AN IC/MCM TIMING-DRIVEN PLACEMENT ALGORITHM FEATURING EXPLICIT DESIGN SPACE EXPLORATION

by

Henrik Esbensen and Ernest S. Kuh

Memorandum No. UCB/ERL M95/20

29 March 1995

AN IC/MCM TIMING-DRIVEN PLACEMENT ALGORITHM FEATURING EXPLICIT DESIGN SPACE EXPLORATION

by

Henrik Esbensen and Ernest S. Kuh

Memorandum No. UCB/ERL M95/20

29 March 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Title (?)

An IC/MCM Timing-Driven Placement Algorithm Featuring Explicit Design Space Exploration

Henrik Esbensen   Ernest S. Kuh

Dept. of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720, USA

Abstract

A genetic algorithm for building-block placement of ICs and MCMs is presented which simultaneously minimizes layout area and an Elmore-based estimate of the maximum path delay while trying to meet a target aspect ratio. Explicit design space exploration is performed by using a vector-valued, 3-dimensional cost function and searching for a set of distinct solutions representing the best tradeoffs of the cost dimensions. From the output solutions, the designer can choose the solution with the preferred tradeoff. This approach eliminates the inherent weight and constraint specification problems of existing multi-objective placement methods, in which a weighted sum is minimized subject to user-defined constraints. Very good experimental results are obtained for various placement problems.

1 Introduction

During placement of an integrated circuit (IC) or a multichip module (MCM) the objective is to find a solution which is satisfactory with respect to each of a number of competing criteria. Most often specific constraints has to be met for some criteria, while for others, a good tradeoff is wanted. However, at this point in the design process, the available information as to which values are obtainable for each criteria is based on relatively rough estimates only. Consequently, the designers notion of the overall design objective is rarely clearly definable.

Virtually all existing placement tools considering more than one optimization criterion minimizes a weighted sum of some criteria subject to constraints on others. E.g., if k criteria are considered, the objective is to minimize the single valued cost function

$$c = \sum_{i=1}^{j} w_i c_i \quad \text{subject to} \quad \forall\ i = j+1, \ldots, k : c_i \leq C_i \tag{1}$$

for some j, $1 \leq j \leq k$. Here c_i measures the cost of the solution with respect to the i'th criterion and the w_i's and C_i's are user-defined weights and bounds, respectively. For example, if area is minimized subject to a bound on delay we have $k = 2$ and $j = 1$.

However, in practice it may be very difficult for the designer to specify a set of bounds and weights which makes the placement tool find a satisfactory solution. If the bounds are too tight, a solution may not be found at all, and it is far from clear how to derive a set of weight values from the vaguely defined design objectives. Furthermore, depending on the nature of the c_i functions, the relative magnitude of the $w_i c_i$ terms may change during the optimization process itself, in which case constant weights are unlikely to keep the cost function properly balanced throughout the process.

Our work is motivated by the need to overcome these problems. A building block placement algorithm for both ICs and MCMs is presented, which supports *explicit* design space exploration in the sense that 1) a set of alternative solutions rather than a single solution is generated, and 2) solutions are characterized explicitly by a cost value for each criterion instead of a single, aggregated cost value. The algorithm simultaneously minimizes layout area, the maximum path delay and the deviation from a target aspect ratio. A 3-dimensional, vector-valued cost function measures solution quality for each criterion independently, and the algorithm searches for a set of alternative, good solutions where "good" is defined relative to a user-defined goal vector specifying ideal values for each criterion. The goal vector eliminates the use of both the weights and the bounds of (1) and as opposed to the bounds, the goal values need not be obtainable. From the output set of solutions, the designer chooses a solution representing the preferred tradeoff.

Apart from supporting explicit design space exploration as described, the approach has two other significant characteristics:

- It is based on the genetic algorithm (GA), which is well suited for multi-objective optimization in the above sense since it operates on a set of solutions simultaneously [9].
- Delay minimization is path-based, i.e., path delay is explicitly modelled and minimized. This is in contrast to net-based placement approaches, in which given path constraints are initially converted into net constraints. While simpler, net-based approaches usually overconstrains the problem, thereby potentially excluding good solutions from being found [10].

Previous work on explicit design space exploration in CAD is still very limited, but approaches for scheduling and channel routing are presented in [5]. In recent years an increasing number of GA applications in CAD are reported [6]. However, we are only aware of three previous GA approaches to building-block placement [3, 4, 8], none of which optimizes delay. Despite the fact that delay is inherently path oriented, timing-driven, path-based placement algorithms are rare. The few existing approaches includes [12, 14, 15], all of which, however, relies on very simple net models (stars and bounding boxes). In contrast, the approach presented here approximates net topology by a minimum spanning tree, in which the Elmore delay is computed.

The remaining of this paper is organized as follows. Section 2 presents the problem definition used. The algorithm is described in Section 3 and in Section 4 experimental results are presented. Conclusions are given in Section 5.

2 Problem Definition

The placement model described in Section 2.1 is relevant for both MCMs and for IC technologies with at least two metal layers available for routing. Section 2.2 characterizes the solution set searched for by the algorithm.

2.1 Placement Model

The given input is

- A set of rectangular building blocks of arbitrary sizes and aspect ratios with a set of pins located anywhere within each block.
- A specification of all nets, including for each net 1) the capacitance of each sink pin, and 2) a designated source pin p, its driver resistance and an associated internal delay $t(p)$ in the block $m(p)$ to which p belongs. $t(p)$ is the time it takes a signal to travel through $m(p)$ to p.
- A specification of a set of paths $\mathcal{P}$. A path connects two registers of distinct blocks, i.e., it is an alternating sequence of wires passing through blocks and net segments. For a sink pin p, denote by $s(p)$ the source pin of the net to which p belongs. Each path $P \in \mathcal{P}$ is then uniquely specified by an ordered set of sink pins $P = \{p_0, p_1, \ldots, p_{l-1}\}$ of distinct nets, such that $m(p_i) = m(s(p_{i+1})), i = 0, 1, \ldots, l-2$. Each path in an MCM will have length $l = 1$ assuming that all signals are latched at the inputs of the components.
- The routing wire resistance $\bar{r}$ and capacitance $\bar{c}$ per unit wire length.

Each output solution is a specification of

- An absolute position of each block so that no blocks are closer than a specified minimum distance $\lambda \geq 0$. This parameter allow physical constraints (design rules) to be met and is not intended for routing area allocation. It is assumed that routing is performed mainly on top of the blocks.
- An orientation and reflection of each block. Each block in an IC can be orientated and reflected in eight distinct ways, while for MCMs, four distinct orientations/reflections exists.

For a given placement, the topology of each net is approximated by computing a minimum spanning tree MSpT over all pins of the net. The delay $D(P)$ of each path $P = \{p_0, p_1, \ldots, p_{l-1}\}$ is then estimated as

$$D(P) = \sum_{i=0}^{l-1} [E(p_i) + t(s(p_i))] \tag{2}$$

where $E(p_i)$ is the Elmore delay [7] from $s(p_i)$ to p_i computed in the MSpT. The objective wrt. delay is to minimize the maximum delay $\max\{D(P) \mid P \in \mathcal{P}\}$. The MSpT computation is a relatively accurate estimation of the net topology, cf. Section 1. Furthermore, for ICs the topology dependent Elmore delay estimate has high fidelity in the sense that a solution which is near-optimal according to the estimate will also be near-optimal wrt. actual delay [2].

2.2 What is a "Good" Tradeoff ?

Let Π be the set of all placements and $\Re_+ = [0, \infty[$. The cost of a solution is defined by the function $c = (c_a, c_r, c_d) : \Pi \mapsto \Re_+^3$. c_a is the layout area, i.e., the area of the smallest rectangle enclosing all blocks. $c_r = |r_{actual} - r_{target}|$ is the distance between the actual aspect ratio r_{actual} (height divided by width) and a user-defined target aspect ratio r_{target}. Finally, c_d is the maximum path delay, computed as described in Section 2.1.

Required/wanted values for each criterion is expressed by the designer in the form of a goal vector $g = (g_a, g_r, g_d) \in (\Re_+ \cup \{\infty\})^3$. Goals need not be obtainable, but merely specifies the properties of the ideal solution. For example, $g = (0, 0.1, \infty)$ specifies that the smallest possible area is wanted (the goal 0 will never be met), that r_{actual} should be within 0.1 from r_{target} and that any delay is satisfactory (since any delay is smaller than ∞). As knowledge of obtainable values is obtained, the designer can refine g in succeeding executions of the algorithm.

To guide the search towards a set of solutions representing "good" tradeoffs, a notion of relative solution quality is needed, which takes the goal vector into account. Let $x, y \in \Pi$, $c(x) = (x_1, x_2, x_3)$, $c(y) = (y_1, y_2, y_3)$, $g = (g_1, g_2, g_3)$, $i \in \{1, 2, 3\}$. The relation x *dominates* y, written $x <_d y$, is defined by

$$x <_d y \quad \Leftrightarrow \quad (\forall\, i : x_i \leq y_i) \wedge (\exists\, i : x_i < y_i) \tag{3}$$

Following [9], we then define the relation x *is preferable to* y, written $x \prec_p y$, as follows, depending on how x compares to the goal vector : If x satisfies all goals, i.e., $\forall\, i : x_i \leq g_i$ then

$$x \prec_p y \quad \Leftrightarrow \quad (x <_d y) \vee (\exists\, i : y_i > g_i) \tag{4}$$

If x satisfies none of the goals, i.e., $\forall\, i : x_i > g_i$ then

$$x \prec_p y \quad \Leftrightarrow \quad x <_d y \tag{5}$$

Finally, x may satisfy exactly one or two of the goals. I.e., assuming a convenient ordering of the optimization criteria, $\exists\, k \in \{2, 3\} : (\forall\, i < k : x_i \leq g_i) \wedge (\forall\, i \geq k : x_i > g_i)$. Then

$$\begin{aligned} x \prec_p y \quad \Leftrightarrow \quad & [(\forall\, i \geq k : x_i \leq y_i) \wedge (\exists\, i \geq k : x_i < y_i)] & (6) \\ & \vee & \\ & [(\forall\, i \geq k : x_i = y_i) \wedge & (7) \\ & \{((\forall\, i < k : x_i \leq y_i) \wedge (\exists\, i < k : x_i < y_i)) \vee (\exists\, i < k : y_i > g_i)\}] & (8) \end{aligned}$$

The right hand side of (6) states that x dominates y wrt. the dimensions for which x does not satisfy the goals. (7) and (8) states that in the special case when x equals y wrt. the non-satisfied dimensions, then x is still preferable to y if it either dominates y wrt. the satisfactory dimensions or if y does not satisfy a goal satisfied by x. Notice from (6) that when two solutions satisfy the same subset of goals, they are considered equal with respect to these goals, regardless of their specific values in these dimensions. In other words, when goals are satisfied, they are "factored out", focusing the search on the remaining, unsatisfactory dimensions.

The algorithm outputs a set of distinct solutions Φ_0 which are the "best" found in the sense defined by $\prec_p$. As a special case, if $g = (0, 0, 0)$ the algorithm searches for (a sample of) the Pareto-optimal set, i.e., the set of solutions, in which no solution can be improved in any dimension without being deteriorated in another. Since $\forall\, x, y \in \Phi_0 : \neg(x \prec_p y)$ the output solutions represents distinct design alternatives.

3 Description of the Algorithm

The concept of genetic algorithms, introduced in 1975 by John Holland [13], is based on natural evolution. In nature, the individuals constituting a population adapt to the environment in which they live. The fittest individuals have the highest probability of survival and tend to increase in numbers, while the less fit individuals tend to die out. This *survival-of-the-fittest* Darwinian principle is the basic idea behind the GA. The algorithm maintains a *population* of *individuals*, each of which corresponds to a specific solution to the optimization problem considered. Based on a given cost function, a measure of *fitness* defines the relative quality of individuals. An evolution process is simulated, starting from a set of random individuals. The main components of this process are *crossover*, which mimics propagation, and *mutation*, which mimics the random changes occurring in nature. After a number of *generations*, highly fit individuals will emerge corresponding to good solutions to the optimization problem.

A *phenotype* is the physical appearance of an individual, while a *genotype* is the corresponding representation or genetic encoding of the individual. Crossover and mutation are performed in terms of genotypes, while fitness/cost is defined in terms of phenotypes. For a given genotype, the corresponding phenotype is computed by a *decoder*. A good introduction to genetic algorithms is given in [11]. Section 3.1 outlines our algorithm, Section 3.2 presents the genotype and its interpretation, and Sections 3.3 and 3.4 describes the genetic operators and the selection strategy, respectively.

3.1 Overview

```
generate($\Phi$);
repeat :
  select $\phi_1, \phi_2 \in \Phi$;
  def$\psi_2$ := crossover($\phi_1, \phi_2, \psi_1, \psi_2$);
  mutate($\psi_1$);
  update($\Phi, \phi_1, \psi_1$);
  if def$\psi_2$ :
    mutate($\psi_2$);
    update($\Phi, \phi_2, \psi_2$);
  if converged() :
    optimize($\Phi_0$);
  diversify($\Phi$);
until converged() or terminate();
output $\Phi_0$;
```

Figure 1: *Outline of the algorithm.*

Fig. 1 outlines our GA. Let $\Phi = \{\phi_0, \phi_1, \ldots, \phi_{N-1}\}$ denote the current population. The *rank* $r(\phi)$ of $\phi \in \Phi$ is the number of currently existing individuals which are preferable to ϕ, i.e., $r(\phi) = |\{\gamma \in \Phi \,|\, \gamma \prec_p \phi\}|$. Furthermore, let $\Phi_0 = \{\phi \in \Phi \,|\, r(\phi) = 0\} \subseteq \Phi$, i.e., Φ_0 is the current best solutions. Initially, Φ is constructed by routine *generate* from distinct, random individuals (line 1). One iteration of the repeat loop (lines 2-13) corresponds to the simulation of one generation. Throughout the optimization $N = |\Phi|$ is kept constant.

In each generation, two parent individuals ϕ_1 and ϕ_2 are selected for crossover as described in Section 3.4 (line 3). The crossover operator, described in Section 3.3, generates offspring ψ_1 and possibly ψ_2 (line 4). defψ_2 is true if and only if ψ_2 was defined. The algorithm is a steady-state GA, which means that only one crossover operation is performed per generation. Routine *mutate*, described in Section 3.3, subjects the generated offspring to random changes (lines 5, 8) and the resulting individuals are inserted in Φ by routine *update* (lines 6, 9). In general, *update*(Φ, ϕ, ψ) replaces parent ϕ by offspring ψ. However, if $(r(\phi) = 0) \wedge \neg(\psi \prec_p \phi)$, another poor solution with high rank is replaced instead. Fig. 2 describes the detailed replacement scheme.

```
if $(r(\phi) > 0) \vee (\psi \prec_p \phi)$ :
  $\Phi := (\Phi \setminus \{\phi\}) \cup \{\psi\}$;
else
  $\Phi_\psi := \{\gamma \in \Phi \,|\, \psi \prec_p \gamma\}$;
  if $\Phi_\psi = \emptyset$ :
    $\Phi_\psi := \{\gamma \in \Phi \,|\, \neg(\gamma \prec_p \psi) \wedge r(\gamma) > r(\psi)\}$;
  if $\Phi_\psi \neq \emptyset$ :
    randomly select $\gamma \in \Phi_\psi$ with $r(\gamma)$ maximal;
    $\Phi := (\Phi \setminus \{\phi\}) \cup \{\gamma\}$;
```

Figure 2: *Routine update(Φ, ϕ, ψ). Notice that Φ may be unaltered.*

Routine *converged* (line 10) detects if no improvement has occurred in S consecutive generations, that is, if Φ_0 has not changed in this period. In that case routine *optimize* (line 11) attempts to optimize all rank zero individuals by simple hillclimbing. On each individual $\phi \in \Phi_0$ a sequence of H mutations is tried. Each mutation yielding γ from ϕ is only executed if $\gamma \prec_p \phi$. With frequency F (a number of generations) routine *diversify* (line 12) ensures that all solutions represents distinct cost values. For all pairs $\phi, \psi \in \Phi \mid c(\phi) = c(\psi)$, routine *diversify* executes a sequence of mutations on ψ until $c(\psi) \neq c(\phi)$. The algorithm terminates (line 13) when either a) the process has converged, or b) Φ_0 contains a solution satisfying all goals or c) a cpu-limit T has been exceeded (detected by routine *terminate*). Finally, Φ_0 is the output set of solutions (line 14).

Whenever routines *update*, *optimize* and *diversify* replaces or alters individuals of Φ the ranks of all individuals are updated. As described above, routines *update* and *optimize* only replace, resp. modify, a rank zero individual if the new individual is preferable to the existing one.

Furthermore, routine *diversify* leaves the subset of distinct solutions in Φ_0 unaltered. Hence, in the sense inferred by $\prec_p$ the solution set Φ_0 is monotonically improving as a function of time. For single-valued cost functions, GAs keeping the current best solution throughout the optimization are often referred to as *elitist* GAs. Thus, the above scheme can bee seen as a generalization of the elitist GA to vector-valued domains.

3.2 Solution Representation and Decoder

Given a problem instance with b blocks, the representation, or genotype, of a placement consist of two parts:

a) An inverse Polish expression of length $2b-1$ over the alphabet $\{0,1,\ldots,b-1,+,*\}$, where the operands $0,1,\ldots,b-1$ denotes block identities and $+,*$ are operators. Each operand occurs exactly once in each Polish expression. As in [4], an expression uniquely specifies a slicing-tree for the placement, with $+$ and $*$ denoting a horizontal and a vertical slice, respectively.

b) A bitstring of length q, where $q = 2b$ for ICs and $q = b$ for MCMs. The bitstring represents the reflection of each block. In an IC each block can be reflected in four distinct ways (reflection in one or both of the axis) without changing the contour of the block. The reflection of the i'th block is specified by bits $2i$ and $2i+1$. Similarly, each block in an MCM can be reflected in two distinct ways (180 degree rotation) as specified by the i'th bit for the i'th block.

Let Ω denote the set of all such representations and let $d : \Omega \mapsto \Pi$ be the decoder. The search space $\hat{\Pi}$ considered by the algorithm is defined as the image of d, i.e., $\hat{\Pi} \equiv d(\Omega) \subset \Pi$. For a given representation $\omega \in \Omega$ the corresponding placement, or phenotype, $d(\omega) \in \Pi$ is computed in five steps as follows (illustrated in Fig. 3):

1. From the unique slicing tree specified by the Polish expression, the orientation (possible 90 degree rotation) of each block is determined such that layout area is minimized. The orientations are computed using an exact algorithm by Stockmeyer [16] which guarantees that a minimum area layout for the given slicing-structure is obtained. The reflection of each block is as specified by the bitstring.

2. Absolute coordinates are determined for all blocks by a top-down traversal of the slicing tree. At each operator node, if relative movement of the two subtrees along the slicing axis is possible, the centerpoints of the subtrees are aligned (perpendicular to the slicing axis).

3. The layout is compacted, first vertically and then horizontally. The compaction algorithm is a simplified version of the one-dimensional channel compaction algorithm presented in [18], which adapts a scan-line approach. Area and aspect ratio of the layout is now computed.

4. For each net, a MSpT over all its pins is computed as an approximation of the net topology. The Elmore delay from the source pin to each sink is computed by traversing the MSpT.

5. Each path delay is computed using (2) and the delay of ω is the maximum path delay.

Since more than one slicing tree may exist for a slicing structure the decoder d is not injective. The genetic representation and the decoder defines the search space $\tilde{\Pi}$ considered by the algorithm as the set of all possible oriented (step 1), centered (step 2) and compacted (step 3) slicing structures. Note that $\tilde{\Pi}$ is not a subset of the set of slicing structures and vice versa.

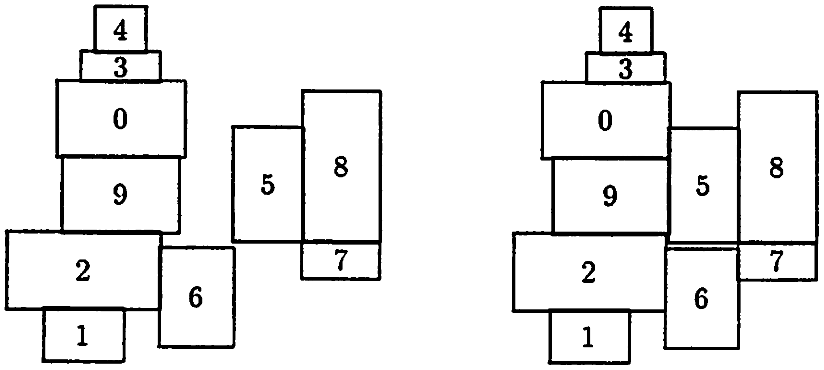


Figure 3: *Given 10 blocks and the Polish expression 1 2 + 6 * 9 0 + + 3 4 + + 5 * 7 8 + *, the placement on the left is the result of step 2 of the decoding. Subtrees are recursively centered and oriented optimally. Since the height of the layout is determined by the blocks 1,2,9,0,3,4, no blocks are moved when attempting vertical compaction. Subsequent horizontal compaction moves blocks 8,7,5,9 and 0 towards the left, so that blocks 2,6,7 now determines the width of the layout. The placement to the right is the result of compaction (step 3), i.e., the final placement.*

At first sight the use of Stockmeyers algorithm (step 1) as well as the compactor (step 3) may seem to introduce an unwanted bias towards optimizing area at the cost of delay. And indeed, simple pathological examples can be constructed in which each of these algorithms does increase the path delay. However, experiments performed using the layouts described in Section 4 clearly showed that for a given, fixed Polish expression, each of the algorithms of steps 1 and 3 also have a positive effect on the path delay in the vast majority of all cases. Hence, in practice these algorithms are believed to be beneficial wrt. path delay as well as area. Intuitively, this is because the path causing the maximum delay will connect a relatively large number of blocks, which are more likely to be further apart in a larger layout.

Since thousands of representations will be decoded during a typical execution the computational complexity of d is crucial. While step 2 requires linear time in the number of blocks b, Stockmeyers algorithm is quadratic and so is our implementation of the compactor. The time consumption of step 4 is dominated by the MSpT computations using Prims algorithm, which is quadratic in the number of pins of the net. Step 5 is linear in the total number of path segments p_{seg}, i.e., the accumulated number of sink pins of all paths. In total, each computation of d requires time $O(b^2 + \sum n_i^2 + p_{seg})$, where n_i is the number of pins of the i'th net and the sum is over all nets.

3.3 Genetic Operators

Given the genotypes of two parent individuals ϕ_1 and ϕ_2, the crossover operator (line 4 on Fig. 1) generates one or two new individuals ψ_1 and possibly ψ_2 as follows. The Polish expression parts and the bitstring parts of the new genotypes are constructed independently. The expression parts are constructed using one of four distinct operators CO1, C02, C03 and C04, introduced and described in detail in [4]. The operator applied is chosen uniformly at random. Each of CO1, C02 and C03 generates a single offspring, while C04 generates two offspring. When using CO1, ψ_1 inherits the order and positions of all operands from ϕ_1. Similarly when using CO2, ψ_1 inherits the order and positions of all operators from ϕ_1, i.e., C02 preserves the slicing structure. In addition to also preserving the complete slicing structure from one parent, C03 preserves a complete subtree. Finally, C04 generates two offspring by interchanging two subtrees of ϕ_1 and ϕ_2. If this is not possible while preserving feasibility of the generated expressions, C04 fails and one of the other operators are applied instead. The bitstring part of ψ_1 and, when C04 is applied, ψ_2, is generated using uniform crossover [11, 13] : Bit values are inherited independently of each other and the value of the i'th bit is inherited from either ϕ_1 or ϕ_2 with probability 0.5.

The mutation operator (also applied by routines *optimize* and *diversify*) similarly treats the Polish expression and the bitstring independently and the expression is mutated using the four mutations of [4] : A pair of operands can be interchanged, a pair of operators can be interchanged, the type of an operator can be changed and an operator and an operand can be interchanged. Only the latter operation requires a check for feasibility of the produced expression. Each possible and feasible mutation of the Polish expression is performed independently and in random order with a small user-defined probability p_{mut}. The bitstring is subjected to pointwise mutation [11, 13] : Each bit is independently inverted with probability p_{mut}.

A crucial property of the crossover operator as well as the mutation operator is that they preserve feasibility, i.e., only feasible genotypes, which can be interpreted by the decoder, are ever generated. If feasibility were not preserved by the operators, either a "repair" algorithm would be required in the decoder, slowing down the algorithm, or a cost penalty method would be needed, jeopardizing a main objective of our approach, the need to eliminate weight factors.

3.4 Selection Strategy

The scheme for selection of parents for crossover (line 3 of Fig. 1) should enforce the principle of survival-of-the-fittest. The particular scheme chosen here is in line with the suggestions of [9]. The parents are selected independently of each other, subject only to the requirement that they are distinct. Assume that the current population $\Phi = \{\phi_0, \phi_1, \ldots, \phi_{N-1}\}$ is sorted in ascending order according to rank, i.e., $r(\phi_0) \leq r(\phi_1) \leq \ldots \leq r(\phi_{N-1})$. Each parent is selected by a two step procedure. In the first step the rank of the parent is chosen to be $r(\phi_k)$, where k is determined by

$$k = \frac{N[\beta - \sqrt{\beta^2 - 4(\beta - 1)\chi}]}{2(\beta - 1)} \tag{9}$$

Here χ is a uniform stochastic variable on $[0, 1]$ and β is a user-defined parameter, $1 < \beta \leq 2$. Using (9), which was introduced in [17], the probability of selecting a given individual increases linearly with the index of the individual and the probability of selecting ϕ_0 is β times the probability of selecting the median individual $\phi_{N/2}$. Then, in the second step, the parent is selected uniformly at random among all individuals having rank $r(\phi_k)$.

In the traditional GA, selection is based on a fitness function, which defines the relative quality of each individual by a (non-trivial) transformation of the cost values, cf. Section 3. In contrast, the rank-based selection scheme described eliminates the need for a fitness function.

4 Experimental Results

A performance evaluation based on comparison to an existing approach is unfortunately very difficult to establish for a number of reasons. Firstly, the placement model, which extends to MCMs by assuming routing on top of blocks, is not compatible to the IC models applied by previous building-block approaches. Secondly, there are no building-block benchmarks which includes appropriate timing information. And thirdly, it is inherently difficult to fairly compare the 3-dimensional optimization approach to existing 1-dimensional approaches. Sections 4.1 and 4.2 describe the alternative evaluation approach taken and Section 4.3 presents the results.

4.1 Test Examples

Five circuits were used for testing, cf. Table 1. A1 is a pathological example which we constructed such that the global optimum values for area and delay are known. This facilitates an absolute measure of performance. The 20 blocks of A1 fit together in a rectangular area with no empty space and 17 paths, each of which is a two-pin net, connect neighboring blocks so that only eight points exists in the search space, in which both the globally optimal area and the globally optimal delay are obtained[1].

circuit	type	blocks	block area	nets	pins	paths	p_{max}
A1	IC	20	42.00	17	34	17	1
xeroxT	IC	10	19.35	206	706	115	9
ami33T	IC	33	1.16	110	530	204	32
ami49T	IC	49	35.45	431	1,058	123	48
SPERT	MCM	20	4,311.30	210	750	540	1

Table 1: *Main characteristics of test examples. The columns are : type (IC/MCM), no of blocks, total block area in mm^2, no. of nets, pins and paths. The no. of nets and pins includes only those nets/pins, which are part of a path. p_{max} is the length of the longest path in terms of no. of involved nets.*

xeroxT, ami33T and ami49T are constructed from the building block benchmarks xerox, ami33 and ami49, respectively, by adding the required timing information. The original specifications of these circuits are unaltered and paths are generated in a random fashion, adding a few nets

[1]For a given orientation and reflection of A1, only one global optimum exists, which can then be oriented and reflected in eight distinct ways.

to assure that the longest path in each circuit connects all blocks. The output driver resistances, the internal block delay associated with each source pin and the capacitance of sink pins are assigned randomly according to a normal distribution $N(\mu, \sigma)$ (mean μ and standard deviation σ) for each quantity. For the driver resistances, $(\mu, \sigma) = (100, 20)$ Ω, for the internal block delays, $(\mu, \sigma) = (0.5, 0.2)$ ns, and for the input capacities, $(\mu, \sigma) = (15.3, 3.0)$ fF/μm. These mean values are representative of a 0.8 μm CMOS process [2, 15].

Finally, SPERT is a specification of an MCM consisting of a vector processor (ASIC), 16 SRAMs and 3 buffer components. SPERT is the key component of a dedicated hardware system for neural net based speech recognition, currently being developed at the International Computer Science Institute (ICSI) in Berkeley, California [1].

For all ICs, the routing wire resistance $\bar{r}$ is 0.03 $\Omega/\mu m$ and the capacitance $\bar{c}$ is 0.352 $fF/\mu m$, again typical for an 0.8 μm CMOS process [2]. The minimum block spacing λ is 0, i.e., blocks can be abutted. For the MCM, we assume $\bar{r} = 0.008$ $\Omega/\mu m$, $\bar{c} = 0.06$ $fF/\mu m$, and a minimum spacing of $\lambda = 5.0$ mm.

4.2 Method

The algorithm is implemented in 5,000 lines of C and runs on a DEC station 5000/125. The layouts obtained are compared to those obtained by a random walk in the space $\tilde{\Pi}$ for 10 CPU-hours. This algorithm, denoted RW10, simply generates genotypes at random, evaluates them using the decoder, and stores the best (rank 0) solutions found. Since the cost of previously examined points in a random walk does not affect the future search, RW10 can be considered using the same 3-dimensional cost function as the GA, and hence the two approaches can be compared. Furthermore, the use of RW10 reveals some information on the difficulty of searching in $\tilde{\Pi}$.

For all executions of the GA on all examples, the following fixed set of control parameters were used : Population size $N = 40$, selection bias $\beta = 1.8$, mutation rate $p_{mut} = 0.0005$, diversity frequency $F = 100$ generations. The hillclimber attempts $H = 1,000$ mutations on a given individual, and the search is considered converged if no improvement has been observed for $S = 10,000$ consecutive generations. The time limit is $T = 1$ CPU-hour. For all circuits, $r_{target} = 1.0$ and $g = (0, 0.2, 0)$, i.e., any aspect ratio in the range from 0.8 to 1.2 is satisfactory and both area and delay should be as small as possible.

4.3 Performance

A single execution of the GA was performed for each circuit. For A1 and xeroxT, the algorithm terminated due to convergence after 33 and 42 CPU-minutes, respectively. The 1 hour CPU-limit caused termination for the remaining circuits. The number of decodings performed during optimization varied from 28,838 (SPERT) to 111,287 (A1). In comparison, RW10 performed from about 3×10^5 (SPERT) to 4×10^6 (A1) decodings. For ami33T and SPERT, the GA never executed the hillclimber since Φ_0 was always updated within 10,000 generations.

Figures 4 and 5 compares the solution sets Φ_0 found by the GA and RW10 for ami33T and ami49T, respectively. The tradeoffs found by the GA are significantly better in all dimensions. Similar results were obtained for A1, xeroxT and SPERT, although the distance between the solution sets found by the two algorithms are smaller for these circuits, as expected because of the significantly smaller search spaces.

For each of the circuits A1, xeroxT and SPERT and each of the algorithms, Figures 6, 7 and 8 shows the subsets of the output sets Φ_0, which satisfies the aspect ratio goal. From these sets the designer can choose the preferred tradeoff between area and delay. As can be seen from Figures 6, 7 and 8 the GA always finds the best solutions wrt. both area and delay except in the case of A1. Here RW10 finds a placement with the globally optimal area 42.0 mm^2 (with aspect ratio 6/7, i.e., satisfying the aspect ratio goal), whereas the smallest area found by the GA is 49.0 mm^2. Although the GA obtains the best delay values, none of the algorithms are close to the globally optimal delay of only 8.873 ps.

For each of the five circuits, Figures 9 through 11 shows the smallest area solution found by the GA which satisfies the aspect ratio goal. The empty space, i.e., the ratio of the area not occupied by blocks to the total layout area, varies from 4.2 % for SPERT[2] to 14.3 % for A1. Smaller layouts can easily be obtained by optimizing for area only, i.e., using the goal vector $g = (0, \infty, \infty)$. For example, a sample execution of the GA using this goal vector yielded a placement for SPERT of size 6,628.17 mm^2. However, the cost is an aspect ratio of 12.25 which illustrates the need for considering all objectives simultaneously.

Since the solution set Φ_0 improves monotonically with time, cf. Section 3.1, better results are guaranteed to be obtained by spending more CPU-time, as long as the algorithm does not converge. Fig. 11 (right) shows the minimum area layout obtained for ami49T after 10 CPU-hours, which can be compared to the result after 1 CPU-hour shown on Fig. 11 (left). The minimum area solution has improved with respect to both area and delay while still satisfying the aspect ratio goal.

5 Conclusions and Future Work

A genetic algorithm for building-block placement of ICs and MCMs has been presented, which minimizes area and path delay while attempting to meet a target aspect ratio. The key feature of the approach is its capability to explore the 3-dimensional design space explicitly without expressing cost in terms of weighted sums and/or hard constraints as is done in existing approaches. Instead the designer directly specifies a goal value for each cost dimension, which may or may not be obtainable, and the algorithm outputs a set of good solutions, from which the designer can choose the preferred tradeoff.

This work illustrates that the genetic algorithm is very well suited for this type of multi-dimensional optimization. The search for a set of solutions as opposed to a single solution is conveniently expressible in terms of the genetic algorithm since it already performs optimization by manipulating a solution set. In contrast, it is not clear how to use e.g. simulated annealing for this type of optimization.

The experimental work includes results for a real-world design and shows that the solution sets found represent good, balanced tradeoffs compared to a 10 CPU-hour random walk restricted

[2]For SPERT a minimum distance between blocks of $\lambda = 5$ mm is required. The layout is 4.2 % larger than required to satisfy this spacing, assuming an aspect ratio of 1.

to the same search space. Furthermore, the runtime requirement of 1 CPU-hour or less is very reasonable from a practical point of view.

This work can be continued in numerous ways of which only three possibilities are mentioned here. Currently, I/O pins/pads are ignored for simplicity but should of course be added to the placement model. More work on how to control and maintain diversity among the solutions during the optimization is also needed. And finally, it would be very interesting to add a measure of routing congestion as a fourth independent optimization dimension, which would allow (an estimate of) routing to directly influence the placement.

Acknowledgements

The authors would like to thank James Beck and Brian Kingsbury at the International Computer Science Institute in Berkeley, California, for providing us with their data for the SPERT design. This research was partially supported by SRC grant no. 95-DC-324 and NSF grant no. MIP 91-17328. The first author is supported mainly by the Danish Technical Research Council.

References

[1] K. Asanović, J. Beck, "T0 Engineering Data, Revision 0.14," Technical Report, International Computer Science Institute, 1994.

[2] K. D. Boese, A. B. Kahng, B. A. McCoy, G. Robins, "Fidelity and Near-Optimality of Elmore-Based Routing Constructions," *Proc. of the Intl. Conf. on Computer Design*, pp. 81-84, 1993.

[3] H. Chan, P. Mazumder, 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, Nov. 1991.

[4] J. P. Cohoon, S. U. Hedge, W. N. Martin, D. Richards, "Distributed Genetic Algorithms for the Floorplan Design Problem," *IEEE Transactions on Computer-Aided Design*, Vol. 10, pp. 484-492, April 1991.

[5] P. Dasgupta, P. Mitra, P. P. Chakrabarti, S. C. DeSarkar, "Multiobjective Search in VLSI Design," *Proc. of The 7th International Conference on VLSI Design*, pp. 395-400, 1994.

[6] R. Drechsler, H. Esbensen, B. Becker, "Genetic Algorithms in Computer Aided Design of Integrated Circuits," Technical Report 17/94, Computer Science Department, Johann Wolfgang Goethe University, Frankfurt, Germany, 1994.

[7] W. C. Elmore, "The Transient Response of Damped Linear Network with Particular Regard to Wideband Amplifiers," *J. Applied Physics*, Vol. 19, pp. 55-63, 1948.

[8] H. Esbensen, P. Mazumder, "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*, pp. 211-214, 1994.

[9] C. M. Fonseca, P. J. Fleming, "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization," *Proc. of the Fifth International Conference on Genetic Algorithms*, pp. 416-423, 1993.

[10] T. Gao, P. M. Vaidya, C. L. Liu, "A Performance Driven Macro-Cell Placement Algorithm," *Proc. of the 29th Design Automation Conference*, pp. 147-152, 1992.

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

[12] T. Hamada, C.-K. Cheng, P. M. Chau, "Prime : A Timing-Driven Placement Tool using A Piecewise Linear Resistive Network Approach," *Proc. of the 30th Design Automation Conference*, pp. 531-536, 1993.

[13] J. H. Holland, "Adaption in Natural and Artificial Systems," *University of Michigan Press*, Ann Arbor, MI., 1975.

[14] M. A. B. Jackson, E. S. Kuh, "Performance-Driven Placement of Cell Based IC's," *Proc. of the 26th Design Automation Conference*, pp. 370-375, 1989.

[15] A. Srinivasan, K. Chaudhary, E. S. Kuh, "RITUAL : A Performance-Driven Placement Algorithm," *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, Vol. 39, pp. 825-840, Nov. 1992.

[16] L. Stockmeyer, "Optimal Orientations of Cells in Slicing Floorplan Designs," *Information and Control*, Vol. 57, pp. 91-101, 1983.

[17] D. Whitley, "The Genitor Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best," *Proc. of the Third International Conference on Genetic Algorithms*, pp. 116-121, 1989.

[18] X.-M. Xiong, E. S. Kuh, "Geometric Approach to VLSI Layout Compaction," *International Journal of Circuit Theory and Applications*, Vol. 18, pp. 411-430, 1990.

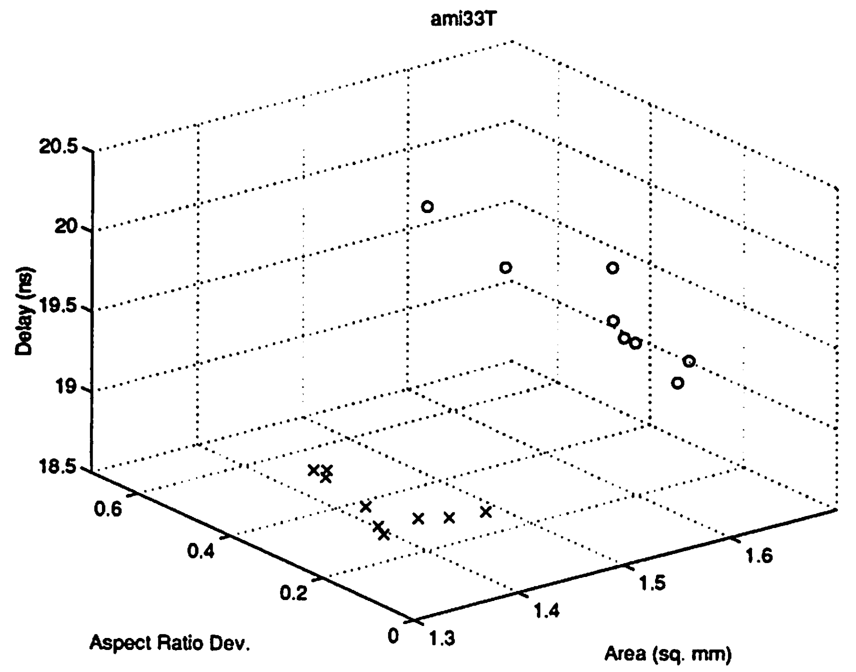


Figure 4: *Comparison of the solution sets Φ_0 found by RW10 and the GA for ami33T. The* o*'s are RW10 solutions and the* $\times$*'s are GA solutions. The three dimensions (x,y,z) correspond to the three cost dimensions $c = (c_a, c_r, c_d)$, respectively. Although it is not clear from this figure, three of the nine solutions found by the GA satisfies the aspect ratio goal of 0.2.*

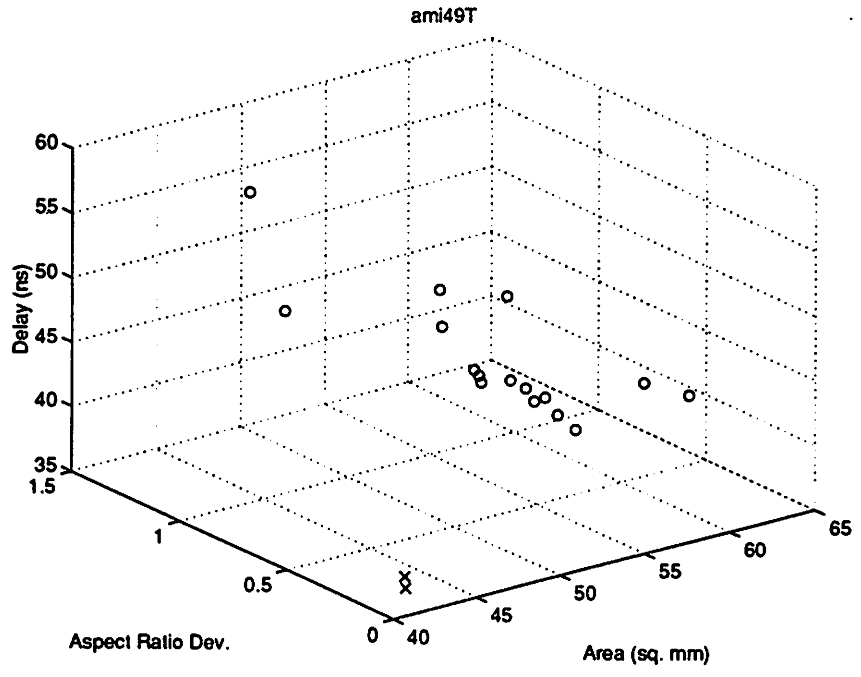


Figure 5: *Comparison of the solution sets Φ_0 found by RW10 and the GA for ami49T.*

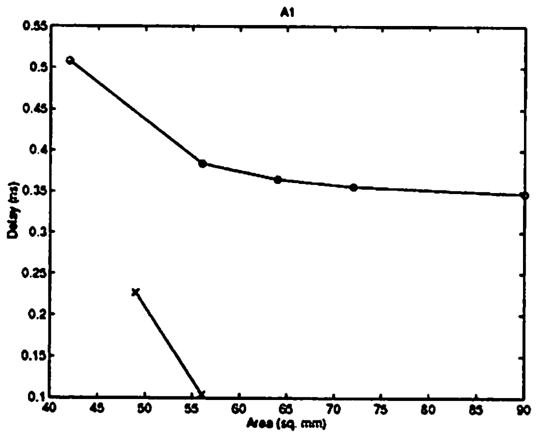


Figure 6: *Found area/delay tradeoffs for A1, all of which satisfies the aspect ratio goal. The* o*'s are RW10 solutions and* × *'s are GA solutions.*

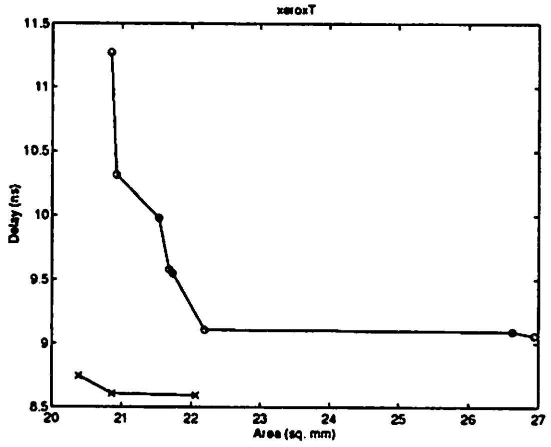


Figure 7: *Found area/delay tradeoffs for xeroxT, all of which satisfies the aspect ratio goal.*

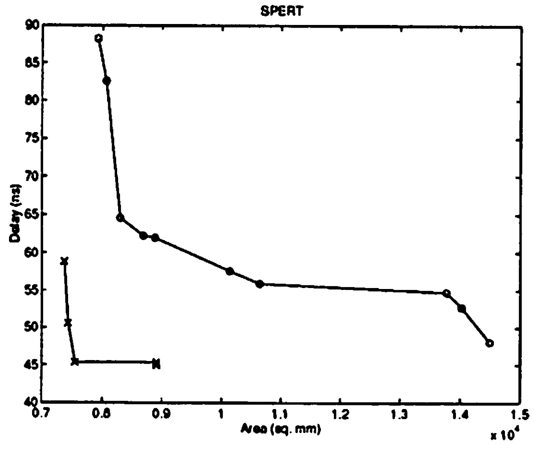
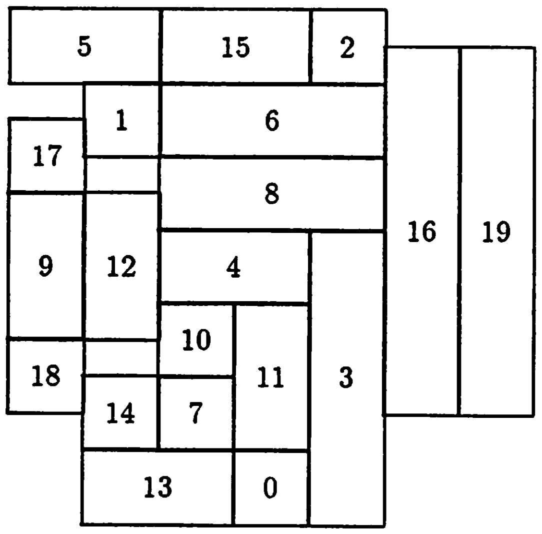


Figure 8: *Found area/delay tradeoffs for SPERT, all of which satisfies the aspect ratio goal.*



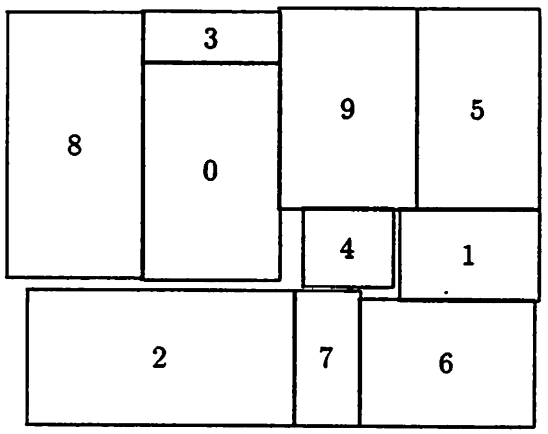
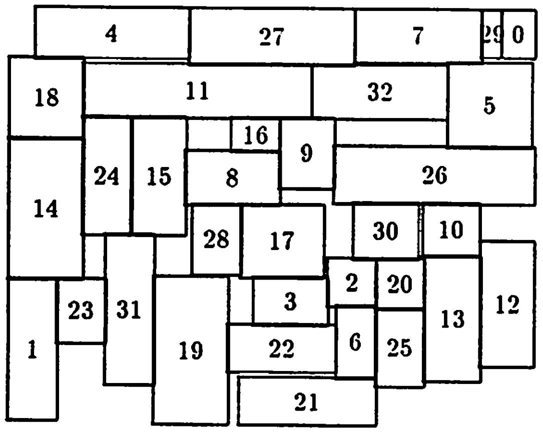


Figure 9: *Left: A1, area = 49.00 mm^2 (empty space 14.3 %), r_{actual} = 1.00, delay = 0.23 ns. Right: xeroxT, area = 20.38 mm^2 (empty space 5.1 %), r_{actual} = 0.80, delay = 8.74 ns.*



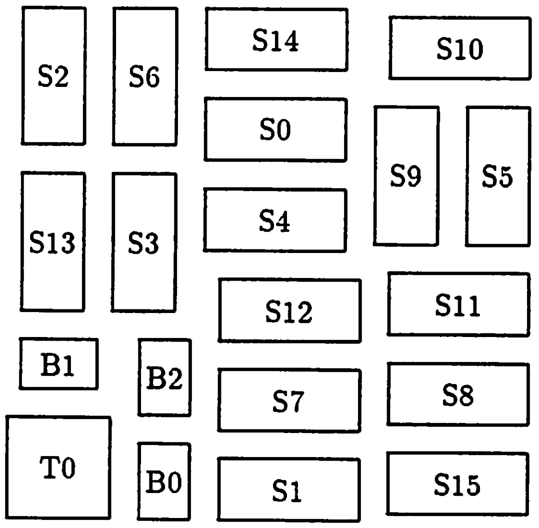


Figure 10: *Left: ami33T, area = 1.35 mm^2 (empty space 14.1 %), r_{actual} = 0.81, delay = 18.75 ns. Right: SPERT. T0 is the vector processor, the S-blocks are the SRAMS and the B-blocks are the buffers. Area = 7377.09 mm^2, r_{actual} = 1.00, delay = 58.84 ns. Excess area, apart from the min. spacing required, is 4.2 % (at aspect ratio 1).*

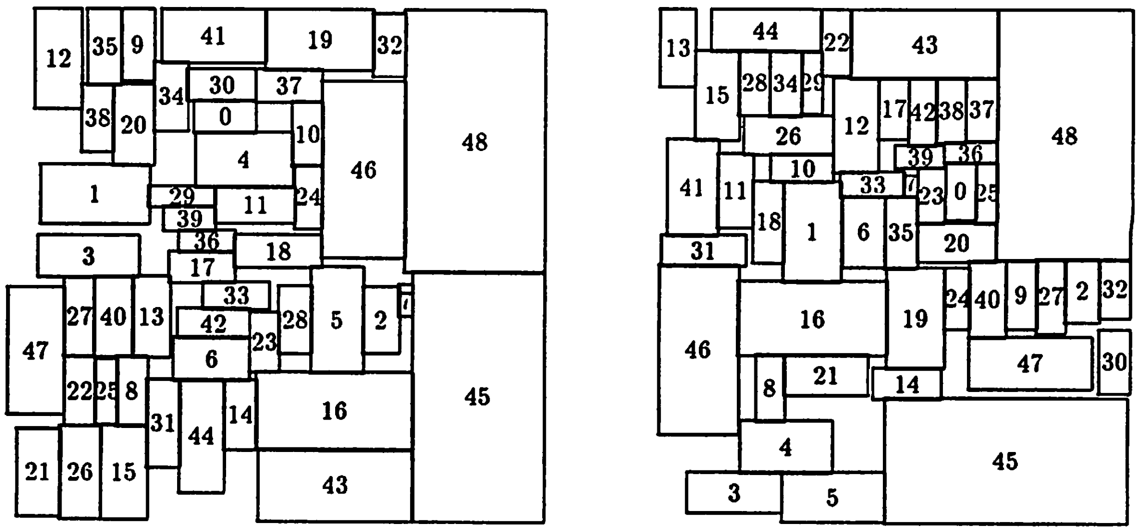


Figure 11: *Left: ami49T (1 CPU-hour), area = 41.02 mm² (empty space 13.6 %), r_{actual} = 0.97, delay = 37.77 ns. Right: ami49T after 10 CPU-hours, area = 39.85 mm² (empty space 11.0 %), r_{actual} = 1.11, delay = 33.31 ns.*