Timing Driven Genetic Algorithm for Standard-cell Placement *

Sadiq M. Sait   Habib Youssef   Khaled Nassar   Muhammad S. T. Benten
Department of Computer Engineering
King Fahd University of Petroleum and Minerals
Dhahran-31261, Saudi Arabia
e-mail: facy009@saupm00.bitnet

Abstract

In this paper we present a timing-driven placer for standard-cell IC design. The placement algorithm follows the genetic paradigm. At early generations, the search is biased toward solutions with superior timing characteristics. As the algorithm starts converging toward generations with acceptable delay properties, the objective is dynamically adjusted toward optimizing area and routability. Experiments with test circuits demonstrate delay performance improvement by up to 20%. Without any noticeable loss in solution quality, sizable reduction in runtime is obtained when population size is allowed to decrease in a controlled manner whenever the search hits a plateau.

1 Introduction

Placement consists of assigning cells of a given circuit to physical locations on a 2-dimensional layout surface. The cells may be standard-cells, macro blocks, etc. In this work we consider standard-cell placement. Until recently, the total wirelength was a widely used measure of the quality of placement. However, due to advances in VLSI technology, sizes of transistors have been decreasing and their switching speeds increasing. In the last two decades the scaling has been so drastic, that there has been a tremendous increase in the importance of interconnect delays with respect to the overall speed performance of the circuit.

The speed of a circuit is determined by the time it takes for a signal to travel on its *longest* path. A signal traveling on any path π is constrained to reach the path end point no later than its latest required arrival time ($LRAT_\pi$). A design is free from long path timing problems if, for every path π,

$$T_\pi \leq LRAT_\pi \qquad (1)$$

where T_π is the overall delay on path π. The problem of performance driven placement consists of finding suitable locations of cells so as to minimize the total wirelength and area, while satisfying Equation 1 for each path π.

In this work a linear cell delay model is used. The delay TD of a cell is computed as follows,

$$TD = BD + LD + ID \qquad (2)$$

*This research is supported by KFUPM Project # COE/VLSIDESIGN/162

where, LD is the load delay due to loading pins of the net driven by the cell, and ID is the interconnect delay on the net. Expressions for these quantities are given below.

$$LD = LF \times C_{in} \qquad (3)$$

$$ID = LF \times C_{net} + R_{net} \times (C_{net} + C_{in}) \qquad (4)$$

where, LF is the load factor, C_{in} is the total input capacitance of the loading cells, R_{net} is the total interconnect resistance of the net, and C_{net} is the total interconnect capacitance of the net (fringe plus surface).

Most contemporary VLSI placement tools incorporate timing performance aspects in their objective functions[1]. Methods for generating constraints on sizes of nets to guarantee performance are reported in [10, 15]. These methods consist of distributing slacks on the nets. The final layout that satisfied these net bounds was guaranteed to satisfy path timing constraints for desired performance. In [14], factors which are highly correlated with the path delays are combined into a score function. Path criticality is decided on the basis of path scores. The predicted critical paths are used by the placement procedure.

In [4, 12] performance driven placement is solved using mathematical programming. In [7], a placement solution that satisfies multiple objectives, which are, area, routability, and timing, is produced using fuzzy logic rules. In [13], the application of a constructive successive augmentation methodology to VLSI placement under constraints on routability, area, and timing is presented.

Iterative and non-deterministic techniques have also been employed. In [2], the authors employ simulated annealing to improve both the wirelength and performance. Other iterative nondeterministic techniques that have been applied to the placement are genetic algorithm (GA) [1, 11] and simulated evolution [6]. But for both, the placement objective was the minimization of wirelength, timing performance was not an issue.

In this work we describe a timing driven genetic algorithm for placement. Initially, a number of

[1]The number of various approaches reported on the aspect of timing driven placement is too large to enumerate in a conference paper. The review is limited to a small subset of the works the authors are most familiar with.

placement configurations are constructively produced. Then, the genetic algorithm is used to iteratively search for a new solution that combines the good characteristics of the initial configurations. The overall objective is two-fold; (1) satisfy path timing constraints and (2) minimize overall wiring length (area).

The principle reason for selecting the genetic algorithm is that it allows several placement configurations to be maintained. This makes the technique easy to adapt to the multi-objective nature of the placement problem in general and timing-driven placement in particular. Further, the genetic algorithm has hill climbing capability, making it suitable for the placement problem which is characterized by several noisy (a large number of local optima) and conflicting objectives.

1.1 Timing Prediction

Long path timing problems are caused by large interconnect delays. Obviously, the critical paths are those that are most inclined to exhibit a long path problem. The timing data passed by the timing analysis program to the placement procedure consist of a set of the most critical paths. This set is predicted using the notion of α-criticality. Our prediction approach proceeds as follows. From past layouts with similar complexity, the average and standard deviation of net lengths are estimated for each type of net (2 pin-, 3 pin-,..., k pin-nets). These are converted to capacitances for the particular technology of the design at hand. Let T_π and S_π be the overall delay (including the net delay estimations) and standard deviation along path π. Let $T_{\max}$ be the estimated delay of the longest path in the design, that is,

$$T_{\max} = \max_{\pi}\{T_\pi\} \tag{5}$$

A path π is called *α-critical* if and only if,

$$T_\pi + \alpha \times S_\pi \geq T_{\max} \tag{6}$$

The parameter α acts as a confidence level. The larger α is, the larger is the number of predicted critical paths, and the higher is the probability of including all potentially critical paths. Typical values of α are: $\alpha \times S_\pi \leq 5ns$. This prediction approach was effective in predicting all of the critical paths in the designs we experimented with.

2 TDGAP: Timing Driven Genetic Algorithm for Placement

Genetic algorithm (GA) is a search technique which emulates the natural process of evolution as a means of progressing toward the optimum [3]. GA has been applied in solving various optimization problems including those in VLSI physical design [1, 11].

The structure of the genetic algorithm for timing-driven placement referred to as TDGAP is given in Figure 1. In this work the cost function used includes both the timing performance of the circuit and the total wirelength.

Given an initial solution, TDGAP conducts a search in the solution space using its operators and functions to repetitively modify the population of placement configurations. The search continues until all the timing constraints are satisfied or the population has reached maturity. Timing constraints consist of delay bounds on the critical paths of the circuit.

```
ALGORITHM ($TDGAP$)
  $N_p$= Population Size.
  $N_g$= Number of Generations.
  $N_o$= Number of Offsprings.
  $P_c$= Crossover Probability.
  $P_\mu$= Mutation Probability.
  Begin
  Initial_Population($N_p$)
    For $j$ = 1 to $N_p$
      Evaluate $Fitness$(Population[$j$])
    EndFor
    For $i$ = 1 to $N_g$
      For $j$ = 1 to $N_o$
        $(s,t) \leftarrow Choose\_parents$ (*CHOICE*)
        With probability $P_c$ Perform Crossovers
          Offspring[$j$] $\leftarrow$ $Crossover(x,y)$
      EndFor
      (*SELECTION of next generation*)
      Population $\leftarrow$ $Select$(Population, offspring, $N_p$)
      For $k$ = 1 to $N_p$
        With probability $P_\mu$ Perform Mutation
          Population[$k$] $\leftarrow$ $Mutation$(Population[$k$])
      EndFor
      For $m$ = 1 to $N_p$
        Evaluate $Fitness$(Population[$m$])
      EndFor
    EndFor
  Return highest scoring individual in Population.
  End
```

Figure 1: General algorithm of TDGAP.

2.1 Solution Representation

Each individual (solution) in the population is encoded (represented) as a set of rows. Each row contains modules (genes) that are represented as a set of three integers indicating the cell serial number, the row number, and the displacement from the left edge of the layout.

2.2 Initial Population Constructors

Initial solution construction is very critical to GA. Five initial population constructors were investigated. They are: (1) Constructor IPC_1 selects modules at random and places them in rows; (2) Constructor IPC_2 attempts to clusters cells affecting the same path; (3) Constructor IPC_3 is similar to IPC_2 except that IPC_2 places cells left to right, starting from row 0, while IPC_3 places cells starting from the middle row and proceeding outward; (4) Constructor IPC_4 combines individuals from IPC_1 and IPC_3; and

(5) Constructor IPC_5 is similar to constructor IPC_4 with the difference that it includes in its initial population a placement configuration obtained using the mincut partitioning algorithm of the OASIS system [8]. Therefore, the initial population constructed with IPC_5 consists of three classes of individuals: (1) placements obtained randomly, (2) placements constructively built to exhibit good timing characteristics, and (3) placements constructively obtained with mincut partitioning to exhibit good wirelength characteristics.

2.3 Choice Function

The choice function adopted is based on the *stochastic remainder without replacement* scheme. This selection scheme has been proven to be superior over the expected value scheme [3] and works as follows. Let $exp\_count(\mathcal{P}(i))$ be the value of the expected count of an individual $\mathcal{P}(i)$.

$$exp\_count(\mathcal{P}(i)) = \frac{cost(\mathcal{P}(i))}{\overline{cost}} \qquad (7)$$

where,

$$cost\mathcal{P}(i) = \text{ cost value of individual } \mathcal{P}(i) \qquad (8)$$

and,

$$\overline{cost} = \frac{1}{N_p} \times \sum_{i=1}^{N_p} cost(\mathcal{P}(i)) \qquad (9)$$

For each individual $\mathcal{P}(i)$, $\lfloor exp\_count(\mathcal{P}(i)\rfloor$ instances of $\mathcal{P}(i)$ are included in a list L. The fractional part $f_i = exp\_count(\mathcal{P}(i) - \lfloor exp\_count(\mathcal{P}(i)\rfloor$ is interpreted as a probability, that is, with probability f_i one more instance of $\mathcal{P}(i)$ is included in the list L. This operation is repeated until all individuals are processed. Following this step, two parents at a time are randomly selected from the list L and, with a probability P_c, they are crossed to produce an offspring. In each generation, at most N_o offsprings are bred.

2.4 Crossover $\mathcal{X}$

Crossover is the most important genetic operator and has the most effect on the convergence rate and the quality of solution. Two types of crossover operators $\mathcal{X}_1$ and $\mathcal{X}_2$ are considered in TDGAP. Both use information passing from one parent to the other, but they differ in the way they pass information. Both operators are aimed at improving the timing aspects of the reported α-*critical* paths. They try to make the offspring inherit some of the satisfied paths (paths with no timing problems) from its parents. Operator $\mathcal{X}_1$ achieves this by maintaining the same locations of the cells affecting these satisfied paths. Operator $\mathcal{X}_2$, however, keeps the cells affecting the satisfied paths within a certain window.

Let $\mathcal{P}(s)$ and $\mathcal{P}(t)$ be the passing and target parents respectively, and CP be the set of the α-*critical* paths of the circuit. Operator $\mathcal{X}_1$ operates in the following way. An identical copy ($\mathcal{P}(o)$) of $\mathcal{P}(t)$ is made. A critical path cp is selected from CP according to a criterion that will be explained later. Then, the set β of cells affecting cp is identified. The goal of $\mathcal{X}_1$ is to reconfigure offspring $\mathcal{P}(o)$ such that the cells in β occupy the same locations as in the passing parent. Collisions are resolved by interchanging the locations of the two modules. The steps of $\mathcal{X}_1$ operation are illustrated in Figure 2. In this figure, the set $\beta = \{c_1, c_2, c_3, c_4\}$. Then, the cells of β in $\mathcal{P}(o)$ are moved to the same locations as they were in $\mathcal{P}(s)$. As shown in Figure 2(c), cells c_1 and d_1 interchange locations.

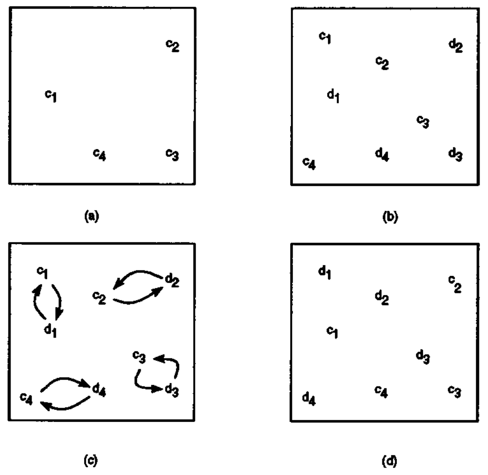

Figure 2: Crossover operator $\mathcal{X}_1$. $\beta = \{c_1, c_2, c_3, c_4\}$. (a) Parent 1 (passing parent), (b) Parent 2 (target parent), (c) Information passing, (d) Offspring.

Crossover operator $\mathcal{X}_2$ operates in the following manner. It starts like $\mathcal{X}_1$ by making a copy $\mathcal{P}(o)$ from $\mathcal{P}(t)$, selecting a critical path cp from CP, and identifying the set β of cells affecting cp. The size and location of the smallest bounding window ω_s that encloses the cells of β in $\mathcal{P}(s)$ is determined. A window ω_t is also determined in parent $\mathcal{P}(t)$ with the same dimensions and location as ω_s in $\mathcal{P}(s)$. Comparing the contents of these two windows, three sets are defined:

σ = cells $\in \beta - \omega_t$, (cells in β but not in ω_t);
ρ = cells $\in \omega_s - (\omega_t + \beta)$, (cells $\in \omega_s$ but are neither in ω_t nor in β); and
η = cells $\in (\omega_s \cap \omega_t)$.
Then operator $\mathcal{X}_2$ reconfigures $\mathcal{P}(o)$ as follows. It first defines a window ω_o in $\mathcal{P}(o)$ of the same size and location as that of ω_s. After that, it scans the contents of ω_o cell by cell and one row at a time. Then, for each scanned cell e_i, operator $\mathcal{X}_2$ works according to the algorithm shown in Figure 3. The operation of $\mathcal{X}_2$ is depicted in Figure 4.

Selection of critical path cp

For each placement configuration $\mathcal{P}(i)$ in current generation, the critical paths are maintained in two lists: (1) CP_i^+ is the list of the critical paths with positive slack, and (2) CP_i^- is the list of the critical paths with negative slack. Let $\mathcal{P}(s)$ and $\mathcal{P}(t)$ be the source (passing) and target parents respectively. The process of selecting a cp is aimed at generating an off-

```
ALGORITHM(Reconfigure)
Stop=0
Repeat
 If $e_i \in \eta$ Then
  skip this cell and go to the next one,
 ElseIf $\sigma \not\subseteq \emptyset$ Then
  Begin
   pick a module $e_j$ from $\sigma$ and swap $e_i$ and $e_j$,
   remove module $e_j$ from $\sigma$
  End
 ElseIf $\rho \not\subseteq \emptyset$ Then
  Begin
   pick a module $e_j$ from $\rho$ and swap $e_i$ and $e_j$,
   remove module $e_j$ from $\rho$
  End
 Else
  Begin
   skip this cell (all other cells will stay in their locations)
   Stop=1
  End
Until(all cells $\in \omega_o$ are scanned or Stop=1)
```

Figure 3: An algorithm used by $\mathcal{X}_2$ to reconfigure an offspring ξ_o.

spring with better timing characteristics. The critical path selection algorithm proceeds as follows:

1. If target parent has no timing problems, that is $CP_t^- = \emptyset$, then select at random a *cp* from CP_s^+;

2. else, if there is a critical path *cp* with long path timing problem in target parent, but is problem free in passing parent, that is $cp \in CP_s^+ \cap CP_t^-$, then select this path;

3. else, if there is a *cp* with a long path timing problem in both the target and passing parents ($cp \in CP_s^- \cap CP_t^-$), but with better timing (larger slack) in the passing parent, then select this path;

4. else, select a *cp* at random.

2.5 Selection (§) of the Next Generation

The crossover applications have the effect of augmenting the current generation with its offsprings. The selector function is used to maintain the population size fixed, that is to decide which individuals to use as part of the next generation. We experimented with four selector functions. Let $\mathcal{P}$ be the current population and $\mathcal{J} = \mathcal{P} \cup Offsprings$; then the selector functions proceed as follows:

- Selector $\S_1$ selects from $\mathcal{J}$ the best scoring individual and $N_p - 1$ other individuals at random, where $N_p = \mid \mathcal{P} \mid$.

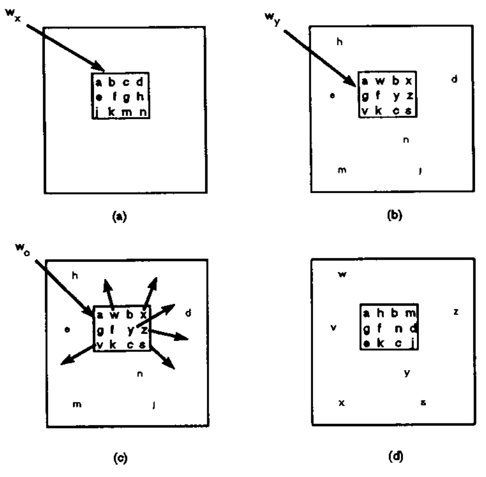

Figure 4: Crossover operator $\mathcal{X}_2$. β={a,h,m,n}, σ={h,m,n}, ρ={d,e,j}, η={a,b,c,f,k,g}. (a) Parent 1 (passing parent), (b) Parent 2 (target parent), (c) Information passing, (d) Offspring.

- Selector $\S_2$ selects the best 10% of N_p and the rest are selected at random.

- Selector $\S_3$ selects all N_p individuals from $\mathcal{J}$ at random.

- Selector $\S_4$ selects individuals on a competitive basis with each individual $\mathcal{P}(j)$ having a probability *Prob*(j) to be selected, where

$$Prob(j) = \frac{score(\mathcal{P}(j))}{\sum_{i=1}^{q} score(\mathcal{P}(i))} \quad (10)$$

  where, $q = \mid \mathcal{J} \mid$, and $score(\mathcal{P}(i))$ is the fitness of individual $\mathcal{P}(i)$. With this selector, the algorithm has a higher probability than with other selectors to be trapped into local minima. This might happen because individuals with low fitness values are quickly discarded at the early generations.

2.6 Mutation μ

Two mutation operators μ_1 and μ_2 were investigated. Operator μ_1 is targeted toward improving the timing of the placement, while operator μ_2 is targeted at improving the wirelength of the placement. Both operators use the notion of *center-of-mass* in order to improve the path timing and length of selected nets. In TDGAP, except for the best individual, any individual in the newly selected generation may be a candidate for mutation.

Mutation operator μ_1 works as follows. For each individual $\mathcal{P}(i)$ that is selected for mutation, first a critical path *cp* is randomly selected from the set CP_i^- (those paths with long path problems in $\mathcal{P}(i)$). Next, a module e_s which is affecting the performance of *cp* is randomly selected. The module e_s is pairwise interchanged with the module at the center of mass of

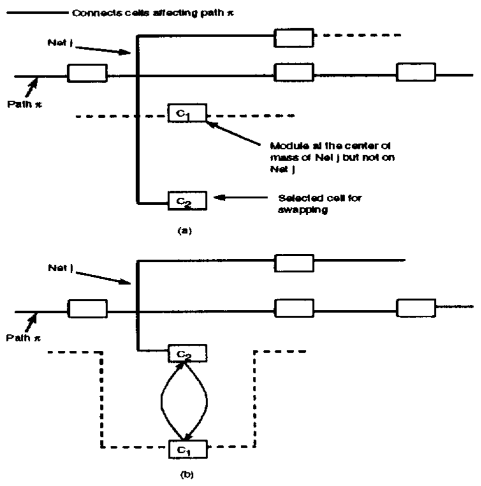
Connects cells affecting path π
Net j
Path π
C1
Module at the center of mass of Net j but not on Net j
C2
Selected cell for swapping
(a)
Net j
Path π
C2
C1
(b)

Figure 5: The operation of mutation operator μ_1. (a) Before mutation, (b) after mutation.

the selected critical net. This mutation operation is illustrated in Figure 5.

Mutation operator μ_2 operates in a manner similar to that of operator μ_1. It starts by selecting at random a two-pin net, where neither of the two pins is an I/O pad. Then, one of the two modules is chosen to be swapped with a module at the location of the center of the selected net.

The requirement that the selected net be a two-pin net is motivated by experimental observations. Analysis of several layouts revealed that most of the two-pin nets that are on critical paths have their modules separated by large distances. This wide separation has two undesirable effects: (1) it increases the number of feedthroughs and (2) it increases the total wirelength.

2.7 Score Function

The score function is a combination of three terms. The three terms are directed toward the improvement of the circuit performance and total wirelength. Let $\mathcal{P}(i)$ and $\mathcal{P}(j)$ be two individuals in current population $\mathcal{P}$; if $Score(\mathcal{P}(i)) > Score(\mathcal{P}(j))$ then individual $\mathcal{P}(i)$ is fitter than individual $\mathcal{P}(j)$. The score of a given individual $\mathcal{P}(i)$ is computed as follows:

$$Score(\mathcal{P}(i)) = w_1 \times S_i \ + \ w_2 \times W_i \ + \ w_3 \times R_i \quad (11)$$

where w_1, w_2, and w_3 are different weights assigned for each term. S_i and W_i are measures of the timing and wirelength aspects of individual $\mathcal{P}(i)$, while R_i is a relaxation factor. It is a measure of the amount by which satisfied paths can be made longer and remain problem free. This is to give the wirelength metric a chance to improve.

2.8 Experiments and Results

The genetic algorithm is a very elaborate and relatively hard to tune algorithm. It is harder than other iterative heuristics such as simulated annealing [5] and simulated evolution [6]. The tuning of the genetic algorithm parameters to a particular problem (such as placement) requires extensive experimentation. Such an extensive experimentation has been conducted on TDGAP. The operators that performed best with respect to wirelength and timing are indicated[2] in Table 1. These operators showed a superior performance as well as faster convergence.

Constructor function	IPC_5
Crossover operator	$\mathcal{X}_1$
Mutation operator	$\mu_1 \cup \mu_2$
Selector function	$\S_1$
Population size	24
Crossover probability	0.5 - 0.7
Mutation probability	0.1

Table 1: A summary of the genetic parameters that were found to perform better than others with TDGAP.

We noticed that the population fitness improves very rapidly during the early generations. The change in the population fitness is less rapid as the number of generations increases, until it becomes insignificant. The reason is that, the improvements in the population are caused by crossovers and mutations involving high scoring individuals. Therefore, toward the middle and later generations, the role of low scoring individuals becomes insignificant as a source of new fit individuals for the next generations. Hence, it seems reasonable to allow the population size to progressively decrease (in a controlled manner) with the number of generations. Such a decrease will cause a sizable reduction in run time without any noticeable change in solution quality.

We experimented with this idea using CRC16 (see Table 3) as a test case. In a first strategy the population size was fixed to 30 individuals. In a second strategy, the population was allowed to progressively decrease. A reduction procedure determines when to reduce the population size and by how much. The performance of the best solution is checked periodically every *Reduction_Period*. If no timing improvement by at least 3% is achieved during the last *Reduction_Period*, the population size is reduced by 20%. This reduction is allowed as long as the population size does not become less than half of the initial population size. The *Reduction_Period* parameter is also dynamically decreasing. Initially it is assigned a large value. Then each time the population size is checked, the *Reduction_Period* is reduced by a *Period_Factor*. This is performed because the convergence rate of TDGAP in the early generations is higher than in the later ones. Thus, the reduction procedure monitors the performance after shorter periods in those generations where the improvement is too slow. For the CRC16 test case, we set the *Reduction_Period* equal to 5000 generations and the *Period_Factor* equal to 5%.

[2] discussions and graphs are omitted due to lack of space.

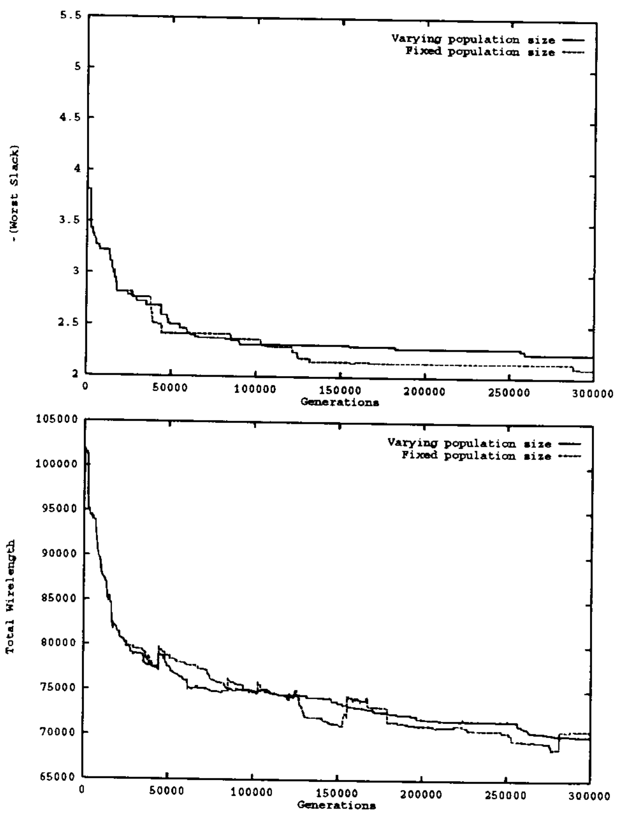

Figure 6: Timing and wirelength performance of the best solution with dynamic and fixed population size. The clock period taken is smaller by $3ns$ than for results reported in Table 2.

Figure 6 shows the timing and area performance of the best solution with dynamic and fixed population size. Both cases were run for 300,000 generations. A summary of the initial and final values of the best solution with dynamic and fixed population sizes is given in Table 2. From this table we note that the quality of the results in both cases were of comparable quality, but the total run time for the case of dynamic population size has decreased by 46%. However more experimentation is required to tune the reduction schedule (that is, *Reduction_Period* and *Period_Factor*).

We run TDGAP on the following circuits (Table 3).

1. **Ckl**: A sample AHPL model that performs part of the stop and wait protocol.
2. **CRC16**: A 16-bit Cyclic Redundancy Checker.
3. **Highway**: A traffic light controller (test case from the OASIS system).
4. **Fract**: A fractional multiplier. The description of this circuit is given in [9].

A summary of the initial and final values of the best solution for all test cases with respect to timing

Best Solution	Dynamic Population		Fixed Population	
	Slack	Area	Slack	Area
Initial	-2.47	87786.3	-2.47	87786.3
Final	+0.61	69959.1	+0.78	70580.8
Run Time %	54%		100 %	

Table 2: A summary of the initial and final values of the best solution with dynamic and fixed population size for wirelength and timing performance. Slack values are given in *ns*.

Circuit Name	# of cells	# of critical paths	# of rows in final layout
Ckl	209	200	8
CRC16	209	330	9
Highway	56	14	4
Fract	149	368	6

Table 3: Characteristics of the circuits used.

and area metrics is given in Table 4. The slack values given are obtained after the placement phase, but before routing. The placements obtained by OASIS and TDGAP were evaluated with respect to timing as well as overall wirelength. Timing performance improvements of up to 20% were obtained. The area values given are obtained after completing the routing phase and generating the layout. The *Magic* layout system has been used to view the layouts of the circuits and get their actual height and width. The improvement achieved by TDGAP with respect to timing aspects has resulted in a slight increase in the overall area (between 1% and 9%).

One observation that is in order concerns the genetic control parameters. From experiments we identified two parameters that, if allowed to adapt, would lead to shorter run time and superior results: (1) the crossover probability; and (2) the mutation probability. From experimentation, starting with a high crossover probability of 90% then gradually reducing it to 70% seems to be a better choice then keeping it constant. A similar strategy can be also applied for the mutation probability, that is, starting with relatively large value of 20% and then gradually reducing it until it reaches 10%. These observations require further investigations.

TDGAP is implemented in the C language. Experiments were performed on a 64-bit DEC Alpha workstation that is running OSF/1 operating system at the speed of 100 MIPS. Although run-time reported ranged from 5 to 14 hours, as can be seen from Figure 6, solutions improves by 80% within the first 10-15% of the time. That is, a solution close to the best reported is obtained within the first one or one and a half hours of run-time.

Circuit Name	Clock period	Run time in Hrs	Area			Slack		
			TDGAP	Increase	OASIS	TDGAP	Impr	OASIS
Ck1	21 *ns*	10.1	928 ×1016	9.1%	923 × 936	+0.52	8.1%	-1.14
CRC16	18 *ns*	14.4	1131 × 968	5.5%	1072 × 968	+0.61	17.7%	-2.47
Highway	20 *ns*	5.4	478 × 496	6.2%	465 × 480	+0.51	11.4%	-1.72
Fract	38 *ns*	9.0	798 × 824	5.9%	768 × 808	+0.31	6.5%	-2.14

Table 4: Area and Slack performance comparison between TDGAP and OASIS.

3 Conclusions

In this paper we presented a timing-driven placement program. The placement procedure follows the genetic algorithm. The program uses path timing data from a timing analyzer. The timing analyzer uses a new technique and a new criterion (*α-criticality*) to predict the critical paths prior to placement.

Extensive experiments were conducted to tune the parameters of the program and evaluate the extent of timing improvement. Experimental results show that timing improvement of upto 20% can be achieved by our placement system without any change to the logic of the circuit. The sizable timing improvement is accompanied with a very slight increase in the area of the circuit. We also experimented with the idea of gradually decreasing the population size as the population matures. This idea resulted in considerable reduction in run-time (by about 50%) without any noticeable loss in solution quality. We believe, that with a better tuned reduction schedule run-time can be further reduced (again with no loss in solution quality).

Acknowledgments

Authors acknowledge King Fahd University of Petroleum and Minerals for all support. This research is supported by KFUPM Project # COE/VLSIDESIGN/162

References

[1] J. P. Cohoon and W. D. Paris. Genetic placement. *IEEE Transactions on Computer Aided Design*, CAD-6:956–964, November 1987.

[2] W. Donath et al. Timing-driven placement using complete path delays. *Proceedings of 27th Design Automation Conference*, pages 84–89, June 1990.

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

[4] M. A. B. Jackson and E. S. Kuh. Performance-driven placement of cell-based IC's. *Proceedings of 26th Design Automation Conference*, pages 370–375, June 1989.

[5] S. Kirkpatrick, Jr. C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):498–516, May 1983.

[6] R. M. Kling and P. Banerjee. Empirical and theoretical studies of the simulated evolution method applied to standard cell placement. *IEEE Transactions on Computer Aided Design*, CAD-10:1303 - 1315, October 1991.

[7] Rung-Bin Lin and E. Shragowitz. Fuzzy logic approach to placement problem. *Proceedings of 29th Design Automation Conference*, pages 153–158, 1992.

[8] MCNC Group. *OASIS 2.0 Reference Manual*, 1990.

[9] H. Troy Nagle et al. *Intro to Computer Logic.* Prentice Hall, 1975. page 461.

[10] R. Nair et al. Generation of performance constraints for layout. *IEEE Transactions on Computer Aided Design*, CAD-8(8):860–874, August 1989.

[11] K. Shahookar and P. Mazumder. A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Transactions on Computer Aided Design*, 9(5):500–511, May 1990.

[12] Arvind Srinivasan, Kamal Chaudhary, and Ernest S. Kuh. Ritual: a performance-driven placement algorithm. *IEEE Transactions on Circuits and Systems*, pages 825–840, November 1992.

[13] S. Suthanthavibul, E. Shragowitz, and Rung-Bin Lin. An adaptive timing-driven placement for high performance VLSI's. *IEEE Transactions on Computer Aided Design*, 12(10):1488–1498, October 1993.

[14] H. Youssef et al. Critical path issue in VLSI design. *Proceedings of International Conference on Computer-Aided Design*, pages 520–523, 1989.

[15] H. Youssef and E. Shragowitz. Timing constraints on signal propagation in VLSI. *Proceedings of International Conference on Computer-Aided Design*, pages 24–27, 1990.