EXPLORER : An Interactive Floorplanner for Design Space Exploration

Henrik Esbensen*
Avant! Corporation
1208 East Arques Avenue
Sunnyvale, CA 94086, USA
esbensen@avanticorp.com

Ernest S. Kuh
Department of EECS
University of California
Berkeley, CA 94720, USA

Abstract

An interactive floorplanner based on the genetic algorithm is presented. Layout area, aspect ratio, routing congestion and maximum path delay are 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 have to be met for some criteria, while for others, a good tradeoff is wanted. The approach taken by virtually all existing 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 scalar-valued cost function

$$c = \sum_{i=1}^{j} w_i c_i \quad \text{s. t. } \forall\ i = j+1, \ldots, k:\ c_i \leq C_i \quad (1)$$

for some j, $1 \leq j \leq k$. Here c_i is 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.

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 overall design objective is rarely clearly definable. Consequently, it may be very difficult 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 a clear notion of the overall design objective, the use of (1) causes serious difficulties : If the 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-dominated, but non-convex point, it can never be found using (1).

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:

1) The maximum routing congestion is minimized, thereby improving the likelihood that the generated floorplans are routable without further modification.

2) The delay minimization is path based, while most timing-driven floorplanning and placement approaches are net-based and therefore may over-constrain the problem [7].

3) 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 significant extensions of the approach described in [3].

*This work was done while the author was at Department of EECS, University of California, Berkeley, CA 94720, USA.

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

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

A floorplanning problem is specified by the following :

1) 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 and exact pin locations are known. For a flexible implementation, the area is given but the aspect ratio and exact pin locations are unknown.

2) A specification of all nets and a set of paths. Capacitances of sink pins, driver resistances of source pins, internal block delays and capacity and resistance of the interconnect is also needed to calculate path delays.

3) Technology information such as the number of routing layers available on top of blocks and between blocks.

Each output solution is a specification of the following :

1) A selected implementation for each block.

2) 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.

3) An absolute position of each block so that no pair of blocks are closer than a specified minimum distance. Since multi-layer designs are considered, it is assumed that a significant part of the routing is performed on top of the blocks.

4) An orientation and reflection of each block. The term *orientation* of a block refers to a possible 90 degree rotation, while *reflection* refers to the possibility of mirroring the block around a horizontal and/or a vertical axis.

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 scalar-valued cost measure.

Instead of weights and bounds, the user defines a *goal and feasibility vector pair* $(g, f) \in G$, where $G = \{(g, f) \in \Re_{+\infty}^n \times \Re_{+\infty}^n \mid \forall i : g_i \leq f_i\}$, $\Re_{+\infty} = [0, \infty]$. 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.

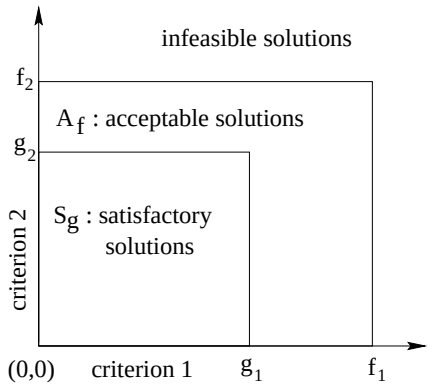

Figure 1: *The sets of satisfactory and acceptable solutions, illustrated in two dimensions.*

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 $x, y \in \Pi$. Then x *dominates* y, written $x <_d y$, if and only if $(\forall i : c(x)_i \leq c(y)_i) \wedge (\exists i : c(x)_i < c(y)_i)$. 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

$$x \prec y \quad \Leftrightarrow \quad (x <_d y) \vee (y \notin S_g) \tag{2}$$

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

$$x \prec y \quad \Leftrightarrow \quad (x <_d y) \vee [(x \in A_f) \wedge (y \notin A_f)] \tag{3}$$

Finally, x may satisfy some but not all goals. Assuming a convenient ordering of the optimization criteria, $\exists k : (\forall i < k : c(x)_i \leq g_i) \wedge (\forall i \geq k : c(x)_i > g_i)$. Then $x \prec y$ if and only if

$$[(\forall i \geq k : c(x)_i \leq c(y)_i) \wedge (\exists i \geq k : c(x)_i < c(y)_i)] \tag{4}$$
$$\vee$$
$$[(x \in A_f) \wedge (y \notin A_f)] \tag{5}$$
$$\vee$$
$$[(\forall i \geq k : c(x)_i = c(y)_i) \wedge \tag{6}$$
$$\{((\forall i < k : c(x)_i \leq c(y)_i) \wedge \tag{7}$$
$$(\exists i < k : c(x)_i < c(y)_i))$$
$$\vee$$
$$(\exists i < k : c(y)_i > g_i)\}] \tag{8}$$

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 (4) 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.

Using $\prec$ the solutions of a given set Φ can be ranked : $r(\phi, \Phi) = |\{\gamma \in \Phi | \gamma \prec \phi\}|$ 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

An overview of the GA used in Explorer is given in Section 3.1, while Section 3.2 focus on the key issue of the algorithm : the representation of a floorplan and its interpretation, as defined by the decoder. Section 3.3 describes how the user controls the optimization process interactively. For brevity, familiarity with GAs is assumed. An introduction to GAs can be found in [8].

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), replacing a poor solution. 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.

There are four types of interaction through the graphical user interface, gui (lines 7, 9, 11, 12). The update and optimization operations (lines 7-8 and 9-10) as well as the display function (line 11) are described in 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(Φ);
02   repeat :
03      select φ1, φ2 ∈ Φ;
04      crossover(φ1, φ2, ψ);
05      mutate(ψ);
06      insert(Φ, ψ);
07      if gui(update) :
08         adjust (g, f);
09      if gui(optimize) :
10         hillclimber(φ, k, (g', f'));
11      gui(display);
12   until gui(terminate);
13   output Φ0;
```

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 identifies the implementation selected for the i'th block.

b) A string of real values specifying aspect ratios of selected flexible implementations. The i'th value specifies the aspect ratio of the i'th selected flexible implementation.

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 [11], 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, used when routing the nets. The i'th integer identifies the critical sink of the i'th net.

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 dimensions of each selected flexible block is computed from its aspect ratio and its fixed area. The dimensions of all blocks are then known.

2) From the slicing-tree specified by the Polish expression the orientation of each block is determined such that layout area is minimized. An algorithm by Stockmeyer [10] is used, guaranteeing a minimum area layout for the given slicing-structure.

3) Absolute coordinates are determined for all blocks by a top-down traversal of the slicing-tree.

4) The layout is compacted, first vertically and then horizontally. The area c_{area} is computed as the smallest

rectangle enclosing all blocks and the aspect ratio cost is computed as $c_{ratio} = |r_{actual} - r_{target}|$, 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.

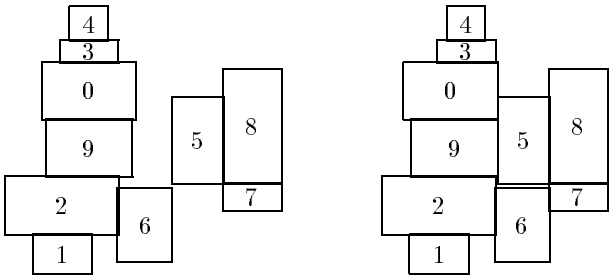

Figure 3: *Given 10 blocks and the Polish expression 1 2 + 6 * 9 0 + + 3 4 + + 5 * 7 8 + *, the floorplan on the left is the result of step 3. No blocks are moved when attempting vertical compaction, but subsequent horizontal compaction moves blocks 8,7,5,9 and 0 towards the left, so that blocks 2,6,7 now determine the width of the layout. The floorplan to the right is the result of compaction (step 4).*

5) A global routing graph $G = (V, E)$ is constructed, forming a uniformly spaced lattice, covering the layout. 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 pins of flexible implementations 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]. For each net, SERT-C minimizes the Elmore delay from the source to the critical sink specified by the representation.

7) The maximum path delay c_{delay} is determined by computing all path delays. For each net segment of a path, its Elmore delay is calculated in the corresponding Elmore-optimized Steiner tree and the appropriate internal block delays are added to obtain the total path delay. Since the Steiner trees are a very accurate estimation of the net topologies, c_{delay} is an accurate estimate.

8) The maximum routing congestion is estimated as

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

where cap(e) denotes the capacity of edge e (depending on possible blocks at that location) and usage(e) is the number of nets using e. I.e., c_{cong} is the maximum percentage by which an edge capacity has been exceeded. The smaller c_{cong} is, the fewer nets needs to be rerouted to obtain 100% global routing completion.

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 extensively studied in the GA literature and described in e.g. [8].

A crucial property obtained by the representation, the decoder and the genetic operators is that feasibility is preserved. I.e., only feasible representations, which can be interpreted by the decoder, are ever generated.

3.3 Interactive Control of the Search

Explorer provides the user with 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. An example graph is shown in Fig. 4. Based on this information the user can alter the optimization process at any time as described in the following.

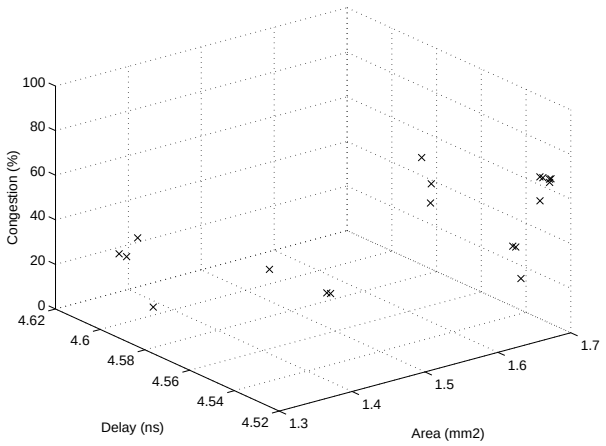

Figure 4: *An example graph showing 3 dimensions of a set of current best solutions. All solutions are non-dominated in the 4-dimensional cost space.*

As the optimization progresses and knowledge of obtainable cost tradeoffs is gained, the user can adjust the values of (g, f) (lines 7 and 8 of Fig. 2), thereby re-defining the notions of satisfactory and 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. 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 space.

The user can also execute a hillclimber on a specified individual ϕ (lines 9 and 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 $\neg(\phi \prec \phi')$. 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.

4 Experimental Results

It is inherently difficult to fairly compare our 4-dimensional optimization approach generating a solution *set* to existing 1-dimensional approaches generating a *single* solution. However, comparisons to simulated annealing and random search have been established.

4.1 Test Examples and Method

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 and the required timing information is added. spertF is an MCM designed at the International Computer Science Institute in Berkeley, California.

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.*

Explorer is implemented in C and executed 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].

Since RW does not rely on cost comparisons, it can use the same 4-dimensional cost function as Explorer, allowing the two approaches to be directly compared. In contrast, the traditional SA algorithm relies on absolute quantification of change of cost, which therefore has to be a scalar. Using a SA cost function of the form (1), it is far from clear how to fairly compare the *single* solution output by the SA algorithm to the *set* of solutions output by Explorer. Therefore, comparisons of Explorer to SA is based on optimizing one criterion only, in which case the output of Explorer reduces to a single solution.

4.2 One-Dimensional Optimization

One-dimensional optimization for area and delay was performed, for which Explorer uses the goal vectors $g = (0, \infty, \infty, \infty)$ and $g = (\infty, \infty, 0, \infty)$, respectively. Explorer is executed non-interactively.

Fig. 5 illustrates the results. For each circuit and each of the two criteria, the three algorithms was executed 10 times each and the result indicated by a bar. The center point of each bar indicates the average result obtained in the 10 runs and the height of each bar is two times the standard deviation. For each circuit and criterion, the average result of RW is normalized to 1.00.

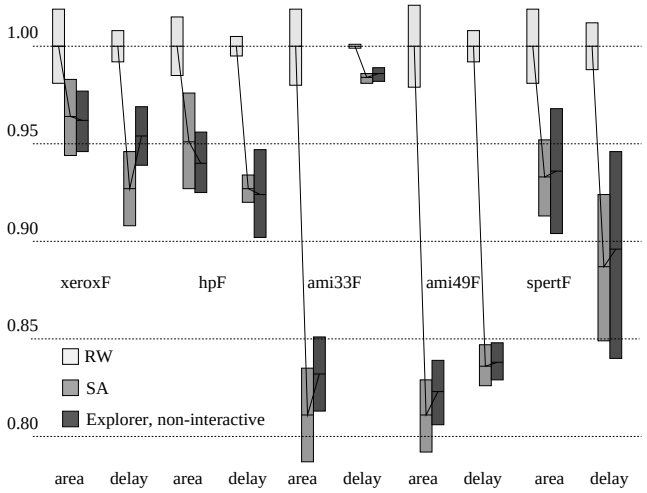

Figure 5: *Comparison of the performance of Explorer, SA and RW for one-dimensional optimization.*

The SA was executed first, and the consumed average CPU-time enforced on Explorer and RW as a CPU-time limit. The exact same average time consumption is thus obtained for all algorithms, at the cost of giving the SA approach an advantage. Average CPU-time per run varied from 39 seconds for area optimization of xeroxF to about 65 minutes for delay optimization of ami49F. As expected, both Explorer and SA performs significantly better than RW in all cases. Overall, the performance of Explorer and SA is very similar, indicating that the efficiency of the genetic algorithm used by Explorer is comparable to that of SA.

4.3 Four-Dimensional Optimization

Optimizing all four criteria simultaneously, interactive and non-interactive executions of Explorer are compared to RW. Explorer uses the target aspect ratio $r_{target} = 1.0$, the goal vector $g = (0, 0.2, 0, 50)$ and the feasibility vector $f = (1.5B, 0.5, \infty, 400)$, where B is the sum of the areas of all blocks of the circuit in question. For each circuit, RW is executed 10 times using a 5 CPU-hour time limit. In non-interactive mode, Explorer is also executed 10 times per circuit, but using a 1 CPU-hour limit. In interactive mode, a single execution was performed for each circuit, defining the time limit as 1 hour, wall-clock time, i.e., including the time spent using the interface.

The results are shown in Table 2. The set quality values are obtained using the set quality measure introduced in [4], which accounts for the (g, f) values specified. A smaller value means a higher quality. 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 execu-

Circuit	Output set size					Set quality		
	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 and the value in brackets is the standard deviation. For Explorer, the output set size is limited to 40.*

tions 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 search efficiency.

This performance gain is especially significant for the spertF layout. Feasible solutions were obtained interactively by executing direction-oriented hillclimbing on solutions outside but close to A_f. Only one of the sets generated non-interactively contained feasible solutions.

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 the need for iterations of floorplanning 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 and the required runtime is very reasonable from a practical point of view. Furthermore, the mechanisms provided for user-interaction are observed to improve the search efficiency significantly over non-interactive executions.

Acknowledgments

Dongsheng Wang at University of California, Berkeley, CA, implemented the simulated annealing algorithm. The research was supported by SRC contract no. 95-DC-324, NSF contract no. MIP 91-17328 and the Danish Technical Research Council.

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,* pp. 170-175, 1996.

[4] H. Esbensen, E. S. Kuh, "Design Space Exploration Using the Genetic Algorithm," *Proc. of the IEEE International Symposium on Circuits and Systems,* Vol. IV, pp. 500-503, 1996.

[5] P. J. Fleming, A. P. Pashkevich, "Computer Aided Control System Design Using a Multiobjective 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] L. Stockmeyer, "Optimal Orientations of Cells in Slicing Floorplan Designs," *Information and Control*, Vol. 57, pp. 91-101, 1983.

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