Evolutionary Algorithms for the Physical Design of VLSI Circuits

James Cohoon[1], John Karro[2], and Jens Lienig[3]

[1] Department of Computer Science
University of Virginia
Charlottesville, VA 22903, U.S.A.
E-mail: cohoon@virginia.edu

[2] Computer Science Program
Oberlin College
Oberlin, OH 44017, U.S.A.
E-mail: john.karro@oberlin.edu

[3] Faculty of Electrical Engineering and Information Technology
Dresden University of Technology
Dresden, 01062, Germany
E-mail: jens@ieee.org

Abstract. Electronic design automation (EDA) is concerned with the design and production of VLSI systems. One of the important steps in creating a VLSI circuit is physical design. The input to the physical design step is a logical representation of the system under design. The output of this step is the layout of a physical package that optimally or nearly optimally realizes the logical representation. Physical design problems are generally combinatorial in nature and have very large problem sizes, thus necessitating the use of heuristics such as evolutionary algorithms. We review evolutionary algorithms for physical design and observe and analyze the common traits of the superior contributions. We also discuss important requirements for evolutionary-based approaches for even greater acceptance within the VLSI community.

1 Introduction

Electronic systems are at the core of our everyday lives. For example, they are in our financial networks, mass transit, telephone systems, power plants, and personal computers. Electronic systems are increasingly based on complex VLSI (Very Large Scale Integration) integrated circuits, or as they are known in the vernacular, chips.

A VLSI chip today can contain more than 100 million transistors. One of the main factors contributing to this rapid increase in complexity is an important computer science application area – electronic design automation (EDA). EDA systems are able to simplify the otherwise extremely complex design process of VLSI chips by hiding low-level circuit theory and device physics details from the designer. This encapsulation allows the designer to concentrate on the functionality of the circuit and ways to optimize that functionality.

An EDA system supports descriptions of hardware at many levels of abstraction. Hence it enables designers to work progressively down from an abstract level of design to the layout level. A layout is a complete geometric representation (i.e., a set of rectangles) of the masks which define how the individual layers of the circuit are to be produced.

EDA researchers have created methodologies and tools that allow circuit designers to not merely produce feasible systems, but rather optimal or near-optimal systems composed of tens of millions of circuit elements. The typical figures of merit for VLSI systems are concerned with maximizing reliability and circuit speed while minimizing the size of the physical package, power consumption, etc.

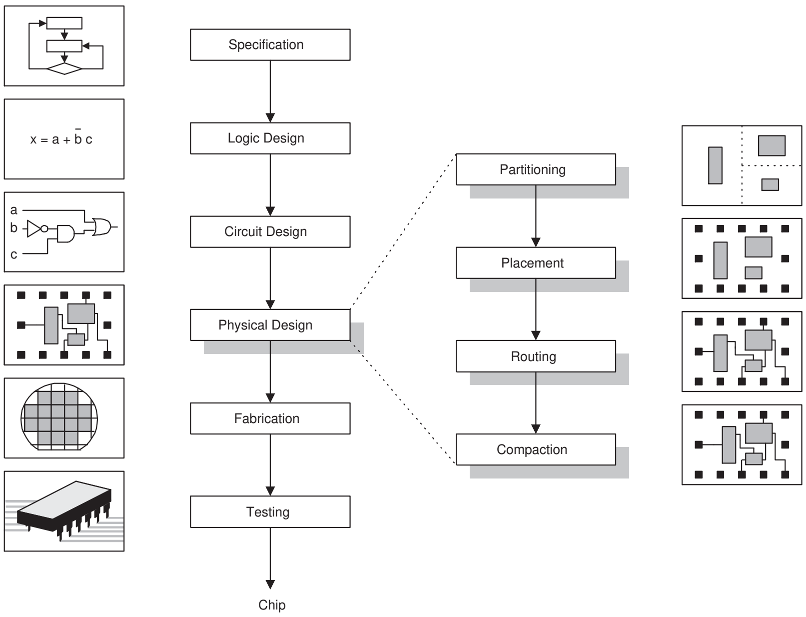


Fig. 1. Typical steps of chip design

In creating a VLSI system, a designer will typically iterate through six major steps (see Figure 1):

- *Specification*: produce a functional specification of the system under design.
- *Logic design*: transform the functional specification into a logical representation, typically via Boolean expressions.
- *Circuit design*: represent the logic representation as a circuit using components from an available library of modules (e.g., AND, NOT, and OR gates, standard cells, or building block macros).

- *Physical design*: translate the circuit design into a physical package representation. This representation is specified through a set of mask descriptions, which define how the individual layers of the integrated circuit are to be produced.
- *Fabrication*: use the physical package representation to fabricate an actual integrated circuit.
- *Testing*: determine whether there are manufacturing errors that prevent the integrated circuit from implementing the functional specification.

Because of the desire for optimality, the problems arising in the design and testing steps are generally combinatorial in nature. And because EDA problem instances are much larger than the instances of traditional problems, EDA problems are particularly difficult as compared to typical combinatorial optimization problems. For example, a circuit can be easily composed of over ten million circuit elements. As such, EDA practitioners have a strong tradition of quickly considering and adapting new and alternative solution techniques. Simulated annealing, an optimization technique that emulates the annealing of crystals, is a good example of this. This combinatorial optimization method was first proposed in the literature in 1983 by [Kirpatrick et al. (1983)] and by the following year, the major EDA conferences (e.g., *ACM-IEEE DAC* 1984 and *IEEE ICCAD 1984*) had multiple sessions on simulated annealing for EDA. Early adoption was again repeated on neural networks that simulate the organizing principles of nervous systems (e.g., [Yu (1989)] and [Zhang and Mlynski (1990)]).

There is an extensive amount of EDA literature associated with each of the system design steps. Detailed coverage of each of these steps – even limited to the application of evolutionary algorithms to EDA – is beyond the scope of a single chapter. Therefore, we limit ourselves to the application of evolutionary algorithms to physical design. Because physical design is rich in optimization problems, there have been many applications of evolutionary algorithms to it. (For more general overviews of the EDA field itself, we refer the reader to the three classic texts: [Preas and Lorenzetti (1988)], [Sherwani (1999)], and [Sarrafzadeh and Wong (1996)].)

Our discussion continues with a brief overview of physical design. Next we examine evolutionary algorithms and present a prototypical evolutionary algorithm for circuit partitioning, one of the major steps in physical design. We then present a systematic review of evolutionary algorithm investigations for the physical design process. These contributions are generally different than standard evolutionary algorithm investigations. One major difference is that the crossover and mutation operators for physical design evolutionary algorithms are typically very problem-specific. This specificity occurs because of the extreme importance of determining very high-quality solutions – therefore expert information on the likely form of solutions is included as much as possible. Another major difference is the concern for robustness. There exists a rich collection of design automation benchmarks (e.g., [EDA Benchmarks (1997)] and [IEEE International Testing Conference (1999)]). For a solution method to be accepted, it must be demonstrated to work consistently well on current benchmarks.

We then consider recent trends and discuss what actions must be taken to see even more use of evolutionary algorithms in physical design. We finish this discussion with a summary of our analysis.

2 Physical Design Overview

In physical design, the circuit designer produces a description of the physical layout of the circuit. The description is an assignment of circuit elements including interconnections to geometric coordinates. Depending upon the nature of a particular circuit element, an element may be placed either on a single planar layer or on several contiguous planar layers. The layout must satisfy the requirements of the fabrication technology (e.g., sufficient spacing between components of the circuit and sufficient feature size) and must optimize system characteristics (e.g., timing, speed and size).

The input to the physical design step is a logical description of the circuit that has been augmented to include a specification of the particular circuit elements implementing the various logic functions. The input is generally given as a *netlist*, where a *net* is a collection of circuit elements that must be interconnected for the circuit to be realized. Because circuit elements may in general be part of several different nets, a particular point of contact on the circuit element is specified for each of its interconnections. The point of contact is known as a *pin*. Due to its complexity, the physical layout problem is generally divided into four subproblems that are solved sequentially: partitioning, placement, global and detailed routing, and compaction (see Figure 1). Although these subproblems are NP-hard, they reduce the practical complexity to a manageable level.

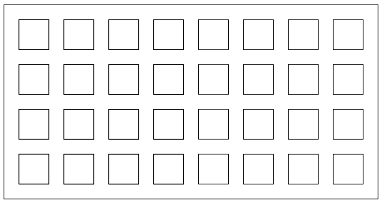
Partitioning is the task of dividing a circuit into subcircuits. The decomposition is performed in order to reduce the problem size. The reduction can be necessary because of problem complexity or because the physical package on which the circuit is to be implemented is of insufficient size for the entire circuit or both. The partitions are then implemented separately. The goal is to partition the circuit so that the sizes of the various partition elements are within prescribed ranges and so that the complexity of the interconnections between the partition elements is minimized. We note that the partitioning problem and the other physical layout subproblems may be invoked in a hierarchical fashion. For example, a circuit may be first partitioned into a collection of boards that comprise the backplane of the circuit. The boards may be further partitioned into the chips that make up a given board. A chip itself may be partitioned to produce a floorplan where the major circuit elements are to reside, and so on.

Placement assigns the circuit elements to their geometrical locations on the package. In placement terminology, a circuit element is known as a *cell*. Depending upon where in the invocation hierarchy a placement tool is invoked, a cell may be a single transistor, Boolean operator, functional unit, subcircuit, and so on. Depending upon the fabrication technology, the cells may be placed according to a particular design style (see Figure 2). In *standard cell design*, the circuit elements are rectangular with uniform height but varying widths and are placed in rows. The layout spaces between the rows and along the perimeter are called *channels* and are used for routing interconnections. (If more than two layers are available for routing, the area "above" the cells is used for routing as well.) In *macro cell design*, the cells can be of different sizes and rectilinear shapes and can be placed irregularly. For each of these design styles, a typical goal is to minimize the total layout area of the chip. Another typical goal is to minimize the length of the longest interconnection. Achieving this goal usually satisfies timing constraints. In *gate matrix design*, the circuit elements are rectangular, of uniform size, and are constrained to lie at positions arranged in a matrix pattern. Here the placement objective is to ensure routability by minimizing the congestion of the interconnections between the cells. Another popular design style utilizes FPGAs (field-programmable gate arrays) in a manner similar to gate matrix design.

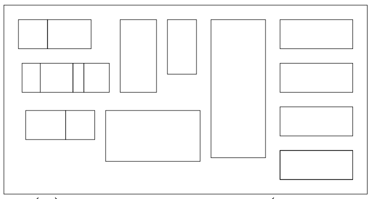
Global and detailed routing follow the placement activity. The tools for this phase determine the paths of the interconnections between the cells laid out during the placement procedure. The goal is to connect all pins that belong to the same net, subject to certain physical constraints (e.g., to minimize the overall area size), technological constraints (e.g., derived from mask requirements such as not to violate minimum spacing rules) and electrical constraints (e.g., to meet timing requirements). A global router typically works with many nets simultaneously and determines a coarse routing that loosely describes the forms of the various interconnections. The description for a particular net is typically the sequence of tiles (or channels) that the interconnection traverses. A detailed router can then take the *course routing* and assign specific resources and paths within the tile (channel) to realize the interconnection. Detailed routers typically route nets one at a time.

Depending upon the design style, routing may be the final step in physical design. For example, such is the case for gate matrix and FPGA design. However, for other design styles, such as macro cells, *compaction* is usually the final step in the physical layout design. A *compactor* transforms the symbolic layout produced by the preceding steps into a mask layout, i.e., the geometric mask features on the silicon. The objective of compaction is to minimize the size of the resulting circuit layout without impacting performance characteristics.

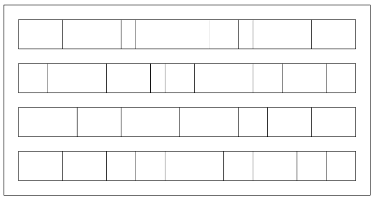
Once a circuit has been fabricated, tests are needed to determine whether defects have been introduced in the manufacturing process. For an overview of



(a) Gate matrix design



(b) Macro cell design (custom design)



(c) Standard cell design

Fig. 2. Layout styles

testing consider [Abramovici et al. (1994)] and [Bushnell and Agrawal (2000)], and for its application to genetic algorithms consider [Mazumder and Rudnick (1999)]. For some individual genetic algorithms applied in testing approaches consider [Corno et al. (1996)], [Ökmen et al. (1997)], [Rudnick et al. (1997)], [Saab et al. (1992)], and [Srinivas and Patnaik (1993)].

Logic Synthesis Although the primary interest of this chapter lies in partitioning, placement, routing, and compaction, we note that there is also evolutionary algorithms literature on related problems such as logic synthesis and testing. Logic synthesis is concerned with determining the optimal representation of a circuit with respect to a given design style. Clearly, different representations will lead to different circuit placement and routing characteristics. For an overview of logic synthesis consider [Villa et al. (1997)]

and [Cong and Ding (1996)] and for its application to evolutionary algorithms consider [Drechsler (1998)]. For individual genetic algorithm investigations examining logic synthesis issues consider [Becker and Drechsler (1994)], [Ohmori and Kasai (1997)], [Thomson and Miller (1997)], and [Vemuri and Vemuri (1991)]].

3 Evolutionary Algorithms Overview

Evolutionary algorithms are search algorithms that operate by evolving a population of solutions through repeated transformations. The fact that there is a population of solutions being simultaneously manipulated is one of the major differences between evolutionary algorithms and traditional search algorithms such as backtracking. Another important difference is that the operators performing the transformations on the population are drawn from evolution in nature.

There are three basic models of evolutionary algorithms: genetic algorithms (GA) [Holland (1975)], evolutionary programming (EP) [Fogel et al. (1966)], and evolutionary strategies (ES) [Rechenberg (1973)]. Although the three methods have some similarities, they were developed independently. For individual overviews of the GA, EP, and ES methods see respectively [Mitchell (1996)], [Fogel (1995)], and [Bäck (1996)]. For a general overview of these strategies, see elsewhere in this text, as well as [Bentley (1996)] and [Fogel (1995)].

In an *evolutionary algorithm*, associated with each solution is a *score*, which is a measure of the solution's *fitness*. In GA parlance, the encoding used to represent a solution is normally known as a chromosome or *string*. Typically considerable effort is spent by all three paradigms in developing an encoding that facilitates transformations. One of the principal transformation mechanisms for both the GA and ES methods is *crossover*. In a crossover, two solutions are combined to produce a new solution, where the new solution is called an offspring. The other principal evolutionary algorithm transformation mechanism is *mutation*. In a mutation, an existing solution is randomly modified. The GA, ES, and EP models all use this operator. In a GA, the mutation generally occurs in-place, i.e., the existing solution itself is modified. A standard GA limits the application of the mutation operator to offspring. However, there are many GAs that ignore this limitation. In the EP method, the mutation is made to a copy of a parent. (The EP chooses the parent in a manner that favors the more fit solutions and the parent's performance in a tournament.) ES variants that do not use the crossover operator apply the mutation operator in a manner similar to the EP method. ES variants that do use the crossover operator apply the mutation operator to the offspring in an in-place manner.

A basic evolutionary algorithm begins by constructing an initial *population* of solutions. The initial population is typically a collection of randomly generated solutions. As a result, it is expected that a diverse collection of solution characteristics will be generated. In particular, among the solutions of the initial population we expect to find both good and bad characteristics.

The evolutionary algorithm then produces a series of successor populations or generations. The production of a new generation begins by first performing a series of crossovers. For each crossover, two parents are chosen probabilistically from the prior generation. If the GA model is being followed, parent selection is done in a manner that probabilistically favors the more fit solutions. If the ES model is being followed, all parents have an equal probability of being favored. Next, the mutation operator is applied as described previously. Afterwards, a selection process chooses which members of the prior generation and the offspring are to survive to form the basis of the next generation. In a GA, the survival selection process is similar to crossover selection, i.e., the more fit solutions are probabilistically favored to survive. In a typical ES method, the best offspring solutions are the ones selected for survival. Note that some ES implementations are similar to the EP process that selects the best solutions from the combined offspring and parent population. This selection completes the construction of the new generation.

By favoring the more fit solutions in crossover and in survival, the evolutionary algorithm's search is directed towards solutions with superior characteristics. By performing incremental mutations, new solution characteristics can be introduced, which in turn prevents premature convergence of the population to one that contains merely locally optimal characteristics. Together these evolutionary processes are sufficient to produce an algorithm that converges in the limit to an optimal solution [Goldberg (1989)].

```
Initialize population to represent a random collection
   of parent solutions.
Evaluate the fitness of all members of the population.
While the population has not converged:
      Initialize the offspring population to be empty.
      While the number of offspring is insufficient do:
            Select two new members a and b of the
              parent population.
          Crossover a and b to produce offspring c.
          Add c to the offspring population.
      End
      Initialize the number of mutated members to 0.
      While the number of mutated members is insufficient:
          Randomly select a member to mutate.
       End
       Select the members of the parent and offspring
          populations to become the new parent population.
End
```

Fig. 3. A basic GA-like evolutionary algorithm

An algorithm description of a GA-like evolutionary algorithm is given in Figure 3. The outer loop iterates until the algorithm has converged. The test for convergence can be as simple as determining whether the solution quality is acceptable or running until a desired number of iterations have been performed. A more complicated test can measure the amount of improvement expected in future iterations and then halt when the expectation is sufficiently low. A population size is often used that is proportional to the problem instance. The proportionality is typically either linear or quadratic. With regard to the number of crossovers and mutations, one simple rule of thumb is three crossovers for every mutation.

Multi-objective Optimization One of the benefits of using a genetic algorithm is the relative ease of using them to solve multi-objective optimization. In such cases, we are searching a solution space with the objective of finding the solution that optimizes over two or more independent criteria. One of the first successful attempts to address such problems was devised by [Schaffer (1984]. Further research in the area is summarized in [Fonseca and Fleming (1995)] and [Goldberg (1989)].

4 A Sample Evolutionary Algorithm for Partitioning

We next describe a simple evolutionary algorithm for a basic partitioning problem. We do so by providing methods for the initialization of the population, a crossover operator, and a mutation operator.

The partitioning problem that we consider has the following description:

- Input: A list $N = \{N_1, \cdots, N_n\}$ of n nets. A list $C = \{C_1, \cdots, C_m\}$ of m circuit elements.
- Output: A bipartition B of the circuit elements to sides 0 or 1 that minimizes the number of *cut nets* – nets that have circuit elements on both sides of the partition. If $B_i = 0$, then element C_i lies on side 0, and if $B_i = 1$, then element C_i lies on side 1. Approximately one-half of the circuit elements are assigned to side 0 and the other half are assigned to side 1. The difference in the number of elements on the two sides can be no greater then k percent of the total number of elements.

Although simplistic, the problem is sufficiently close to standard EDA partitioning problems to acquaint the reader with the problem.

A straightforward encoding for a population member would be a string of m bits, where the i^{th} bit in the string is 0, if the solution being represented has C_i on side 0; the i^{th} bit in the string is 1, if the solution being represented has C_i on side 1. The score of a solution would be the number of nets cut by the solution being represented. The fitness of a solution could be the difference of its score from m.

To construct an initial population, one could generate a series of random bit patterns of length m such that the difference in the number of zeros and ones in a given string pattern is no greater than k percent.

A variety of straightforward crossover operators exist. For example, a one-point crossover operator [Goldberg (1989)], acting on parents a and b, first randomly picks an index i into a. Then, the first i bits of parent a are concatenated with the last $m-i$ bits of parent b to produce an offspring c. If feasible solutions are required and c is infeasible, then excess zeros or ones can be randomly complemented. As an example, suppose there are eight circuit elements with $a = 10100110$ and $b = 01100101$ and the one-point index is 5. The offspring c is $\underline{10100}101$ (the underlined portion comes from a). Because the number of zeros and ones are equal, no feasibility correction is required.

For another example, consider a two-point crossover operator. A two-point crossover, operating on parents a and b, first randomly selects two different indices i and j. If $i < j$, then bits 1 through $i-1$ of b are concatenated with bits i through j of a that are concatenated in turn with the last $m-j$ bits of b to produce the offspring c. If instead $i > j$, then the roles of a and b are reversed (i.e., bits 1 through $i-1$ of a are concatenated with bits i through j of b that are concatenated in turn with the last $m-j$ bits of a to produce the offspring c). As an example with the same values of parent a (10100110) and parent b (01100101) and $i = 6$ and $j = 3$, then offspring $c = \underline{101001}10$ (the underlined portion comes from a). A two-point crossover operator is usually preferred to a one-point crossover operator as more varied combinations of the parents are possible.

Straightforward mutation operators are also possible. One possibility is to randomly select a bit position and to complement its value. If the complementation makes the solution infeasible, a random bit with that same value is selected and complemented. For example, if $a = 10100110$ is selected for mutation and the third bit is to be complemented, the result is $10\underline{0}00110$ (the underlined portion shows the change that is introduced). For reasonable values of k (the maximum percentage difference between the two partition sizes), the new solution would be feasible. For other values of k, either bit 2, 4, 5, or 8 of a would need to be complemented. For a more radical mutation, a range of bits could be complemented.

For small circuits, the above methods produce good solutions. However, because most interesting EDA problems do not have small instances, more involved strategies are necessary. Such strategies are discussed in the next section, where we give a systematic overview of evolutionary algorithms that have been successfully applied to the major steps of the physical design of VLSI circuits. Although EDA did not immediately add evolutionary algorithms to its basic tool chest, evolutionary algorithms have been consistently used in the field since 1987.

5 Application of Evolutionary Algorithms to the VLSI Physical Design

In this section we look at the four primary steps of design automation (partitioning, placement, routing, and compaction), and provide an overview of the work that has been done in applying evolutionary algorithms to these problems. We present a brief description of each major tool, discuss the primary characteristics of the algorithms (e.g., crossover and mutation operators and population representation), and argue the merits and problems of each method.

Ideally, we would provide both analytical runtime bounds of each tool as well as experimental comparisons. This, however, is not currently a feasible goal. Runtime bounds of evolutionary algorithms are, at best, quite difficult. Because of the probabilistic nature of the algorithms, and because of the dependency of the runtime on subtle characteristics of the solution space of any particular problem, it can be almost impossible to achieve a useful bound. It is sometimes feasible to achieve loose worst-case bounds, but the bounds are frequently so loose that they cannot be considered useful information.

In presenting an experimental analysis, we run into the problems of experimental inconstancy between papers. In order to truly compare two evolutionary algorithms, we would need to run them on the same problem instance, using comparable architectures, with consistent parameters. For example, we cannot meaningfully compare the results and runtimes when using two possible population representations if the convergence criteria vary between the experiments. As there is no constant methodology for testing evolutionary algorithms, each research group uses the experiments they see fit. Hence, we cannot meaningfully compare the resulting reports of two methodologies if the researches in question did not employ consistent tests.

Partitioning As mentioned previously, the goal of partitioning is to divide the circuits into smaller parts that would be separately implemented. Thus, in performing the partition we want parts with both similar complexities and minimal mutual dependencies. Mutual dependencies are generally expressed in the form of interconnect complexities between different parts. Most algorithms for partitioning focus on these interconnections between different parts with the goal to minimize overall interconnection costs.

The first evolution-based algorithm for solving the partitioning problem was published by [Saab and Rao (1989)]. In contrast to previous heuristic algorithms that usually optimize on only one constraint, this approach is capable of handling both a number of constraints and a number of objectives. This capability is achieved by using a multi-constraint, multi-objective fitness function.

The algorithm consists of an initial partitioning (based on a bin packing algorithm) and a so-called *evolution function*. The evolution function includes three steps: migration, distribution, and calculation of parameters. The algorithm works with *one* problem solution. A crossover operator is not used.

The algorithm yields multi-way partitions with fairly balanced sizes and a small number of pins for each part. The presented strategy has a reasonable execution speed that is comparable to other published heuristic approaches.

[Hulin (1991)] and [Hulin (1991a)] investigate different coding schemes for the problem of circuit partitioning. The proposed GA is specifically tailored for the partitioning of circuits with complex bit-slice components and uses a special two-step coding of partitions.

The initial population is generated randomly, i.e., the circuit is partitioned by chance and then decoded. The algorithm consists of a crossover, a mutation operator, and a deterministic improvement strategy. The runtime of the algorithm is not competitive.

A parallel GA by [Cohoon et al. (1991)] is based on a population structure that involves subpopulations which have their isolated evolution occasionally interrupted by inter-population communication. Every 50 generations, each subpopulation exchanges 10 – 20 percent of its members with neighboring subpopulations. Partitions with design violations are tolerated by means of a special penalty function. Results show that multiple subpopulations enhance the solution quality by exploiting wider regions of the search space.

A hybrid GA for the ratio-cut partitioning problem is presented by [Bui and Moon (1994)]. (The ratio-cut partitioning problem considers not only the cut size but also the size of the partitions.) Here the problem is formulated in terms of a hypergraph. The modules are mapped to vertices and the nets are mapped to hyper-edges. Each solution in the population is represented by a chromosome. The number of genes in a chromosome equals the number of modules in the graph. The chromosome is a binary string, where each module has a corresponding location on the string. A location has the value 0 if the corresponding module is on the left side of the bipartition, and has value 1 otherwise. Before the GA is executed, the ordering of these genes is determined by a depth-first search to improve the performance of the GA. Traditional crossover and mutation operators are combined with a fast partitioning heuristic applied to each offspring as an improvement operator. The performance of the algorithm is compared with two other partitioning approaches, using benchmark data sets. Averaged over all benchmarks, the presented algorithm achieves better results than the other approaches, with a similar runtime for smaller graphs and a shorter runtime for the largest graphs.

Another hybrid approach is published by [Varanelli and Cohoon (1995)]. It combines a simulated annealing method with a GA. The main motivation for this approach is the parallelization of the simulated annealing strategy by replacing its single solution search process with a population-based approach using a GA. Benchmark comparisons are not presented.

While a number of GAs have been proposed for dealing with the partition problem, none of them are truly competitive with the state-of-the-art tools in the area. See [Alpert and Khang (1995)] for a full survey on partitioning, including a number of the most sophisticated probabilistic and deterministic algorithms for the problem. For a tutorial on GA partitioning, consider [Mazumder and Shahookar (1999)].

Placement The placement step assigns physical locations on the chip to the components (cells, gates, etc.) of the circuit. This assignment is done by considering placement constraints (e.g., no overlap is allowed between cells) and objectives (e.g., fulfilling timing constraints). The objectives differ widely according to the specific design concept. Placement algorithms can be divided into algorithms for standard cell design, macro cell design, gate matrix design, and FPGAs according to the variation in size and location of these cells (see Figure 4).

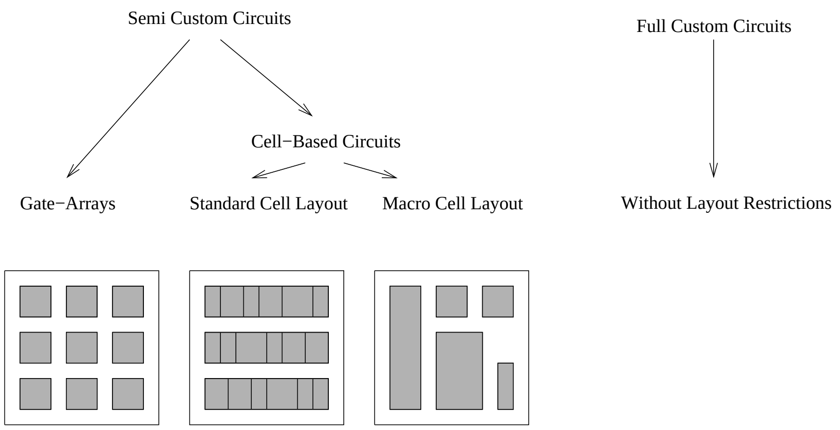


Fig. 4. Placement styles

The task of placement can usually be considered as two related problems: a two-dimensional bin packing problem and the problem of minimizing the interconnection costs. The bin packing problem targets the optimal or sub-optimal composition of components with different sizes and shapes. The minimization of interconnection costs optimizes the length of the routing either by considering the overall routing length (or cost) or by considering the individual interconnection costs between different components.

One of the main tasks of a placement algorithm is the definition of an appropriate cost function which also considers the needs of the subsequent routing procedure. Most placement algorithms are limited to two types of cost functions: they consider either the estimated length of the interconnect (as needed for timing constraints) or the area density of the interconnect (in order to avoid routing congestions).

After the pioneering work of [Cohoon and Paris (1987)], further applications of evolutionary algorithms for standard cell placement have been presented, such as [Kling and Banerjee (1989)], [Kling and Banerjee (1990)], [Shahookar and Mazumder (1990a)], and [Shahookar and Mazumder (1990b)]. The last two approaches work with an initial population that has been generated randomly. Each individual consists of an array of four integers, containing the cell number, the x and y coordinates, and the location of the cell. The population size is kept at 24 individuals. Three crossover operators are applied that cut the parent string at random positions. The crossover operators vary according to their "rebuilding strategy" of the descendants.

The mentioned approaches produce high-quality placements of real-world VLSI circuits that can compete with sophisticated simulated annealing-based placement strategies. The published runtimes are up to 6 hours for the algorithm of [Kling and Banerjee (1989)], while runtimes of up to 12 hours are presented in [Shahookar and Mazumder (1990b)].

[Mohan and Mazumder (1993)] reduced the runtime significantly by developing a parallel implementation of a GA that runs on a distributed network of workstations. The total population is split over different processors and a migration mechanism is used to exchange genetic material between them. While the placement results are similar to a sequential GA, an almost linear speedup can be achieved with this method.

We next discuss investigations that use evolutionary algorithms for macro cell placement. The approach of [Chan et al. (1991)] is based on a two-dimensional bitmap representation of the macro cell placement problem. Another representation scheme, a binary tree, is applied by [Esbensen (1992)]. [Esbensen and Mazumder (1992)] consider a combination of a GA with a simulated annealing-based optimization strategy. The experimental results suggest that a mixed strategy performs better than a pure GA for the macro cell placement problem. The results are better or comparable to previously published results of placement benchmarks. However, the runtime is not as competitive.

[Schnecke and Vornberger (1996)] present a parallel GA for the macro cell placement problem. They use the term "self-adaptation" to describe a search process by which several "islands" execute sequential GAs with different strategies. At fixed intervals these strategies are ranked and each strategy is improved by assimilating the characteristic parameters of the next better strategy. The experimental results suggest that "self-adaptation" improves both the performance and the robustness of the algorithm. Benchmark results and runtimes are not presented.

Since macro cells are constructed in a hierarchical way, their shape may not be fixed. This more general placement problem is called *floorplanning*. A hybrid GA for the floorplan area optimization problem has been presented by

[Rebaudengo and Reorda (1996)]. They introduce several heuristic operators in addition to a pure GA. Their approach adaptively provides the activation probabilities of the operators. Experimental results show that the proposed method is competitive with other approaches in both solution quality and runtime. Since the complexity of the algorithm grows linearly with the problem size, the approach can be applied to floorplans larger than any benchmark previously considered.

An application of a GA for the placement of gate matrix design has been published by [Shahookar et al. (1993)]. The approach uses the Genesis package ([Genesis (1987)]) as the basic GA. This package is modified with a special algorithm for constructing permutations that considers only a small subset of the solution space. The results are compared with only one previously published algorithm. The runtime is in the order of minutes.

It can be observed in the above-mentioned evolutionary algorithms for placement that "traditional" bit-string representations are not used. (The obvious reason is that symbols in the solution string cannot be repeated when applied to the placement problem.) Hence, "traditional" crossover strategies, such as one-point crossover operators, have not been applied either.

As mentioned earlier, crossover is the primary method of optimization during the evolutionary process. In the case of placement, crossover works by combining sub-placements of two different parent configurations in order to generate a new (and perhaps better) placement. This process might generate conflicts that need to be resolved by the crossover operator. Despite the variety of crossover operators presented, all include conflict resolution methods tailored to the placement problem. Three of the most promising ones are mentioned here (see Figure 5, described in [Mazumder and Rudnick (1999)]).

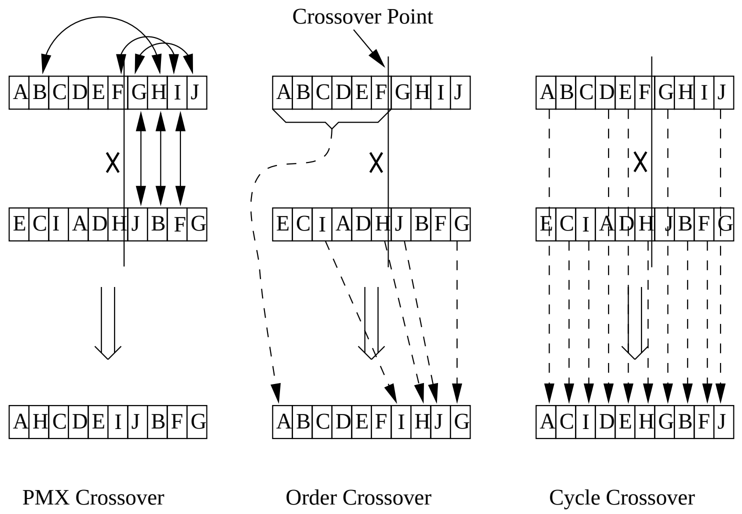


Fig. 5. Crossover operators suitable for placement

The *partially mapped crossover* (PMX) chooses a random cut point in both parents and considers the segments following the cut point as a partial mapping of the cells to be exchanged in the first parent to generate the offspring [Goldberg and Lingle (1985)]. It takes corresponding cells from the segments of both parents, locates both these cells in the first parent, and exchanges them. Hence, a cell segment in the first parent and a cell at the same location in the second parent define which cells in the first parent have to be exchanged to generate an offspring.

The *order crossover* [Davis (1985)] also chooses a random cut point in both parents. It then copies the array segment to the left of the cut point from one parent to the offspring. The remaining (i.e., right) portion of the offspring array is filled by going through the second parent and taking, in order, those elements which were left out.

The *cycle crossover* [Oliver et al. (1985)] generates an offspring in which every cell is in the same location as in one parent or the other. This crossover operator tries to avoid cell conflicts by finding non-overlapping sets of cells to pass from the two parents. Since non-overlapping cells are not contiguous, a special selection process is applied.

While GAs have enjoyed some success when applied to placement algorithms – some of the tools are competitive with conventional algorithms – they take considerably longer to match the non-GA solutions. In the area of gate array placement, tools such as VPR [Betz and Rose (1997)] or Spiffy [Karro and Cohoon (1999)] are able to outperform any GA tools currently in the literature.

For a general survey of placement literatures see [Sherwani (1999)], as well as the work by [Sarrafzadeh and Wong (1996)]. For a tutorial on GAs as applied to the problem of placement, consider the work in [Mazumder and Rudnick (1999)].

Routing Routing is the process of connecting pins subject to a set of routing constraints. The routing of VLSI circuits (and also of Multi-Chip Modules, MCMs) is usually divided into *global routing* (to assign nets into certain routing regions) and *detailed routing* (to assign nets to exact positions inside a routing region).

Routing of MCMs and PCBs (Printed Circuit Boards) is based on fixed routing areas with numerous routing layers. This leads to a non-planar routing topology. This characteristic and the desire to achieve a global-optimal routing result prevent the separation into global and detailed routing. Instead, a so-called “free” (or custom) routing is used that considers the entire routing region of the substrate (see Figure 6).

Global routing algorithms are usually based on graph-search strategies where the nodes of the graph represent separate routing regions (e.g., tiles or channels) of the circuit. The edges of the graph mark the routing capacities of

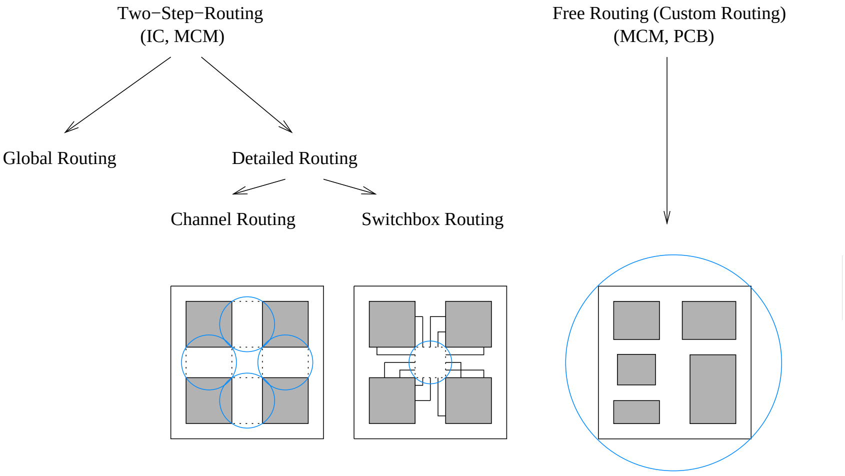


Fig. 6. Routing strategies

the routing regions. These routing capacitances are often estimates because they can be verified exactly only in the subsequent detailed routing.

The regions are routed sequentially during detailed routing. According to the position of the pins, detailed routing can be separated into *channel routing* (pins are only located on two parallel sides of the routing area) and *switchbox routing* (pins are placed on all four sides of the routing area), as shown in Figure 7. Detailed routing sometimes also includes the determination of the exact placement location of the cells and their pins, respectively.

Contrary to placement, where numerous evolutionary algorithms have been published, there have not been as many successful publications of evolutionary routing algorithms. This condition seems to be a consequence of the larger complexity of the routing task. In particular, the appropriate coding of the routing poses a major obstacle with the result that almost all publications in this field are limited by specific constraints induced by the encoding.

Related to the routing problem is the Steiner tree problem – finding a minimum length interconnection in the plane. For an introduction to this problem see [Hwang et al. (1992)]. For a discussion of Steiner-tree-related GAs see [Esbensen and Mazumder (1994)], [Julstrom (1993)], [Kapsalis and Rayward-Smith (1993)], and [Esbensen and Mazumder (1999)].

To our knowledge, only one evolutionary algorithm for global routing has been reported ([Esbensen (1994)]). Before the global routing process itself is initiated, a rectilinear routing graph is extracted from the given placement. Routing is then performed in terms of this graph; that is, computing a global route for a net is done by computing a corresponding path in the routing graph. In other words, each edge of the graph corresponds to a routing channel, and each vertex corresponds to the intersection of two channels. Vertices representing the terminals of the net are added to the routing graph at ap-

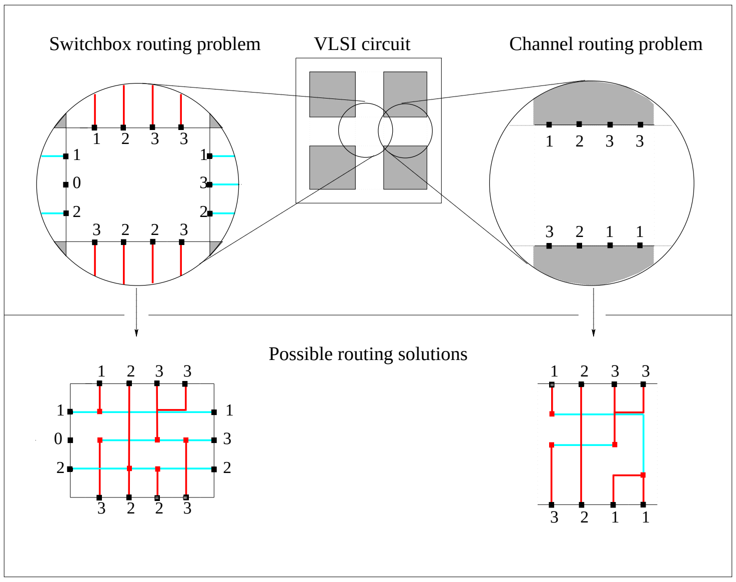


Fig. 7. Switchbox and channel routing

propriate locations. Finding the shortest path for the net is then equivalent to finding a minimum cost subtree in a graph. This graph spans all of the added terminal vertices, assuming that the cost of an edge is defined as its length. When a net has been routed, its terminal vertices are removed from the routing graph, thereby significantly reducing the problem size.

The global route algorithm is based on a two-phase router. In the first phase, a GA for the Steiner problem in the above-mentioned graph is used to generate a number of distinct, alternative routes for each net. Then, in a second phase, another GA selects a specific route for each net (among the alternatives given from the first phase), such that the overall layout area is minimized.

The router is superior to TimberWolfMC ([Sechen (1988)]), a state-of-the-art simulated-annealing-based router, with respect to solution quality. However, the runtime is slower by a factor of 50 which limits practical applications of this algorithm.

[Lin et al. (1989)] developed a rip-up-and-rerouter which is based on a probabilistic rerouting of nets of one routing result. (A rip-up-and-rerouter iteratively improves the routing result by deleting and rerouting previously routed nets.) The initial population (consisting of nets as individuals) is generated with both a shortest path algorithm and a random path strategy. Competitive results are presented for channel and switchbox routing benchmarks. The runtimes vary between a couple of minutes for small examples and up to 24 hours for large channel routing problems.

The router of [Geraci et al. (1991)] combines the steepest descent method with specific features of GAs. The crossover operator is restricted to the exchange of entire nets and the mutation procedure performs only the creation of new individuals. The presented results are limited to simple VLSI problems, and no runtime remarks are made.

Some evolutionary algorithms have been presented for the restrictive channel routing problem; [Rahmani and Ono (1993)] have an example of this, as do [Rao et al. (1994)], and [Rao et al. (1995)]. Here, all vertical net segments are located on one layer and all horizontal segments are placed on a second layer. This and other restrictions make these approaches unusable for real-world VLSI channel routing problems.

The GA for channel routing published in [Lienig and Thulasiraman (1994)] is based on a problem-specific representation scheme. Here the layout is coded in a three-dimensional lattice-like chromosome with the cells representing different coordinate points of the routing solution. The value of a cell indicates which net is routed at this coordinate point in the routing solution. A negative cell value indicates a fixed assignment (e.g., a pin) and zero indicates that the area is unused (Figure 8, as described in [Lienig and Thulasiraman (1994)]).

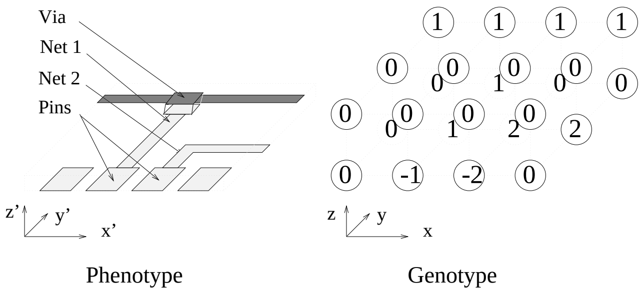


Fig. 8. A sample routing (left) and its genetic encoding (right)

This three-dimensional encoding scheme ensures that good "routing islands" in the routing structure are preserved as compact high-fitness building blocks in the chromosome. Consequently, these building blocks have a high probability of being transferred intact and recombined with other high-quality building blocks in offspring solutions. Furthermore, this encoding scheme enables a simple monitoring of the routing constraints directly in the chromosome.

The genetic operators are also specifically developed for the channel routing problem. The results are either qualitatively similar to or better than the best published results for channel routing benchmarks. The runtime of the algorithm (between one and fifty minutes) is not as competitive.

A GA for the channel routing problem, which includes problem-specific heuristics, is presented by [Göckel et al. (1997a)]. The reported results are identical to those in [Lienig and Thulasiraman (1994)], but with shorter runtimes. An extension of this algorithm for the routing of channels with more than two layers (the so-called "multi-layer detailed routing problem") has also been published [Göckel et al. (1997b)]. Instances of up to five layers and more than 60 nets are considered, with runtimes ranging from several minutes up to 24 hours.

The GA by [Lienig and Thulasiraman (1996)] for switchbox routing is similar to their channel router. The genotype is essentially a lattice corresponding to the coordinate points of the layout (as in the channel router). Crossover and mutation are performed in terms of interconnection segments. The algorithm assumes that the switchbox is expandable in both directions. Subsequently, these extensions are reduced with the goal of reaching the fixed size of the switchbox. While more costly in runtime, on numerous benchmark examples the GA produces solutions with equal or better routing characteristics than the previously best published results.

[Lienig (1997)] presents a parallel GA for the channel and switchbox routing problem. The problem is based on the theory of punctuated equilibria from [Eldrege and Gould (1972)] and [Cohoon et al. (1991)]. A GA with punctuated equilibria is a parallel GA in which independent subpopulations of individuals, each with their own fitness function, evolve in isolation. Periodic exchanges of individuals (migration) are then performed when a state of equilibrium throughout all the subpopulations has been reached (illustrated in Figure 9).

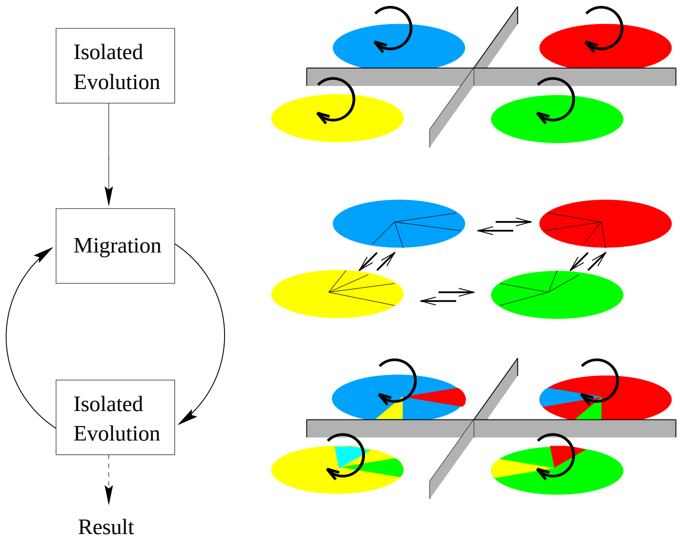


Fig. 9. Punctuated equilibria model with four subpopulations as shown in [Lienig(1997)]. Subpopulations evolve in isolation ("Isolated Evolution"), periodically interrupted by a limited exchange of individuals ("Migration").

A set of n individuals (problem solutions) is assigned to each of the N processors, for a total population size of $n \times N$. The set assigned to each processor is considered to be a subpopulation. The processors are connected by an interconnection network with a torus topology. Thus, each processor (subpopulation) has exactly four neighbors.

First, the main process creates an initial subpopulation at each processor. This initial subpopulation consists of randomly constructed (i.e., not optimized) routing solutions. They are designed by a random routing strategy which connects net points in an arbitrary order with randomly placed interconnections. The main process consists of a number of iterations, called epochs. During an epoch, each processor, disjointly and in parallel, executes the sequential GA on its subpopulation for a certain number of generations. Afterwards, each subpopulation exchanges a specific number of individuals (migrants) with its four neighbors. Here, the individuals themselves are exchanged, i.e., the migrants are removed from one subpopulation and added to another. Hence, the size of the subpopulations remains the same after migration and the assimilation of migrants is simply a fitness recalculation. The process continues with the separate evolution of each subpopulation during the next epoch. At the end of the process, the best individual seen during execution constitutes the final routing solution.

Experimental results showed that, when applied to the routing problem, the parallel GA – based on concepts of punctuated equilibria – consistently performs better than a sequential GA.

In investigating the algorithm's parameters, the following conclusions have been reached by [Lienig (1997)]:

- A small number of migrants (1 to 3 per neighbor) combined with a "moderate" epoch length (approximately 5 to 10 percent of the total number of generations) leads heuristically to the best results.
- *Variable* epoch lengths determined via equilibrium measures within subpopulations achieve overall results that are slightly better than those obtained with (near-)optimized *fixed* epoch lengths. Practical applications of this "strict punctuated equilibria method" require the user to weigh the advantage of this "self-adjustment" against its main drawback, decreased time efficiency.
- Quality constraints on the migrants (e.g., to be above median fitness) do not improve the overall behavior of the algorithm. On the contrary, placing quality requirements on the selection of migrants led to premature stagnation.
- Given a sufficient number of individuals per subpopulation, a larger number of parallel evolving subpopulations will produce better routing results (for a fixed number of total evaluations). The size of the problem and the minimal subpopulation size have a direct correlation that must be taken into account when dividing a population into subpopulations.

Similar to placement, all published evolutionary algorithms for routing differ widely from "traditional" GAs. This is due to sophisticated representation schemes that often incorporate layout-specific constraints. Genetic operators, such as crossover and mutation, include the consideration of design rules in order to avoid unresolvable conflicts in the offspring(s). A significant improvement regarding solution quality, robustness, and runtime has been achieved by using parallel implementation strategies.

As with partition and placement, evolutionary-algorithm-based routing approaches have proved effective, but are generally not competitive with the state-of-the-art tools in the literature in terms of runtime and robustness. For discussions of such routing tools, see [Sherwani (1999)].

Compaction As mentioned earlier, compaction transforms the symbolic layout into a mask layout with the goal of minimizing the size of the resulting circuit representation. It usually consists of two separate steps: (1) transforming the symbolic layout to a mask layout, and (2) compacting the layout area. Both steps have to be performed under strict consideration of the design rules. The first step results from the use of symbols as representation of circuit components ("schematic-driven design"or SDC) that obviously cannot consider *all* geometrical features of these components. Generating the mask layout also includes detailed characteristics of the final fabrication technology.

The subsequent minimization of the layout area utilizes layout regions which allow a compaction in one (or two) direction(s) based on their occupation. (For example, an area with only horizontal routing segments without vias usually allows a vertical compaction.) Due to the used strategies, compaction algorithms are divided into one-dimensional algorithms (compaction in one direction at a time), two-dimensional algorithms (compaction in the x- and y-direction simultaneously), and topological algorithms (moving separate cells according to routing constraints).

To the best of our knowledge, the only applications of an evolutionary algorithm for compaction have been advanced by two papers: [Fourman (1985)] and [Goodman et al. (1994)]. (In a related paper, [Hsieh et al. (1988)] make use of similar principles to apply the technique of *simulated annealing* to the compaction problem.) In the first of these, [Fourman (1985)] describes two prototypes of GAs that perform compaction of a symbolic circuit layout.

The algorithm considers symbolic placement and routing results which are characterized by their routing requirements and the technological constraints. These designs are encoded into individuals. The individuals undergo an evolutionary process with the selection based on both the number of design rule violations and the size of the allocated area.

Although Fourman's results are limited to very simple layout structures, he does propose a new problem-specific representation for layout design that includes constraints of the compaction process. Neither runtimes nor real VLSI applications are discussed.

In [Goodman et al. (1994)], the authors investigate a methodology known as *hierarchical chromosome representation* (HCR) and its application to several related problems – including compaction. By applying this new methodology to the compaction problem, it is argued that many of the constraints and limitations of the techniques from [Fourman (1985)] can be overcome – but the discussion is only theoretical. No actual code or results are presented.

6 Conclusions

Since the problems encountered in the physical design of VLSI circuits are very complex, evolutionary algorithms have considerable potential in this field. Accordingly, there has been a significant increase in evolutionary algorithms that have been developed for VLSI physical design applications. This development is characterized by some specific features, such as:

- Successful evolutionary algorithms in VLSI design differ widely from the "traditional evolutionary algorithm concept" in that they incorporate problem-specific knowledge into their operators. Furthermore, universal fixed-length binary implementations are often replaced by problem-specific representation schemes.
- Combinations of evolutionary algorithms with other optimization strategies, e.g., simulated annealing or fast deterministic algorithms, are no longer an exception. Their incorporation often seems to be the only way to obtain an approach that offers a competitive runtime.
- Useful parallel implementations of evolutionary algorithms have emerged in VLSI design. The increasing availability of high-speed local computer networks and inexpensive parallel computers encourages the implementation of parallel evolutionary algorithms with improved solution quality and runtime characteristics.

While these trends look promising, we cannot ignore the fact that most evolutionary-based VLSI design algorithms are not applicable in real-world VLSI design systems. This condition is underlined by the fact that currently no commercially available EDA system contains any evolutionary-based algorithms.

What are the reasons for this obvious conflict between promises of an emerging solution technique and the problems of successful adoption? One investigation [Lienig (1996)] using different software vendors showed two major drawbacks of evolutionary algorithms: robustness and failed expectations. Evolutionary algorithms are of probabilistic nature, hence *one* execution does not necessarily deliver the best possible result. This property is one of the reasons that many very good results claimed in publications could not be repeated in successive runs using practical applications – often only "fine tuning" of parameters would have lead to the expected superior results.

These are some of the reasons that are responsible for the gap between successful academic results with evolutionary-based VLSI physical design algorithms and the obvious neglect of these algorithms in commercial design tools. However, we believe that the above-mentioned problems can be avoided if the following guidelines are considered when developing evolutionary algorithms for the VLSI design problem:

- Since evolutionary algorithms are probabilistic approaches, it is very important to investigate the influence of the randomness on the solution quality. A VLSI designer will always hesitate to use an algorithm he/she has to run several times in order to achieve a usable and competitive result. Furthermore, the parameters used for running the evolutionary algorithm should be designed in a way to provide an easy adjustment to the specific design problem without extensive experimentation.
- The probabilistic nature of the algorithm should be discussed whenever solutions of evolutionary algorithms are presented. In other words, it is necessary to clearly express how the presented results have been obtained. The randomness of the results also has to be taken into account when comparing the runtime of an evolutionary algorithm with deterministic algorithms that require only one execution to achieve their results.
- Whenever possible, the presented approach should be compared with state-of-the-art algorithms regarding both solution quality and runtime. Many of the surveyed evolutionary algorithms are competitive with respect to the solution quality only. However, to be of interest in the VLSI design area, a measure of running time must also be included. Less interesting to the community is the number of recombinations, points visited in the search space, and so on.
- Constraints of other algorithms have to be taken into account when comparing with their results. For example, it is not a fair comparison when the routing quality of an evolutionary algorithm is expressed only in terms of the number of vias ([Geraci et al. (1991)]) and then compared with the results of other approaches that minimize the net length concurrently.
- Due to the large number of CAD tools in VLSI design, benchmark data are available for all major design steps, e.g., [EDA Benchmarks (1997)]. An evolutionary algorithm developed for VLSI design will not create any interest within the VLSI community unless its performance is tested with the appropriate benchmarks. It is only by examining the results of these benchmarks that we can compare a particular evolutionary algorithm with any other given approach. The test examples should be large, reflecting real-world VLSI design problems.

7 Summary

We have presented a survey of evolutionary algorithms that have been applied in VLSI physical design. We have also provided guidelines to make evolutionary algorithms more competitive among physical design algorithms for VLSI layout. We believe that such an approach is an important and necessary part of the continued growth of this field.

Acknowledgments

We would like to acknowledge support of the research received from the Virginia Aerospace Consortium, from the National Science Foundation under grants CDA 9634333, CCR 9224789, MIP 9107717, DUE 9554715, and DUE 9653413, from the Department of Computer Science at the University of Virginia, the German Science Foundation (DFG) and from Dresden University of Technology.

References

[Abramovici et al. (1994)] Abramovici, M., Breuer, M. A., Friedman, A. D. (1994) **Digital Systems Testing and Testable Design**, IEEE Press, Piscataway, NJ

[Alpert and Khang (1995)] Alpert, C. J., Khang, A. B. (1995) *Recent Directions in Netlist Partitioning: A Survey*, **Integration: The VLSI Journal**, **19**, 1-81

[Bäck (1996)] Bäck, T. (1996) **Evolutionary Algorithms in Theory and Practice**, Oxford University Press, New York

[Becker and Drechsler (1994)] Becker, B., Drechsler, R. (1994) *OFDD Based Minimization of Fixed Polarity Reed-Muller Expressions Using Hybrid Genetic Algorithms*, **IEEE International Conference on Computer Design**, 106-110

[Bentley (1996)] Bentley, P. (1996) **Evolutionary Design by Computers**, Morgan Kaufmann Publishers, San Francisco, CA

[Betz and Rose (1997)] Betz, V., Rose, J. (1997) *VPR: A New Packing, Placement and Routing Tool for FPGA Research*, **International Workshop on Field Programmable Logic and Applications**, 213-222

[Bui and Moon (1994)] Bui, T. N., Moon, B. R. (1994) *A Fast and Stable Hybrid Genetic Algorithm for the Ratio-Cut Partitioning Problem on Hypergraphs*, **Proc. of the ACM-IEEE Design Automation Conference**, 664-669

[Bushnell and Agrawal (2000)] Bushnell, M. L., Agrawal, V. D. (2000) **Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits**, Kluwer Academic Publishers, Boston, MA

[Chan et al. (1991)] Chan, H., Mazumder, P., Shahookar, K. (1991) *Macro-Cell and Module Placement by Genetic Adaptive Search with Bitmap-Represented Chromosome*, **Integration, The VLSI Journal**, **12**, 49-77

[Cohoon and Paris (1987)] Cohoon, J. P., Paris, W. D. (1987) *Genetic Placement*, **IEEE Trans. on Computer-Aided Design**, **6**, 956-964

[Cohoon et al. (1991)] Cohoon, J. P., Hegde, S. E., Martin, W. N., Richards, D. S. (1991) *Distributed Genetic Algorithms for the Floorplan Design Problem*, **Computer-Aided Design of Integrated Circuits and Systems**, **10**, 483-492

[Cohoon et al. (1991)] Cohoon, J. P., Martin, W. N., Richards, D. S. (1991) Genetic Algorithms and Punctuated Equilibria in VLSI *Parallel Problem Solving from Nature*, H. P. Schwefel and R. Männer, eds., **Lecture Notes in Computer Science**, 496 (Berlin: Springer Verlag), 134-144

[Cong and Ding (1996)] Cong, J., Ding, Y. (1996) *Combinational logic synthesis for LUT Based Field Programmable Gate Arrays*, **ACM Transactions on the Design of Electronic Systems**, **1**, 145-204

[Corno et al. (1996)] Corno, F., Prinetto, P., Rebaudengo, M., Reorda, M. S. (1996) *GATTO: a Genetic Algorithm for Automatic Test Pattern Generation for Large Synchronous Sequential Circuits*, **IEEE Transactions on Computer-Aided Design**, **15**, 943-951

[Davis (1985)] Davis, L. (1985) *Applying Adaptive Algorithms to Epistatic Domains*, **Proc. Int. Joint Conference on Artificial Intelligence**

[Drechsler (1998)] Drechsler, R. (1998) **Evolutionary Algorithms for VLSI CAD**, Kluwer Academic Publishers, Boston, MA

[EDA Benchmarks (1997)] **EDA Benchmarks 1997**, WWW: http://www.cbl.ncsu.edu/ or email: benchmarks@cbl.ncsu.edu

[Eldrege and Gould (1972)] Eldredge, N., Gould, S. L. (1972) *Punctuated equilibria: An Alternative to Phyletic Gradulism*, **Models of Paleobiology**, Freeman, Cooper and Co., San Francisco, CA, 82-115

[Esbensen (1992)] Esbensen, H. (1992) *A Genetic Algorithm for Macro Cell Placement*, **Proc. of the European Design Automation Conference**, 52-57

[Esbensen (1994)] Esbensen, H. (1994) *A Macro-Cell Global Router Based on Two Genetic Algorithms*, **Proc. of the European Design Automation Conference**, 428-433

[Esbensen and Mazumder (1992)] Esbensen, H., Mazumder, P. (1992) *SAGA: A Unification of the Genetic Algorithm with Simulated Annealing and its Application to Macro-Cell Placement*, **Proc. of the 7th International Conference on VLSI Design**, 211-214

[Esbensen and Mazumder (1994)] Esbensen, H., Mazumder, P. (1994) *Genetic Algorithm for Steiner Problems in a Graph*, **European Design and Test Conference**, Paris, 402-406

[Esbensen and Mazumder (1999)] Esbensen, H., Mazumder, P. (1999) *Macro Cell Routing*, **Genetic Algorithms for VLSI Design, Layout and Test Automation**, P. Mazumder and E. M. Rudnick, eds., Prentice-Hall, Upper Saddle River, NJ, 70-106

[Fogel (1995)] Fogel, D. (1995) **Evolutionary Computation: Toward a New Philosophy of Machine Intelligence**, IEEE Press, New York

[Fogel et al. (1966)] Fogel, L. J., Owens, A. J., Walsh, M. J. (1966) **Artificial Intelligence through Simulated Evolutions**, Clearinghouse for Federal Scientific and Technical Information, Springfield, VA

[Fonseca and Fleming (1995)] Fonseca, M., Fleming, P. J. (1995) *An Overview of Evolutionary Algorithms in Multiobjective Optimization*, **Evolutionary Computation**, **3**, 1-16

[Fourman (1985)] Fourman, M. P. (1985) *Compaction of Symbolic Layout using Genetic Algorithms*, **Proc. of the First International Conference on Genetic Algorithms**, 141-153

[Genesis (1987)] Grefenstette, J., Schraudolph, N. (1987) **A User's Guide to GENESIS 1.2 UCSC**, CSE Dept., University of California, San Diego

[Geraci et al. (1991)] Geraci, M., Orlando, P., Sorbello, F., Vasallo, G. (1991) *A Genetic Algorithm for the Routing of VLSI Circuits*, **Euro Asic '91**, Parigi, 27-31 Maggio, 218-223

[Göckel et al. (1997a)] Göckel, N., Pudelko, G., Drechsler, R., Becker, B. (1997) *A Hybrid Genetic Algorithm for the Channel Routing Problem*, **Proc. of the International Symposium on Circuits and Systems**, 675-678

[Göckel et al. (1997b)] Göckel, N., Pudelko, G., Drechsler, R., Becker, B. (1997) *A Multi-Layer Detailed Routing Approach Based on Evolutionary Algorithms*, **Proc. of IEEE International Conference on Evolutionary Computation**, 557-562

[Goldberg (1989)] Goldberg, D. E. (1989) **Genetic Algorithms: Search, Optimization and Machine Learning**, Addison-Wesley, New York

[Goldberg and Lingle (1985)] Goldberg, D. E., Lingle, R. (1985) *Alleles, Loci and the Traveling Salesman Problem*, **Proc. International Conference on Genetic Algorithms**

[Goodman et al. (1994)] Goodman, E., Tetelbaum, A. Y., Kureichik, V. (1994) **A Genetic Algorithm Approach to Compaction, Bin Packing, and Nesting Problems**, GARAGe Technical Report

[Holland (1975)] Holland, J. H. (1975) **Adaptation in Natural and Artificial Systems**, University of Michigan Press, Ann Arbor, MI

[Hsieh et al. (1988)] Hsieh, T. M., Leong, H. W., Liu, C. L. (1988) *Two-Dimensional Layout Compaction by Simulated Annealing*, **Proc. IEEE International Symposium on Circuits and Systems**, Espoo, Finland, **3**, 2439-2443

[Hulin (1991)] Hulin, M. (1991) *Analysis of Schema Distributions*, **Proc. of the Fourth International Conference on Genetic Algorithms**, 204-209

[Hulin (1991a)] Hulin, M. (1991) *Circuit Partitioning with Genetic Algorithms Using a Coding Scheme to Preserve the Structure of a Circuit*, **Parallel Problem Solving from Nature**, H. P. Schwefel and R. Männer, eds., Lecture Notes in Computer Science, 496 (Berlin: Springer Verlag), 75-79

[Hwang et al. (1992)] Hwang, F. K., Winter, P., Richards, D. S. (1992) **The Steiner Tree Problem**, Elsevier Science, Amsterdam.

[IEEE International Testing Conference (1999)] **IEEE International Testing Conference**, Atlantic City, NJ

[Julstrom (1993)] Julstrom, D. A. (1993) *A Genetic Algorithm for the Rectilinear Steiner Problem*, **International Conference on Genetic Algorithms and their Applications**, 231-236

[Kapsalis et al. (1993)] Kapsalis, A., Rayward-Smith, V. J., Smith, G. D. (1993) *Solving the Graphical Steiner Tree Problem Using Genetic Algorithms*, **Journal of the Operational Research Society**, **44**, 397-406

[Karro and Cohoon (1999)] Karro, J., Cohoon, J. (1999) *A Spiffy Tool for the Simultaneous Placement and Global Routing of Three-Dimensional Field Programmable Gate Arrays*, **Ninth Great Lakes Symposium on VLSI**, Ann Arbor, MI, 226-227

[Kirpatrick et al. (1983)] Kirkpatrick, S., Gelatt, C. D., Vecci, M. P. (1983) *Optimization by Simulated Annealing*, **Science**, **220**, 45-54

[Kling and Banerjee (1989)] Kling, R. M., Banerjee, P. (1989) *ESP: Placement by Simulated Evolution*, **IEEE Trans. on Computer-Aided Design 8**, 245-256

[Kling and Banerjee (1990)] Kling, R. M., Banerjee, P. (1990) *Optimization by Simulated Evolution with Applications to Standard Cell Placement*, **Proc. of the 27th ACM-IEEE Design Automation Conference**, 20-25

[Lienig (1996)] Lienig, J. (1996) **Evolutionary Algorithms Applied to VLSI Physical Design**, (in German), Fortschrittberichte VDI, Reihe, 20, VDI-Verlag, Duesseldorf

[Lienig (1997)] Lienig, J. (1997) *A Parallel Genetic Algorithm for Performance-Driven VLSI Routing*, **IEEE Trans. on Evolutionary Computation**, **1**, 29-39

[Lienig and Thulasiraman (1994)] Lienig, J., Thulasiraman, K. (1994) *A Genetic Algorithm for Channel Routing in VLSI Circuits*, **Evolutionary Computation**, **1**, 293-311

[Lienig and Thulasiraman (1996)] Lienig, J., Thulasiraman, K. (1996) *GASBOR: A Genetic Algorithm for Switchbox Routing in Integrated Circuits*, **Journal of Circuits, Systems, and Computers**, **6**, 359-373

[Lin et al. (1989)] Lin, Y.-L., Hsu, Y.-C., Tsai, F.-S. (1989) *SILK: A Simulated Evolution Router*, **IEEE Trans. on Computer-Aided Design**, **8**, 1108-1114

[Mazumder and Rudnick (1999)] Mazumder, P., Rudnick, E. (1999) **Genetic Algorithms for VLSI Design, Layout & Test Automation**, Prentice Hall, Upper Saddle River, NJ

[Mazumder and Shahookar (1999)] Mazumder, P., Shahookar, K. (1999) *Partitioning* in **Genetic Algorithms for VLSI Design, Layout and Test Automation**, P. Mazumder and E. M. Rudnick, eds., Prentice Hall, Upper Saddle River, NJ, 38-68

[Mitchell (1996)] Mitchell, M. (1996) **An Introduction to Genetic Algorithms**, The MIT Press, Cambridge, MA

[Mohan and Mazumder (1993)] Mohan, S., Mazumder, P. (1993) *Wolverines: Standard Cell Placement on a Network of Workstations*, **IEEE Trans. on Computer-Aided Design**, **12**, 1312-1326

[Ohmori and Kasai (1997)] Ohmori, K., Kasai, T. (1997) *Logic Synthesis using a Genetic Algorithm*, **Internal Symposium on IC Technologies, Systems, and Applications**, 200-203

[Oliver et al. (1985)] Oliver, I. M., Smith, D. J., Holland, J. R. C. (1985) *A Study of Permutation Crossover Operators on the Traveling Salesman Problem*, **Proc. International Conference on Genetic Algorithms**, 224-230

[Ökmen et al. (1997)] Ökmen, C., Keirn, M., Krieger, R., Becker, B. (1997) *On Optimizing BIST-Architecture by Using OBDD-Based Approaches and Genetic Algorithms*, **IEEE VLSI Test Symposium**, 426-431

[Preas and Lorenzetti (1988)] Preas, B. T., Lorenzetti, M. J. (1988) **Physical Design Automation of VLSI Systems**, Benamin/Cumming Publishing, Menlo Park, CA

[Rahmani and Ono (1993)] Rahmani, A. T., Ono, N. (1993)*A Genetic Algorithm for Channel Routing Problem*, **Proc. of the Fifth International Conference on Genetic Algorithms**, 494-498

[Rao et al. (1994)] Rao, B. B. P., Patnaik, L. M., Hansdah, R. C. (1994) *A Genetic Algorithm for Channel Routing Using Inter-Cluster Mutation*, **Proc. of the First IEEE International Conference on Evolutionary Computation**, 97-103

[Rao et al. (1995)] Rao, B. B. P., Patnaik, L. M., Hansdah, R. C. (1995) *An Extended Evolutionary Programming Algorithm for VLSI Channel Routing*, **Evolutionary Programming IV: Proc. of the 4th Annual Conference on Evolutionary Programming**, J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, eds., MIT Press, Cambridge, MA, 521-544

[Rebaudengo and Reorda (1996)] Rebaudengo, M., Reorda, M. S. (1996) *GALLO: A Genetic Algorithm for Floorplan Optimization*, **IEEE Trans. on Computer-Aided Design**, **15**, 943-951.

[Rechenberg (1973)] Rechenberg, I. (1973) **Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution**, Frommann-Holzberg, Stuttgart -Bad Cannstatt

[Rudnick et al. (1997)] Rudnick, E. M., Patel, J. H., Greenstein, G. S., Niermann, T. M. (1997) *A Genetic Algorithm Framework for Test Generation*, **IEEE Trans. on Computer-Aided Design of Integrated Systems**, **16**, 1034-1044

[Saab and Rao (1989)] Saab, Y., Rao, V. (1989) *An Evolution-Based Approach to Partitioning ASIC Systems*, **Proc. of the ACM-IEEE Design Automation Conference**, 767-770

[Saab et al. (1992)] Saab, D. G., Saab, G. Y., Abraham, J. A. (1992) *CRIS: a Test Cultivation Program for Sequential VLSI Circuits*, **IEEE/ACM International Conference on Computer-Aided Design**, 216-219

[Sarrafzadeh and Wong (1996)] Sarrafzadeh, M., Wong, C. (1996) **An Introduction to VLSI Physical Design**, McGraw Hill, New York

[Schaffer (1984] Schaffer, J. D. (1984) **Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms**, Unpublished doctoral dissertation, Vanderbilt University, Nashville, TN

[Schnecke and Vornberger (1996)] Schnecke, V., Vornberger, O. (1996) *An Adaptive Parallel Genetic Algorithm for VLSI-Layout Optimization*, **Parallel Problem Solving from Nature**, H. P. Schwefel and R. Männer, eds., Lecture Notes in Computer Science, 1141 (Berlin: Springer Verlag), 859-868

[Sechen (1988)] Sechen, C. (1988) **VLSI Placement and Global Routing Using Simulated Annealing**, Kluwer Academic Publishers, Boston, MA

[Shahookar et al. (1993)] Shahookar, K., Khamisani, W., Mazumder, P., Reddy, S. M. (1993) *Genetic Beam Search for Gate Matrix Layout.* **Proc. of the 6th International Conference on VLSI Design** , 208-213

[Shahookar and Mazumder (1990a)] Shahookar, K., Mazumder, P. (1990) *GASP - A Genetic Algorithm for Standard Cell Placement*, **Proc. of the European Design Automation Conference**, 660-664

[Shahookar and Mazumder (1990b)] Shahookar, K., Mazumder, P. (1990) *A Genetic Approach to Standard Cell Placement using Meta-Genetic Parameter Optimization* **IEEE Trans. on Computer-Aided Design**, **9**, 500-511

[Sherwani (1999)] Sherwani, N. (1999) **Algorithms for VLSI Physical Design Automation, Third Edition**, Kluwer Academic Publishers, Boston, MA

[Srinivas and Patnaik (1993)] Srinivas, M., Patnaik, L. M. (1993) *A Simulation-Based Test Generation Scheme Using Genetic Algorithms*, **International Conference on VLSI Design**, 132-135

[Thomson and Miller (1997)] Thomson, P., Miller, J. F. (1997) *Comparison of AND-XOR Logic Synthesis Using a Genetic Algorithm Against MISII for Implementation on FPGAs*, **IEE and IEEE International Conference on GA Applications in Engineering**

[Varanelli and Cohoon (1995)] Varanelli, J. M., Cohoon, J. P. (1995) *Population-Oriented Simulated Annealing: A Genetic/Thermodynamic Hybrid Approach to Optimization*, **Proc. of the Sixth International Conference on Genetic Algorithms**, 174-181

[Vemuri and Vemuri (1991)] Vemuri, R., Vemuri, R. (1991) *Genetic Synthesis: Performance-Driven Logic Synthesis Using Genetic Evolution*, **IEEE Great Lakes Symposium on VLSI Systems**, Kalamazoo, MI, 312-317

[Villa et al. (1997)] Villa, T., Kam, T., Brayton, R. K., Sangiovanni-Vincentelli, A. (1997) **Synthesis of Finite State Machines: Logic Optimization**, Kluwer Academic Publishers, Boston, MA

[Yu (1989)] Yu, M. L. (1989) *A Study of the Applicability of Hopfield Decision Neural Nets to VLSI CAD*, **Proc. of the 26th ACM-IEEE Design Automation Conference**, 412-417

[Zhang and Mlynski (1990)] Zhang, C. X., Mlynski, D. A. (1990) *VLSI Placement With a Neural Network Model*, **Proc. of the International Symposium on Circuits and Systems**, 475-478