85] for the GA. To apply new GA ideas, operations, codings, selections, and structures that were successful for the bin packing problems.
2-D formulation: Minimize the area of the layout, while preserving the design rules and not altering the function of the circuit; both x and y coordinates of elements can be changed simultaneously.	See section 2.1.	GA's [78] and simulated annealing [85]	Many inequalities lead to long CPU time. Simple genetic operations.	

3. PROJECT DESCRIPTION

3.1. Project Statement

The focus of this research is on the solution of the integrated circuit layout compaction, bin packing, and nesting problems on the basis Genetic Algorithms (GAs). It has to enhance the linkage between symbolic level modeling of the layout and the mask level design of hierarchical, ultra fast, low power analog and logic circuits. The current status of the GAs to physical design link has several shortcomings. The dramatic increase in the complexity of VLSI designs has driven a rapid conversion from hand design to the extensive use of computer-aided layout editor to support the layout process. Symbolic layout tools free the designers from design rule consideration and allow them to focus on the topology of the design, thus increasing design productivity.

Compaction is the process of producing an area-efficient physical layout from a symbolic layout while enforcing the separation (design rules), electrical connectivity, and user-specified requirements. It is an active area of research in automatic VLSI layout design. Layout compaction is the process that takes an existing layout and produces a new layout while minimizing some geometric aspect (usually size and now it is also important to minimize the circuit delay). This process preserves the underlying circuit integrity and enforces design rule correctness. As the complexity of layout and demands to its quality increase, the corresponding increases in execution time and memory usage become a problem in many compaction systems if a good solution has to be found. Because of this, compaction results produced by non-evolution design tools usually produce the layout with the non-minimal area and they are not easily flexible to the change of the design rules.

3.2. Goals Statement

Most compaction, bin packing, and nesting algorithms implemented today have a high time computational complexity, $O(N^m)$, where $m>1$ and N is the problem size, and find non-optimal solutions. Therefore, it is very important to try to reduce the size of the problem and find new approaches that solve the compaction, bin packing, and nesting problems more effectively.

These problems may be viewed abstractly as problems of optimization in the presence of constraints. Genetic algorithms provide a means of guiding the search for fast good solutions. This project is based on the premise that the methodology of GAs will produce more good compaction. The project goals are to develop new methodologies, abstractions, models, and approaches which lead to more efficiently defined and utilized CAD algorithms and tools. This algorithms will include mechanism of natural evolution. The new methodologies will also incorporate important improvements in genetic algorithms and physical layout. This is to be accomplished by defining a new GA methodology and developing a useful set of rules related genetic algorithms and layout VLSI systems. It is asserted that, even though more information is to be considered during compaction, our strategy will tend to reduce layout design time due

to an evolutionary and adaptive to nature approach to generating layout in which a solution of the compaction problem found at each step is done using a non-random, knowledge-based initial population.

3.3. The Approach

3.3.1. The Representation

Let us first briefly examine the effects of the classic genetic operators on examples of straightforward structures relevant to the compaction (bin packing) problems. We will show why we think this is not the best structures for these problems.

First of such structures is the following straightforward encoding scheme, in which we represent explicitly the position of each element of the electronic circuit via x and y coordinates of (for example) its top left corner. For example,

$(x_1,y_1)(x_2,y_2)...(x_n,y_n)$

would encode the solution where the first element (component or fragment of connection) is located at the position (x_1,y_1), the second at the position (x_2,y_2), and so on.

An advantage of this representation is the simplicity with which one can go from this encoding to the positions of the elements on the layout. A disadvantage is that the crossover and mutation operations will produce illegal solutions (violations of design rules) in most cases, and the power of the GA will be seriously impaired. Moreover, it is difficult to modify GA operators so as to preserve

1) the contact between to rectangular elements (fragments) of the same connection; and

2) "good" neighboring elements from the standpoint of connectivity (i.e., keeping elements with many connections between them "close" on the chromosome and on the layout. Another disadvantage is that elements which are close to each other in the layout might be separated in the chromosome. The standard two-point crossover would not tend to preserve valuable schemata.

Second of the very straightforward representations is the order-based encoding scheme. Such a representation simply consists the set of possible permutations of the elements that correspond to the electronic components, together with some heuristic "decoding" procedure for their placement given a particular ordering. The output of the heuristic mechanism is the actual layout (placement of components together with connections) corresponding to the given chromosome. The decoding mechanism usually proceeds by considering the elements one by one in the order specified on the chromosome, and placing them and their corresponding connections in accordance with the design rules within the area available. Of course, the decoding procedure

does not, in general, guarantee optimal decoding of any particular sequence of component; for example, it might involve a rule which tries to move each new component as far to the top and left as is allowed by the components already decoded, using the design rules.

A disadvantage of this type of representation is that the ordering crossovers will tend to utilize context-dependent information out of context during recombination. Indeed, given the mechanism of decoding the chromosome, it is clear that the meaning of a gene on the chromosome depends heavily on all the genes that precede it on the chromosome. If, for example, we swap two adjacent genes (elements) in the chromosome, it will very likely lead to widespread changes in the layout, for all elements "downstream" of the first element of the two swapped. So the standard GA operations does not effectively preserve useful schemata ("building blocks" are not used well).

In the proposed work, we will investigate a hierarchical chromosome representation (HCR) for layout problems, including both compaction and bin packing problems. This representation is designed to preserve building blocks in chromosomes. An example from this family of representations will be described briefly here. In most cases, we will consider only 2-level representations, for simplicity, but the results generally hold for a k-level HCR, k>2, as well. The encoding scheme makes uses genes to represent groups of circuit elements. The rationale is that in the problems being considered, it is the groups of elements and their location which are the meaningful building blocks, i.e., the smallest piece of a solution which can convey information on the expected quality of the solution they are part of. This is crucial: indeed, the very idea behind the GA paradigm is to perform an exploration of the search space, so that promising regions are identified, together with an exploitation of the information thus gathered, by an increased search effort in those regions. If, on the contrary, the encoding scheme does not allow the building blocks to be exploited (i.e. transmitted from parents to offspring, thus allowing a continuous search in their surroundings) and simultaneously to serve as estimators of quality of the regions of the search space they occupy, then the GA strategy inevitably fails and the algorithm performs in fact little more than a random search or naive evolution.

In 2-level representation, each chromosome might consist (for example) of several groups G1,G2,...,Gm of elements (or subgroups for k-level HCR) and their relative (within the group) coordinates. For instance, an example chromosome might be (for convenience of reading, we put each group in a separate row or rows)

<
{ [2] }(x_{G1},y_{G1}), // group G1 of one element, level 0
{ [1(x_1,y_1)], [4(x_4,y_4)], [7(x_7,y_7)] }(x_{G2},y_{G2}), // group G2 of three elements, level 1
{ [3(x_3,y_3)], [5(x_5,y_5)] }(x_{G3},y_{G3}), // group G3 of two elements
{ [6(x_6,y_6)], [8(x_8,y_8)], [9(x_9,y_9)] }(x_{G4},y_{G4}) // group G4 of three elements, level 1
>(0,0).

Fig. 4(a,b) illustrate this example.

A 3-level HCR chromosome might be

```
<
  {                                                    // beginning of group G5,  level 2

    { [2] }(x_G1,y_G1),                                // group G1 of one element, level 0
    { [1(x_1,y_1)], [4(x_4,y_4)], [7(x_7,y_7)] }(x_G2,y_G2) },  // group G2 of three elements,  level 1
  }(x_G5,y_G5),                                        // end of group G5
  {                                                    // beginning of group G6,  level 2
    { [3(x_3,y_3)], [5(x_5,y_5)] }(x_G3,y_G3),         // group G3 of two elements,  level 1
    { [6(x_6,y_6)], [8(x_8,y_8)] }(x_G4,y_G4),         // group G4 of three elements,  level 1
  }(x_G6,y_G6),                                        // end of group G6
  { [9] }(x_G7,y_G7)                                   // group G7 of one element, level 0
>(0,0).
```

Fig. 4(c,d) illustrate this example.

Each group of elements (subgroups) is a piece of layout -- elements and connections in some area (or bin, for the bin packing problem). At the beginning, when we are given some initial layout to be compacted, each group contains only one element. During the GA process, some groups are merged into new, larger groups (of a higher hierarchical level).

3.3.2. The Genetic Operators

It is important that the primary GA search operators now work not with elements, but rather with groups.

We can schematically describe an example crossover operator as follows:

1. Select at random some groups, Gf, from the first parent (father).

2. Select at random some groups, Gm, from the second parent (mother), which do not contain elements (groups) from GF.

3. For elements (or groups) that are not included into Gf or Gm, form a hierarchical substructure using heuristic algorithms, which can add expert knowledge and sound heuristics to the process.

4. Apply steps 1 through 3 to the two parents with their roles permuted in order to generate the second child (if desired).

By changing numbers of groups at steps 1 and 2, we can select the desired balance between the GA and the heuristic strategies. At step 3, we can, to the extend found to be desirable, use valuable information extracted from the initial (symbolic) layout in order to form groups. This information can also be used for forming the initial population, if desired.

It is also interesting to note that some undefined parameters of the heuristic algorithm can be treated as another set of variables for the GA to optimize. In particular, one of the authors

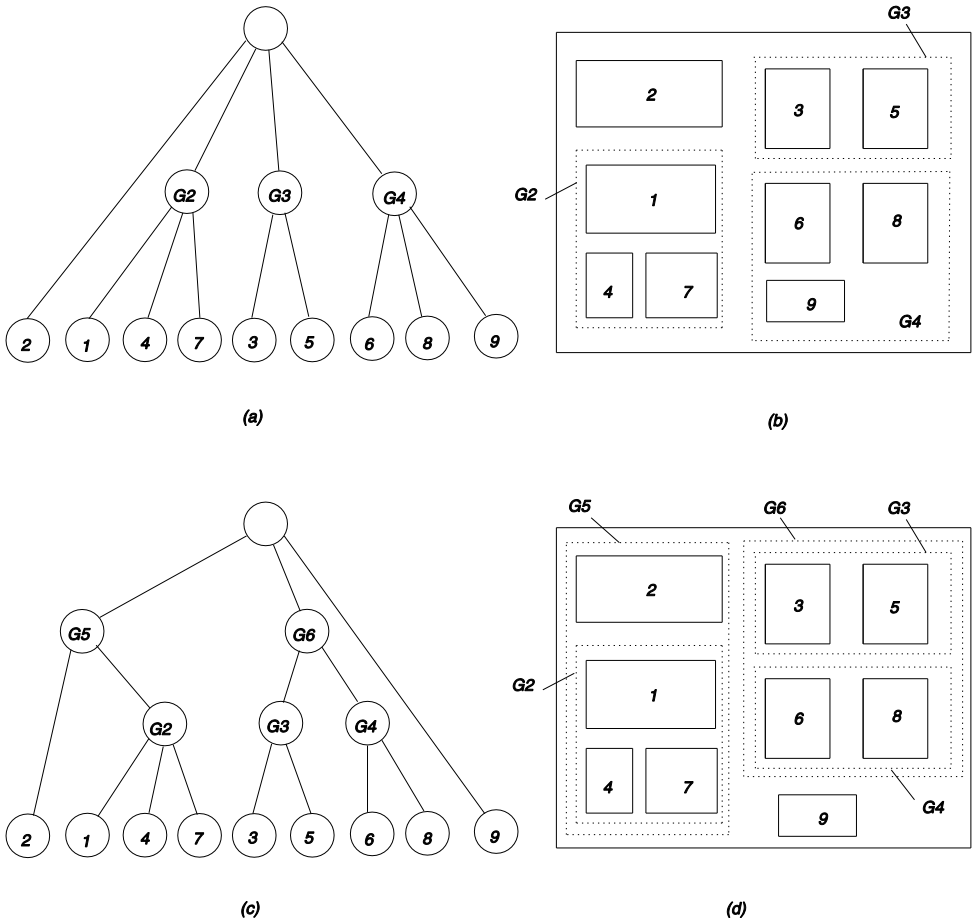


Fig. 4. Examples of HCR

[16] has developed several procedures for systematically extending arbitrary order-based (permutation-type) operators so as to search simultaneously for optimal values of a number of ordinary binary mapped, fixed-point variables. The GA architecture described below can, we believe, make effective use of this capability to improve the performance of the heuristics as it searches for optimal positions of the elements in the layout.

A mutation operator also should work with groups rather than elements. A general strategy for this operator is the "elimination" of an existing group, after which each element (subgroup) of the group becomes a separate group, or is redistributed among remaining subgroups.

For the proposed representation, we can use as the decoding mechanism (and for implementing step 3 of the crossover) one of the existing silicon compilers [2], or at least the methods of such a program. A silicon compiler uses a similar representation of the circuit, and it finds coordinates for each element (group) and each connection between elements (groups).

During each step of this compilation, two or more groups are merged into one new group, with simultaneous determination of relative coordinates of elements and connections within the group formed. Finally, all groups are merged into one, which represent the whole layout.

3.3.3. The Fitness Function

The simplest fitness function, F_G to use to estimate the quality of group G (or the whole layout) is the ratio of the area of the group layout which is effectively used (not wasted) to the whole area, A_G of the group layout:

$$F_G = \frac{A_G - A_d}{A_G},$$

where A_d is the "dead" area of the group (the area that is not occupied by elements or connections), $0<F_G\leq 1$. (The higher the fitness, the better the layout.)

As work progresses it is expected that other terms will be added to the fitness function, reflecting other desired qualities of the solution sought, and compensating for any undesired effects found to be introduced by the heuristics employed. However, the area efficiency is expected to remain a key element of the fitness function.

3.3.4. The Parallel GA Architecture

One of the authors, together with his graduate students, has been investigating various architectures for parallel and distributed GA's for over ten years [17-19]. He first realized the benefits of non-panmictic breeding in GA search in 1976, when an MSU Ph.D. student revealed to his doctoral committee his mathematical proof that group selection was possible using ordinary Darwinian principles, so long as isolated ("island") subpopulations with only infrequent immigration were provided.

This has led to a succession of distributed and parallel GA architecture implementations, including five software systems embodying different GA tools and different parallel/distributed architectures. Our GA Research and Applications Group is currently using (and making available) three of these sets of research tools. The first is for micro-grained parallelism, based on GENESYS, and most effectively used on Unix-based symmetric multiprocessors, although it may also be run on distributed workstations. It holds little interest for solving of the most difficult problems, as it enables only a hardware speedup of the solution of a single-population GA problem.

The second is for island parallelism, is based on GAUCSDx.x, and designed for loosely coupled, heterogeneous Unix workstations. It can be run in a "polite" mode in which any workstation checkpoints and "dies" whenever a user sits down at the console, and the system is extremely tolerant of failures/reboots of any nodes.

The third is also for island parallelism, and runs not only on Unix workstations, but also on arbitrary networks of PC clones, so long as a shared file space is available (this latter capability was developed particularly to allow our collaborators in Russia and China access to a parallel GA toolkit). To facilitate its use by remote users, it was developed starting from the 'C' version of Goldberg's Simple Genetic Algorithm (as presented so effectively in his introductory book [5]), but completely rewritten and extended manyfold. This system, called the GALOPP System (Genetic Algorithm Optimized for Portability and Parallelism), is now approximately 30,000 lines of code, features a large variety of crossover operators, selection methods, performance and diversity measures, and a flexible scheme for inter-population communication. The user can specify a problem to solve using a template provided, by doing as little as filling in an objective function, or can exert tight control over the algorithm through a series of skeleton callback functions, all without modifying any source code except the user application file.

The primary tool for this project will be the latter system, running primarily in a distributed workstation environment. The parallel architecture to be used has been dubbed the "island injection Genetic Algorithm", or iiGA. It utilizes subpopulations organized into three or more "layers," in which each higher layer represents a higher degree of abstraction of the problem (in this case, for example, higher-level groups). Migration of individuals, when it occurs, is usually performed primarily within each layer, however, individuals can also migrate from a higher layer to a lower layer (but not in the other direction). When this occurs, the representation is remapped by desegregating groups into their component constituents at the lower layer. The lack of "back contamination" means that the higher-level subpopulations are able to act as a continuing source of diversity and good high-level building blocks for the lower layers' populations, immune to the contaminating influence of the lower-layer populations to direct search toward the local suboptimal solutions they have found.

The iiGA architecture is not required for relatively simple optimization problems, but the complexity of the compaction task easily warrants such an approach. A great deal of experimentation must be done to determine the best combination of representation/operator definition/selection method/fitness function/parallel architecture for a class of bin packing and compaction problems. However, the lessons we have learned in applying the iiGA to problem of composite material structures optimization (2-D and 3-D)[19], lowest-energy molecular configurations, and classification of high-dimensionality empirical data, will serve well in facilitating progress on the bin packing and compaction problems.

3.4. Sequence of Problem Refinement

Work on GA architecture and tuning can progress at the same time as a sophisticated package for decoding chromosomes into full-blown layouts is being developed. As envisioned, the GA work would proceed in three partially overlapping stages:

Task 1 *GA design methodology development for the hierarchical chromosome representation (HCR).*

Objective To develop the k-level HCR, which can be applied to the bin packing and layout VLSI problems. To develop a formal definition and structure for a new representation (HCR), corresponding GA operators, and GA design methodology, which is based on HCR and incorporates heuristics and expert knowledge.

Significance This new design methodology and HCR will profit from the many significant developments in GA theory and practice made by our group and others, in providing a basis for effective solving of compaction, bin packing, and nesting problems. This is very important, because the problems are NP-hard and reasonable approximation algorithms cannot guarantee solutions that are near to optimal for many practical situations [25].

Approach Our approach is to develop a hierarchical description of the chromosome, in which each group (node of HCR) represents a building block, and corresponding GA operators. The work is based on new results in GAs -- new genetic operations, encodings, selection methods, and structures (Table 1-Table 5) -- which can incorporate heuristic algorithms, domain knowledge, and local hillclimbing.

Task 2 *Bin packing and nesting problems.*

Objective To investigate new methodology and some new GA ideas for the 1-D, 2-D, and 3-D bin packing and 2-D nesting problems.To generalize the formulation of the 2-D (3-D) bin packing problems by allowing sizes of the bin be variables and introducing a new type of boxes -- non-rectangular and with adjustable sizes -- and a new criterion: the minimum bin area (volume).

Significance These generalizations make the formulation of the 2-D (3-D) bin packing problems closer to the compaction problem and the nesting problem. These approaches will be used for improving the solutions of the 2-D (3-D) bin packing problems, and these ideas will be incorporated and used in fostering development of compaction GAs for the layout of real VLSI systems and for addressing nesting

problems.

Approach — We will use our methodology and HCR in a series of experiments with tailored representations in 2-D (3-D) bin-packing and nesting problems. We will explore some new schemes for selection: random with pressure and random-direct selection, and will investigate new (or newly optimized) genetic operations, encodings, selection methods, and structures. (See Table 1 - Table 5). We will investigate these effects of various parallel GA architectures, including especially our own "island injection GA" architecture, which appears to be perfectly "matched" with the HCR, on the solution of these problems.

Task 3 *Module placement (simplified compaction) problem.*

Objective — Development of and experimentation with a highly simplified decoding and fitness function, based on tracking only the number of connections between each pair of blocks, but abstracting the details of routing the connections.

Significance — This approach will be used for more effective solving of the so-called module placement problem [2], as an approximation to the 2-D (and 3-D) compaction of the 2-D (and 3-D) layout problems. It will create the basis for extension of these results to the compaction problem. Moreover, this problem is important in itself, and practically all industrial CAD systems include some tools to address it.

Approach — We will use the proposed GA methodology based on HCR. We will attempt to produce good placement using GAs with HCR. We will accomplish several serials of experiments to find optimal hierarchical structure parameters and values for adjustable GA parameters. Our algorithm is to be also in progress in making the genetic operators robust to quantity of data, variation in dimensions of boxes, and variation in the aspect ratio of the bin.

Task 4 *Compaction problem.*

Objective — Development of a silicon-compiler-logic-based compaction algorithm, and experimentation with this representation on real (particularly, the most important fragments of some modern types of VLSI systems) and benchmark problems. To investigate new genetic ideas, operations, encodings, selection methods, and structures that were successful for solving the bin packing and nesting problems in order to create a new compaction methodology on the basis of GAs and receive better compaction solutions.

Significance It is expected that the new GA methodology will produce a better packing density of the layout, smaller die area of the layout, and smaller signal delays along critical paths. It has to be done more effectively than another techniques. This is very important, because the problem is NP-hard. Improvement of the density and the layout area will result in improving practically all characteristics of the VLSI system.

Approach We will create new GA operators on the basis of HCR and silicon compiler merge-algorithms. We will consider a generalized formulation of the compaction including the objective function of the circuit performance. We will extent the compaction operations by using 90^{o} turns and mirror reflections of modules and groups. We will investigate the possibility of incorporating the simulated annealing approach [85] into GA methodology for dealing with compaction. Our crossover operator will produce only solutions without design-rule violations and in this way, reduce tremendously the execution time. It can be based in part on some ideas developed in [128-130]. Using important information of the initial layout to generate the initial population will also be exploited. We will do experiments to find an effective and efficient structure for the HCR, and will search for good settings for GA parameters.

3.5. Contribution and Impact

The results of this research effort enhance design automation frameworks and specific design methodologies in several significant ways. First, the development of a useful set of genetic operations, rules and design oriented models representing the layout for very high speed and low power integrated circuits has the potential to improve the performance of both the designs and algorithms. Second, research and potential discoveries relating to GAs for the compaction optimization problem will have an impact on GA development in general. Applying genetic algorithms for compaction and bin packing is likely to generate high-quality solutions to these problems and to other knowledge-oriented tasks in which the objects or events to be sequenced are unique.

4. BIBLIOGRAPHY

[1] Chu, N.-A., Dardy, H., "A Survey of High Density Packaging for High Performance Computing Systems", *Journal of Microelectronic Systems Integration*, Vol. 1, No. 1, pp. 3-27, 1993.

[2] *Physical design automation of electronic system*, edited by Bryan T. Preas and Michael J. Lorenzetti. The Benjamin/Cummings Publishing Company, Inc., 1988.

[3] Thomas Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, Inc., 1990.

[4] Holland, J. H., *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Application to Biology, Control, and Artificial Intelligence*. University of Michigan, 1975.

[5] Goldberg D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc., 1989.

[6] *Handbook of Genetic Algorithms*, Edited by Lawrence Davis, Van Nostrand Reinhold, New York, 1991.

[7] Sannier, A. and E. Goodman, "Genetic Learning Procedures in Distributed Environments," *Proc. of the 2nd Int. Conf. on Genetic Algorithms and their Applications*, Lawrence Erlbaum Associates, 1987.

[8] Punch, W.F., Goodman, E. D., et al, "Further Research on Feature Selection and Classification Using Genetic Algorithms", *Proc. of the 5th Int. Conf. on Genetic Algorithms*, Urbana-Champaign, July 1993, pp. 557-564.

[9] Kureichik, V. M., Liah, A. V., "Parallel evolution (genetic) algorithm of graph division," *Proc. USSR Conf. Graphs Algorithms in CAD*, Academy of Sciences , Novosybirsk, 1989, (in Russian).

[10] Kureichik, V. M., Lebedev, B.K., Liah, A. V., "Problems of Evolution Adaptation in CAD", *Novintech journal*, No. 3, Moscow, 1991, (in Russian).

[11] Kureichik, V. M., "Genetic Algorithms in CAD," *Proc. Russia Conf. AI in CAD*, Gelendzik, September 1993, (in Russian).

[12] Kureichik, V. M., "Evolution Simulating and CAD Systems of Computers," *Proc. Int. Conf. Problems of CAD Systems*, Gurzuf, Ukraine, 1993.

[13] Kureichik, V. M., Mizuk, N. V., Liah, A. V., "Tasks of Evolution Modeling in Intellectual CAD", *Proc. Int. Workshop, CAD-93*, July 1993, Moscow, Russia.

[14] Tetelbaum, A.Y. "Estimation of the Layout Characteristics", *Mathematical and Computer Modeling*, Vilnus, 1987, No.2.

[15] Tetelbaum, A.Y. "Hierarchical Approach to VLSI Circuit Design", *MicroElectronics*, Moscow, 1981, No.2 (50).

[16] Goodman, E.D., *An Introduction to GALOPPS -- The "Genetic Algorithm Optimized for Portability and Parallelism" System*, Case Center Technical Report # 940401, Michigan State University, May, 1994, 58pp.

[17] Goodman, E.D. and Sannier, A.P., "Genetic Learning Procedures in Distributed Environments," *Proc. Second Int'l Conf. on Genetic Algorithms and their Applications*,

Laurence Erlbaum Associates, Inc., Cambridge, MA, 1987.

[18] Punch, W.F., Goodman, E.D., Pei, M., Enbody, R., Lai, C.-S., and Hovland, P., "Further Research on Feature Selection and Classification Using Genetic Algorithms," *Proc. Fifth Int'l Conf. On Genetic Algorithms and their Applications*, Morgan Kaufmann Publishers, Inc., July, 1993, pp.557-564.

[19] Lin, S.-C., Punch, W.F., and Goodman, E.D., "Course-Grain Parallel Genetic Algorithms: Categorization and New Approach," *IEEE Conf. on Parallel and Distributed Processing*, October, 1994, (accepted for publication).

[20] Falkenauer, E., Delchambre, A., "A Genetic Algorithm for Bin Packing and Line Balancing," *Proc. of the 1992 IEEE International Conf. on Robotics and Automation*, vol. 2, Piscataway, NJ, 1992, pp. 1186- 1192.

[21] Kao, C.-Y., "A Stochastic Approach for The One-Dimensional Bin-Packing Problems", Proc., *Int. Conf. on Systems, Man, and Cybernetics*, pp. 1545-1551, Chicago, October, 1992.

[22] Pargas, R. P., Jain, R., "A Parallel Stochastic Optimization Algorithm for Solving 2D Bin Packing Problems", in *Proc. 9th. Conf. on AI for Applications*, March, 1993, Orlando, Florida, pp. 18 - 25.

[23] Coffman, E. G. Jr., Garey, M. R., and Johnson, D. S., "Dynamic Bin Packing", *SIAM J. COMPUT*. vol. 12, No. 2, May 1983, pp.227-258.

[24] Baker, B. S. and Schwarz, J. S., "Shelf Algorithms For Two-dimensional Packing Problems", *Siam J. COMPUT*. vol. 12, No. 3, August 1983, pp.508-525.

[25] D.Smith, "Bin Packing With Adaptive Search," in *Proc. 1st Int. conf. on Genetic Algorithms and their Applications*, July 1985, pp.202-207.

[26] Lin, J.-L., Foote, B., Pulat, S., at. al., "Hybrid Genetic Algorithm for Container Packing in Three Dimensions", in *Proc. 9th. Conf. on AI for Applications*, March, 1993, Orlando, Florida, pp. 353 -359.

[27] Fujita, K., Akagji, S., Kirokawa, N., "Hybrid approach for optimal nesting using a genetic algorithm and a local minimization algorithm", in *Proc. of the 19th Annual ASME Design Automation Conference*, part 1, New York, 1993, pp. 477-484.

[28] Dyckhoff, H., "A typology of cutting and packing problems", *European Journal of Operating Research* 44, 1990, pp.145-159.

[29] Cheok, B. T., and Nee, A. Y. C., "Algorithms for nesting of ship/offshore structural plates," In *Proc. Advances in Design Automation*, vol.2, ASME 1991, pp.221-226.

[30] Jain, P., Fenyes, P., and Richter, R., "Optimal blank nesting using simulated annealing", in *Proc. Advances in Design Automation*, vol.2, ASME 1991, pp. 109 - 116.

[31] Dunlop, A.E., "Symbolic layout IM: the translation of symbolic layouts into mask data," *Journal of Digital Systems*, vol.5, no.4, pp. 429-451, 1981.

[32] Wolf, W., *Two-dimensional Compaction Strategies*, Ph. D. thesis, Stanford university, March 1984.

[33] Dunlop, A.E., "Symbolic layout IM: the translation of symbolic layout into mask data," *Proc. 17th Design Automation Conf.*, pp. 595-602, ACM/IEEE, 1980.

[34] Boer, D.G., "Symbolic layout compaction benchmarks - results," *Digest Intl. Conf. on*

CAD, pp. 209-217, October 1987.

[35] Doenhardt, J., and T. Lengauer, "Algorithmic aspects of one-dimensional layout," *IEEE Trans. on CAD, vol. CAD-6*, no.5, pp. 863-878, IEEE, 1987.

[36] Schlag, M., Y.Z. Liao, and C.K. Wong, "An algorithm for optimal two-dimensional compaction of VLSI layouts," *Integration*, vol.1, no.2, 3, pp.179-209, September 1983.

[37] Burns, J.L., and A.R. Newton, "Efficient constraint generation for hierarchical compaction," *Proc., Intl. Conf. on CAD*, pp. 197-200, IEEE Computer Society, October 1987.

[38] Bamji, Cyrus S., and Varadarajan Ravi, "MSTC: A Method for Identifying Overconstraints during Hierarchical Compaction," *Proc. 30th DAC*, pp.389-394, June 1993, ACM/IEEE.

[39] Boyer, D. G., "Split grid compaction for a virtual grid symbolic design system," *Digest Intl. Conf. on CAD*, pp.134-137, November 1987.

[40] Tan, D., and N. Weste, "Virtual grid symbolic layout 1987," *Proc. Intl.Conf. on CA*D, pp.192-196, October 1987.

[41] Boyer, D.G., and N. Weste, "Virtual grid compaction using the most recent layers algorithm," *Digest Intl. Conf. on CAD*, pp.92-93, September 1983.

[42] Akers, S.B., J.M. Geyer, and D.L. Roberts, "IC mask layout with a single conductor layer," *Proc. 7th Design Automation Workshop*, pp.7-16, 1970.

[43] Dao, J., Matsumoto, N., Hamai T., Ogawa, C., Mori, S., "A Compaction method for full chip VLSI layouts", *Proc. 30th Design Automation Conf.*, pp. 407-412, ACM/IEEE, 1993.

[44] Pan, P., Dong, Sai-Keung, and Liu, C.L., "Optimal graph constraint reduction for symbolic layout compaction", *Proc. 30th Design Automation Conf.*, pp. 401-406, ACM/IEEE, 1993

[45] Kingsley, C., *Earl: An Integrated Circuit Design Language*, M.S. Thesis, California Institute of Technology, June, 1982.

[46] Hedges, T., W. Dawson, and Y.E Cho, "Bitmap graph build algorithm for compaction," *Digest Intl. Conf. on CAD*, pp. 340-342, September 1985.

[47] Hedlund, K.S., "Electrical optimization of PLAs," *Proc., 22nd DAC*, pp.681-687, June 1985.

[48] Burns, J.l., and A.R. Newton, "SPARCS: a new constraint- based IC symbolic layout spacer," trans., *IEEE Custom Integrated Circuits Conf.*, pp. 534-539, IEEE, May 1986.

[49] Mehlhorn, K., *Data Structures and Algorithms 2: graph algorithms and NP-completeness*, Springer-Verlag, Berlin, 1984.

[50] Lengauer, T., "On the solution of inequality systems relevant to IC-layout," *Journal of Algorithms*, vol.5, pp.408-421, 1984.

[51] Cook, P.W., "Constraint solver for generalized IC layout," *IBM Journal of Research and Development*, pp. 581-589, September 1984.

[52] Liao, Y.-Z., and C.K. Wong,"An algorithm to compact a VLSI symbolic layout with mixed constraints," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol.CAD-2, no.2, pp.62-69, 1983.

[53] Van Der Woude, M., and X. Timermans, "Compaction of hierarchical cells with minimum

and maximum compaction constraints," *Proc., Intl. Symposium on Circuits and systems*, pp. 1018-1021, 1983.

[54] Lo, C.-Y., R. Varadarajn, and W.H. Crocker, "Compaction with performance optimization," *Proc.,Intl. Conf. on Circuits and Systems*, pp.514-517, 1987.

[55] Kedem, G., and H. Watanabe, "Graph-optimization techniques for s-2 IC layout and compaction," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. CAD-3, no.1, pp. 12-20, 1984.

[56] Entenman, G., and S. W. Daniel, "A fully automatic hierarchical compactor," *Proc., 22nd DAC*, pp. 69-75, ACM/IEEE, June 1985.

[57] Reichelt, M., and W. Wolf, "An Improved cell model for hierarchical constraint-graph compaction," *Digest Intl. Conf. on CAD Design*, pp. 482-485, September 1986.

[58] Mosteller, R.C., A.H. Frey, and R. Suaya, "2-D compaction: a Monte Carlo method," *Proc., Conference on Advanced Research in VLSI*, pp. 173-197, MIT Press, 1987.

[59] Lee, J. F., and Tang, D. T., "HIMALAYAS - a hierarchical compaction system with a minimized constraint set," IBM Research Report, T. J. Watson Research Center, Yorktown Heights, Ny, 1992.

[60] Marple, D., "A hierarchy preserving hierarchical compactor," in *Proc. 27th DAC*, pp. 375-381, 1990,ACM/IEEE.

[61] Wolf, W. H., Mathews, R. G., Newkirk, J. A., and Dutton r. W., " Algorithms for optimizing two-dimensional symbolic layout compaction," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. CAD-7, no. 4, pp. 451-466, April 1988.

[62] Shin, H., Sangiovanni-Vincentelli, A. L., and Sequin, C. H.., "Two-dimensional module compactor based on zone-refining," in *Proc. Intl. Conf. on Computer Design*, 1987, pp. 201-203.

[63] Wolf, W., "An experimental comparison of 1-D compaction algorithms", pp.165-180 in *1985 Chapel Hill Conf. on VLSI*, Computer Science Press, Rockvile, Md, May 1985.

[64] Bamij, C. S., and Varadarjan, R.,"Hierarchical pitch matching compaction using minimum design", in *Proc. 29th DAC*, pp. 311-317, 1992, ACM/IEEE.

[65] Yao, S.-Z., Cheng, C.-K., Dutt, D., Naxar, S., and Lo, C.-Y., "Cell-Based Hierarchical Pitchmatching Compaction Using Minimal LP", in *Proc. 30th DAC*, pp. 395-400, 1992, ACM/IEEE.

[66] Floyd, R., "Algorithm 97, shortest path," *Communication of the ACM*, vol.5, no.6, p.345, June 1962.

[67] Eichenberger, P., *Fast Symbolic Layout Translation for Custom VLSI Integrated Circuits*, Ph. D. thesis, Stanford University, April 1986.

[68] Bullman, W. R., Davieu, L.A., Moscovitz, H. S., and O'Donnell, G. D., PANDA - a module assembler for the IDA system, personal communication, September 1986.

[69] Kollaritsch, P., and B. Ackland, "COORDINATOR: a complete design-rule enforced layout methodology," *Proc. Intl. Conf. on CAD*, pp. 302-307, October 1986.

[70] Maley, F. M., "Compaction with performance optimization, " *Proc. Intl. Conf. on Circuits and Systems*, pp. 514-517, 1987.

[71] R. Ghandrasekharam, S. Subhramanian and S.Chadhury. "Genetic algorithm for node

partitioning problem and application in VLSI design", *IEEE Proceedings*, vol.140, No.5, Sept.1993.

[72] G. von Laszewski, "Intelligent Structural Operators for the K-Way Partitioning Problem", In *Proc. of the 4th Int. Conf. on Genetic Algorithms*, San Diego,1991.

[73] R.M. Kling and P. Banerjee, "ESP: Placement by Simulated Evolution", *IEEE Trans. on CAD*, vol. 8, No.3, March 1989, pp.245-256.

[74] J.P. Cohoon, W.D. Paris, "Genetic Placement, *IEEE Trans. on CAD*, vol.6, No.6, November 1987, pp.956-964.

[75] J.P. Cohoon, S.U. Hegde, W.N. Martin and D.S. Richards, "Distributed Genetic Algorithms for Floorplan Design Problem", *IEEE Trans. on CAD*, vol.10, No.4, April 1991, pp.483-492.

[76] A.T. Rahmani, N. Ono. A Genetic Algorithm for Channel Routing Problem. In *Proc. 5th Int. Conf. on Genetic Algorithms*, Urbana-Champaign,1993.

[77] Y.L. Lin, Y.C. Hsu, F.S. Tsai, "SILK: A Simulated Evolution Router", *IEEE Trans. on CAD*, vol.8, No.10, October 1989, pp.1108-1114.

[78] M.Fourman, "Compaction of Symbolic Layout using Genetic Algorithms," in *Proc. 1st Int. Conf. on Genetic Algorithms and their Applications*, July, 1985, pp. 141-153.

[79] Yao, A. C.-C., "New Algorithms for Bin Packing", *Journal of the ACM*, vol. 27, No. 2, April 1980, PP. 207-227.

[80] Johnson, D.S., Demers, A., Ullman, D.J., Garey, M.R., and Graham, R.L., "Worst-case performance bounding for simple one-dimensional packing algorithms", *SIAM J. Comput.*, vol. 3, No. 4, December 1974, pp. 299-325.

[81] Corcoran, A.L. and R.-L. Wainwright. "LibGA: A User-friendly Workbench for Order-based Genetic Algorithm Research", In *Proceedings of the ACM SIGAPP Symposium on Applied Computing*, Indianapolis, Indiana, February 14-16, 1993, pp. 111-118.

[82] Corcoran, A. L. 111, Wainwright, R. L., "Genetic Algorithm for Packing in Three Dimensions ", *Proc., of the 1992 ACM/SIGAPP Symposium on Applied Computing SAC '92*, New York, 1992, pp. 1021- 1030.

[83] Esbensen, H., "Genetic algorithm for macro cell placement", *Proc. European Design Automation Conference -EURO-VHDL '92*, Hamburg, Germany, 1992, pp. 52-57.

[84] D.F. Wong, H.W. Leong and C.L. Liu, *Simulated Annealing for VLSI Design*, Kluwer Academic Publishers, Boston, 1988.

[85] Hsieh, T. M., Leong, H. W., Liu, C. L., "Two-Dimensional Layout Compaction by Simulated Annealing", in *Proc. IEEE International Symposium on Circuits and Systems*, June 1988, Espoo, Finland, vol. 3, pp. 2439 - 2443

[86] Eshelman, L. J., Caruna, R. A., Schaffer, D. J.,"Biases in the Crossover Landscape", *Proc. of the 3rd Int. Conf. on Genetic Algorithms*, June 1989, Arlington, VA, pp. 10-19.

[87] De Jong, K. A., *Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. Dissertation, University of Michigan , Ann Arbor, MI, 1975.

[88] Davis, L., "Applying adaptive algorithms to epistatic domains" *Proc. 9th Int. Joint. Conf. Arti. Intell.*, Los Angeles, 1985, pp. 162- 164.

[89] Shang, Y. and Li, G.-J., "New Crossover Operators in Genetic Algorithms", *Proc. of the 1991 IEEE Int. Conf. on Tools for AI*, San Jose, CA, November 1991, pp. 150-153.

[90] Goldberg, D. E., and Lingle, R., "Alleles, loci, and the traveling salesman problem", *Proc. Int. Conf. Genetic Algorithms and Their Applications*, Pittsburgh, PA, 1985, pp. 154-159.

[91] Oliver, I. M., Smith D. J., and Holland, J. H., "A study of permutation crossover operators on the traveling salesman problem", *Proc. 2nd Int. Conf. Genetic Algorithms*, Cambridge, MA, July 1987, pp.82-89.

[92] Grefenstette, J., Gopal, R., Rosmaita, B., Van Gucht, D., "Genetic Algorithms for the Traveling Salesman Problem", *Proc of the 1st Int. Conf. on Genetic Algorithms and their Applications*, Pittsburgh, PA, July 1985, pp. 160-168.

[93] Oosthuizen, D., "SUPERGRAN: A connectionist approach to learning, integrating genetic algorithms and graph induction", *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, Cambridge, MA, July 1987, pp. 132-139.

[94] Schaffer, J., Morishima, A., "An Adaptive Crossover Distribution Mechanism for Genetic Algorithms, *Proc, of the 2nd Int. Conf. on Genetic Algorithms*, Cambridge, MA, July 1987, pp.36-40.

[95] Syswerda, G., "Schedule Optimization Using Genetic Algorithms" in *Handbook of Genetic Algorithms*, edited by L. Davis, van Nostrand Reinold, New York, 1991, pp. 332-349.

[96] Ackley, D. H., *A Connectionist Machine for Genetic Hillclimbing*, Kluwer Academic Publishers, Boston, MA, 1987.

[97] Davidor, Y., "Analogous Crossover", *Proc. of the 3rd Int. Conf. Genetic Algorithms*, Arlington, VA, June 1989, pp. 98-103.

[98] Louis, S. J., Rawlins, J. E., "Designer Genetic Algorithms : Genetic Algorithms in Structure Design," *Proc. of the 4th Int. Conf. on Genetic Algorithms*, San Diego, CA, July 1991, pp. 53-60.

[99] Whitley, D., Starkweather, T., and Fuquay, D., "Sche*duling Problems and the Traveling Salesman: the genetic edge recombination operator", Proc. of the 3rd Int. Conf. on Genetic Algorithms and Their Applications*, Arlington, VA, June 1989, pp. 133-140.

[100] Potts, J. C., Giddens, T. D., and Yadav, Surya B., "The Development and Evaluation of an Improved Genetic Algorithm Based on Migration and Artificial Selection", *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 24, No. 1, January 1994, pp. 73-86.

[101] Grefenstette, J. J., "Optimization of Control Parameters for Genetic Algorithms", *IEEE Trans. on Systems Man, and Cybernetics*, vol. sms-16, No. 1, January/February 1986, pp. 122-128.

[102] Shahookar, K., and Mazumder, P., "A Genetic Approach to Standard Cell Placement Using Meta-Genetic Parameter Optimization", *IEEE Trans. on CAD*, vol. 9, No. 5, May 1990, pp. 500-511.

[103] Cohoon, J. P., Hegde, S. U., Martin, W. N., Richards, D., "Punctuated Equilibria: A Parallel Genetic Algorithm", *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, Cambridge , MA, July 1987, pp. 148-154.

[104] Eshelman, L. J., "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination", in *Foundations of Genetic*

Algorithms, edited by Gregory J. E. Rawlins, Morgan Kaufmann Publishers, San Mateo, California, 1991, pp. 265-283.

[105] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proc. 2nd Int. Conf. on Genetic Algorithms*, Cambridge, MA, 1985, pp. 14-21.

[106] A. Brindle, Genetic algorithms for functional optimization, Ph.D. thesis, Univ. of Alberta, Alberta, 1981.

[107] D. Whitley, "Using reproductive evaluation to improve genetic search and heuristic discovery," in *Proc. 2nd Int. Conf. on Genetic Algorithms*, Cambridge, MA, 1985, pp. 108-115.

[108] J.J. Grefenstette and J. E. Baker, "How genetic algorithm work: A critical look at implicit parallelism," in *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Mateo, CA, 1989, pp. 20-27.

[109] T. Back and F. Hoffmeister, "Extended selection mechanism in genetic algorithms," in *Proc 4th Int. Conf. on Genetic Algorithms*, San Diego, CA, 1991, pp. 92-99.

[110] D. Whitley, "The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best," in *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Mateo, Ca, 1989, pp. 116-121.

[111] L. J. Eshelman and J. D. Schaffer, "Preventing premature convergence in Genetic Algorithms by preventing incest," in *Proc. 4th Int. Conf. on Genetic Algorithms*, San Diego, CA, 1991, pp. 115-122.

[112] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proc. 2nd Int. Conf. on Genetic Algorithms*, Cambridge, MA, 1987, pp. 41-49.

[113] K. Deb and D.E. Goldberg, "An investigation of niche and species formation in genetic function optimization," in *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Mateo, CA, 1989, pp. 42-50.

[114] D. J. Cavicchio, "Reproductive adaptive plans," in *Proc. of the ACM 1972 Annual Conf.*, 1972, pp. 1-11.

[115] Bosworth, J., Foo, N., and Zeigler, B. P. *Comparison of genetic algorithms with conjugate gradient methods (CR-2093)*. Washington, DC: National Aeronautics and Space Administration.

[116] Fogel, L. J., Owens, A. J., and Walsh, M. J. *Artificial intelligence through simulated evolution*. John Wiley Publisher, New York, 1966.

[117] Syswerda, G., "Schedule Optimization Using Genetic Algorithms", in *Handbook of Genetic Algorithms*, Edited by Lawrence Davis, Van Nostrand Reinhold, New York, 1991, pp. 332-349.

[118] Liepins, G. E., and Potter, W. D., "A Genetic Algorithm to Multiple-Fault Diagnosis", in *Handbook of Genetic Algorithms*, Edited by Lawrence Davis, Van Nostrand Reinhold, New York, 1991, pp. 237-250.

[119] Montana, D. J., "Automated Parameter Tuning for Interpretation of Synthetic Images", in *Handbook of Genetic Algorithms*, Edited by Lawrence Davis, Van Nostrand Reinhold, New York, 1991, pp. 282-311.

[120] Fogarty, T.C., "Varying the probability of mutation in the genetic algorithm", in *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Diego, CA, 1989, pp. 104-109.

[121] Frantz, D. R. *Non-linearities in genetic adaptive search*. (Ph. D, University of Michigan), Dissertation Abstracts International, 33(11), 5240B-5241B, 1972, (University Microfilms No.73-11,116).

[122] Bagley, J. D. *The behavior of Adaptive systems which employ genetic and correlation algorithms*. (Ph. D, University of Michigan), Dissertation Abstracts International, 28(12), 510B, 1967, (University Microfilms No.68-7556).

[123] Mange, A. P., Mange, E. J., *Genetics: Human Aspects*. Saunder College, Philadelphia, 1982.

[124] Lakhani, G., and R. Varadarajan, "A wire-length minimization algorithm for circuit layout compaction," *Proc., ISCAS-87*, pp. 276-279, May 1987.

[125] Lin, S.L., and J. Allen, "Minplex - a compactor that minimizes the bounding rectangle and individual rectangles in a layout, " *Proc., 23rd DAC*, pp. 123-130, June 1986.

[126] C. Sechen, *VLSI Placement and Global Routing using Simulating Annealing*, Kluwer Academic Publishers, Boston,1988.

[127] S. Goto, eds., *Design Methodologies*, North - Holland Amsterdam - New York Oxford - Tokyo, Vol.6,1986.

[128] Tetelbaum, A.Y. "Force Embedding of a Planar Graph", in *Proc. of 26th IEEE Southeastern Symposium on System Theory*, Ohio, USA, March 1994, pp. 2-6.

[129] Tetelbaum, A.Y. "Component Placement of VLSI Gate Arrays", *Electronic Design Automation*, Kiev, 1991, No. 44.

[130] Tetelbaum, A.Y., Shramchenko, B.L. and Demyanenko, O. "Automatic Placement for ARM2-01 Workstation", *Electronic Equipment and Computer Design Automation*, Moscow, 1992, No. 1.