PROCEEDINGS OF AN INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS AND THEIR APPLICATIONS

July 24-26, 1985
at
Carnegie-Mellon University
Pittsburgh, PA

Sponsored By

Texas Instruments, Inc.

U.S. Navy Center for Applied Research
in Artificial Intelligence
(NCARAI)

John J. Grefenstette
Editor

Compaction of Symbolic Lagout using Genetic Algorithms

Michael P. Fourman
Dept of Electrical and Electronic Engineering
Brunel University, Uxbridge Middx., U.K.
michael%brueer@ucl-cs.AC.UK

Introduction.

Design may be viewed abstractly as a problem of optimisation in the presence of constraints. Such problems become interesting once the space of putative solutions is too large to permit exhaustive search for an optimum, and the payoff function too complex to permit algorithmic solutions. Evolutionary algorithms [Holland 1975] provide a means of guiding the search for good solutions. These algorithms may be viewed as embodying an informal heuristic for problem solving along the lines of

> "To find a better strategy try variations on what has worked well in the past."

Here, a "strategy" is an attempt at a solution. A strategy will generally not address all the constraints imposed by the problem. The algorithms we are considering guide the search by comparing strategies. We represent this comparison by the relation

a beats b

(which will usually be be a partial order, but need not be total). We call strategies which satisfy all the constraints of the problem "solutions". In general, solutions should beat other strategies and, of course, some solutions will beat others. Abstractly, the algorithms merely search for strategies which are well-placed in this ordering.

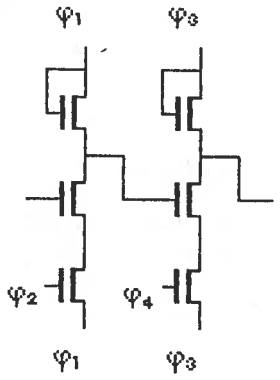
Many problems in silicon design involve intractable optimisation problems, for example, partitioning, placement, PLA folding and layout compaction. We say a

problem is intractable when the combinatorial complexity of the solution space for the problem makes exhaustive search impossible, and the varied nature of the constraints which must be satisfied makes it unlikely that there is a constructive algorithmic solution to the problem. Automatic solution of such problems requires efficient search of the solution space. Simulated annealing has been applied to the first three problems [Kirkpatrick *et al.* 1983], branch and bound techniques have been applied to layout compaction [Schlag *et al.* 1983]. In this paper we report on the application of a genetic algorithm to layout compaction.

The first prototype solved a highly simplified version of the problem. It produced layouts of a given family of rectangles under the constraint that no two shall overlap, with cost given by the area of a bounding box. A more realistic prototype deals with the layout of a family of rectangular modules with a single level of interconnect. These prototypes allow the designer to add his ideas to the evolving population of layouts and thus supplement rather than replace his expertise.

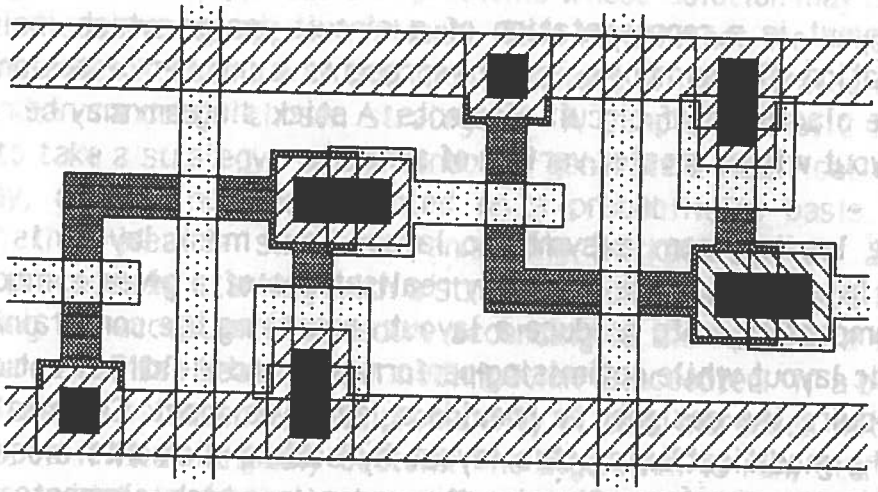
Symbolic Layout.

A circuit diagram conveys connectivity information:

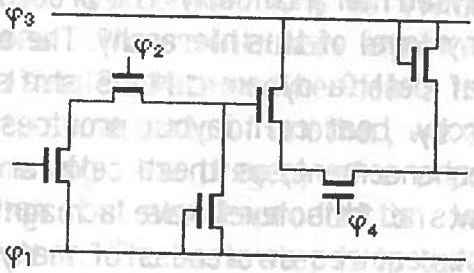


To manufacture the circuit this must be tranformed to a representation in terms of layout elements, each layout element must be assigned an absolute mask position. A layout diagram conveys this mask-making information. The passage from a circuit diagram to a layout may be divided into three stages: firstly the topology (relative positioning of layout elements) of the layout is designed and represented by a **symbolic layout**, then a mask level is assigned to each wire in the circuit - the design is now represented by a **stick diagram**, finally the mask geometry (absolute size and positions) is created. Engineers commonly use these

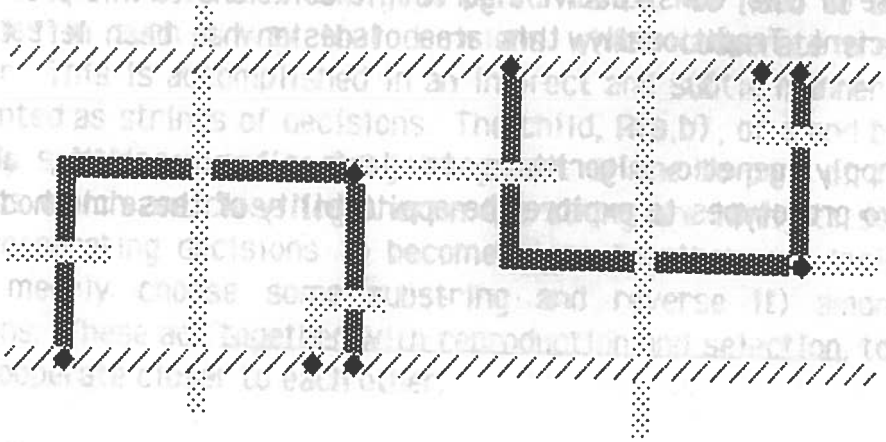
intermediate notations to represent the intermediate stages in the design process. Here is a mask layout for our circuit:



Here is a symbolic version of this layout:



The corresponding stick diagram is:



A symbolic layout is a representation of a circuit design which includes some layout information. The symbolic layout represents a number of design decisions on the relative placement of circuit elements. A stick diagram may be regarded as a symbolic layout with a greater variety of symbols.

The procedure leading from a symbolic layout to a mask layout is a form of **compaction**. In general, there are many realisations of a given symbolic layout. The aim of compaction is to produce a layout respecting the constraints implicit in the symbolic layout while optimising performance and yield. Current compaction algorithms require the designer to provide a layout as input. Compaction usually consists of the modification of this layout by sliding elements closer together while retaining the topology. Clearly, the order in which elements are moved affects the result. Most algorithms simply compact in each coordinate direction in turn.

Modern designs are modularised hierarchically. The process of symbolic layout and compaction may occur at any level of this hierarchy. The example we have used for illustration above is a leaf cell (a dynamic NMOS shift register cell) from the bottom level of the hierarchy. Leaf cell layout provides great opportunities for area reduction and yield enhancement, as these cells are replicated many times and any small improvements at this level have a magnified effect on the chip. Optimising leaf cell layout requires awareness of many interacting constraints and complex cost functions (for example connectivity constraints given by the circuit design, geometric constraints given by the process design rules, and the cost functions arising from performance requirements and knowledge of yield hazards). Because of this, constructive algorithmic solutions to this problem have not proved efficient. Traditionally, this area of design has been left to human experts.

We hope to apply genetic algorithms to leaf-cell compaction, and have implemented two prototypes to explore the applicability of these methods in this domain.

Genetic Algorithms.

Genetic algorithms are applicable to problems whose solution may be arrived at by a process of successive approximations. This means that we need to be able to modify strategies in such a way that modifications of good strategies are likely to be better than randomly chosen strategies. A simple heuristic in this setting would be to take a strategy, **a**, and randomly generate a modification, **M(a)**, of it which may, or may not, be accepted on a probabilistic basis. An algorithm embodying this idea is simulated annealing [Kirkpatrick *et al.* 1983]. The algorithm procedes by starting with a strategy and repeatedly modifying it in this way, varying the acceptance procedure according to the value of a variable called **temperature**. If **M(a)** beats **a**, the modification is accepted. If **a** beats **M(a)**, the modification may be accepted (the probability of this increases with temperature and decreases if **M(a)** is badly beaten). The algorithm is run, starting at a high temperature which is gently lowered. This simulates the mechanism whereby a physical system, gently cooled, tends to reach a low-energy equlibrium position. Genetic algorithms apply where the strategies have more structure. (In fact, in most applications of simulated annealing, this extra structure is available.) Strategies are represented as conjunctions of elementary restrictions on the search space, or **decisions**. The evolutionary algorithm produces a population of strategies, rather than a single strategy. The idea is that by combining some parts of one good strategy with some parts of another, we are likely to produce a good strategy. Thus in generating the progeny of a population, we allow not only modifications or **mutation**, but also **reproduction** which combines part of one strategy with part of another. The basic step is to take a population and produce a number of progeny using a combination of mutation and reproduction. The progeny compete with the older generation, and each other, for the right to reproduce.

If reproduction is to maintain good performance, we need to be able to divide strategies in such a way that decisions which cooperate are likely to stay together. This is accomplished in an indirect and subtle manner. Strategies are represented as strings of decisions. The child, **R(a,b)**, of **a** and **b** is generated by randomly splitting **a** and **b** and joining part of one to part of the other. Thus, decisions which are close together in the string are likely to stay together. To allow cooperating decisions to become close together, we include **inversions** (which merely choose some substring and reverse it) among the possible mutations. These act together with reproduction and selection, to move decisions which cooperate closer to each other.

Nothing analogous to the temperature used in simulated annealing appears explicitly in the genetic algorithm. The likelihood that a nascent individual will survive to reproduce depends on the degree of competition it experiences from the rest of the population. As the population adapts, the competition hots up -which has the same effect as the cooling in the simulation of annealing.

Although genetic algorithms may be seen as a generalisation of simulated annealing, mutation plays a subsiduary rôle to reproduction. The population at any generation should be viewed as a repository of information summarizing the results of previous evaluations. Individuals which perform well survive to reproduce. Reproduction acts to propagate combinations of decisions occuring in these individuals. The better an individual performs, the longer it will survive and the more chances it has to reproduce. The relative frequencies with which various groups of decisions occur in the population record the degree to which they have been found to work well together. Holland has shown that (under appropriate statistical assumptions) the effect of the genetic algorithm is to use this information to effect an optimal allocation of trials to the various combinations of genes.

Applying the genetic algorithm to compaction.

The genetic algorithm evolves populations of individuals. In our implementation, each individual is characterised by a chromosome which is a string of genes. The length of chromosomes is not fixed. New individuals are produced by a stochastic mix of the classic genetic operators: crossover, mutation and inversion. Crossover picks two individuals at random from the population, randomly cuts their chromosomes and splices part of one with part of the other to form a new chromosome. Mutation picks an individual from the population and, at a randomly chosen number of points in its chromosome, may delete, create or replace a gene. Inversion reverses some substring of a randomly selected chromosome.

A Simple Layout Problem.

The layout problem addressed by our first prototype may be thought of as a form of 2-dimensional binpacking: A collection of rectangles is to be placed in the plane to satisfy certain design rules and minimise some cost function.

The simplest version of this problem (the one we address) has rectangles of fixed sizes, the design rule that distinct rectangles should not overlap, and cost given by area of a 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 four elementary constraints

p above q
p below q
p left_of q
p right_of q

is satisfied. Then, for a problem with **n** rectangles, we have $N = n^2 - n$ pairs and, *a priori*, 4^N elements in our search space. In fact, this estimate of the size of the problem is unreasonably large, there are ways of reducing the search space significantly; for example, "branch and bound" procedures have been used [Schlag *et al.* 1983].

Layout Strategies.

We consider layout strategies which consist of **consistent** lists of elementary constraints (as above). Given such a list, 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. (The procedure which interprets the constraints is very **un**intelligent. For example, it interprets '**p above q**' by ensuring that the y-coordinate of the bottom of **p** is greater than that of the top of **q**, even if **p** is actually placed far to the right of **q** (because of other constraints). Any inconsistent lists of constraints produced by the genetic operators are discarded.

Selection criteria.

Populations of consistent lists of constraints are evolved using various orderings for selection. When defining a selection criterion, various conflicting factors must be addressed. For example, our simplest criterion attempts firstly 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 numberof violations, strategies with smaller bounding boxes win. This simple prioritising of concerns led to the generation of some rather 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. Typically, these would have large areas and redundant constraints. The algorithm would later have to spend time refining these crude attempts. In an attempt to mitigate this effect, we added a further selection, favouring shorter chromosomes all other things being equal. Smith has pointed out that implementations of the genetic algorithm allowing variable length chromosomes tend to produce ever longer chromosomes (as chromosomes below a certain length are selected against). We did not find this an overwhelming problem as longer chromosomes were more likely to be rejected as inconsistent by the evaluation function. Nevertheless, we did find that the performance of the algorithm was improved by introducing a selection favouring shorter chromosomes.

We also experimented with trade-offs between the various criteria, established by computing a composite **score** for each strategy and letting the strategy with the better score win. We found that the genetic algorithm was remarkably robust in optimising the various scoring functions we tried. However, the results were often unexpected; the algorithm would find ways of exploiting the trade-offs provided in unanticipated ways. We have not yet found a selection criterion of this type which works uniformly well, over a range of examples. However, by tuning the selection criterion to the example, good solutions have been obtained.

A better way of combining our various concerns was found. Rather than address the concerns serially, or try to address all the concerns at once, we select a concern randomly each time we have a selection to make. A number of predicates for comparing two individuals were programmed. (For example, comparing areas of bounding boxes, comparing areas of design rule violations, comparing the areas of rectangles placed.) Each time we are asked to compare two individuals, we non-deterministically choose one of these criteria and apply it, ignoring the others. This works surprisingly well. It is easy to code in new criteria and to adjust the algorithm by changing the relative frequencies with which the criteria are chosen. The resulting populations show a greater variability than with deterministic selection, and alleles which perform well in some respects, but would have been selected out with our earlier deterministic approach, are retained.

Results.

Most of our experiments with this prototype have been based on problems with a large amount of symmetry, for which it is easy (for us) to enumerate the optimal solutions. If we actually wanted to solve **these** problems, other approaches

exploiting the symmetries available would certainly be more efficient. However, for the purpose of evaluating the performance of the genetic algorithm, we claim these examples are not too misleading. The algorithm is not provided with any knowledge of the symmetries of the problem nor of the arithmetical relationships between the sizes of the rectangles. For the purposes of evaluating the applicability of the genetic algorithm to layout compaction, the prototype is probably pessimistic. Real layout problems are far more constrained (by, for example, connectivity constraints). This not only reduces the size of the search space *per se*, but also appears to localise the interdependence of various genes making the problem more suitable for the genetic algorithm.

A naive analysis of a very simple example is instructive. The example consists of six rectangles, three 3 x 1 (horizontal) and three 1 x 3 (vertical). A minimal solution of this problem was found (consistently) in under 50 generations with 20 progeny per generation (1000 points of the search space evaluated).

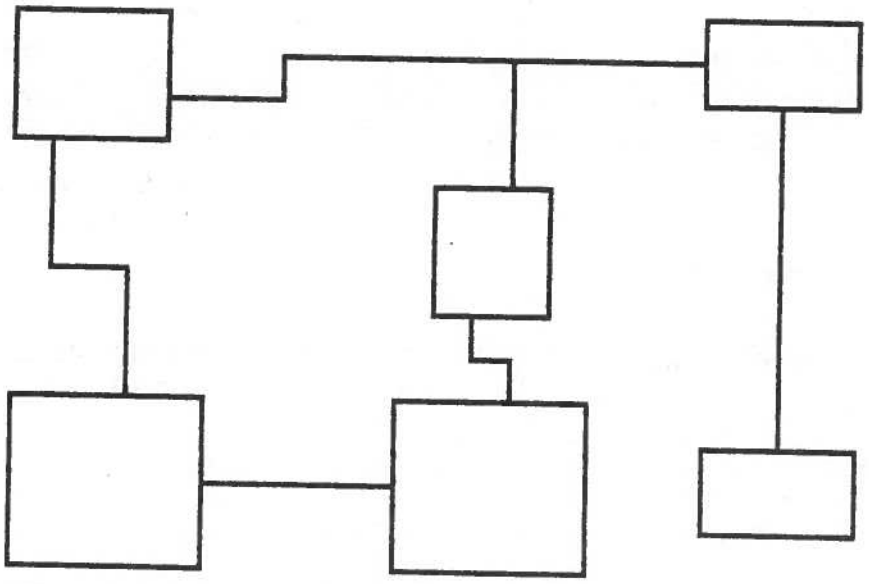
A solution to this problem must say how each of these rectangles is constrained, both horizontally and vertically. Thus the search space has 6^{12} (about 2×10^9) points. The problem has 8 basic solutions and a symmetry group of order 36. There are about 7.5×10^6 points/solution. Of these, we only examine some 10^3.

Representing Layout.

Our first prototype deals with a problem which has little direct practical significance for VLSI layout. (However, Rob Holte has pointed out that scheduling problems from operations research might be represented by minor variations on our prototype problem.) As a next step towards a practical layout tool, we have implemented a system which compacts a simple form of symbolic layout. The problem is to formalise the constraints implicit in the symbolic layout, and to find a representation, suitable for the genetic algorithm, for layout strategies.

We consider a symbolic layout of blocks connected by wires. The rectangles (blocks) are of fixed size and may be translated but not rotated. The interconnecting lines (wires) are of fixed width but variable length. The interconnections shown must be maintained, and no others are allowed. In addition, there are design rules which prohibit unconnected pairs of tiles (wires or blocks) from being placed too close together.

This form of the symbolic layout problem was introduced by [Schlag *et al.* 1983]. Here is their example of a simple symbolic layout:



We represent the problem at two levels:

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. as before), we use a fixed list of **structural constraints**, to represent the information in the symbolic layout, and **fundamental constraints** which represent the size limitations on tiles. Structural constraints have the following forms

v crosses h, N b v, S b v, E b h, W b h

where **v**, **h** are vertical and horizontal wires and **b** is a block. These constraints allow us to stipulate 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, x_posn v, whose names are self-explanatory. For each tile, we generate a list of fundamental

constraints expressing the relationship between the primitive layout elements arising from it. This representation allows both blocks and wires to stretch.

The example above is represented by declaring the widths of the wires and sizes of the blocks and then specifying the following list of constraints. (We use a LISP list syntax as it is more widely familiar, actually, our implementation is written in ML.):

```
((E B1 H2)
(crosses V3 H2)
(crosses V3 H3)
(crosses V4 H3)
(N B4 V4)
(W B5 H3)

(S B1 V1)
(crosses V1 H1)
(crosses V2 H1)
(N B2 V2)

(E B2 H5)
(W B3 H5)

(S B4 V6)
(crosses V6 H4)
(crosses V5 H4)
(N B3 V5)

(S B5 V7)
(N B6 V7))
```

Again, we evolve 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. Solutions to this problem were found in around 200 generations of 20 progeny, and this was reduced to around 150 generations when the algorithm was given a few "hints" in

the form of extra constraints. Watching the evolving populations showed that progress was rapid for around 50 generations. Thereafter, the algorithm appeared to get stuck for long periods on local minima (in the sense that one 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. This might suggest that either the population size (100) or the probability of mutation being used as an operator (0.1) is too small. We have not yet experimented with variations on these parameters. We think that better solutions would be either to introduce a further element of competition into the genetic algorithm by penalising configurations which become too numerous (implementing this is problemaical), or to evolve a number of populations allowing a limited degree of "intermarriage" (We are currently implementing the latter approach. If it is successful it will be a good candidate for parallel implementation.)

Conclusions.

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. We have experimented with various selection criteria based on combinations of the total intersection area, I, of overlap involved in design-rule violations, and the area, A, of a bounding rectangle. Experiments were made to compare various performance criteria based on combinations of the number of design-rule violations, and the area of a bounding rectangle. From our experience with the prototype, it appears that the choice of a selection criterion is an essential difficulty in applying the genetic algorithm to layout. The problem is that we must evolve populations of partial solutions (strategies), while the optimisation task is defined in terms of a cost function defined on layouts (solutions). To extend a (technology imposed) cost-function, defined on solutions, to the space of strategies, in such a way that the genetic algorithm will produce a solution (rather than just a high-scoring strategy), is a non-trivial task.

We intend to experiment with our second prototype in various ways before going on to implement a "real" system dealing with design-rules for a practical multi-layer technology. We will continue to experiment with selection criteria and we are

implementing the idea of having several weakly interacting populations running in parallel, described above. We also intend to integrate other, rule-based, methods with the genetic algorithm, automating the provision of "hints". Thus, a number of suggestions for strategies would be generated and passed to the genetic algorithm which would then explore combinations and variations of these.

Acknowledgements.

I would like to thank Steve Smith for introducing me to Genetic Algorithms, and Robert Holte for many stimulating discussions, his criticism and encouragement have been invaluable.

References.

Holland, John H. 1975. Adaptation in natural and artificial systems. Ann Arbor, University of Michigan Press.

Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi 1983. Optimisation by simulated annealing. *Science,* 1983, 220, 671-680.

Schlag, M., Y.-Z. Liao, and C.K. Wong 1983. An algorithm for optimal two-dimensional compaction of VLSI layouts. *INTEGRATION, the VLSI journal* 1 (1983) 179-209.

Smith, S.F. 1982. Implementing an adaptive learning system using a genetic algorithm. Ph.D. thesis. U. of Pittsburgh 1982.