EXPLORER VERSION 1.0 USERS GUIDE

by

Henrik Esbensen

Memorandum No. UCB/ERL M96/72

26 November 1996

EXPLORER VERSION 1.0 USERS GUIDE

by

Henrik Esbensen

Memorandum No. UCB/ERL M96/72

26 November 1996

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Contents

1 Introduction

This users guide describes the use of Explorer Version 1.0, which is the implementation of the floorplanner for integrated circuits (ICs) presented in [4, 5].

Explorer is based on the genetic algorithm (GA) and simultaneously optimizes layout area, aspect ratio, routing congestion and maximum path delay. It is designed for design space exploration: In a single execution, a *set* of alternative floorplan solutions are generated, representing alternative tradeoffs of the four cost dimensions.

A unique feature of Explorer is that neither weights nor traditional bounds are used in the problem formulation. The competing optimization criteria are never combined into a scalar-valued cost function. Instead, the search is driven by a vector-valued cost function and a partial ordering on the cost space, which is defined by the user. Furthermore, Explorer supports an interactive search process. As knowledge of obtainable tradeoffs is gained, the user can interactively re-define the notion of a "good" or a "bad" tradeoff, thereby focusing the search on the region of interest in the cost space.

An input floorplanning problem is specified by the following:

1. A set of blocks, each of which have one or more possible implementations. Each implementation is either fixed (hard macro) or flexible (soft macro).
2. A set of IO-pins/pads.
3. A set of nets and a set of paths to be considered when minimizing the maximum path delay. Associated timing information, such as capacitances of sink pins and driver resistances of source pins, is also needed.
4. Technology information, e.g., the number of routing layers available on blocks and between blocks.

For each output solution, Explorer computes the following:

1. A selected implementation for each block.
2. An aspect ratio for each selected flexible implementation.
3. An approximate pin assignment for each pin of selected flexible implementations.
4. An absolute position for each block.
5. An orientation and reflection for each each block.
6. An absolute position for each IO-pin/pad.

While Explorer is a floorplanner, the provided mechanisms for GA-based, interactive, multi-objective optimization are of a general nature. Consequently, Explorer is believed to be well suited for more general studies of this type of optimization process, and in this context, the floorplanning problem can be seen as a non-trivial, example application. Interactive, multi-objective optimization poses some very interesting and open research problems [7]. For example, the issue of comparing solution sets as needed to evaluate the performance of different set-generating heuristic approaches is not yet solved satisfactory [3, 8]. Another research topic is whether the performance of the multi-objective GA-based search can be improved by using external controllers based on e.g. fuzzy logic, as studied in [9].

Familiarity with [4] is assumed throughout this users guide and a copy of [4] is therefore included in Appendix F. Issues not discussed in [4], such as e.g. IO-pin handling, are covered in this guide as needed. For example, IO-pin handling is described in Sections 5.2 and 5.4 as an integrated part of the relevant file format descriptions. Throughout this document, the `typewriter` font is used for keywords in files and for commands, while *italics* are used for menu selections on the interactive interface.

2 Required software and installation

The main part of Explorer is implemented in C and is compiled, while the interactive interface is implemented in Matlab and is interpreted. The source code consists of about 10,000 lines of C and about 1,000 lines of Matlab code. To install and use Explorer, the following additional software is required/recommended:

Software	Needed for	Importance
C compiler	compilation of source code	Required
Matlab	use of interactive features and some output files	Strongly recommended
Latex, viewer	viewing floorplan results	Recommended

Table 1: *Required and recommended software.*

Since it is possible to execute Explorer non-interactively, Matlab is not strictly needed. However, it is strongly recommended since the interactive interface constitutes some of the most interesting features of Explorer. Furthermore, the floorplan results obtained by interactive executions are generally much better than those obtained by non-interactive executions. If Latex and a method of viewing compiled Latex files (dvi-files) is available, plots of floorplan results can be displayed by compiling one of the output files generated by Explorer.

Explorer 1.0 is distributed as a single compressed tar-file named Explorer.tar.Z. When installing the software as described below, all files will be placed in the same directory, i.e., no new directories or subdirectories are created at any step of the process. Installation is done in four steps:

1. In the target directory, unpack the files by typing
   ```
   uncompress Explorer.tar.Z
   tar xvf Explorer.tar
   ```
   The result is a total of 57 files, as listed in Table 2.

2. If Latex is not available:

   (a) In makefile, remove latex.o from `OBJFILES`

   (b) In line 27 of database.h, redefine the constant `Latex` to `false`

3. If Matlab is available:

   (a) In makefile, redefine `MLIB` and `MINCLUDE` as the locations of the Matlab library libmat.a and the Matlab include files, respectively.

   Otherwise, if Matlab is not available:

   b In makefile, remove matlab.o from `OBJFILES`

   c In makefile, delete the path of `MLIB` such that the line reads `MLIB =`

   d In line 28 of database.h, redefine the constant `Matlab` to `false`

4. Compile the C code by typing `make`

File Name(s)	No. files	Contents
⋆.c ⋆.h	24	C source code
makefile	1	makefile
⋆Script.m	21	Matlab code
⋆.cir	10	sample floorplanning problems
test.par	1	sample parameter file

Table 2: *Overview of the files of the Explorer distribution.*

Explorer was developed and tested on a DEC 5000/125 workstation under Unix Ultrix V4.2A, using gcc, Matlab 4.2c and TeX 3.1415. The use of two machine-dependent functions in the C source code may require slight modifications:

- In database.c and statistics.c, the date and elapsed time is obtained using the functions `time()`, `difftime()`, `ctime()` and `asctime()`.

- Random number sequences are generated using `lrand48()` in database.h and `srand48()` for initialization in database.c

In case one or both of these functions are unavailable, they should be easy to replace, since they are only used at the specified locations.

The sample floorplanning problems include most of those used in [4]. They consist of pairs of the form <name>.cir and <name>F.cir. In <name>.cir, all blocks are fixed, while in <name>F.cir, all blocks are flexible. This is the only difference between the two examples in each pair, i.e., all block areas as well as the specifications of nets and paths are identical. The largest example in terms of number of blocks is ami49.cir/ami49F.cir. When optimizing this circuit and using the default parameter settings, cf. Section 5.1 and Appendix A, Explorer requires about 8 MB of memory on the DEC 5000/125.

3 Execution and input/output files

Explorer is a stand-alone tool. Table 3 gives an overview of the required input files and generated output files.

File Name	I/O	Contents	Section
<name>.par	I	Parameter file containing all control parameters	5.1
<circuit>.cir	I	Specification of a floorplanning problem	5.2
<name>.log	O	Logfile of the execution	5.3
<name><no>.pla	O	A set of files, each containing one floorplan result	5.4
<name>Plot.tex	O	Plots of all floorplan results To be compiled using Latex	5.5
<name>Result.m	O	Information on the cost tradeoffs of the floorplan results To be displayed using Matlab	5.6
<name>Stat.m	O	Information on the optimization process To be displayed using Matlab	5.7

Table 3: *Overview of input (I) and output (O) files read/written by Explorer. Each file is described in more detail in the Sections listed in the rightmost column. Depending on the installation procedure followed, Matlab and/or Latex-related output files may not be generated.*

The output floorplan results corresponds to the output set denoted Φ_0 in [4] (line 13 of Fig. 2). All files are ASCII text files. In addition to these output files, various information on the progress of the optimization process is written to stdout during execution.

Explorer is executed by the command

```
Explorer <name>
```

where <name>.par is the name of a parameter file. In addition to the control parameters, the parameter file also contains the name of the circuit specification to read, i.e., <circuit>.cir. After installing Explorer as described in Section 2, a test execution can be performed using the included sample files: The command

```
Explorer test
```

should generate the files test.log, test0.pla, test1.pla, ..., test<n>.pla[1], testPlot.tex if Latex is available, and testResult.m and testStat.m, if Matlab is available. This execution will require 60 seconds, elapsed time, and is non-interactive, i.e., does not use Matlab.

[1]Since the termination criterion of Explorer is specified in terms of absolute, elapsed time, the number of generations performed using a specific .par file is machine dependent, and consequently, the number of .pla files generated is machine dependent.

4 The interactive interface

An execution of Explorer will be interactive if `Interactive = yes` in the parameter file (see Section 5.1). When executed interactively, Explorer consists of two separate processes: a) The main process, corresponding to the main algorithm and the C source code, and b) a Matlab process, corresponding to the Matlab code, which handles interpretation of the interactive interface only. Initially, the Matlab process is started by the main process, and communication between the two processes is performed using a library of communication routines provided by Matlab and written in C [15].

The remaining of this Section describes how to use the interactive interface. Section 4.1 presents general information about the interface while Section 4.2 describes how to select the information displayed via the interface. Section 4.3 then describes how to adjust control parameters, including the goal and feasibility vectors, and use of the hillclimber is explained in Section 4.4.

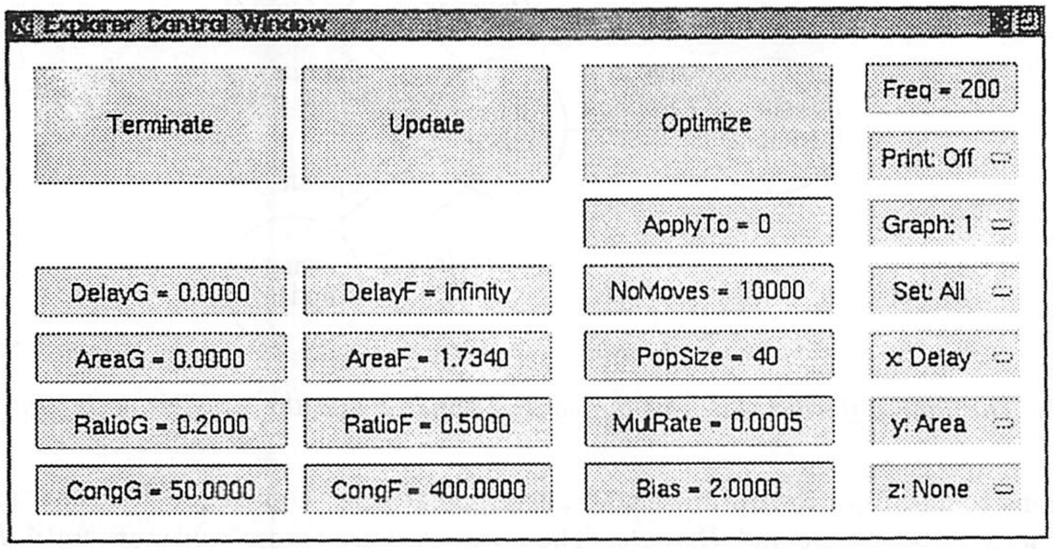
4.1 Interface overview

The interface consists of three different windows. Two windows are opened by Explorer, and are entitled "Explorer Control Window" and "Explorer Display Window" and the third is the window from which Explorer was started. In the following, these windows are referred to as the control window, the display window and the output window, respectively. The control window is used to control the optimization process and contains a number of buttons, menus and input fields. The display window is used for displaying graphs of the cost tradeoffs found so far, while textual information on the progress of the optimization process is provided in the output window, cf. Figure 1. The control window accepts user-input, while the display and output windows are used for output only.

```
GenNo 161 :  Delay = 4.81   Area = 10.963   Ratio = 1.00   Cong = 50
GenNo 200 :  ASize =  0  FrontSize =  7
GenNo 205 :  Delay = 5.05   Area = 10.860   Ratio = 0.87   Cong = 20
GenNo 214 :  Delay = 4.93   Area = 10.860   Ratio = 0.87   Cong = 10
GenNo 219 :  Delay = 4.78   Area = 11.172   Ratio = 0.91   Cong = 0
GenNo 259 :  Delay = 4.45   Area = 11.240   Ratio = 1.03   Cong = 30
GenNo 274 :  Delay = 4.84   Area = 10.052   Ratio = 0.74   Cong = 20
GenNo 300 :  ASize =  0  FrontSize =  8
GenNo 320 :  Delay = 4.96   Area = 10.331   Ratio = 0.84   Cong = 30
```

Figure 1: *Sample output in the output window. Each line listing cost values indicates that the best solution set Φ_0 has been updated by insertion of a solution with these cost values. In addition, the size of two important subsets are printed every 100 generations. The notation used here will be described in the following Sections.*

In the following, the term 'button' is used to refer to buttons, menus and input fields alike. Similarly, the terms 'clicking on' or 'selecting' can refer to any operation the user can do in the control window, including e.g. selecting another item of a menu or changing the value in a numerical field.


Explorer Control Window

Terminate | Update | Optimize | Freq = 200
Print: Off
ApplyTo = 0 | Graph: 1
DelayG = 0.0000 | DelayF = Infinity | NoMoves = 10000 | Set: All
AreaG = 0.0000 | AreaF = 1.7340 | PopSize = 40 | x: Delay
RatioG = 0.2000 | RatioF = 0.5000 | MutRate = 0.0005 | y: Area
CongG = 50.0000 | CongF = 400.0000 | Bias = 2.0000 | z: None

Figure 2: *The Explorer Control Window.*

The control window is shown in Figure 2. Every button is either green or red. It is important to always remember the following basic rule for operating the interface:

At any time, it is legal to click on a button if and only if it is green

When clicking on a green button, some red buttons may turn green and some green buttons may turn red, thereby indicating the next set of legal operations. Clicking on a red button is possible, but the effect is undefined. This is a consequence of a simple synchronization scheme used between the main process and the interactive interface.

The control window can be in one of four different states: initial, update, optimize and terminate, as illustrated in Figure 3. Each state represents a set of legal operations and have corresponding sets of green and red buttons. Each directed arc in Figure 3 represents a possible state transition and its label indicate the name of the button to click to perform that transition. When starting Explorer, the control window will be in the initial state. In this state, only three buttons are green: *Update*, *Optimize* and *Terminate*. Clicking one of these will cause a state transistion to the state of the corresponding name, cf. Figure 3. The update state allow the user to alter the displayed graphs, as described in Section 4.2 or to adjust the control parameters, as described in Section 4.3. The optimize state allows execution of the hillclimber, as described in Section 4.4. A transition from the initial state to the update state will cause the label of the *Update* button to change to *Continue* and at the same time, the color of the button will change to red. Therefore, to return from the update state to the initial state, the same button should be clicked, cf. Figure 3, when it has turned green again. This will happen when Explorer is ready, that is, when the main process is synchronized with the interface process. A message in the output window will then also indicate that *Continue* can be selected. The very same scheme is used

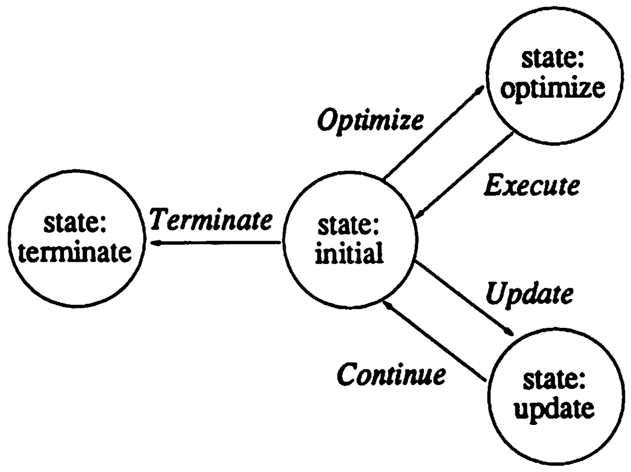


Figure 3: *The four different states of the control window and the possible state transitions.*

for transitions to and from the optimize state, except that here the label of the transition button alternates between *Optimize* and *Execute.* The optimization process continues until *Terminate* is selected, after which all buttons will turn red. It may take a few seconds before termination is detected and completed.

4.2 Defining the displayed graphs

The current state of the optimization process is continously visualized in the display window in the form of one, two, three or four graphs, each of which are either 2- or 3-dimensional. Each graph displays a set of points in the cost space, with each point corresponding to a specific solution in the current solution set. The user selects which subset is being displayed in each graph, and which cost-dimension correspond to which axis. The axis are labeled according to the current selection and the title of each graph indicates which subset is currently being displayed in the graph as well as the current size of that subset.

The graphs in the display window are defined by five buttons in the rightmost column of the control window, which are accessible when the control window is in the update state. The buttons are *Graph*, *Set*, *x*, *y* and *z*. Each of these is a menu with a pre-defined set of choices, and the current selection is displayed on the button itself. The *Graph* button determines which graph is currently being (re-)defined. The *Set* button determines which subset is being displayed in the graph, and the *x*, *y* and *z* buttons determines which cost dimensions correspond to the first, second and third axis, respectively. Since each graph has its separate definition, the settings of the *Set*, *x*, *y* and *z* buttons displayed at any given time only applies to the graph selected by the *Graph* button at that time. So to alter the definition of, for example, the 3rd graph, first select *Graph:3* on the *Graph* button. The current definitions for the 3rd graph is then displayed on the four other buttons and can be altered. Table 4 lists the five possible choices for the *Set* button together with the corresponding notation used in [4].

The possible choices for each of the *x*, *y* and *z* buttons are listed in Table 5. In addition to the choices listed in Table 5, the *z* button also has the selection *None*, which means that no cost

Set	[4] notation	Meaning
All	Φ	All existing solutions
A	$\Phi \cap A_f$	The subset of solutions satisfying the feasibility bounds
S	$\Phi \cap S_g$	The subset of solutions satisfying the goals
Front	Φ_0	The best solutions, relative to the goals and feasibility bounds
None	N/A	No set - the graph will not be displayed

Table 4: *Notation for sets.*

dimension is mapped to the third axis. Hence, if *None* is selected, the graph is 2-dimensional and otherwise, it is 3-dimensional.

Explorer will display graph 1, graph 2, etc, according to their definitions, until it meets the first graph definition for which the selected set is *None*. Therefore, if k graphs are wanted, $k = 1, 2, 3, 4$, they have to be defined as graphs 1 through k. The default graph definition used when starting Explorer is a single, 2-dimensional graph, showing the delay versus area tradeoffs of all solutions. If more than one graph is displayed, it is advisable to resize the display window.

Selection	Meaning
Delay	Maximum path delay in ns
Area	Layout area in mm^2
Ratio	Aspect ratio deviation, i.e., distance to the target aspect ratio
Cong	Maximum routing congestion

Table 5: *Notation for optimization criteria, selectable on the* x, y *and* z *buttons.*

Since Explorer optimizes four cost dimensions simultaneously, and only 2- and 3-dimensional graphs can be displayed, it may be difficult to obtain all four cost values of a specific solution by inspecting the graphs only. The *Print* feature (rightmost row, second button from the top) aims at reducing this problem. The button has five selections: *All*, *A*, *S*, and *Front*, which are sets defined as in Table 4, and *Off*. If a set is selected, the current cost values (4-tuples) of all solutions in that set are listed in the output window whenever the graphs are updated. If *Off* is selected, this feature is not used.

The graphs in the display window are updated every time the main process synchronizes with the interface process. The frequency of synchronization is defined by the *Freq* button, located in the top right hand corner of the control window. The numerical value is the number of generations between synchronizations. A smaller value will cause more frequent updates of the graphs and, if *Print* is used, more frequent listings of cost values. In addition, the worst-case waiting time for a transition from the update or optimize states back to the initial state will be reduced. However, the drawback of a smaller *Freq* value is that the main process will be wasting more time synchronizing with the interface and waiting for user input instead of performing optimization.

4.3 Adjusting control parameters

The control parameters adjustable by the user are the goal and feasibility vectors, described in [4], the population size, the mutation rate and a parameter concerning selection pressure. As the graph definition parameters described in Section 4.2, the control parameters are adjustable when the control window is in the update state, cf. Section 4.1.

Eight buttons in the lower left corner of the control window corresponds to the goal and feasibility vector pair (g, f) defined in [4]. These buttons are labelled pairwise as <criterion>*G* and <criterion>*F*, where <criterion> is one of *Delay*, *Area*, *Ratio* and *Cong*. The criteria correspond to the four cost criteria optimized, as listed in Table 4, and the *G* or *F* suffix used on each button indicate the goal or the feasibility value, respectively. For example, *AreaG* is the goal value for the area criterion, and *CongF* is the feasibility value for routing congestion. Since the range of all goal and feasibility values is $[0,\infty]$, cf. [4], any non-negative real value as well as the character string `infinity` can be entered.

The *PopSize* button (third column, fourth button from the top) is the population size. It can be changed to any integral value greater than or equal to 2. When increasing the population size, new randomly generated solutions will be added. If the population size is reduced, solutions are deleted in decreasing order of rank.

The *MutRate* button (third column, second button from the bottom) specifies the mutation rate. It is defined as the probability that a possible mutation is actually performed. Since the number of possible mutations vary with the given floorplan problem, this definition is to some extent problem-independent. Any real value in $[0,1]$ is allowed.

The last control parameter is the *Bias* button (bottom of third column). This parameter controls the selection pressure, following the scheme introduced in [6, 11]: Let β be the value of the *Bias* parameter, and assume that the current population $\Phi = \{\phi_0, \phi_1, \ldots, \phi_{N-1}\}$ is sorted in ascending order according to rank, i.e., $r(\phi_0) \le r(\phi_1) \le \ldots \le r(\phi_{N-1})$, where $r(\phi) = r(\Phi, \phi)$ is the rank of ϕ in Φ. When selecting a parent ϕ for crossover, the probability that $r(\phi)$ equals $r(\phi_k)$, written $P[r(\phi) = r(\phi_k)]$, decreases linearly with k, and $P[r(\phi) = r(\phi_0)] = \beta P[r(\phi) = r(\phi_{N/2})]$. Furthermore, all solutions having the same rank have the same probability of being selected. Hence, as β is increased, more solutions are selected among the best existing solutions. The *Bias* value (β) can be any real value in $]1,2]$.

The initial values of all control parameters discussed in this Section are read by Explorer from the parameter file <name>.par Whenever a control parameter has been changed interactively, it will take some time, i.e., some number of generations, for the effect of the change to propagate and visibly effect the optimization process. Therefore, control parameters should generally not be altered too frequently.

4.4 Applying the hillclimber

As described in [4], the hillclimber attempts a sequence of mutations on a specified solution. Each attempted mutation yielding ϕ' from ϕ is executed if and only if $\neg(\phi \prec \phi')$. Hence, the hillclimber may improve, and will never deteriorate, the optimized solution in the sense defined by the preference relation $\prec$.

The hillclimber is applied as follows:

1. Click *Optimize* to bring the control window to the optimize state, from which the hillclimber is accessible.
2. Select the solution to optimize, and write its identity (index in the population) in the *ApplyTo* button. A list of identities and corresponding cost values to choose from will appear in the output window next time the main program and the interface synchronizes. The listed set is the one currently selected on the *Print* button. If *Print* is *Off*, the *Front* set, i.e., Φ_0, is listed.
3. Write the number of mutations to attempt in the *NoMoves* button. The initial value of this parameter is defined in the parameter file.
4. Optionally, any of the goal and feasibility values can be changed before executing the hillclimber, thereby constraining the direction of the hillclimbing in the cost space, cf. [4]. If goal and/or feasibility values are changed, the new values will be in effect until changed again, in a subsequent *Update* or *Optimize* operation. In other words, to apply specific goal and feasibility values to the hillclimbing process only, it has to be followed by an *Update* operation, in which the original goal and feasibility values are restored.
5. Click *Execute* to execute the hillclimber. When completed, the obtained improvement will be described in the output window. Explorer will then switch to the initial state and the optimization process will resume.

5 File descriptions

In this Section the files read and written by Explorer are described. The usage of each file is explained and informal descriptions of syntax and semantics are given. A sample file of most file types can be found in the Appendices.

5.1 File <name>.par

A sample input parameter file is shown in Appendix A. The parameters have the following meaning:

`InputFileName` it the name of the floorplan specification file <circuit>.cir to read. It should be given without the extension.

`RatioTarget` is the target aspect ratio of the output floorplan(s), denoted by r_{target} in [4]. The aspect ratio is defined as height divided by width.

`Interactive` can be either `yes` or `no`. The execution will be interactive, using a Matlab process, if and only if `yes` is specified. If Matlab is not available, `Interactive` has to be `no`.

The eight parameters <criterion>`G` and <criterion>`F`, where <criterion> is either `Delay`, `Area`, `Ratio` or `Cong`, corresponds to the goal and feasibility vectors (g, f) in [4] and are as described in Section 4.3. If executing Explorer interactively, these parameters can be adjusted during execution and the values specified in the parameter file are used as initial values. The units are given in Table 5. Specifically, `RatioG` and `RatioF` specifies absolute distances to `RatioTarget`. As in Section 4.3, the range of each parameter is [0,∞], i.e., any non-negative real value can be entered and ∞ is specified by the character string `infinity`.

`TimeLimit` is only used if `Interactive` is `no`. It then specifies the maximum elapsed time of the execution, in seconds. Explorer will terminate before reaching the specified limit if and only if a solution has been found, which satisfies all goals.

`UseTimeAsSeed`, which can be either `yes` or `no`, and `RandSeed`, which can be any positive integer, controls the random number generation. If `UseTimeAsSeed` is `no`, the value of `RandSeed` will be used to initialize the random number generator, and the execution will be reproducable by simply re-using the same parameter file. If `UseTimeAsSeed` is `yes`, the random number generator is seeded using the system clock and the value of `RandSeed` is not used. However, the actual seed value used will be recorded in the logfile, described in Section 5.3, thereby allowing the execution to be reproduced by setting `UseTimeAsSeed` to `yes` and `RandSeed` to the value read from the logfile.

`PopSize`, `MutRate`, `Bias` and `NoMoves` are as described in Section 4.3. If Explorer is executed interactively, these parameters can be changed during the execution, and the values in the parameter file are used as initial values.

`HillclimbFreq` is only used if `Interactive` is `no`. It then specifies the frequency by which the hillclimber will be executed. Whenever no improvement have occurred for `HillclimbFreq` consequtive generations, that is, Φ_0 has not changed in this period, the hillclimber will be executed on every solution in Φ_0, using `NoMoves` mutations per solution.

The parameter values specified in the sample parameter file in Appendix A can in most cases be used as reasonable defaults, with one exception: A reasonable value of `AreaF` is 1.5 times the total instance area of the circuit in question and should therefore be changed when `InputFileName` is changed. The total instance area of a circuit is recorded in the logfile, i.e., it can quickly be obtained by executing Explorer non-interactively, using `0` as the `TimeLimit`.

If executing Explorer interactively, one can always use zero for all goal values and `infinity` for all feasibility values as the initial setting. As information on the obtainable tradeoffs is gained, the goal and feasibility values can then be adjusted appropriately. However, for non-interactive executions, this default setting is generally not advisable. The more aggressive a goal is defined, the harder will it be to obtain satisfactory values in the other cost dimensions. Therefore, goal values should not be specified beyond what is actually satisfactory.

5.2 File <circuit>.cir

An informal description of the syntax and semantics of the <circuit>.cir file is given below. Keywords are written using capital letters and the `typewriter` font. <n>, <x>, <y>, <val>, <min>, <max> and <t> are all integers. <c>, <r>, <d>, <area>, <low> and <high> are real values and the remaining terminal symbols are text strings. The keywords are described in the order in which they appear in the <circuit>.cir file. Ten sample .cir-files are included in the Explorer distribution, cf. Section 2.

A <circuit>.cir file is initiated by the following specification of technology related information:

TYPE	Specifies the type of the circuit as either IC or MCM. Each block in an IC can be oriented/reflected in eight distinct ways. In contrast, only four distint orientations/reflections exist for a block in an MCM, since MCM blocks are chips and their pins have to face the MCM surface.
ONBLOCKS <n>	Number of metal layers available for routing on top of blocks. This number is assumed constant for all blocks.
BETWEENBLOCKS <n>	Number of metal layers available for routing at locations where there are no blocks.
WIREPITCH <n>	The wirepitch for interconnect routing in μm, assumed constant for all routing layers. When estimating routing congestion it is assumed that all nets are routed using this wirepitch.
CAPACITANCE <c>	Capacitance of interconnect in pF/μm, assumed constant for all routing layers.
RESISTANCE <r>	Resistance of interconnect in kΩ/μm, assumed constant for all routing layers.
SPACING <n>	The minimum required distance between any pair of blocks or a block and an IO-pin, in μm.

The following statements of the <circuit>.cir file specifies constraints on relative IO-pin positions. This is done using a two-dimensional array $A_{s \times t}$ as illustrated in Fig. 4. Each IO pin can be assigned to an entry of A and the physical location corresponding to entry (i, j) will be $(ix/(s-1), jy/(t-1))$, where x and y are the horizontal and vertical dimensions of the layout, respectively[2]. An IO-pin assigned to A by the user is called a *fixed* IO-pin, while the remaining IO-pins are *flexible*. Each flexible IO-pin will be assigned by Explorer to a vacant entry of A not specified as blocked. Since any subset of the entries of A can be specified as blocked, pins can be restricted to placement along the periphery of the layout, they can be uniformly distributed over the entire layout, etc.

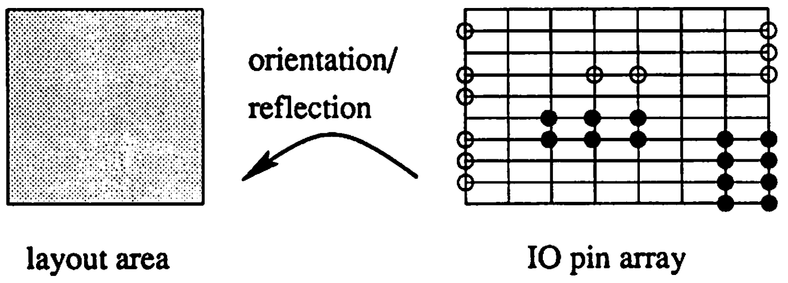


Figure 4: *Specification of constraints on placement of IO-pins. Here A has dimensions* 8×10 *and 11 fixed IO-pins (white circles) are assigned to specific entries of A while 14 entries (black circles) are blocked. The remaining entries are available for flexible IO-pins. A will be oriented and/or reflected and subsequently scaled so that it exactly covers the layout area of the floorplan.*

[2]Relative to the building-blocks, the entire set of IO-pins can be oriented and/or reflected in eight distinct ways, while still satisfying the constraints on relative positions specified by A. The given absolute position of entry (i, j) assumes that the IO pin set is positioned on top of the blocks without changing neither the orientation nor the reflection relative to the blocks.

`IOARRAY` <x> <y>	The x and y dimensions of the IO-pin array A described above (corresponding to s and t). Required whether or not any IO-pins exist.
`IOMINSPACE` <x>	The minimum distance between any pair of IO-pins, in μm. Required whether or not any IO-pins exist.
`IOBLOCKINGS`	Specification of blocked positions in the IO-pin array A. Required whether or not any IO-pins exist. The keyword is followed by zero or more pairs of <IndexRange> specifications. Each consecutive pair specifies a set of entries in A as blocked, i.e., not usable for assignment of flexible IO-pins. The first <IndexRange> in each pair specifies a range for the x-dimension while the second <IndexRange> specifies the y-dimension.
<IndexRange>	Has one of two forms: <val> <min> - <max> Indexing in A starts from 0. For example, the <IndexRange> pair `2 - 5 8` specifies that $A[i][8]$ is blocked for $i = 2, \ldots, 8$. Spaces are required on both sides of the hyphen.
`IOPINS`	Description of zero or more IO-pins. The keyword is required whether IO-pins exist or not. Each pin description has one of two forms: <fixedIOpin> or <flexibleIOpin>.
<fixedIOpin>	Has the form <pinName> <x> <y> <IOpinType>. <x> and <y> specifies the fixed relative position of the pin as indices in A.
<flexibleIOpin>	Has the form <pinName> <IOpinType>. This pin is flexible, i.e., has no constraints on its position.
<IOpinType>	Has one of four forms : `I` <c> `O` <r> `B` <c> <r> `@` `I` is an input pin (sink) and <c> its input capacitance. `O` is an output pin (source) and <r> its output driver resitance. `B` is a bidirectional pin (i.e., has both capacitance and resistance). `@` is a pin for which no type or electrical specification is available. <c> values are in pF and <r> values in kΩ.

The IO-pin specification above is followed by a specification of the available implementations of the blocks. At least one implementation has to be specified and each specification has one of the forms <fixedBlock> or <flexibleBlock>. The implementation specifications are followed by at least one instantiation of the form <instance>.

<fixedBlock>	Describes a fixed block and has the form `BLOCK` <name> <x> <y> followed by zero or more descriptions of pins, each of which has the form <fixedBlockPin>. <name> is the name of the implementation and <x> and <y> its horizontal and vertical dimensions, respectively, in μm.

<fixedBlockPin>	has the form <name> <x> <y> <blockPinType>. <x> and <y> specifies the position of the pin as the distance from the lower left corner of the block, in μm.
<blockPinType>	Has one of four forms : I <c> O <r> <d> B <c> <r> <d> @ I is an input pin (sink) and <c> its input capacitance. O is an output pin (source), <r> its output driver resitance and <d> its internal block delay. B is a bidirectional pin (i.e., has both capacitance, resistance and internal block delay). @ is a pin for which no type or electrical specification is available. <c> values are in pF, <r> values in kΩ and <d> values in ns.
<flexibleBlock>	Describes a flexible block and has the form FLEXBLOCK <name> <area> <low> <high> followed by zero or more descriptions of pins, each of which has the form <flexibleBlockPin>. <name> is the name of the implementation and <area> is its fixed area in μm^2. <low> and <high> specifies the lower and upper bound on the aspect ratio of the implementation, respectively.
<flexibleBlockPin>	has the form <name> <blockPinType>.
<instance>	Instance specification of the form INSTANCE <instName> <cellNameList> where <instName> is the instance name and <cellNameList> is a list of one or more names of alternative implementations separated by spaces. Each listed name refers to a previously defined implementation, which can be used to implement the instance. A flexible implementation can only be instantiated once.

The instantiations are followed by a specification of zero or more nets, and then zero or more paths. A path connects either two registers of distinct blocks or an IO-pin and a register, i.e., it is an alternating sequence of wires passing through blocks and net segments. For a sink pin p, let $m(p)$ denote the block to which p belongs[3] and let $s(p)$ denote the source pin of the net to which p belongs. Each path 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 $m(p_i) = m(s(p_{i+1})), i = 0, 1, \ldots, l-2$. $s(p_0)$ or p_{l-1} may be an IO pin. Each net specification has the form <net> and each path specification has the form <path>. The last path specification is followed by the keyword END, which is the last word in <circuit>.cir.

[3]If p is an IO pin, $m(p)$ is p itself.

<net>	A net specification of the form `NET` <netName> followed by an unordered list of two or more pin specifications of the form <pinRef>. Each net has exactly one pin of type `O`, which is the unique source pin.
<pinRef>	has one of two forms : <instName> <pinName> `IO` <pinName> The first form specifies an instance and a pin in that instance. A pin with name <pinName> has to exist in every implementation listed for the instance <instName> when declaring it. The second form specifies an IO-pin, i.e., a pin listed in the `IOPINS` section.
<path>	A path specification of the form `PATH` <pathName> followed by an ordered list of one or more pins of the form <pinRef>. This specification follows the path definition above, i.e., each pin is a sink pin of a distinct net, and the nets connects a sequence of blocks. An IO-pin can only be the first or the last pin of a path. If a net contains a pin of type @ it can not be part of any path, since it is then not possible to compute the path delay.

5.3 File <name>.log

A sample output logfile is shown in Appendix B. It contains the following information:

- Date and time of the start and end of the execution.
- An exact copy of the used parameter file <name>.par, thereby allowing the execution to be reproduced. If the system clock was used for initialization of the random number generator, the actual seed value can also be found in the logfile, cf. Section 5.1.
- Main characteristics of the input floorplan problem <circuit>.cir, including number of (flexible) cells, instances, pins, IO-pins, total instance area, etc. Information on net and path sizes is also given.
- Total number of generations, cost evaluations and elapsed time.
- A list of the generations in which hillclimbing was performed, if the execution was non-interactive.

If Explorer terminated normally, i.e., no error occurred, the last two lines of the logfile are

```
Explorer terminated normally <DateAndTime>
End of logfile
```

5.4 Files <name><no>.pla

The generated <name><no>.pla files are intended to be read by a global router and the .pla-file format is intentionally very similar to the input .cir-file. Therefore, the .pla-file format is described in the following by describing all differences from the .cir-file format presented in Section 5.2.

`SPACING`	Is absent in the .pla files since Explorer has placed all blocks and IO-pins such that this space requirement is satisfied.
`IOARRAY, IOMINSPACE, IOBLOCKINGS`	Also absent since this specification of relative IO-pin positions has been replaced by absolute positions.
`IOPINS`	Since absolute pin positions are computed for all IO-pins, every pin description has the form <fixedIOpin>, i.e., <flexibleIOpin> is never used. Furthermore, in <fixedIOpin>, <x> and <y> are now absolute positions in the layout, as opposed to relative positions.
<flexibleBlock>	Now has the form `FLEXBLOCK` <name> <x> <y> followed by descriptions of all pins, each of which has the form <flexibleBlockPin> or <fixedBlockPin>. <x> and <y> is the horizontal and vertical dimensions of the block in μm, as computed by Explorer. Each connected pin will have the form <fixedBlockPin>, in which <x> and <y> is the position assigned to the pin by Explorer. This position is an approximation only. Each pin is placed as close as possible to the mass center of the net. Consequently, many pins tend to be assigned to corners of blocks and multiple pins may be assigned to the same location or may be too close. If a pin is not connected, it will have the form <flexibleBlockPin>, i.e., its position is still undetermined.
<instance>	Now has the form `INSTANCE` <instName> <cellName> <x> <y> <t> <cellName> is the implementation of <instName> selected by Explorer. Only implementations, which are selected for at least one instantiation, will appear in the .pla file. <x> and <y> is the absolute position of the lower left corner of the instance *after* transformation of the instance, as defined by <t>. The transformation <t> is an integer in the interval from 0 to 7, as follows : 0 : no transformation 1 : mirror in x and y axis 2 : turn 90 degrees clockwise 3 : turn 90 degrees clockwise, mirror in x and y axis 4 : turn 90 degrees clockwise, mirror in y axis 5 : turn 90 degrees clockwise, mirror in x axis 6 : mirror in x axis 7 : mirror in y axis
<net>	Is as in .cir file, except that each pin description now has the form <netPin>.

<netPin>

Has one of four forms :
<instName> <pinName>
<instName> <pinName> `CRITICAL`
`IO` <pinName>
`IO` <pinName> `CRITICAL`
Each pin appears as in the input .cir file, except that the keyword `CRITICAL` has been added to exactly one sink pin for each net. The keyword indicates that the marked sink was used by Explorer as the critical sink when routing the net using the SERT_C algorithm [1, 4]. This information may be of use to a global router. For nets with only one sink pin, the keyword `CRITICAL` is omitted.

5.5 File <name>Plot.tex

The output file <name>Plot.tex contains plots of the generated floorplan results. It is only generated by Explorer if Explorer was installed following the procedure for systems having Latex available, cf. Section 2. <name>Plot.tex is to be compiled using Latex and it can then be viewed using e.g. xdvi. Each floorplan result in <name>Plot.tex appears as a Latex figure on a separate page. The figure on page *n* corresponds to the floorplan result stored in file <name><n-1>.pla. A sample figure from a <name>Plot.tex file is shown in Appendix C. Each cell on the figure is labelled by the instance name followed by the cell name in brackets. Filled circles are IO-pins, which are also labelled by their names. If routing congestion is optimized it is visualized on the figure as follows: An open circle indicates the location of a routing graph edge, for which the routing capacity was exceeded by more than the goal value. Hence, if no open circles are shown on a figure (as in Appendix C), the routing congestion goal is satisfied everywhere, while an area with many open circles indicates an unresolved congestion problem at that location in the circuit. The figure caption contains a name of the form <name>`G`<no>`P`<id>, where <name> is the same name as used in the file name <name>Plot.tex, <no> is the number of generations of the generating execution and <id> is the identity of this solution in the population. In addition, the caption lists the cost value for each optimized criterion. If applicable, the percentage of the maximum path delay, which was caused by interconnect, is also given.

5.6 File <name>Result.m

The output file <name>Result.m is only generated if Explorer was installed following the procedure for systems having Matlab available, cf. Section 2. The file contains information on the final cost tradeoffs obtained, stored in a form to be used within Matlab. In particular, the information can be visualized using Matlabs facilities for plotting graphs.

A sample <name>Result.m file is shown in Appendix D. It contains eight Matlab vectors, named <name><set><criterion>, where <set> is either `All` or `Front` and <criterion> is either `Area`, `Ratio`, `Delay` or `Cong`. The meaning of <set> and <criterion> are as in Table 4 and Table 5, respectively. The i'th entry in vector <name><set><criterion> is the cost value for <criterion> of the i'th solution in <set> in the final generation of the optimization process.

The vectors are read into Matlab by simply typing the name <name>Result at the Matlab command line, and the tradeoffs of any pair or triple of cost criteria can then be visualized as graphs. For example, if <name> is exp, a two-dimensional graph showing area versus delay tradeoffs of all existing solutions after the final generation is obtained by the command

```
plot(expAllArea,expAllDelay,'o')
```

Similarly, the command

```
plot3(expFrontDelay,expFrontRatio,expFrontArea,'o')
```

generates a three-dimensional graph showing the tradeoffs of delay versus aspect ratio deviation and area represented by the best (Φ_0) solutions after the final generation. This subset of solutions also corresponds to the solution set saved in the <name><no>.pla files and the <name>Plot.tex file. For further use of the vectors within Matlab, the reader is referred to [14].

5.7 File <name>Stat.m

The output file <name>Stat.m is generated if and only if <name>Result.m is generated, and as <name>Result.m, it contains Matlab vectors to be used within Matlab as described in Section 5.6. The information in <name>Stat.m captures main characteristics of the optimization process itself. It can be used to generate two-dimensional graphs showing the development of key quantities as functions of time, as described in the following.

<name>Stat.m contains a total of 14 Matlab vectors. Two of the vectors, <name>`GenNo` and <name>`ElapsedTime`, can be used as the time axis in a graph. <name>`GenNo[i]` is the i'th generation for which information is stored. The frequency of storing information is 100 generations. However, the first entry in <name>`GenNo` is always 0, corresponding to the point in time immediately after generation of the initial population, and the last entry is always the last performed generation, that is, it corresponds to the point in time immediately before termination of the algorithm. <name>`ElapsedTime[i]` is the actual elapsed time in seconds spent up to and including generation number <name>`GenNo[i]`.

Each of the remaining 12 vectors described below specifies a quantity, which can be plotted as a function of time. The i'th entry of each of these vectors specifies the value of that quantity at time <name>`GenNo[i]`, or equivalently, at time <name>`ElapsedTime[i]`.

<name>`FrontSize` is the size of the subset *Front*, or Φ_0, i.e., the number of solutions having zero rank. Similarly, <name>`ASize` is the size of the subset A, or $\Phi \cap A_f$, i.e., the number of feasible solutions. The vectors <name>`AvgRank` and <name>`MaxRank` specifies the average and maximum rank of solutions in the population, respectively.

Finally, eight vectors are named <name><criterion><Stat>, where <criterion> is either `Area`, `Ratio`, `Delay` or `Cong` and <Stat> is either `Avg` or `St`. <criterion> specifies an optimization criterion as in Table 5, while the suffix `Avg` refers to average value and `St` refers to standard deviation. Each vector specify either the average or the standard deviation of some cost criteria, calculated over the complete population, i.e., the set *All*, or Φ. For example, <name>`AreaSt[i]` is the standard deviation of area in the complete population at generation <name>`GenNo[i]`.

Assume that <name> equals `exp`. As an example on how the file expStat.m can be used to generate informative graphs in Matlab, the commands

```
plot(expGenNo,expAvgRank)
hold
plot(expGenNo,expMaxRank)
```

will produce a graph showing the average as well as the maximum rank of the population as functions of the generation number. Similarly, the command

```
plot(expElapsedTime,expDelayAvg)
```

will produce a graph of the average delay of solutions in the population as a function of actual elapsed time.

In addition to graph generation as described in this and the preceding Section, the files <name>Result.m and <name>Stat.m can be used in numerous other ways within Matlab to further analyze both floorplan results and optimization processes, by utilizing other Matlab capabilities. For example, sets of floorplan results from different executions of Explorer can be compared, as well as different optimization processes, by using multiple <name>Result.m and/or <name>Stat.m files simultaneously.

6 Organization of the C source code

This Section outlines the organization of the C source code. The purpose is to provide a basic overview, which could be useful if one wishes to modify the implementation. Each source file is listed below together with comments on its contents.

database.h	Declarations of types and global variables used throughout the program.
database.c	Definitions of global variables and routines used throughout the program.
main.c	The main routine, outlining the top-level algorithm.

The following files constitute the GA-part of the code:

initPop.c	Generation of the initial, random population
reproduce.c	Performs one steady-state generation, i.e., selection for crossover, calls of crossover and mutation and insertion of the result(s) into the population.
rank.c	Maintains the rank of all solutions, corresponding to $r(\Phi, \phi)$ in [4]. The routine `preferable()` implements the preference relation denoted $\prec$ in [4].
orderCrossover.c	Contains a generic routine for crossover on permutations of integers. This is an implementation of the edge recombination operator presented in [12].
crossover.c	All other crossover operators, including operators introduced in [2].
mutation.c	All mutation operators.
optimize.c	The hillclimber.

The following files implements the decoder, i.e., the cost computation of a solution, as described in Section 3.2 of [4].

routing.h	Declarations of data structures used in evalDelay.c, pinPosition.c, steiner.c and congestion.c
evalDelay.c	Contains the top-level decoder routine, **Decode**, and routines for computing maximum path delay.
evalArea.c	Computation of aspect ratios for flexible blocks and initial absolute placement of all blocks from a given slicing structure (Polish expression). Includes Stockmeyers algorithm [10].
compact.c	Compaction of the initial placement to obtain the final placement. Implements a simplified version of the 1-dimensional compactor presented in [13].
pinPosition.c	Determines the absolute positions of all pins for a given placement.
steiner.c	Computes an Elmore-optimized Steiner tree for each net, embedded in the global routing graph. The main algorithm is an implementation of the SERT-C algorithm introduced in [1].
congestion.c	Contains routines to determine the maximum routing congestion by examining the global routing graph. This graph is not explicitly constructed in the implementation, but is implicitly represented by the data structure `netSegments`.

I/O and communication with the Matlab process is implemented by the following group of files:

matlab.c	Contains all C interface routines handling communication with the independent Matlab process.
parFileParser.c	Parser for the parameter file <name>.par
placeFileParser.c	Parser for the input circuit file <circuit>.cir. Also constructs many data structures.
latex.c	Generation of the output file <name>Plot.tex
output.c	Generation of the output files <name><no>.pla
statistics.c	Collects statistics of the optimization process and generates the files <name>.log, <name>Result.m and <name>Stat.m.
checks.c	These routines are used for debugging only. They perform various internal consistency checks of datastructures, and are activated by setting the constant `debugFlag` in the file database.h to the value `true`.

7 Questions and Comments

This software is made available AS IS with no obligation of providing technical support. Neither the Electronics Research Laboratory nor the University of California make any warranty about the software, its performance or its conformity to any specification.

However, questions, comments, improvements and bug-reports are very welcome, and can be sent to software@eecs.berkeley.edu or directly to

Henrik Esbensen
Avant! Corporation
1208 East Arques Avenue
Sunnyvale, CA 94086, USA
Email: Henrik_Esbensen@avanticorp.com

Acknowledgments

The author would like to thank Professor Ernest S. Kuh, who supervised the Explorer project, Dr. Dongsheng Wang, who implemented pin assignment in Explorer, and Dr. Michael Lee, Tiangxiong Xue and Premal Buch, who contributed by many constructive discussions.These colleagues are all at University of California, Berkeley. Financial support was provided 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, "Design Space Exploration Using the Genetic Algorithm," *Proc. of the IEEE International Symposium on Circuits and Systems,* Vol. IV, pp. 500-503, 1996.

[4] H. Esbensen, E. S. Kuh, "EXPLORER: An Interactive Floorplanner for Design Space Exploration," *Proc. of the European Design Automation Conference,* pp. 356-361, 1996.

[5] H. Esbensen, E. S. Kuh, "A Performance-Driven IC/MCM Placement Algorithm Featuring Explicit Design Space Exploration," to appear in *ACM Transactions on Design Automation of Electronic Systems,* 1997.

[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] C. M. Fonseca, P. J. Fleming, "Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms I : A Unified Formulation," *Research Report 564*, Dept. of Automatic Control and Systems Eng., University of Sheffield, U.K., January 1995.

[8] C. M. Fonseca, P. J. Fleming, "An Overview of Evolutionary Algorithms in Multiobjective Optimization," *Evolutionary Computation,* Vol. 3, No. 1, pp. 1-16, 1995.

[9] M. A. Lee, H. Esbensen, "Automatic Construction of Fuzzy Controllers for Evolutionary Multiobjective Optimization Algorithms," *Proc. of the IEEE International Conference on Fuzzy Systems,* pp. 1518-1523, 1996.

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

[11] 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.

[12] D. Whitley, T. Starkweather, D. Fuquay, "Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator," *Proc. of the Third International Conference on Genetic Algorithms,* pp. 133-140, 1989.

[13] 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.

[14] "MATLAB Reference Guide", *The MathWorks Inc.*, Mass., 1992.

[15] "MATLAB External Interface Guide", *The MathWorks Inc.*, Mass., 1992.

A Sample <name>.par file

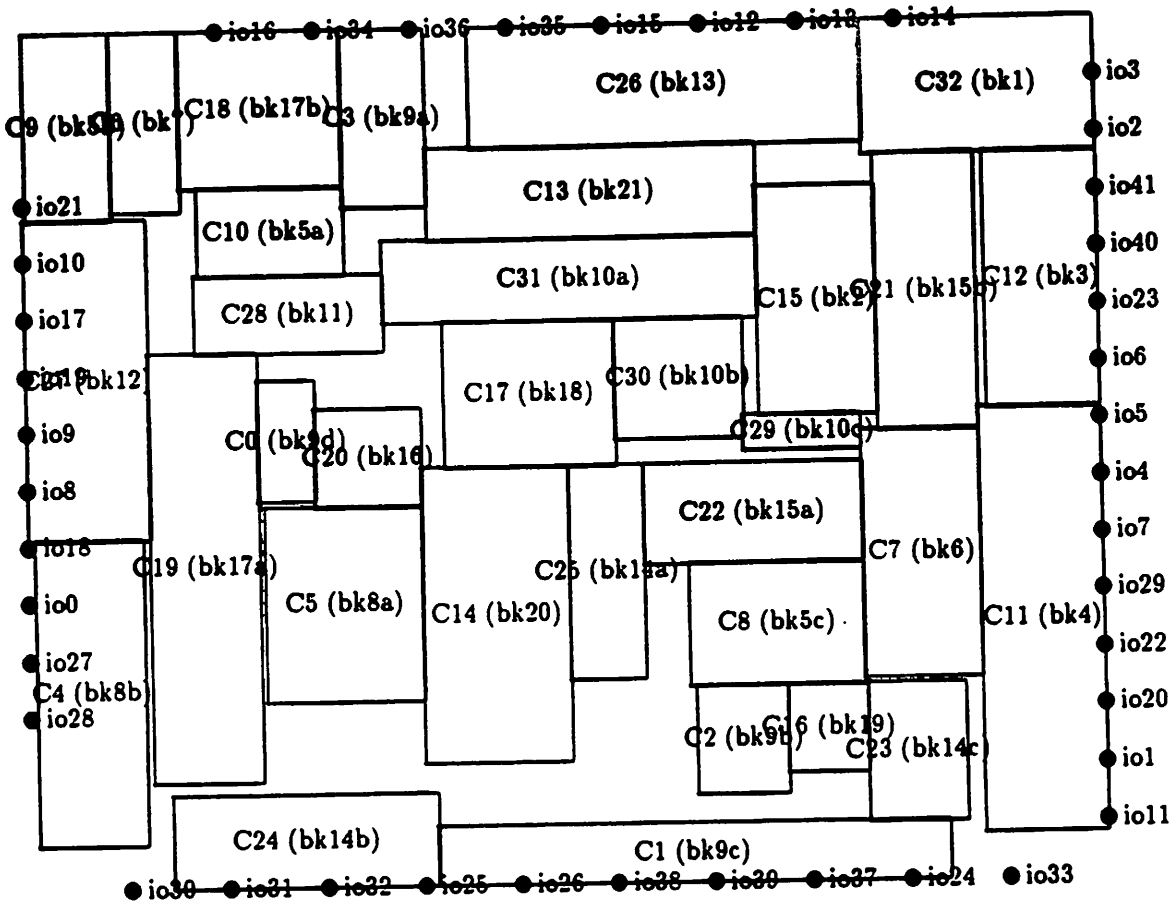
```
Explorer Parameter File

InputFileName = ami33F
RatioTarget   = 1.0

Interactive = no

DelayG = 0
AreaG  = 0
RatioG = 0.2
CongG  = 50

DelayF = infinity
AreaF  = 1.734
RatioF = 0.5
CongF  = 400

TimeLimit     = 3600
UseTimeAsSeed = no
RandSeed      = 21420

PopSize = 40
MutRate = 0.0005
Bias    = 2.0

NoMoves       = 10000
HillclimbFreq = 100000
```

B Sample <name>.log file

```
                    Explorer Logfile

Explorer started Sun Dec  1 02:53:10 1996

Copy of the parameter file used:
---------------------------------
Explorer Parameter File

InputFileName = ami33F
RatioTarget   = 1.0

Interactive = no

DelayG = 0
AreaG  = 0
RatioG = 0.2
CongG  = 50

DelayF = infinity
AreaF  = 1.734
RatioF = 0.5
CongF  = 400

TimeLimit     = 3600
UseTimeAsSeed = no
RandSeed      = 21420

PopSize = 40
MutRate = 0.0005
Bias    = 2.0

NoMoves      = 10000
HillclimbFreq = 100000
---------------------------------

Problem characteristics:

  Total no. of cells    =  33
  No. of flexible cells =  33
  No. of instances = 33
  No. of instantiations of flex. cells = 33
```

```
 Minimum total instance area = 1.156449 sq. mm
 Pins :
 IO-pins with fixed positions =   42
 IO-pins placed automatically =    0
 IO-pins, total               =   42
 Instance pins, total         =  480
 Connected pins, total        =  522

 Size of IO-array    =    192 (  16 x   12)
 Blocked positions   =      0
 Occupied positions  =     42 (fixed pins)
 Available positions =    150 (for free pins)

 No. of nets = 123
 No. of nets with auto-placed IO-pins = 0
 Net characteristics:
   104 nets with  2 pins
    10 nets with  3 pins
     2 nets with  4 pins
     1 nets with  9 pins
     2 nets with 35 pins
     1 nets with 44 pins
     1 nets with 47 pins
     1 nets with 50 pins
     1 nets with 56 pins

 No. of paths = 233
 Path characteristics:
   133 paths with  1 segments
    67 paths with  2 segments
    14 paths with  3 segments
    17 paths with  4 segments
     1 paths with  6 segments
     1 paths with 10 segments

Total number of generations = 34454
Total number of cost evaluations = 43259
Total elapsed time = 3616.0 seconds
Reason for termination:  time limit exceeded
Hillclimbing was never performed

Explorer terminated normally Sun Dec  1 03:53:26 1996
End of logfile
```

C Sample page of compiled <name>Plot.tex file



e7G34454P1 :	Area	:	1.341	mm^2
	Aspect ratio	:	0.81	
	Delay	:	5.33	ns (interconnect : 16.3 %)
	Congestion	:	50.00	% excess

D Sample <name>Result.m file

```
e3AllArea = [ 1.344; 1.341; 1.306; 1.309; 1.368; 1.307; 1.314;
 1.331; 1.315; 1.347; 1.308; 1.315; 1.335; 1.316;
]
e3AllRatio = [ 0.211; 0.192; 0.203; 0.201; 0.234; 0.202; 0.203;
 0.190; 0.200; 0.220; 0.202; 0.202; 0.190; 0.205;
]
e3AllDelay = [ 5.356; 5.325; 5.334; 5.334; 5.374; 5.331; 5.371;
 5.331; 5.318; 5.322; 5.332; 5.353; 5.301; 5.368;
]
e3AllCong = [ 50.000; 50.000; 50.000; 50.000; 50.000; 50.000; 40.000;
 50.000; 50.000; 50.000; 50.000; 50.000; 50.000; 60.000;
]
e3FrontArea = [ 1.344; 1.341; 1.306; 1.309; 1.368; 1.307; 1.314;
 1.331; 1.315; 1.347; 1.308; 1.315; 1.335; 1.316;
]
e3FrontRatio = [ 0.211; 0.192; 0.203; 0.201; 0.234; 0.202; 0.203;
 0.190; 0.200; 0.220; 0.202; 0.202; 0.190; 0.205;
]
e3FrontDelay = [ 5.356; 5.325; 5.334; 5.334; 5.374; 5.331; 5.371;
 5.331; 5.318; 5.322; 5.332; 5.353; 5.301; 5.368;
]
e3FrontCong = [ 50.000; 50.000; 50.000; 50.000; 50.000; 50.000; 40.000;
 50.000; 50.000; 50.000; 50.000; 50.000; 50.000; 60.000;
]
```

E Sample <name>Stat.m file

```
e6GenNo = [ 0; 100; 200; 300; 400; 500;]
e6ElapsedTime = [ 5.000; 16.000; 28.000; 39.000; 50.000; 61.000;]
e6FrontSize = [ 5; 12; 1; 1; 4; 1;]
e6AvgRank = [ 4.000; 1.925; 1.725; 2.525; 5.100; 6.375;]
e6MaxRank = [ 17; 5; 4; 4; 7; 9;]
e6ASize = [ 0; 0; 1; 1; 4; 6;]
e6AreaAvg = [ 2.932; 2.233; 2.041; 1.990; 1.900; 1.866;]
e6AreaSt = [ 0.504; 0.278; 0.238; 0.214; 0.145; 0.147;]
e6DelayAvg = [ 5.770; 5.649; 5.599; 5.578; 5.568; 5.557;]
e6DelaySt = [ 0.123; 0.086; 0.071; 0.068; 0.067; 0.066;]
e6RatioAvg = [ 0.466; 0.367; 0.338; 0.286; 0.300; 0.328;]
e6RatioSt = [ 0.419; 0.334; 0.282; 0.199; 0.231; 0.220;]
e6CongAvg = [ 57.583; 57.667; 45.750; 51.000; 44.750; 46.583;]
e6CongSt = [ 28.684; 32.670; 18.288; 22.782; 23.284; 23.752;]
```

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

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

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

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.

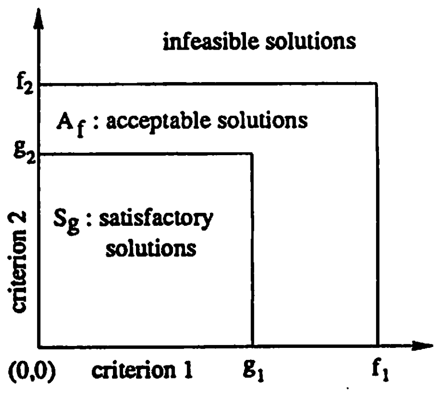


Figure 1: *The sets of satisfactory and acceptable solutions, illustrated in two dimensions.*

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($\Phi$);
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 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.

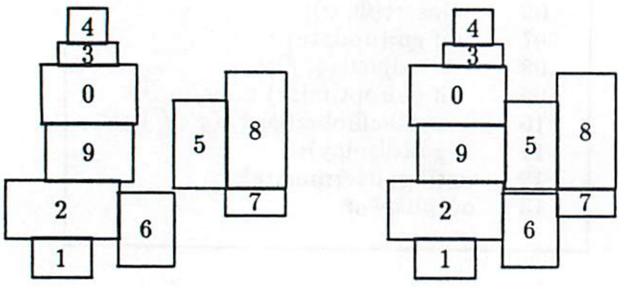


Figure 3: *Given 10 blocks and the Polish expression 1 2 + 6 * 9 0 + + 3 4 + + 5 * 7 8 + *, the floorplan on the left is the result of step 3. No blocks are moved when attempting vertical compaction, but subsequent horizontal compaction moves blocks 8,7,5,9 and 0 towards the left, so that blocks 2,6,7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

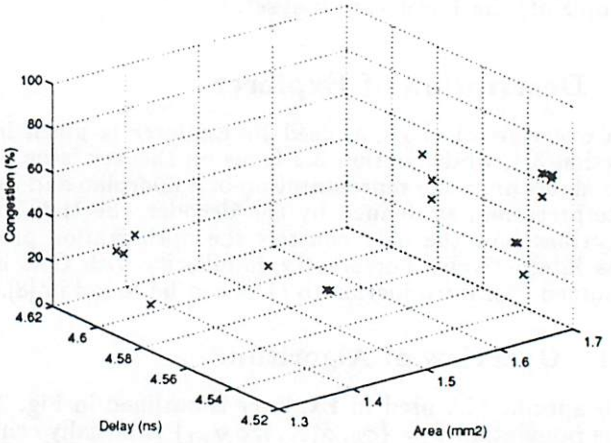


Figure 4: *An example graph showing 3 dimensions of a set of current best solutions. All solutions are non-dominated in the 4-dimensional cost space.*

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

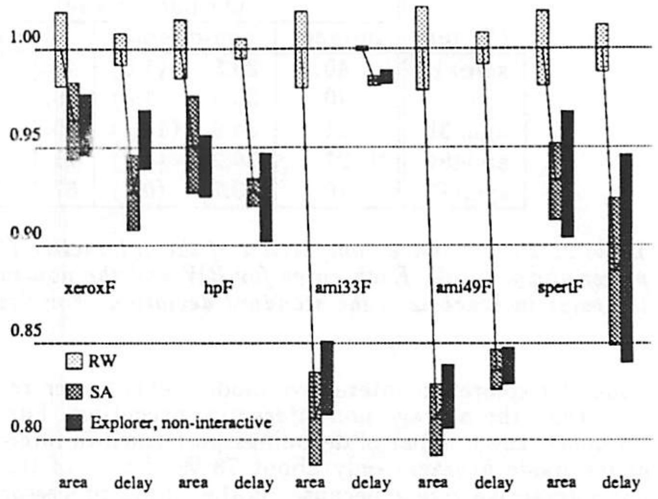


Figure 5: *Comparison of the performance of Explorer, SA and RW for one-dimensional optimization.*

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.

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.