AN INTERACTIVE FLOORPLANNER FOR DESIGN SPACE EXPLORATION

by

Henrik Esbensen and Ernest S. Kuh

Memorandum No. UCB/ERL M96/4

1 February 1996

AN INTERACTIVE FLOORPLANNER FOR DESIGN SPACE EXPLORATION

by

Henrik Esbensen and Ernest S. Kuh

Memorandum No. UCB/ERL M96/4

1 February 1996

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

An interactive floorplanner based on the genetic algorithm is presented. Layout area, aspect ratio, routing congestion and maximum path delay is optimized simultaneously. The design requirements are refined interactively as knowledge of the obtainable cost tradeoffs is gained and a set of feasible solutions representing alternative, good tradeoffs is generated. Experimental results illustrates the special features of the approach.

1 Introduction

When determining the floorplan of an integrated circuit (IC) the objective is to find a solution which is satisfactory with respect to a number of competing criteria. Most often specific constraints has to be met for some criteria, while for others, a good tradeoff is wanted. The approach taken by virtually all existing floorplanning and placement tools is to minimize a weighted sum of some criteria subject to constraints on others. I.e., 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. A main reason for the predominant use of the formulation (1) is its convenience from an algorithmic point of view.

However, at the floorplanning stage of the design process, the expected values of the cost criteria are based on relatively rough estimations only. Furthermore, the available information on obtainable tradeoffs, e.g. the relationship between area and delay, is very limited or non-existent. Since the notion of a "good" solution inherently depends on which tradeoffs are actually obtainable, the designers notion of the overall design objective is rarely clearly definable. Consequently, it may in practice be very difficult for the designer to specify a set of weight and bound values that makes a tool based on the formulation (1) find a satisfactory solution.

Even when assuming that the designer has a clear notion of the overall design objective, the use of (1) causes serious difficulties : If the specified bounds are too loose, perhaps a better solution could have been found, while if they are too tight, a solution may not be found at all. Furthermore, the minimum of a weighted sum can never correspond to a non-convex point of the cost tradeoff surface, regardless of the weights [5]. In other words, if the designers notion of the "best" solution corresponds to a non-convex point, it can *never* be found by minimizing c in (1), even though the solution is non-dominated.

Our work is motivated by the need to overcome these fundamental problems. A floorplanning tool called Explorer is presented, which performs *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. Explorer simultaneously minimizes layout area, deviation from a target aspect ratio, routing congestion and the maximum path delay. Guided interactively by the user, Explorer searches for a set of alternative, good solutions. The notion of a "good" solution is gradually refined by the user as the optimization process progresses and knowledge of obtainable tradeoffs is gained. Consequently, no a priori knowledge of obtainable values is required. From the output solution set, the user ultimately chooses a specific solution representing the preferred tradeoff. Since the use of (1) is abandoned the above mentioned problems concerning weight and bound specification are eliminated.

Explorer has three additional significant characteristics:

- The maximum routing congestion is minimized, thereby improving the likelihood that the generated floorplans are routable without further modification.
- Despite the fact that delay is inherently path oriented, most timing-driven floorplanning and placement approaches are net-based and therefore usually over-constrains the problem [7]. One of the few existing path-based placement approaches is [10], which, however, relies on a very simple net model. Explorer obtains a more accurate path delay estimate by approximating each net by an Elmore-optimized Steiner tree.
- Explorer is based on the genetic algorithm (GA), since it is particularly well suited for (interactive) design space exploration [6]. We are only aware of one previous GA-based approach to floorplanning [2] which, however, does not consider delay or routing congestion or explores the design space.

The work presented in this paper is based on several significant extensions and improvements of the placement approach described in [3].

2 Problem Formulation

Section 2.1 presents our definition of the floorplanning problem while Section 2.2 describes the specification of a "good" solution.

2.1 The Floorplanning Problem

The floorplanning problem is specified by the following input :

- A set of *blocks*, each of which has $k \geq 1$ alternative *implementations*. Each implementation is rectangular and either *fixed* or *flexible*. For a fixed implementation, the dimensions, area and exact pin locations are known. For a flexible implementation, the area is given but the aspect ratio and exact pin locations are unknown. The capacitance of each sink pin and the driver resistance and internal delay $t(p)$ for each source pin p is given for all pins. $t(p)$ is the time it takes a signal to travel through $m(p)$, the implementation to which p belongs, and reach p.
- A specification of all nets and a set of paths $\mathcal{P}$. A path 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 $m(p_i) = m(s(p_{i+1}))$ for $i = 0, 1, \ldots, l-2$.
- Technology information : Number of metal layers available for routing on top of blocks and between blocks, denoted by l_{block} and l_{space}, respectively. The routing wire resistance $\bar{r}$ and capacitance $\bar{c}$ per unit wire length, and the wire pitch w_{pitch}.

Each output solution is a specification of the following :

- A selected implementation for each block.
- For each selected flexible implementation i, its dimensions w_i and h_i such that $w_i h_i = A_i$ and $l_i \leq h_i/w_i \leq u_i$, where A_i is the given area of implementation i and l_i and u_i are given bounds on the aspect ratio of i, which is assumed to be continuous.
- An absolute position of each block so that no pair of 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. Since multi-layer designs are considered, it is assumed that a significant part of the routing is performed on top of the blocks.
- An orientation and reflection of each block. Throughout this paper, the term *orientation* of a block refers to a possible 90 degree rotation, while *reflection* of a block refers to the possibility of mirroring the block around a horizontal and/or a vertical axis. Each block can be oriented and/or reflected in a total of eight distinct ways.

This problem formulation is quite general and also applies to multi-chip modules (MCMs). IO-pins/pads are also handled by Explorer, but for brevity this aspect is not discussed.

2.2 What is a "Good" Tradeoff ?

Let Π be the set of all floorplans and $\Re_+ = [0, \infty[$. The cost of a solution is defined by the vector-valued function $c : \Pi \mapsto \Re_+^4$, $c(x) = (c(x)_1, c(x)_2, c(x)_3, c(x)_4)$, which will be described in Section 3.2. This Section describes how to specify what a "good" cost tradeoff is, and how to compare the cost of two solutions without resorting to a single-valued cost measure.

Instead of weights and/or bounds, the user specifies preferences by defining a *goal and feasibility vector pair* $(g, f) \in G$, where $G = \{(g, f) \in \Re_{+\infty}^n \times \Re_{+\infty}^n \mid \forall i : g_i \leq f_i\}$ and $\Re_{+\infty} = [0, \infty]$. For the i'th criterion, g_i is the maximum value wanted, if obtainable, while f_i specifies a limit beyond which solutions are of no interest. For example, the 3'rd criterion minimized by Explorer is path delay. $g_3 = 5$ and $f_3 = 18$ states that a delay of 5 or less is wanted, if it can be obtained, while a delay exceeding 18 is unacceptable. A delay between 5 and 18 is acceptable, although not as good as hoped for.

For $(g, f) \in G$, let $S_g = \{x \in \Pi \mid \forall i : c(x)_i \leq g_i\}$ and $A_f = \{x \in \Pi \mid \forall i : c(x)_i \leq f_i\}$ be the set of *satisfactory* and *acceptable* solutions, respectively, cf. Fig. 1. $S_g \subseteq A_f \subseteq \Pi$, i.e., a satisfactory solution is also acceptable. The values specified by (g, f) are merely used to guide the search process and in contrast to traditional, user-specified bounds, need *not* be obtainable. Furthermore, as will be discussed in detail in Section 3.3, (g, f) are defined interactively at runtime. Therefore, the specification of the (g, f) vectors do not cause any of the practical problems caused by traditional weights and bounds, cf. Section 1.

In order for the algorithm to compare solutions, a notion of relative solution quality is needed, which takes the goal and feasibility vectors into account. Let $x, y \in \Pi$. The relation x *dominates* y, written $x <_d y$, is defined by

$$x <_d y \quad \Leftrightarrow \quad (\forall\, i : c(x)_i \leq c(y)_i) \wedge (\exists\, i : c(x)_i < c(y)_i) \tag{2}$$

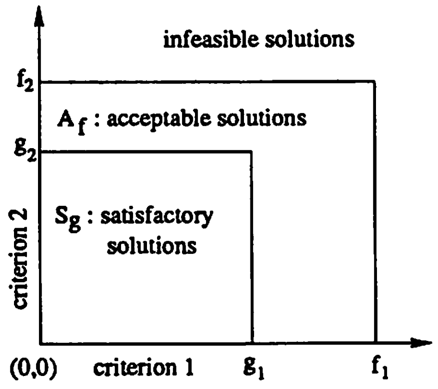

Figure 1: *The sets of satisfactory and acceptable solutions, illustrated in two dimensions.*

For a given $(g, f) \in G$ the relation x *is preferable to* y, written $x \prec y$, is then defined as follows, depending on how $c(x)$ compares to g : If x satisfy all goals, i.e., $x \in S_g$, then

$$x \prec y \quad \Leftrightarrow \quad (x <_d y) \vee (y \notin S_g) \tag{3}$$

If x satisfies none of the goals, i.e., $\forall\, i : c(x)_i > g_i$ then

$$x \prec y \quad \Leftrightarrow \quad (x <_d y) \vee [(x \in A_f) \wedge (y \notin A_f)] \tag{4}$$

Finally, x may satisfy some but not all goals. Assuming a convenient ordering of the optimization criteria, $\exists\, k \in \{2,3,4\} : (\forall\, i < k : c(x)_i \leq g_i) \wedge (\forall\, i \geq k : c(x)_i > g_i)$. Then

$$\begin{aligned}
x \prec y \quad \Leftrightarrow \quad & [(\forall\, i \geq k : c(x)_i \leq c(y)_i) \wedge (\exists\, i \geq k : c(x)_i < c(y)_i)] & (5)\\
& \vee & \\
& [(x \in A_f) \wedge (y \notin A_f)] & (6)\\
& \vee & \\
& [(\forall\, i \geq k : c(x)_i = c(y)_i) \wedge & (7)\\
& \{((\forall\, i < k : c(x)_i \leq c(y)_i) \wedge (\exists\, i < k : c(x)_i < c(y)_i)) \vee (\exists\, i < k : c(y)_i > g_i)\}] & (8)
\end{aligned}$$

This definition of $\prec$ assures that a satisfactory solution is always preferable to a non-satisfactory solution and an acceptable solution is always preferable to an unacceptable solution. Furthermore, from (5) it follows 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. Hence, when goals are satisfied, they are "factored out", focusing the search on the remaining, unsatisfactory dimensions.

The above definition of $\prec$ is introduced in [4] and extends the definition first introduced in [6] by adding the feasibility vector f and the concept of acceptable solutions. The purpose of f is described in Section 3.3.

Using $\prec$ the solutions of a given set Φ can be ranked. Let $r : \Pi \times 2^{\Pi} \mapsto N$ be defined by $r(\phi, \Phi) = |\{\gamma \in \Phi | \gamma \prec \phi\}|$. $r(\phi, \Phi)$ is the *rank* of ϕ with respect to Φ, i.e., the number of solutions in Φ which are preferable to ϕ. Furthermore, let $\Phi_0 = \{\phi \in \Phi | r(\phi, \Phi) = 0\} \subseteq \Phi$, i.e., Φ_0 is the subset of best solutions in Φ with respect to $\prec$. Explorer outputs a set of distinct rank zero solutions Φ_0, i.e., the best found cost tradeoffs. As a special case, if $g = (0,0,0,0)$ and $f = (\infty, \infty, \infty, \infty)$ the algorithm searches for (a sample of) the Pareto-optimal set.

3 Description of Explorer

The concept of genetic algorithms is based on natural evolution [8]. 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. 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.

Rather than altering given solutions directly, the crossover and mutation operators process internal *representations* of solutions. The solution corresponding to a given representation is computed by a function known as the *decoder*. The key issue when developing a GA is the design of a suitable representation of a solution. Section 3.1 outlines the GA used in Explorer, and Section 3.2 present the floorplan representation and the decoder. Finally, Section 3.3 describes how the user controls the optimization process interactively.

3.1 Overview of Algorithm

The specific GA used in Explorer is outlined in Fig. 2. The population $\Phi = \{\phi_0, \phi_1, \ldots, \phi_{N-1}\}$ is initially constructed by routine *generate* (line 1) from random individuals. One iteration of the repeat loop (lines 2-12) corresponds to the simulation of one generation.

In each generation, two parent individuals ϕ_1 and ϕ_2 are selected for crossover (line 3). Each parent is selected at random with a probability inversely proportional to its rank, thereby enforcing the principle of survival-of-the-fittest. The crossover operator generates the offspring ψ (line 4), which is then subjected to random changes by routine *mutate* (line 5) and inserted into Φ by routine *insert* (line 6). An inserted individual ψ replaces a maximum rank solution to which it is preferable or, if ψ is not preferable to any solution, it replaces a maximum rank solution, which is not preferable to ψ. The insertion scheme ensures that a solution ψ can never replace ϕ if $\phi \prec \psi$. Hence, in the sense inferred by $\prec$ the set of best solutions Φ_0 is monotonically improving as a function of time.

There is four types of interaction through the graphical user interface, gui (lines 7, 9, 11, 12). By selecting an update operation (line 7) the user can alter the values of the goal and feasibility vectors (g, f) (line 8). If choosing an optimization operation (line 9) a hillclimber will be executed on a selected individual ϕ (line 10). The current state of the optimization process is continuously displayed to the user (line 11). The update, optimize and display operations are the topic of Section 3.3. The optimization process continues until the user selects termination (line 12), at which time Φ_0 is the output set of solutions (line 13).

```
01  generate($\Phi$);
02  repeat :
03      select $\phi_1, \phi_2 \in \Phi$;
04      crossover($\phi_1, \phi_2, \psi$);
05      mutate($\psi$);
06      insert($\Phi, \psi$);
07      if gui(update) :
08          adjust ($g, f$);
09      if gui(optimize) :
10          hillclimber($\phi$, $k$, $(g', f')$);
11      gui(display);
12  until gui(terminate);
13  output $\Phi_0$;
```

Figure 2: *Outline of the algorithm.*

3.2 Representation and Decoder

The representation of a floorplan having b blocks consist of five components a) through e) :

a) A string of b integers specifying the selected implementations of all blocks. The i'th integer k_i identifies the implementation selected for block i and satisfies $0 \leq k_i \leq b_i - 1$ where $b_i \geq 1$ is the number of existing implementations of the i'th block.

b) A string of real values specifying aspect ratios of selected flexible implementations. The i'th value r_i specifies the aspect ratio of the i'th selected flexible implementation and satisfies $l_i \leq r_i \leq u_i$, cf. Section 2.1.

c) An inverse Polish expression of length $2b - 1$ over the alphabet $\{0, 1, \ldots, b-1, +, *\}$. The operands $0, 1, \ldots, b-1$ denotes block identities and $+, *$ are operators. The expression uniquely specifies a slicing-tree for the floorplan, as first introduced in [12], with $+$ and $*$ denoting a horizontal and a vertical slice, respectively.

d) A bitstring of length $2b$ representing the reflection of all blocks. The reflection of the i'th block is specified by bits $2i$ and $2i + 1$.

e) A string of integers specifying a critical sink for each net, to be used when routing the nets. The i'th integer c_i identifies the critical sink of net i and satisfies $0 \leq c_i \leq s_i - 1$ where $s_i \geq 1$ is the number of sinks of net i.

Given a representation of the above form, the decoder computes the corresponding floorplan and its cost $c = (c_{area}, c_{ratio}, c_{delay}, c_{cong})$ in eight steps as follows :

1. The implementation selected for each block is specified directly by component a) of the representation. The dimensions of each selected flexible block is computed from its aspect ratio, specified by component b), and its fixed area. The dimensions of all blocks are then known.

2. From the slicing-tree specified by the Polish expression, component c), the orientation of each block is determined such that layout area is minimized. The orientations are computed using an exact algorithm by Stockmeyer [11] which guarantees a minimum area layout for the given slicing-structure. The reflection of each block is as specified by component d) of the representation.

3. 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.

4. The layout is compacted, first vertically and then horizontally, using a simple one-dimensional compaction algorithm. Then the area and aspect ratio of the layout can be computed. c_{area} is the area of the smallest rectangle enclosing all blocks and $c_{ratio} = |r_{actual} - r_{target}|$, where r_{actual} is the actual aspect ratio of the layout and r_{target} is a user-defined target aspect ratio. Fig. 3 illustrates the first four steps of the decoding process.

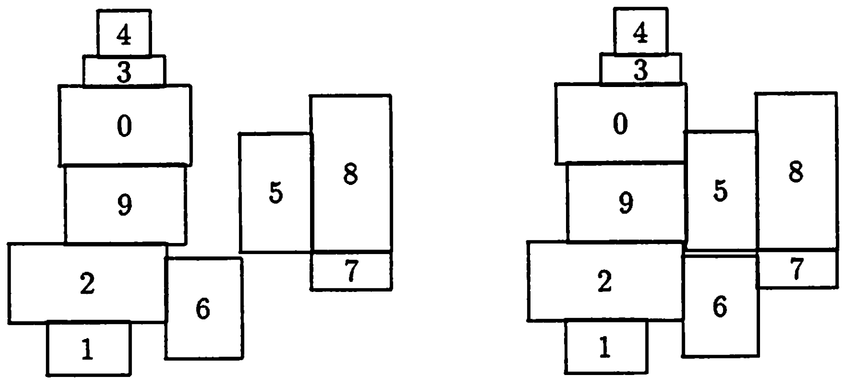

Figure 3: *Given 10 blocks and the Polish expression 1 2 + 6 * 9 0 + + 3 4 + + 5 * 7 8 + *, the floorplan on the left is the result of step 3 of the decoding. Subtrees are recursively centered and oriented optimally. Since the height of the layout is determined by the blocks 1,2,9,0,3,4, no blocks are 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 floorplan to the right is the result of compaction (step 4), i.e., the final floorplan.*

5. A global routing graph $G = (V, E)$ is constructed which forms a two-dimensional, uniformly spaced lattice, exactly covering the entire layout. The pitch of the lattice is determined by the user-defined parameter g_{pitch}. Each pin is then assigned to the closest vertex in V. When computing this assignment, the exact pin locations are used for pins of fixed implementations, while for flexible implementations, all pins are assumed to be located at the center of the block.

6. The topology of each net is approximated by a Steiner tree embedded in G. Each Steiner tree is computed independently by the SERT-C algorithm ("Steiner Elmore Routing Tree with identified Critical sink") introduced in [1]. SERT-C minimizes the Elmore delay from the source to a designated critical sink, illustrated in Fig. 4. For each net the critical sink is specified by component e) of the representation. Since all pins were mapped to V in the previous step all trees are embedded in G.

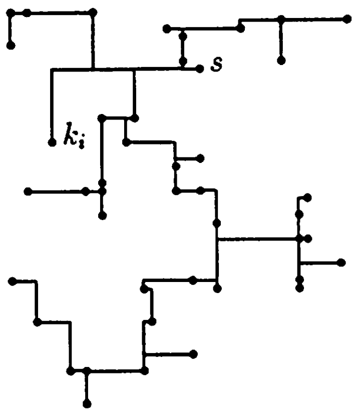

Figure 4: *The Elmore-optimized Steiner tree computed by SERT-C for a 44-pin net. s is the source and k_i the selected critical sink.*

7. The delay $D(P)$ of each path $P = \{p_0, p_1, \ldots, p_{l-1}\}$ is estimated as

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

where $d_E(p_i)$ is the Elmore delay from $s(p_i)$ to p_i computed in the Elmore-optimized Steiner tree. The delay cost c_{delay} is the maximum path delay, i.e., $c_{delay} = \max_{P \in \mathcal{P}}\{D(P)\}$.

8. Finally, the maximum routing congestion is estimated. For each edge $e \in E$, cap(e) denotes the capacity of e and equals $g_{pitch} \times l_{block}/(2 \times w_{pitch})$ if (a part of) e is on top of a block, and $g_{pitch} \times l_{space}/(2 \times w_{pitch})$, otherwise. The division by 2 reflects the assumption that in each layer the routing will mainly be either horizontal or vertical. usage(e) is the number of nets using e. The congestion cost c_{cong} is computed as

$$c_{cong} = 100 \times \max \left[\max_{e \in E} \left\{ \frac{\text{usage}(e) - \text{cap}(e)}{\text{cap}(e)} \right\} , 0 \right]$$

   i.e., c_{cong} is the maximum percentage by which an edge capacity has been exceeded.

Although the nets are routed independently in step 6, the resulting Steiner trees represents a very accurate estimation of the net topologies compared to the estimations of previous approaches, cf. Section 1, which in turn indicates that c_{delay} is an accurate estimate. The Steiner trees also allow the computation of c_{cong}. The smaller c_{cong} is, the fewer nets needs to be rerouted to obtain 100% global routing completion and, by assumption, the easier is the global routing task.

The crossover operator as well as the mutation operator (lines 4 and 5 of Fig. 2) operates on each of the five components of the representation independently. While the Polish expressions are handled by highly specialized operators introduced in [2], the remaining components are handled by standard operators (uniform crossover and pointwise mutation) extensively studied in the GA literature. For brevity, the reader is referred to e.g. [8] for a description of these operations. A crucial property of both the crossover and the mutation operators is that they preserve feasibility, i.e., only feasible representations, which can be interpreted by the decoder, are ever generated.

3.3 Interactive Control of the Optimization Process

In order for the user to interactively control the design space exploration process, Explorer provides continuously updated information on the current state of the optimization (line 11 of Fig. 2). The information is visualized in the form of graphs showing the cost tradeoffs of the solutions obtained so far. Up to four independent graphs can be shown simultaneously, and each graph can be 2- or 3-dimensional with any of the four cost criteria on any axis. The solution set displayed in each graph can be either the entire current population Φ, the current acceptable solutions $\Phi \cap A_f$, the current satisfactory solutions $\Phi \cap S_g$ or the current best solutions Φ_0. An example graph is shown in Fig. 5. Based on this information the user can alter the optimization process at any time by either adjusting parameters (lines 7 and 8 of Fig. 2) or executing a hillclimber (lines 9 and 10 of Fig. 2) as will be described in the following.

As the optimization progresses and knowledge of obtainable cost tradeoffs is gained, the user can adjust the values of the goal and feasibility vectors (g, f), thereby re-defining the notion(s) of satisfactory and/or acceptable solutions. When doing so, the ranking of all solutions will be updated, which in turn affects the selection for crossover, i.e., the sampling of the search space, cf. Section 3.1. Consequently, when re-defining (g, f), the focus of the exploration process will change accordingly, allowing the user to "zoom in" on the desired region of the search space. At any time, g specifies which solutions are considered ideal and S_g serves as an attractor towards which the algorithm will try to move the solutions. On the other hand, f specifies which solutions are *not* wanted, and the algorithm will try to move the solutions out of the infeasible region $\Pi \backslash A_f$. In this sense, g is a "carrot" and f is a "stick" by which the user steers the exploration of the search space.

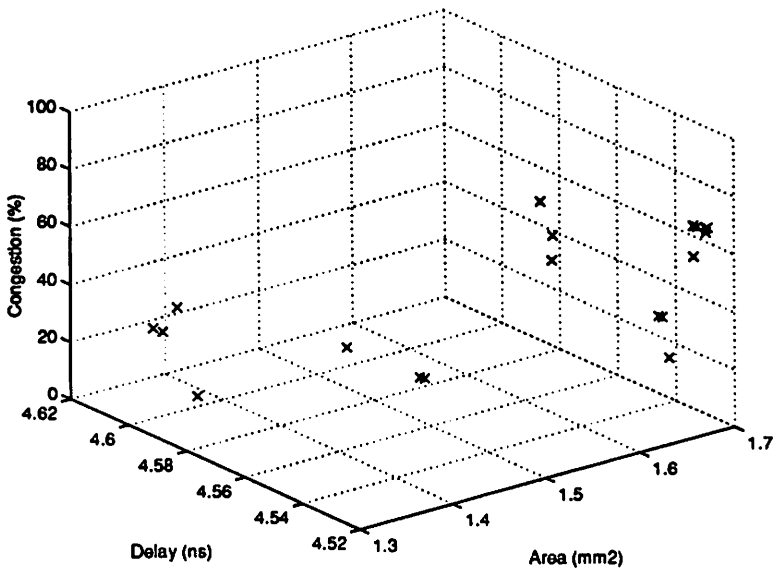

Figure 5: *An example graph showing 3 dimensions of a set of current best solutions. All solutions are non-dominated in the 4-dimensional cost space.*

The user can also execute a hillclimber on a specified individual ϕ (line 10 of Fig. 2). The hillclimber simply tries a sequence of k mutations on ϕ. Each mutation yielding ϕ' from ϕ is performed if and only if $\phi' \prec \phi$. The hillclimber also takes a goal and feasibility vector pair (g', f') as argument, which defines the preference relation $\prec$ to use when deciding which mutations to actually perform. This allows hillclimbing to be direction-oriented in the cost space. As an example, (g', f') can be defined so that ϕ is optimized wrt. delay only, allowing e.g. a 10 % increase in area while keeping ϕ's cost values in the two remaining dimensions unchanged or improved. Direction-oriented hillclimbing is another way of directing the focus of the exploration process towards the region of interest.

4 Experimental Results

Evaluating performance by comparison to an existing approach is complicated by a number of factors. Firstly, the assumption that routing is done mainly on top of the blocks is not compatible to earlier channel-based floorplan models applied by previous approaches. Secondly, there are no floorplanning benchmarks which includes appropriate timing information. Thirdly, while Explorer is designed for interactive use, most existing approaches are non-interactive, complicating the issue of runtime comparison. But the most significant obstacle is the inherent difficulty of fairly comparing a 4-dimensional optimization approach generating a solution *set* to existing 1-dimensional approaches generating a *single* solution. However, using the examples described in Section 4.1, comparisons to simulated annealing and random search have been established as described in Section 4.2. Results are presented in Sections 4.3 and 4.4.

4.1 Test Examples

The characteristics of five of the circuits used for evaluation are given in Table 1. xeroxF, hpF, ami33F and ami49F are constructed from the CBL/NCSU building-block benchmarks xerox, hp, ami33 and ami49, respectively, aiming at minimal alterations of the original specifications. All blocks are defined as flexible by fixing the area of each block to its original value but allowing the aspect ratio to vary in $[r - 0.5; r + 0.5]$, where r is the aspect ratio of the original block. Furthermore, the required timing information is added. Paths are generated in a random fashion and internal block delays, output driver resistances and input capacities are assigned randomly assuming normal distributions and using mean values from [1, 10], representative of a 0.8 μm CMOS process.

Circuit	Type	Blocks	Pins	Nets	Paths
xeroxF	IC	10	698	203	86
hpF	IC	11	309	83	88
ami33F	IC	33	522	123	230
ami49F	IC	49	953	408	116
spertF	MCM	20	1,168	248	574

Table 1: *Main characteristics of test examples.*

spertF is an MCM consisting of a vector processor (ASIC), 16 SRAMs and 3 buffer components. It is the key component of a dedicated hardware system for speech recognition based on neural networks, currently being designed at the International Computer Science Institute in Berkeley, California. Again, all blocks are defined as flexible for the purpose of this evaluation.

4.2 Method

Explorer is implemented in C, except for the graphical user interface, which is implemented in Matlab. The source code is approximately 11,000 lines of C and 1,000 lines of Matlab scripts. All experiments are performed on a DEC 5000/125 workstation. Performance is compared to that of a simulated annealing algorithm, denoted SA, and a random walk, denoted RW. Both algorithms uses the same floorplan representation and decoder as Explorer. The RW simply generates representations at random, decodes them and stores the best solutions ever found (in the $\prec$ sense). The SA generates moves using the mutation operator of Explorer and the cooling schedule is implemented following [9].

The use of the same representation and decoder by all three algorithms means that the effects of the representation and the decoder algorithms, e.g., Stockmeyers algorithm and the compactor, does not a priori give an advantage to any of the algorithms. Furthermore, all algorithms

The results are shown in Table 2. The set quality values are obtained by evaluating the quality of each output solution set using the set quality measure introduced in [4]. A smaller value means a higher quality. This measure appropriately accounts for the (g, f) values specified. As can be seen from the Table, there is no simple relation between the size of an output set and its quality. A set consisting of a single solution only is better than any other set if the single solution is preferable to all other solutions obtained.

	Output set size			Set quality		
Circuit	interact	non-interact	RW	interact	non-interact	RW
xeroxF	40	39.5 (1.6)	49.3 (10.1)	0.572	0.741 (0.073)	0.888 (0.045)
hpF	40	39.6 (1.0)	59.1 (14.5)	0.605	0.638 (0.033)	0.822 (0.029)
ami33F	21	34.0 (11.1)	9.7 (3.9)	0.690	0.759 (0.058)	1.152 (0.048)
ami49F	21	36.2 (4.2)	11.4 (4.5)	0.641	0.676 (0.093)	1.197 (0.052)
spertF	10	39.7 (0.7)	57.2 (17.0)	0.096	1.886 (0.640)	2.178 (0.010)

Table 2: *Performance comparison of the interactive ('interact') and non-interactive ('non-interact') modes of Explorer and the RW. Each entry for RW and the non-interactive mode of Explorer is the average value obtained for the 10 runs and the value in brackets is the corresponding standard deviation. For Explorer, the output set size is limited to 40, which is the population size used.*

The output sets obtained by Explorer in 1 hour are always significantly better than those obtained by RW in 5 hours. But more interestingly, all of the five sample executions of Explorer in interactive mode yields better results than the average non-interactive execution. Furthermore, the number of decodings performed in interactive mode averages only about 78 % of that of the non-interactive mode because of the idling processor during user-interaction. Hence, using Explorer interactively significantly improves the efficiency of the search process, yielding a better result quality while performing less computational work.

This performance gain is especially significant for the spertF layout. Feasible solutions (i.e., solutions in A_f) were obtained interactively by executing direction-oriented hillclimbing on solutions outside but very close to A_f, resulting in a very good solution set quality. In contrast, only one of the 10 sets generated in non-interactive mode contained feasible solutions, which is the reason for the very large difference in set quality values and the large standard deviation for the non-interactive runs.

5 Conclusions

An interactive floorplanner based on the genetic algorithm has been presented, which minimizes area, path delay and routing congestion while attempting to meet a target aspect ratio. The key feature is the explicit design space exploration performed, which results in the generation of a solution *set* representing good, alternative cost tradeoffs.

The inherent problem of existing approaches wrt. specification of suitable weights and bounds is solved by eliminating these quantities, and another practical problem, the traditionally required iterations of floorplanning/placement and global routing is significantly reduced by explicitly minimizing routing congestion.

The experimental work includes results for a real-world design. It is shown that the efficiency of the search process is comparable to that of simulated annealing when a comparison is possible. The required runtime ranging from less than one minute to one CPU-hour is very reasonable from a practical point of view. Furthermore, the mechanisms provided for user-interaction are observed to improve the search efficiency significantly compared to non-interactive executions. For these reasons it is concluded that the presented floorplannning tool is of significant practical value.

References

[1] K. D. Boese, A. B. Kahng, G. Robins, "High-Performance Routing Trees With Identified Critical Sinks," *Proc. of the 30th Design Automation Conference*, pp. 182-187, 1993.

[2] 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.

[3] H. Esbensen, E. S. Kuh, "An MCM/IC Timing-Driven Placement Algorithm Featuring Explicit Design Space Exploration," *Proc. of the IEEE Multi-Chip Module Conference,* 1996 (to appear).

[4] H. Esbensen, E. S. Kuh, "Design Space Exploration Using the Genetic Algorithm," *Proc. of the IEEE International Symposium on Circuits and Systems,* 1996 (to appear).

[5] P. J. Fleming, A. P. Pashkevich, "Computer Aided Control System Design Using a Multi-objective Optimization Approach," *Proc. of the IEE Control '85 Conference,* pp. 174-179, 1985.

[6] 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.

[7] 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.

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

[9] M. D. Huang, F. Romeo, A. Sangiovanni-Vincentelli, "An Efficient General Cooling Schedule for Simulated Annealing," *Proc. of the 1986 International Conference on Computer-Aided Design*, pp. 381-384, 1986.

[10] 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.

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

[12] D. F. Wong, C. L. Liu, "A new algorithm for floorplan design," *Proc. of the 23rd Design Automation Conference*, pp. 101-107, 1986.