Hybrid Genetic Algorithms for Constrained Placement Problems

Volker Schnecke and Oliver Vornberger

Abstract— **Genetic algorithms have proven to be a well-suited technique for solving selected combinatorial optimization problems. When solving real-world problems, often the main task is to find a proper representation for the candidate solutions. Strings of elementary data types with standard genetic operators may tend to create infeasible individuals during the search because of the discrete and often constrained search space. This article introduces a generally applicable representation for two-dimensional combinatorial placement and packing problems. Empirical results are presented for two constrained placement problems, the facility layout problem and the generation of VLSI macro-cell layouts. For multi-objective optimization problems, common approaches often deal with the different objectives in different phases, thus are unable to efficiently solve the global problem. Due to a tree-structured genotype representation and hybrid, problem-specific operators, the proposed approach is able to deal with different constraints and objectives in one optimization step.**

Keywords— **VLSI physical design, facility layout problem, combinatorial optimization, multiparent recombination, tree-structured genotype representation**

I. Introduction

Constrained placement problems deal with the computation of an optimal arrangement of items on a planar site. The objective function for these optimization problems is based on the overall rectangular area of occupied space and on additional terms that reflect problem-specific constraints. The basic variants of these problems are the unconstrained two-dimensional packing problem and the quadratic assignment problem. In the case of the packing problem, a set of rectangular blocks has to be arranged such that no blocks overlap each other. The area (or perimeter) of the rectangle circumscribing all blocks has to be minimal, hence the optimal packing pattern is that with minimal waste inside the enveloping rectangle. In the quadratic assignment problem [1], a set of items has to be matched to fixed arranged bins. A flow matrix defines the connectivity between the items. The objective is to find a mapping with minimal flow costs, these being the sum of the products of flow and distance between each pair of items. The quadratic assignment problem is an NP-hard optimization problem [2].

The constrained placement problems that will be described in the following can be seen as an extension both of the packing problem and the quadratic assignment problem. Rectangular blocks are placed with some defined connectivity, which highly influences the objective function. The rectangular blocks do not have fixed shapes. For each block, the area is given and either a set of admissible shapes or an interval for its admissible aspect ratio (width/height ratio) is provided. Thus, in addition to finding an arrangement of the blocks, feasible shapes are also determined. Because of the complexity, exact techniques can only be used to solve trivial instances of these optimization problems (fewer than ten blocks).

Multi-objective optimization problems, like those described in this article, are found in many engineering applications. To deal with the different, often conflicting, objectives, Pareto-optimal [3] or divide-and-conquer approaches [4] are typically used. In the latter, the problem is divided into subproblems, which are solved more or less independently. Multistage approaches for constrained placement problems first compute an approximately optimal arrangement of the blocks based on their connectivity, and then fix their shapes in a second stage. In this article, hybrid genetic algorithms are introduced to deal with both the arrangement and sizing of the blocks in one optimization step. To deal with the shape constraints, *shape functions* [5], a common concept from VLSI design, are used to consider multiple shapes for the blocks during their placement. The connectivity between the blocks is taken into account by iteratively pairing blocks based on the computation of a weighted matching [6]. A novel genotype representation based on binary trees is introduced, and the genetic operators work directly on this tree structure. Mul-

This work has been supported by the German Federal Ministry for Education, Science, Research and Technology (BMBF) in the project 'HYBRID–Application of Parallel Genetic Algorithms in Combinatorial Optimization' under grant 01 IB 405 E3.

V. Schnecke is with the Department of Biochemistry, Michigan State University, East Lansing, MI 48824-1319, USA (e-mail: volker@sol.bch.msu.edu).

O. Vornberger is with the Department of Mathematics/Computer Science, University of Osnabrück, D-49069 Osnabrück, Germany (e-mail: oliver@informatik.uni-osnabrueck.de).

tiparent *gene-pool recombination* is proposed, where subtrees from more than two parent individuals are composed into a tree representing an offspring.

II. Background

The two placement problems addressed in this article are the *facility layout problem* and *VLSI macro-cell layout generation*. In this section, these combinatorial optimization problems are introduced, and related work, with a focus on evolution-based approaches, is described. The facility layout problem, as defined here, is a basic variant of placement problems, whereas the macro-cell layout generation is a real-world optimization problem. At the end of this article, empirical results for benchmark problems for both applications are presented.

A. The facility layout problem

The *facility layout problem* deals with a set of rectangular facilities to be placed at favorable positions on a planar site. A *flow matrix* $M = [m_{ij}]_{i,j=1,\ldots,n}$ defines the weights of connectivity for each pair of facilities. The objective is to find a nonoverlapping arrangement of the facilities with minimal *flow costs* $\sum_{i,j=1}^{n} d_{ij} \cdot m_{ij}$, with d_{ij} being the distance between the centroids of facilities f_i and f_j (Figure 1). The variant of the facility layout problem addressed in this article involves nonidentical, flexible facilities. Here, no fixed dimensions for the facilities are given; only constraints regarding their admissible shapes are provided. These additional degrees of freedom increase the complexity of the optimization problem, since in addition to identifying a proper arrangement of the facilities, their shapes have to be fixed. This can be seen as two separate optimization tasks. Different variants of the facility layout problem have been introduced [7], [8], for instance with an irregular site for placing the facilities, with preoccupied regions on the layout site, or with more than one layout site. Applications of the facility layout problem include planning architectural spaces, such as offices and warehouses, and designing manufacturing cells and control panels.

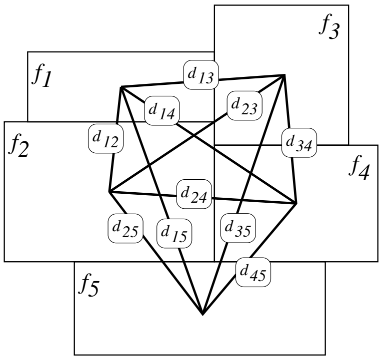


Fig. 1. The schematic representation of a solution to the facility layout problem including five rectangular facilities f_1 to f_5. The objective is to minimize the flow costs based on the sum of products of flow m_{ij} and distance d_{ij} between the centroids for each pair of facilities.

There exist various, in most cases heuristic, solution approaches to the facility layout problem [8]. In multistage techniques, the rectangular facilities are first approximated by circles with a slightly larger area than the area of the actual facilities. An optimal arrangement according to the flow between the centroids of the circles is computed, and the exact shapes of the facilities are determined in a second step after fixing their positions. Examples of this approach are presented by van Camp *et al.* [9] and Tam and Li [10]. Other approaches are based on tree representation of the facility arrangement. Tam uses a genetic algorithm [11] or simulated annealing [12]. In both approaches, all facilities are initially clustered according to their connectivity. These clusters characterize subtrees in a tree containing all facilities. The tree defines a floorplan, i.e., a partitioning of the overall area into separate rooms to which the facilities are assigned. Each inner node of the tree is labeled with the arrangement of the patterns represented by its subtrees (placing the pattern characterized by the left subtree upon, below, left, or right, relative to the other one). The structure of the tree and the mapping of facilities to its leaves is fixed during the first step, then an optimal labeling of all inner nodes is computed by using a genetic algorithm or simulated annealing. Kado *et al.* [13] compare different implementations of genetic algorithms based on Tam's representation, hybridized with clustering methods. They extend Tam's work by searching the space of all possible trees in contrast to the search for optimal labels for the inner nodes in fixed tree structures. Garces-Perez *et al.* [14] use a tree representation without clustering when solving the facility layout problem by genetic programming (GP). However, due to the necessary restriction to fixed-size trees with a predefined set of leaf labels, their approach does not precisely fit the GP paradigm.

B. VLSI macro-cell layout generation

The design of VLSI *(very large scale integrated)* microchips is a process of many consecutive steps including specification, functional design, circuit design, physical design, and fabrication [15]. Macro-cell layout generation is a task in the *physical design cycle*. The circuit is partitioned and the components are grouped in functional units, the *macro-cells*. These cells can be described as rectangular blocks with *terminals* (pins) along their borders. These terminals have to be connected by *signal nets*, along which power or signals (e. g., clock ticks) are transmitted between the various units of the chip. A net can connect two or more terminals, and some nets must be routed to *pads* at the outer border of the layout, since they are involved in the I/O of the chip. The layout defines the positions of the cells and the routes chosen for the signal nets (Figure 2).

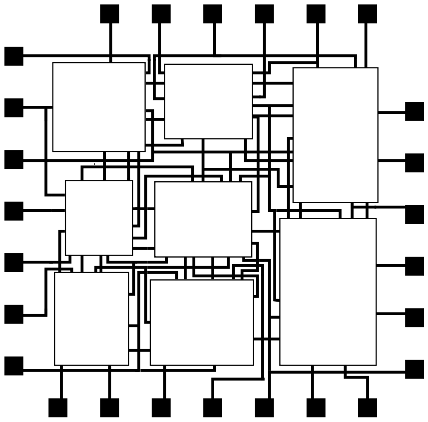


Fig. 2. The schematic representation of a VLSI macro-cell layout, which shows the position of eight cells, the routes for the signal nets, and the I/O pads.

Basically, the placement of macro-cells is quite similar to the facility layout problem, when taking the number of terminals to be connected between two cells as a measure of their connectivity. However, the objective function of this optimization problem is based on the layout area, i.e., the area of the circumscribing rectangle. This area is mainly influenced by the routing space, the area between the cells occupied by the signal net wirings. The computation of the routes for the signal nets is usually separated from the placement task. During placement, an estimated amount of routing space is added between the cells. The estimation of this amount is quite crucial, since adding too much space can lead to sub-optimal layouts. Adding too little space might rule out the optimal (shortest) routes for all nets, or the completion of the interconnections can become impossible. In the latter case, a rearrangement of the cells is necessary. Therefore it is wise to integrate the computation of the routes into the placement task.

In the common, multistage approaches to layout generation, there exist several methods to solve the placement problem, which is the first step in this process [16], [15]. In force-directed placement, cells that are connected by signal nets exert an attractive force on each other, which is proportional to the number of these nets and the distance between the cells. Partition-based methods compute a placement by recursively dividing the set of cells. At the same time, the available chip area is partitioned, and each set of cells is assigned to one of the components. A very popular technique to compute the placement is simulated annealing, which yields high-quality placements, but often requires an excessive amount of computation time.

Because a macro-cell is constructed in a hierarchical manner, its shape is not fixed, but a set of feasible implementations is provided. Thus, like in the facility layout problem, the flexible cells must be sized during or after placement. This more general placement problem is called the *floorplanning problem* in VLSI design [17], [18]. It is usually solved in two steps. First, a relative placement is chosen. This is characterized by a *floorplan* that describes a partitioning of the layout area into a set of rooms to which the cells are mapped (see left side in Figure 3). Then, those shapes of the flexible cells that yield an overall minimal layout are determined. Sizing *(floorplan area optimization)* can be done by using rectangular dualization, partitioning, or linear programming [17], [18].

Cohoon *et al.* [19], [20] offered the classical work on using genetic algorithms for floorplanning. The arrangement of the rooms on the layout surface was represented in the genotype by a postfix notation of the corresponding slicing tree (see Figure 3 for an example of a slicing tree). Chan *et al.* [21] introduce a bit-matrix representation for placement. Here the layout area is divided into squares, and the placement for a single cell is described by a line in the bit-matrix. This encodes information about the occupied squares and the orientation of the cell. In the genetic algorithms of Esbensen [22] or Esbensen and Mazumder [23] a placement is encoded as a binary tree. Each node of the tree represents a cell, and due to a given node order, the placement can be sequentially generated by decoding the genotype.

After either placement or floorplanning, routing of the signal nets is performed. The area of the layout surface that is not occupied by cells is subdivided into routing regions, a region being the empty area between two adjacent cells. During *global routing*, the global routes for all nets are determined. The global route of a net is a collection of those routing regions it covers on its way between the terminals it must connect. The routing regions are typically represented by a graph, and the global routes are computed by finding shortest paths between the terminals (Figure 4). During computation of the global routing, capacity constraints based on the width of the routing regions have to be considered. Thus, the amount of routing area reserved during placement is crucial for the routability of the whole layout, and sophisticated placement approaches try to incorporate as much as possible from the routing into the placement process.

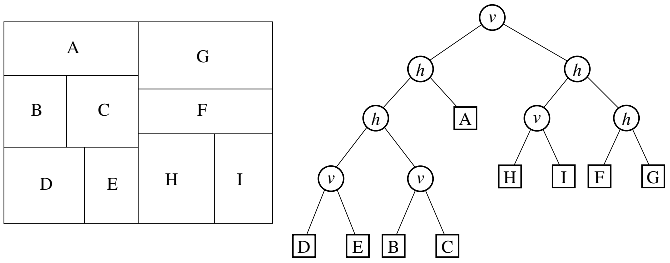


Fig. 3. A pattern describing a floorplan for nine blocks (left) and one corresponding slicing tree (right); the leaves represent the blocks and the inner nodes define the cut direction (v for vertical, h for horizontal) used for recursively partitioning the layout area.

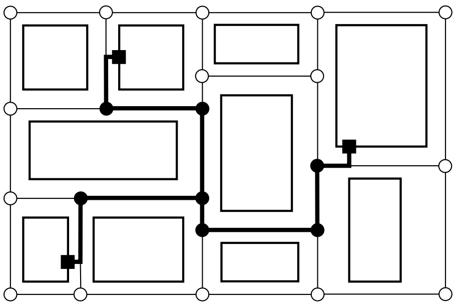


Fig. 4. A routing graph for a layout with ten blocks. The global route for a signal net connecting three terminals is shown by the bold path, which was constructed by using shortest paths in the graph to connect the terminals.

III. Methods

All previous approaches treat the different objectives (placement, sizing, and connectivity) of the optimization problems in separate steps. A single objective is considered in each step, while approximations are used for the others to keep the computation tractable. In contrast, our hybrid genetic algorithms consider the different objectives in a single optimization process. This is possible due to the use of problem-specific heuristics such as slicing trees, shape functions, and iterated matching, which are introduced in this section.

A. *Representing placement patterns by binary slicing trees*

Two-dimensional packing or placement patterns can be characterized by *slicing trees*. A slicing tree defines a hierarchy of cuts needed for recursively partitioning a rectangular block into patterns consisting of smaller blocks. The simplest kind of slicing trees are binary slicing trees, which represent *guillotineable* or *slicing* patterns. According to Stockmeyer [24], a pattern is *slicing*, if it is either a basic block, i.e., an indivisible item, or if there is a line segment (a slice) that divides the enclosing rectangle into two pieces such that each of the pieces is slicing. Figure 3 presents a slicing pattern and a corresponding slicing tree. The inner nodes of the tree are labeled with the cut directions (vertical, horizontal), and the leaves characterize the basic blocks. There is also a bottom-up interpretation of slicing trees. In this case, the label of an inner node defines the relative arrangement (side by side, or one upon the other) of the patterns represented by its subtrees.

B. *Storing different implementations in shape functions*

Shape functions have been introduced by Otten [5] for VLSI layout generation dealing with flexible cells. The flexibility originates from the fact that a cell hierarchically comprises a set of subcells, which can be arranged in different ways. Figure 5 presents a shape function for a cell containing four subcells. Three different arrangements of these cells yield three minimal-area implementations for the macro-cell. These can be represented in a shape function, which defines the admissible shapes, i.e., the relation between area and aspect ratio. Note that a discrete shape function is completely defined by its minimal area implementations.

In the facility layout problem there are no discrete minimal implementations given. Only the area of a facility and a range for the aspect ratio are specified. This information defines a continuous shape function, which can be transformed into a discrete shape function, as shown in Figure 6. Although some information is lost by this transformation, it keeps the computations during the optimization process tractable.

When combining two flexible blocks, their shape functions can be added to compute the shape function of the also flexible *meta-block* (pattern consisting of more than

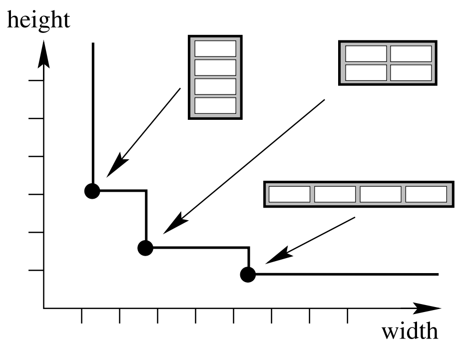


Fig. 5. The shape function for a macro-cell comprising four subcells. The three possible subcell arrangements that yield minimal-area implementations of the macro-cell are shown.

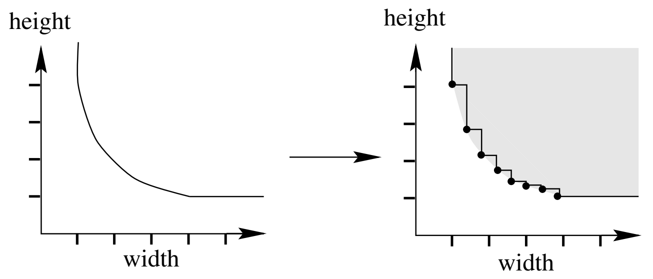


Fig. 6. A transformation of a continuous shape function into a discrete shape function with eight minimal implementations.

one block). If the orientations of the blocks are free, they can be rotated by 90° before being placed on the layout surface. Both possibilities can be considered in the shape function of the resulting meta-block by labeling the minimal implementations with the particular orientations. Figure 7 shows the composition of such a generalized shape function for a meta-block consisting of two fixed-shape blocks with free orientations. The relative arrangement of the blocks is fixed, since they are placed upon each other in all cases. Note that there is one implementation which is dominated by another one: Combination (h, v) with the lower block in horizontal (h) orientation and the upper block in vertical (v) orientation is covered by combination (h, h), since both have the same width, but the latter is less high. In practice, most implementations are dominated, and it is possible to store all nondominated implementations without observing an exponential growth in the number of combined implementations at each level in

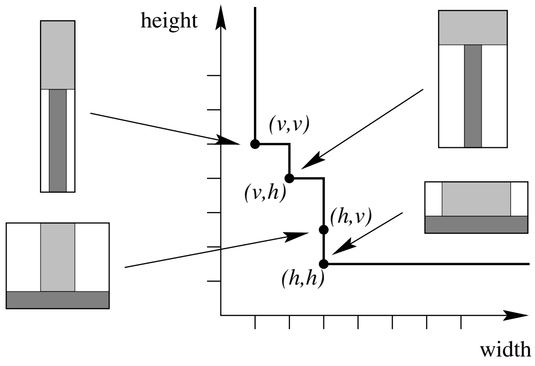


Fig. 7. The generalized shape function for two fixed blocks placed upon each other; each combination is labeled with the orientations of the blocks, for instance, (h, v) means the lower block is in horizontal and the upper block is in vertical orientation

the tree. Empirical results for this will be presented in a later section of this article.

C. Iterative clustering based on connectivity

While slicing trees and shape functions deal with the packing aspect of the optimization problems, another heuristic is necessary to take into account those constraints based on connectivity. When constructing a solution for these problems, care must be taken that highly connected blocks are placed near each other on the layout site. During the composition of a slicing layout, blocks (or meta-blocks) are iteratively paired. This corresponds to composing an inner node of the slicing tree by joining two leaves (or subtrees).

At the beginning of the construction, globally good pairings are identified in the set of all blocks. These build the lowest level of the slicing tree. A good technique for successively pairing items according to a quality function is the *iterated matching* heuristic, which was introduced by Fritsch and Vornberger [6]. It is based on the graph-algorithmic computation of a *maximum weight matching* on a complete graph. The vertices of this graph represent the items to be paired, and each edge is weighted with the value of the quality function for the corresponding pairing. A matching in this graph is a set of node-disjunct edges, and the weight of a matching is the sum of the weights of all edges in this set.

In the case of the facility layout problem, an edge is weighted according to the flow between the facilities represented by the adjacent vertices. For the VLSI layout generation problem, the quality function is based on the number of terminals that have to be connected by signal nets between both cells. A maximum weight matching corresponds to a set of optimal pairings such that a globally maximal number of terminals can be connected inside the resulting meta-blocks. Since paired blocks are adjacent on the final layout, the maximum weight matching ensures short wiring lengths in the case of macro-cell layout generation and low partial flow-cost terms for the facility layout problem.

In the second iteration, the next level of the slicing tree is constructed by computing a maximum weight matching on a graph whose vertices represent meta-blocks, each consisting of two blocks. The quality function is based on the sum of the flow between both sets of facilities contained inside the corresponding meta-blocks for the facility layout problem and on the number of terminals that have to be connected between the cells in both sets for the macro-cell layout generation. The process is iterated until the slicing tree is completed by joining the last two meta-blocks at the root (see Figure 8). If the matching at one level is not

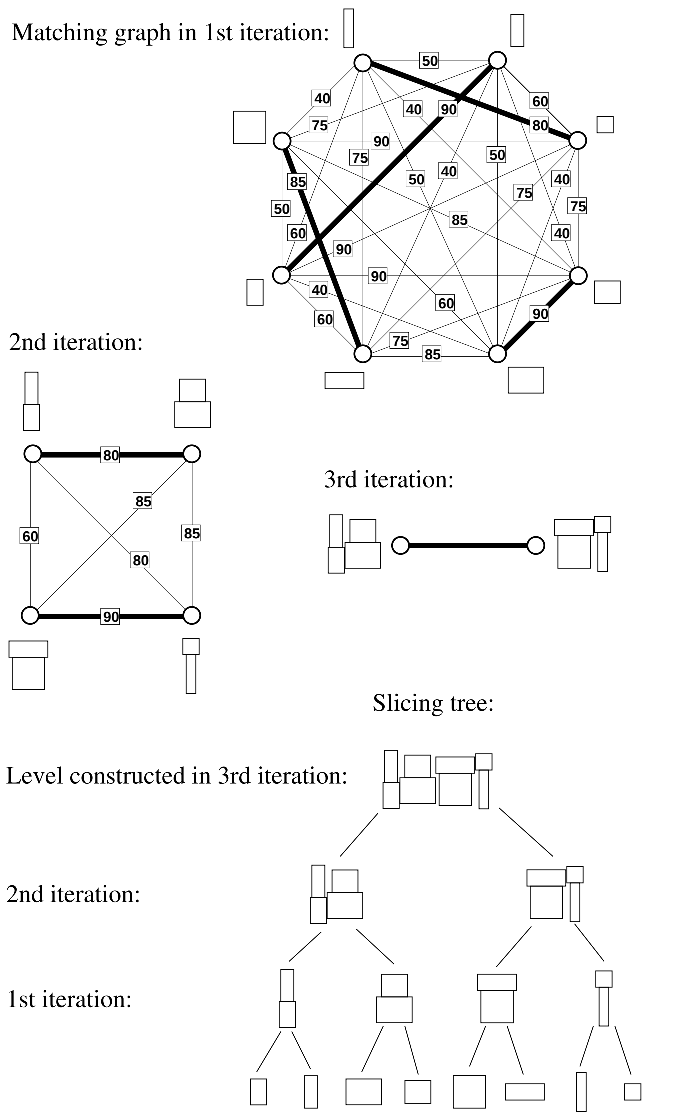


Fig. 8. The construction of a slicing tree using the iterated matching heuristic, e.g., in the second iteration, when the second inner tree level is constructed, the meta-blocks adjacent to the bold edges are combined because the weight of the corresponding matching (80+90) is larger than those of the other two possible matchings (60+85 and 80+85) in that graph.

perfect, that is, not all vertices are adjacent to edges of the matching set, the corresponding blocks or meta-blocks are kept and added to the set of meta-blocks to be matched in the next iteration.

IV. Genetic algorithms with tree-structured genotype representation

This section describes the main features of our hybrid genetic algorithms for the two constrained placement problems. The implementations of these algorithms employ a slicing-tree representation and include shape functions and iterated matching to address multiple design objectives simultaneously. At the end of this section, the hybrid genetic algorithms for the two applications are outlined.

A. *The genotype*

The phenotypic representation for the placement problems is basically the pattern that describes the geometric layout, i.e., shapes and positions of the blocks. In the case of VLSI layout generation, if the computation of the global routes for the signal nets is integrated into placement, characterization of the routes is also part of the phenotype. Binary slicing trees are well suited to represent packing or placement patterns and have already been used in genetic algorithms for two-dimensional problems [19], [20], [14], [13], [25], [11]. The genotype encoding in these approaches is a post- or prefix string defining the structure of the tree and its node labels. During recombination, partial arrangements of blocks are transmitted from parents to offspring. The corresponding operation is the inheritance of subtrees from the parents. Encoding the tree in a string complicates this operation, since the string needs to be decoded into the slicing tree to execute the recombination, then recoded into an offspring chromosome afterwards. There is no reason for using a string encoding except for the analogy to the natural evolution process, where the genetic information is encoded in a DNA string. When directly using the slicing tree as the genotype representation, further decoding or encoding the tree when applying genetic operators is avoided.

The use of trees for genotype coding is already well known from GP [26]. However, in GP the size of the individuals in a population is greatly varied, and usually no restrictions exist regarding the structure of the trees. Trees representing layouts differ from those used in GP in one main point: all trees must have a fixed size, because they have exactly the same set of leaves (objects to place). Therefore, the application of the genetic operators is more complicated than in GP. Problem-specific operators must be used to ensure that only correct offspring are generated.

In the following examples, as for the remainder of this article, blocks or subpatterns in a tree defining a layout or packing pattern are always stacked vertically upon each other. The pattern characterized by the right successor of an inner tree node is always positioned on top of the pattern characterized by its left successor when combining both parts into a pattern or meta-block. Placing these parts next to each other is considered in the next level of the tree by taking into account the rotated variant of this meta-block.

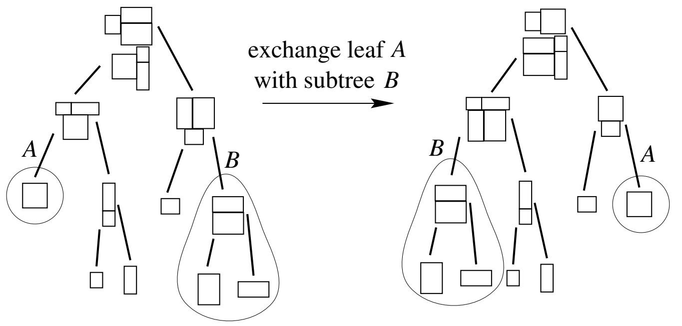


Fig. 9. Mutation by exchanging subtrees. Here a single leaf A is exchanged with a three-node subtree B.

B. Mutation

In our approach, three different mutation operators tailored to handle tree-structured genotypes are used. There are two straightforward mutation operators that change the structure of a tree. Figure 9 shows an example for the operator which exchanges two parts of a tree. Part A is a single leaf, while B is a subtree containing three nodes. At the phenotypic level, this corresponds to exchanging the block represented by leaf A with the placement or pattern for the set of blocks characterized by subtree B.

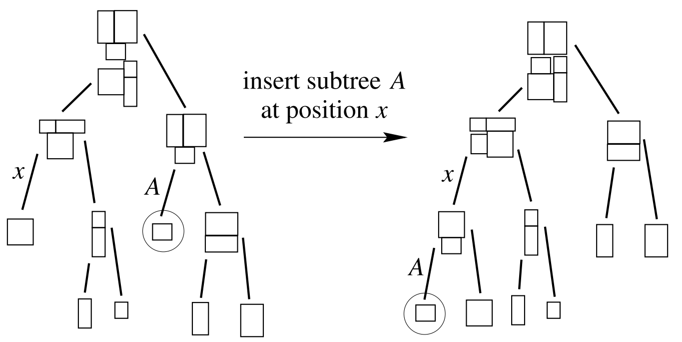


Fig. 10. Mutation by changing the structure of the tree. A single leaf A is cut and inserted at a different position x in the tree.

The second mutation operator (Figure 10) extracts a part of the tree (leaf A) and inserts it at a different position (x). This corresponds to cutting a block or a partial placement out of the complete packing pattern and moving it to a different position. Note that this transformation cannot be performed by the operators which simply exchange leaves or subtrees, so this is an essential mutation operator for a genetic algorithm using tree-structured genotype representation.

A third mutation operator does not directly change the structure of the tree. It converts the pattern described by the slicing tree by changing the orientation of one of the blocks or meta-blocks inside a subpattern characterized by an inner node of the tree. This operator is only necessary if no generalized shape functions are used to encode all possible orientations for the blocks and meta-blocks contained in a particular pattern.

C. Gene-pool recombination

The recombination operator in genetic algorithms is usually a sexual operator, which constructs one or two offspring out of the genetic information encoded in two parent individuals. One obvious way for the creation of one offspring out of two tree-structured individuals is to combine disjoint subtrees of both parents into a tree for the offspring. Such a straightforward recombination operator can be implemented by selecting subtrees from both parents to form a pool of building blocks out of which a new individual is composed. Leaves that are not contained in these subtrees are additionally inserted into this pool. During offspring creation, these subtrees are combined to a complete tree. Here, iterated matching can be used again to identify good pairs of subtrees with regard to their connectivity.

Two-parent or sexual recombination is inspired by the natural evolution process and is the main form of recombination used in artificial evolution. However, multiparent recombination operators have been introduced too. Bersini and Seront [27] used three parents, similar to Mühlenbein's majority vote [28], and Eiben *et al.* [29] use up to ten parents. In the field of evolution strategies, multiparent recombination has also been introduced as 'global recombination' [30], [31]. The extended recombination scheme used in our hybrid genetic algorithms is called *gene-pool recombination*, in reference to the concept established by Mühlenbein and Voigt [32].

In gene-pool recombination, all genetic information from a given set of parents is inserted into the pool, out of which the offspring are created. In our approach, the genetic information encoded in an individual defines the relative positions of the blocks and is represented in the structure the corresponding binary tree. Due to the hierarchical organization of the tree, parts of this information are encoded in each of its subtrees. Thus, every subtree of each parent individual is inserted into the pool. (In the actual implementation the pool does not contain copies of all subtrees, but only pointers to the particular inner nodes of the parent trees). Note that a small subtree occurs more than once in this pool, since it is contained in a set of larger subtrees. Because offspring creation is done by randomly choosing disjoint subtrees out of the pool, this small subtree has a higher chance of being selected. If a small subtree is already included in more than one of the parent individuals, there will be even more copies of it in the pool. This ideally corresponds to the *building block hypothesis* [33]. These building blocks are short, low-order, and highly-fit *schemata* that are sampled, recombined, and resampled during the search. Like those building blocks, smaller trees have a higher chance of being contained in one or more of the offspring that are created out of the genetic material in the pool.

D. Selection and replacement

For choosing individuals as parents for recombination, the fitter individuals have a higher chance of being selected. For mutation, any individual in the population has an equal probability of being chosen; the operator used to generate the offspring is selected randomly from the set of possible mutation operators. An offspring is created either by recombination or mutation. A *steady-state* genetic algorithm is used; thus, an individual may survive for longer than one generation. At the end of each generation, individuals are replaced if the quality of offspring is higher, or if they are quite different from all the members of the current population. The benefit of specialized replacement schemes to maintain diversity in the population has already been investigated by De Jong [34], Goldberg and Richardson [35], and Whitley [36]. More recently, Freisleben and Merz used such a technique for solving the traveling salesman problem [37] and the quadratic assignment problem [38]. In their approaches, the number of noncommon edges in tours or the number of items assigned to different bins, respectively, is taken as a difference measure for two individuals.

The difference between two individuals representing placement patterns is computed at the genotypic level and measured by the number of subtrees that do not contain the same set of leaves. Although this measure does not take into consideration the structure of trees nor the orientations of blocks, it is efficient to compute and serves as a rough measure of diversity.

An offspring is always inserted into the population if its fitness is better than the fitness of the best individual. In this case, the individual with worst fitness is replaced. If an offspring is not better than the best individual, then the individual in the current population that is most similar to this offspring is identified. If their difference is below a given threshold, indicating that they encode similar solutions, then the better of the two is kept in the population. If the offspring is disparate from all individuals, it is considered to encode a significant amount of new genetic information and enters the population by replacing the least fit individual, without consideration of its fitness.

1. Bottom-up construction of the tree:
– Pair blocks based on maximum weight matching
– Consider all possible shapes and orientations in generalized shape functions
2. For all implementations in root node:
– Compute in top-down traversal the orientations and shapes for all facilities
– Transform tree into a geometrical layout, determine positions of the centroids of the facilities, and compute flow costs for this implementation
3. Take minimal flow costs as fitness value

Fig. 11. The operations needed for construction of a single individual in the hybrid genetic algorithm for the facility layout problem. After mutation or recombination step 1 is only executed for all levels in the tree higher than the level where a change occurred.

E. The hybrid genetic algorithm for the facility layout problem

Figure 11 describes the operations which are executed during the computation of a single individual in the hybrid genetic algorithm for the facility layout problem. The slicing tree of an individual is constructed from the bottom-up. All facilities are paired by using iterated matching based on the flow between two facilities or the sum of the flows between facilities comprised in a meta-block, as described in section III-C. Generalized shape functions are used to store all possible implementations for a meta-block based on different orientations of the combined blocks. Thus, a single individual encodes several layouts with different shapes, represented by different implementations stored in the root of the tree. The orientations and shapes of meta-blocks and facilities for each implementation are determined by top-down traversal, and the flow costs for the particular layout are computed. The minimal flow costs of all stored implementations are taken as the fitness of the individual.

Since all possible orientations for the blocks are considered by the generalized shape functions, only the two mutation operators that exchange nodes or change the tree structure are used. After application of a genetic operator, the shape functions are recomputed only for those inner nodes located at higher tree levels than those nodes where a mutation occurred.

F. The hybrid genetic algorithm for macro-cell layout generation

In the hybrid genetic algorithm for the layout generation problem, standard shape functions are used. When two

1. Bottom-up construction of the tree:
– Pair cells based on number of common nets
– Fix orientations for cells
– Compute shape functions for flexible meta-blocks
– Add estimated routing space
2. Choose layout with minimal area of circumscribing rectangle
3. Top-down traversal for sizing flexible cells
4. Compute routing:
– Transform tree into routing graph
– Compute global routes
– Determine and add routing area
5. Bottom-up traversal to update total layout area

Fig. 12. The operations needed for a construction of a single individual in the hybrid genetic algorithm for macro-cell layout generation.

blocks are joined to form a meta-block, their orientations are fixed. This is necessary since the channel between the blocks has been augmented with an estimated amount of routing space, which depends on the positions of the terminals on both blocks and thus, their orientations. Before fixing the orientations, all 16 possible combinations of the blocks relative to each other are checked to identify an arrangement with a maximal number of terminals to be connected on the adjacent sides of both blocks or partial layouts. All nondominated implementations for the flexible meta-blocks are stored in their shape functions. After completion of the tree, the layout with minimal area is identified, the flexible cells are sized, and the tree is transformed into a geometrical layout. For this geometrical layout, the shortest paths between the terminals to be connected by signal nets are determined. At this point, the number of nets in each channel is known, the channel widths are adapted, and the final positions of the cells on the layout are fixed. Details on the computation of the routing in our work can be found in [39]. Figure 12 provides an outline of the computation steps during the construction of an individual.

The main advantage of this approach, in comparison to other common approaches, is that the computation of the global routes for the signal nets is fully integrated into the placement process. The positions of the cells are not fixed until all routes have been determined. Furthermore, all shapes for the flexible cells are stored, and the globally optimal shapes are identified.

TABLE I
THE BENCHMARK CIRCUITS FOR THE MACRO-CELL LAYOUT GENERATION PROBLEM

	$xeroxf$	$xerox$	$ami33$	$ami49$
# cells	10	10	33	49
# nets	203	203	123	408
# terminals	698	698	452	958
# terminals/net	3.43	3.43	3.67	2.35
# I/O terminals	2	2	42	22
# shapes/cell	6.2	1	1	1
cell area [mm^2]	19.4	19.4	1.16	35.1

V. RESULTS

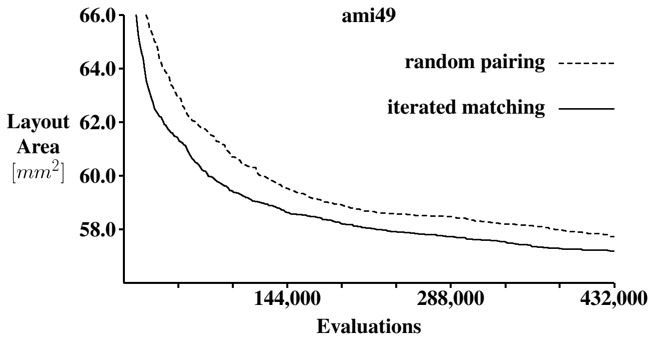
In this section the hybrid genetic algorithms are applied to different benchmark problems, and the results are compared to results published for other techniques.

A. *VLSI macro-cell layout generation*

The hybrid genetic algorithm for the layout generation problem was tested on real-life circuits chosen from a benchmark suite that was released for design workshops in the early 90s and is often referenced in the literature as the *MCNC benchmarks*. They were originally maintained by *North Carolina's Microelectronics, Computing and Networking Center*, but are now located at the *CAD Benchmarking Laboratory (CBL)* at North Carolina State University. These benchmarks are standard problems in macro-cell layout, and the characteristics of the circuits are shown in Table I.

Unless otherwise stated, all results of our hybrid genetic algorithm presented in this section were obtained using a parallel implementation based on the stepping stone model. Several subpopulations each consisting of ten individuals are processed in parallel with periodic migration of individuals between them. In addition to this, different strategies are pursued by the subpopulations, which are dynamically adapted during the search [40]. These strategies differ in the frequency used for the particular mutation operators, in the ratio of mutation to recombination for offspring creation, and in the use of the iterated matching during recombination.

When pairing blocks randomly for VLSI-layout generation, cells are spread arbitrarily with regard to their connectivity on the layouts generated during the search. Iterated matching can be used to enforce highly connected cells to be placed close together. Although computing the matching in a complete graph as we implemented it has cubic runtime, the overhead can be neglected for graphs



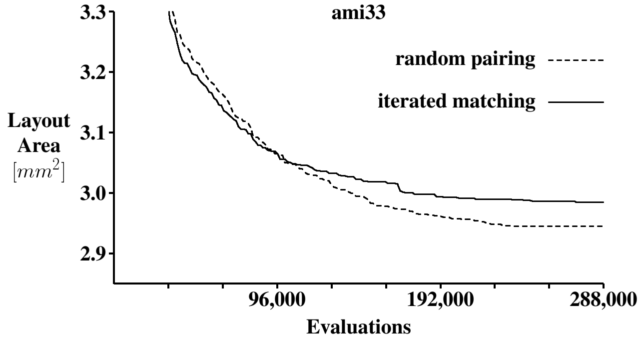


Fig. 13. Progress with respect to the use of the iterated matching heuristic during recombination for VLSI macro-cell layout generation. The average fitness based on ten runs is shown for each case, all parameters were the same, only the recombination operator was changed.

with less than 100 nodes, and the number of meta-blocks to match during recombination is usually much smaller than the number of cells. Figure 13 presents the performance of runs with and without use of iterated matching during recombination. All curves are averages based on ten runs with 16 subpopulations using the same parameters. To avoid side effects (for example compensation of inefficient recombination by increasing the mutation rate), no strategy adaptation has been used. While the use of iterated matching for combining subtrees during recombination is useful when generating a layout for $ami49$, it leads to premature convergence when dealing with $ami33$. A possible explanation for this effect is the deterministic character of the matching heuristic. Especially in combination with gene-pool recombination, when inserting the subtrees of the top 50% of all individuals into the pool, iterated matching tends to pair the randomly chosen subtrees in a similar manner. To overcome this problem and still take advantage of the beneficial effect for larger problems, in the full version of the hybrid genetic algorithm used to obtain the final results, the use of iterated matching during recombination is adapted based on the progress of the evolution [40].

TABLE II
THE LAYOUT AREAS [MM^2] FOR THE VLSI BENCHMARK CIRCUITS (AVERAGE OF 30 RUNS FOR OUR RESULTS)

		$xeroxf$	$xerox$	$ami33$	$ami49$
MBP [41]		—	25.79	2.42	—
TimberWolf [42]		—	—	2.57	—
BB [43]		—	26.17	2.24	51.49
SAGA [23]		—	27.55	—	—
FRODO [44]		29.41	31.13	3.37	60.02
CAR [44]		26.08	28.71	2.64	56.40
Hybrid GA	**Best**	27.34	27.77	2.77	53.49
	Avg	27.72	29.25	2.92	56.74
	σ	0.19	0.47	0.07	1.04

Table II presents the sizes of layouts generated by the hybrid genetic algorithm for the benchmark circuits based on 30 runs for each problem. All runs were done on a network of 16 Motorola MPC 601 processors with a subpopulation of 10 individuals on each processor. The computation times for the largest problems were 34 minutes for $ami33$ and 85 minutes for $ami49$ by running the genetic algorithm for 1800 and 2700 generations, i.e., 288,000 and 432,000 evaluations, respectively. For comparison some previously published results for this benchmark set are listed. The best results are reported by Onodera *et al.* [43]. They use a branch & bound method to place the cells. This approach only scales up to placing six cells, and for larger instances the layout must be composed hierarchically. In particular, for circuit $ami49$, two levels of hierarchy are needed. TimberWolf by Swartz and Sechen [42] and MBP by Upton *et al.* [41] are based on simulated annealing. SAGA by Esbensen and Mazumder [23] is a mixture of a genetic algorithm and simulated annealing. It starts as a genetic algorithm and gradually switches to a simulated annealing process by reducing population size and increasing the mutation rate. Their approach is limited to smaller circuits containing fixed cells. FRODO is a floorplanning tool, based on the work of Lengauer and Müller [45]. The reported results are from the thesis of Pape [44], who uses the placements generated by FRODO as an input to his tool CAR, which refines and topologically compacts the layout.

Although the same set of benchmark circuits was treated in the previously mentioned approaches, the comparison

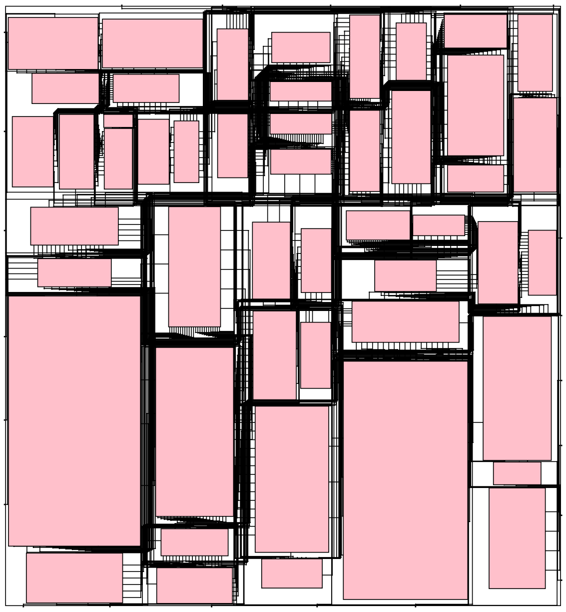


Fig. 14. A layout for circuit $ami49$, area = $55.65 mm^2$

is complicated. The results of TimberWolf have been achieved by combining several tools. BB only determines a placement and estimates the routing space, the presented results have been obtained by using other (commercial) tools for routing and compaction after placement. SAGA and FRODO do only floorplanning, and CAR starts its work with a given placement that has been generated by FRODO. There is always a strict distinction between placement and routing in all these approaches. An integration of the routing into the placement process in VLSI design is desirable, since the accuracy of the routing space estimate determines the accuracy for the assessment of a particular placement. Overestimates can lead to suboptimal layouts, in the case of underestimates the cells have to be rearranged to obtain a routable placement. The hybrid genetic algorithm proposed in this article fully integrates the computation of the global routing into the placement. The presented results are obtained exclusively by the use of this algorithm. From the results of the other approaches only those of SAGA and FRODO are directly comparable to our approach, since in these cases, no detailed routing has been done. The routability of our layouts has been checked by a tool developed at the University of Osnabrück, which also produced the detailed routing for the layout presented in Figure 14. Incorporating a better technique for detailed routing or compacting the final layout after routing, like the other approaches have done, will certainly produce better results. Alternatively, a sophisticated heuristic could be included to determine the demand of routing space inside the channels more accurately. At the moment, an upper bound of routing space is inserted by adding one track for each net in a channel. Nevertheless, even with this simple heuristic our best results are nearly competitive with those presented by the other approaches using specialized tools for each stage of the layout generation process.

B. *The facility layout problem*

The implementation of the hybrid genetic algorithm for the facility layout problem has been tested on a set of eight instances proposed by Tam and Li [10]. These instances are named TL91-X in the following with X representing the number of facilities to place. The areas of the facilities are quite different, for instance in TL91-30 the sizes of the 30 facilities vary between 3 and 36 (avg. 11.9, σ=8.4). A measure for the complexity of the problem with regard to the connectivity is the flow dominance, defined as the coefficient of variation for the entries in the flow matrix [46]. For the problems in the benchmark set, the flow dominance is between 100 and 130, which means that they do not cover a broad range of problems regarding this measure. However, this is a well-known benchmark set, and results for different approaches have been published, which provides a good basis for comparison.

TABLE III

THE NUMBER OF IMPLEMENTATIONS STORED IN THE ROOT NODE OF A SINGLE INDIVIDUAL AND THE AVERAGE COMPUTATION TIME (SUN ULTRA1/140) TO GENERATE AN INDIVIDUAL OF THE INITIAL POPULATION (INIT) AND DURING THE SEARCH (OPT), AVERAGE OF 1000 VALUES IN EACH CASE

	# shapes	**time** [ms]	
		init	**opt**
TL91-5	42.0	5.1	4.6
TL91-6	47.0	6.4	7.0
TL91-7	58.9	9.8	10.4
TL91-8	77.7	11.4	12.0
TL91-12	163.3	29.8	33.9
TL91-15	142.9	42.8	43.1
TL91-20	183.7	70.3	85.2
TL91-30	251.8	169.3	210.8

The continuous shape functions for the facilities were transformed to discrete shape functions with 10 implementations for each facility. Combining two of these facilities and considering all possible orientations for both, there are $4 \cdot 10 \cdot 10$ combinations to consider for the resulting

TABLE IV
THE OBTAINED FLOW COSTS FOR THE FACILITY LAYOUT PROBLEM (AVERAGE OF 30 RUNS), COMPARED TO PREVIOUSLY PUBLISHED RESULTS

		TL91–5	TL91–6	TL91–7	TL91–8	TL91–12	TL91–15	TL91–20	TL91–30
Tam and Li [10]		247	514	559	839	3162	5862	—	—
Kado *et al.* [13]		228	362	559	839	3162	5862	16535	42814
Garces-Perez *et al.* [14]		226	384	568	878	3220	7510	14033	39018
Hybrid GA	**Best**	214	327	629	833	3164	6813	13190	35358
	Avg	214	336	644	886	3203	7004	14333	36984
	σ	0.0	8.6	17.5	9.4	17.0	205.5	367.5	850.8

meta-block. Continuing this computation when constructing the tree representing a complete layout, one might expect an exponential growth for the number of implementations stored in each level of the tree. As shown in section III-B, some implementations are covered by others when constructing a shape function. Table III shows that the vast majority of combinations are redundant, since, even when combining 30 flexible facilities, there exist on average only 251.8 implementations, which are stored in the root node of the tree representing an individual. In the table timing data is also given, which shows the scalability of this approach. The difference between the time to generate an initial individual and the time needed to generate an offspring during the optimization is caused by the overhead to set up the gene pool for recombination.

In Table IV, the results for the facility layout problem are shown, averaged over 30 runs. For this benchmark set, a special objective function, $0.5 \cdot \sum_{j>i} d_{ij}^2 \cdot m_{ij}$, is used for the flow costs of a placement to increase the influence of the distances between facilities. Four subpopulations were used, the size of each was 10, resulting in a total population size of 40. During recombination, the subtrees of the top 50% of all individuals in each subpopulation were inserted into the gene-pool. The number of generations depended on the problem size: the genetic algorithm was run for 1000 generations for problems with less than 10 facilities, 2000 generations for mid-range problems, and 3000 generations for 20 and 30 facilities. This resulted in a computation time of 40 and 100 minutes for the two largest problems on a network of four Sparc Ultra/140 workstations for 120,000 evaluations in either case.

Table IV also includes the best results reported by Tam and Li [10], Kado *et al.* [13], and Garces-Perez *et al.* [14] for comparison. Although Tam and Li originally proposed these instances, they do not present final results for problems TL91-20 and TL91-30, due to difficulties with the scalability of their approach. Kado *et al.* [13] implemented a set of genetic algorithms based on a slicing-tree representation. They were able to obtain better results for the two smallest problems and were the first group to present results for the problems with 20 and 30 facilities. Garces-Perez *et al.* [14] use GP and report better results for the last two problems, whereas the performance for the smaller problems varies. Our approach outperforms the other approaches for the two largest problems, and the flow costs of the best layout for the instance with 30 facilities produced by our hybrid genetic algorithm are 10% and 20% smaller than the results presented by Garces-Perez *et al.* [14] and Kado *et al.* [13], respectively. Their approaches size the facilities after fixing a relative placement and use a continuous representation. While this provides better solutions for smaller problem instances, where our approach ended up with less optimal solutions, considering different shapes in discrete shape functions payed off for the larger problems. From a practical point of view, if better performance for smaller instances is demanded, a refined discretization of the shape functions (i.e. make use of more than ten shapes per block), or even a refinement of the best layout after evaluating a tree should provide better results. However, our goal during the implementation of the hybrid genetic algorithm was to achieve better results for the larger problem instances within a practicable timeframe, since this is a more demanding task.

VI. CONCLUSIONS

Hybrid approaches to two significant combinatorial placement problems have been presented. These are genetic algorithms with nonstandard genotypic representation and specific genetic operators. During the construction of individuals, several problem-specific heuristics address the different objectives and constraints. While the application of the slicing-tree representation and the concept of generalized shape functions deal with the packing aspects, connectivity is considered by using the iterated matching heuristic. As a result, the hybrid genetic algorithms are able to take all constraints into consideration during optimization.

For the generation of VLSI macro-cell layouts, an approach has been introduced that fully integrates the computation of the global routes and the sizing of the flexible cells into the placement task. It is more scalable than most other approaches and can be further improved by incorporating a better heuristic to estimate the area needed to complete the detailed routing of signal nets.

In the case of the facility layout problem, the hybrid approach shows much better scalability than several approaches using the same benchmark set. The instances in this set represent a generic problem type of this domain, which includes various design applications. The proposed approach can certainly be extended to consider shape constraints for the site the facilities have to be placed on and to solve problems containing preoccupied areas. It can also be extended to deal with office or production-hall layout problems where passages are needed between the placed facilities, similar to the routing area in VLSI layouts.

The main feature of the approach introduced here, in comparison with other approaches, is the manner in which block flexibility is treated: During the iterative composition of a placement, several implementations (shapes and orientations) for blocks and meta-blocks are stored. This process can be described as a kind of 'implicit hillclimbing'. Common hillclimbers in genetic algorithms for combinatorial optimization problems randomly explore solutions neighboring the candidate solution encoded in the current individual and accept those with better fitness. In hybrid approaches, local search techniques explore the solution space close to the sample points by applying specialized heuristics. When including problem-specific knowledge during creation of individuals, like in our approach, it is possible to identify unfavorable or redundant partial solutions and consider only the most promising ones. Therefore, each individual in our hybrid genetic algorithms encodes a set of high-quality solutions, the best of which is a local optimum.

Acknowledgment

The authors thank the *Paderborn Center for Parallel Computing (PC^2)* for the opportunity to use their parallel machines. Thanks to Mike Raymer and especially Leslie Kuhn for critical review of the manuscript and to the editor and the anonymous reviewers for valuable comments.

References

[1] P. M. Pardalos, F. Rendl, and H. Wolkowics, "The quadratic assignment problem: A survey and recent developments," in *DIMACS Series Discr. Math. Theor. Comp. Science*, 1994, vol. 16, pp. 1–42.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman, San Francisco, 1979.

[3] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimisation," *Evolutionary Computation*, vol. 3, pp. 1–16, 1995.

[4] K. R. Ryu, J. Hwang, H. R. Choi, and K. K. Cho, "A genetic algorithm hybrid for hierarchical reactive scheduling," in *Proc. of the 7th Int. Conf. on Genetic Algorithms (ICGA)*, Th. Bäck, Ed., San Francisco, 1997, pp. 497–504, Morgan Kaufmann.

[5] R. Otten, "Efficient floorplan optimization," in *Proc. of Int. Conf. on Comp. Design*, Silver Spring, MD, 1983, pp. 499–502, IEEE Comp. Soc. Press.

[6] A. Fritsch and O. Vornberger, "Cutting stock by iterated matching," in *Operations Research Proceedings, Selected Papers of the Int. Conf. on OR 94*, U. Derigs, A. Bachem, and A. Drexl, Eds., Berlin, 1995, pp. 92–97, Springer.

[7] A. Kuziak and S. Heragu, "The facility layout problem," *Europ. J. of Operational Research*, vol. 29, pp. 229–251, 1987.

[8] R. D. Meller and K.-Y. Gau, "The facility layout problem: Recent trends and perspectives," *J. of Manufacturing Systems*, vol. 15, no. 5, pp. 351–366, 1996.

[9] D. J. van Camp, M. W. Carter, and A. Vannelli, "A nonlinear optimization approach for solving facility layout problems," *Europ. J. of Operational Research*, vol. 57, pp. 174–189, 1991.

[10] K. Y. Tam and S. G. Li, "A hierarchical approach to the facility layout problem," *Int. J. of Production Research*, vol. 29, no. 1, pp. 165–184, 1991.

[11] K. Y. Tam, "Genetic algorithms, function optimization, and facility layout design," *Europ. J. of Operational Research*, vol. 63, pp. 322–346, 1992.

[12] K. Y. Tam, "A simulated annealing algorithm for allocating space to manufacturing cells," *Int. J. of Production Research*, vol. 30, no. 1, pp. 63–87, 1992.

[13] K. Kado, P. Ross, and D. Corne, "A study of genetic algorithm hybrids for facility layout problems," in *Proc. of the 6th Int. Conf. on Genetic Algorithms (ICGA)*, L. J. Eshelman, Ed., San Mateo, CA, 1995, pp. 498–505, Morgan Kaufmann.

[14] J. Garces-Perez, D. A. Schoenefeld, and R. L. Wainwright, "Solving facility layout problems using genetic programming," in *Proc. 1st Annual Conf. on Genetic Programming (GP-96)*, J. Koza, D. Goldberg, D. Fogel, and R. Riolo, Eds., Cambridge, MA, 1996, pp. 182–190, MIT Press.

[15] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer, Norwell, 1993.

[16] K. Shahookar and P. Mazumder, "VLSI cell placement techniques," *ACM Computing Surveys*, vol. 23, no. 2, pp. 143–220, June 1991.

[17] Th. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, New York, 1990.

[18] S. M. Sait and H. Youssef, *VLSI Physical Design Automation: Theory and Practice*, McGraw-Hill, New York, 1995.

[19] J. P. Cohoon and W. D. Paris, "Genetic placement," in *Proc. of IEEE Int. Conf. on CAD*, Washington, D.C., 1986, pp. 422–425, IEEE Comp. Soc. Press.

[20] J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. S. Richards, "Distributed genetic algorithms for the floorplan design problem," *IEEE Trans. on CAD*, vol. 10, no. 4, pp. 483–492, April 1991.

[21] H. Chan, P. Mazumder, and K. Shahookar, "Macro-cell and module placement by genetic adaptive search with bitmap-represented chromosome," *Integration, the VLSI journal*, vol. 12, no. 1, pp. 49–77, 1991.

[22] H. Esbensen, “A genetic algorithm for macro cell placement,” in *Proc. of the Europ. Design Automation Conference*, Los Alamitos, CA, 1992, pp. 52–57, IEEE Comp. Soc. Press.

[23] H. Esbensen and P. Mazumder, “SAGA: A unification of the genetic algorithm with simulated annealing and its application to macro-cell placement,” in *Proc. of the 7th Int. Conf. on VLSI Design*, 1994, pp. 211–214.

[24] L. Stockmeyer, “Optimal orientations of cells in slicing floorplan designs,” *Information and Control*, vol. 57, pp. 91–101, 1983.

[25] B. Kröger, “Guillotineable binpacking: A genetic approach,” *Europ. J. of Operational Research*, vol. 84, no. 3, pp. 645–661, 1995.

[26] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*, MIT Press, Cambridge, MA, 1992.

[27] H. Bersini and G. Seront, “In search of a good evolution–optimization crossover,” in *Proc. 2nd Conf. on Parallel Problem Solving from Nature (PPSN II)*, R. Männer and B. Manderick, Eds., Amsterdam, 1992, pp. 479–488, North-Holland.

[28] H. Mühlenbein, “Parallel genetic algorithms, population genetics and combinatorial optimization,” in *Proc. of the 3rd Int. Conf. on Genetic Algorithms (ICGA)*, J. D. Schaffer, Ed., San Mateo, CA, 1989, pp. 416–421, Morgan Kaufmann.

[29] A. E. Eiben, P.-E. Raué, and Z. Ruttkay, “Genetic algorithms with multi-parent recombination,” in *Proc. 3rd Conf. on Parallel Problem Solving from Nature (PPSN III)*, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds. 1994, number 866 in LNCS, pp. 78–87, Springer, Berlin.

[30] Th. Bäck and H.-P. Schwefel, “An overview of evolutionary algorithms for parameter optimization,” *Evolutionary Computation*, vol. 1, pp. 1–23, 1993.

[31] H.-P. Schwefel, *Evolution and Optimum Seeking*, John Wiley & Sons, New York, 1995.

[32] H. Mühlenbein and H.-M. Voigt, “Gene pool recombination in genetic algorithms,” in *Proc. of the Metaheuristics Int. Conf.*, I. H. Osman and J. P. Kelly, Eds., Norwell, 1995, Kluwer.

[33] D. E. Goldberg, *Genetic Algorithms in Search Optimization & Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[34] K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. thesis, University of Michigan, 1975.

[35] D. E. Goldberg and J. Richardson, “Genetic algorithms with sharing for multimodal function optimization,” in *Proc. of the 2nd Int. Conf. on Genetic Algorithms (ICGA)*, J. J. Grefenstette, Ed., Hillsdale, NJ, 1987, pp. 41–49, Erlbaum.

[36] D. Whitley, “The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best,” in *Proc. of the 3rd Int. Conf. on Genetic Algorithms (ICGA)*, J. D. Schaffer, Ed., San Mateo, CA, 1989, pp. 116–121, Morgan Kaufmann.

[37] P. Merz and B. Freisleben, “Genetic local search for the tsp: New results,” in *Proc. of the 1997 IEEE Int. Conf. on Evolutionary Computation*. 1997, pp. 159–164, IEEE Press, NY.

[38] P. Merz and B. Freisleben, “A genetic local search approach to the quadratic assignment problem,” in *Proc. of the 7th Int. Conf. on Genetic Algorithms (ICGA)*, Th. Bäck, Ed., San Francisco, CA, 1997, pp. 465–472, Morgan Kaufmann.

[39] V. Schnecke, *Hybrid Genetic Algorithms for Solving Constrained Packing and Placement Problems*, Ph.D. thesis, Department of Mathematics/Computer Science, University of Osnabrück, Germany, 1996.

[40] V. Schnecke and O. Vornberger, “An adaptive parallel genetic algorithm for VLSI-layout optimization,” in *Proc. 4th Conf. on Parallel Problem Solving from Nature (PPSN IV)*, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Eds. 1996, number 1141 in LNCS, pp. 859–868, Springer, Berlin.

[41] M. Upton, K. Samii, and S. Sugiyama, “Integrated placement for mixed macro cell and standard cell designs,” in *Proc. 27th ACM/IEEE Design Automation Conference*, Los Alamitos, CA, 1990, pp. 32–35, IEEE Comp. Soc. Press.

[42] W. Swartz and C. Sechen, “New algorithms for the placement and routing of macro cells,” in *Proc. IEEE Int. Conf. on Computer Aided Design*, Los Alamitos, CA, 1990, pp. 336–339, IEEE Comp. Soc. Press.

[43] H. Onodera, Y. Taniguchi, and K. Tamaru, “Branch-and-bound placement for building block layout,” in *Proc. 28th ACM/IEEE Design Automation Conference*, Los Alamitos, CA, 1991, pp. 433–439, IEEE Comp. Soc. Press.

[44] M. Pape, *Chip Assembly mit Topologischer Kompaktierung*, Ph.D. thesis, Department of Mathematics and Computer Science, University of Paderborn, 1995.

[45] Th. Lengauer and R. Müller, “Robust and accurate hierarchical floorplanning with integrated global wiring,” *IEEE Trans. on CAD*, vol. 12, no. 6, pp. 802–809, June 1993.

[46] T. E. Vollmann and E. S. Buffa, “The facilities layout problem in perspective,” *Management Science*, vol. 12, no. 10, pp. 450–468, 1966.