Abstract

Name:	Khaled Muhammad Walid Nassar
Title:	Timing Driven Placement Algorithm for Standard-cell Design
Major Field:	Computer Engineering
Date of Degree:	June, 1994

The performance of present VLSI circuits is dominated by interconnect delays. Therefore, there is a great demand to design new placement tools to incorporate performance measures of the circuits. This thesis presents a new method of solving the problem of timing driven placement for standard-cell design style using Genetic Algorithm approach. This problem is solved in an iterative manner using genetic operators. The timing problem is modeled in a path oriented manner, where timing constraints are imposed on total path delays which include both cell and interconnect delays. Improvements up to 17.7 % were obtained with respect to clock speed-up of the tested examples compared to the results obtained by area driven placement tools. However, the improvements in timing have resulted in a little increase between 1.4% and 9.1% in the area of the chip after routing. The thesis work is embodied in a program called Timing Driven Genetic Algorithm for Placement (TDGAP).

Master of Science Degree

King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia

June, 1994

خلاصـــة الرسالـــة

اسـم الطــالــب: خالد محمد وليد نصار
عنوان الدراسـة: خوارزمية لترتيب مواقع الخلايا المنطقية القياسية لتحسين سرعة الأداء
التـخــصــص: هنسة الحاسب الآلي
تاريخ الشهـادة: يونية، ١٩٩٤م

يعتبر التأخير في الموصلات بين الخلايا المنطقية في الدوائر المتكاملـة ذات النطـاق الواسـع العـامل الأساسي المؤثر في أداء هذه الدوائر. لهذا، أصبحت هناك حاجة ماسة لتصميم طريقة لترتيب مواقع هـذه الخلايا بحيث يأخذ بالاعتبار سرعة أداء الدوائر.

ولقد قدمنا في هذا البحث طريقـة جديـدة لـترتيب مواقـع الخلايـا المنطقيـة القياسية لتحسـين السـرعة اللازمة لعمل الدوائر، حيث اعتمدت هذه الطريقة على الخوارزميات الجينية. ولقد تـم حـل هـذه المشـكلة بطريقة التكرار باستخدام العمليات الجينية. وقد تم تصنيف مشكلة الوقت على أساس الطريق الكلـي الـذي يستخدم لنقل الاشارات الكهربائية من المصدر الى الهدف، حيث يشمل هذا الطريق الكلـي التـأخير النـاتج عن الخلايا المنطقية والموصلات بين هذه الخلايا.

ومن نتائج هذا البحث أنه تم الحصول على تحسينات بمقدار يتراوح الى ١٧,٧٪ في سـرعة الدوائـر المختبرة مقارنة مع النتائج الأخرى التي تم الحصول عليها باستخدام برامج الترتيب التي تعتمد فقط علـى تصغير مساحة الدوائر. ومن الجدير بالذكر أن هذه التحسينات في السرعة رافقتهـا بعـض الزيـادة القليلـة في مساحة الدوائر المختبرة بعد اكمال عملية التوصيل بين الخلايا المنطقية بمقدار يـتراوح بيـن ١,٤٪ و ٩,١٪. ولقد تم اتمام هذا البحث في صيغة برنامج أطلق عليه اسم:

"Timing Driven Genetic Algorithm for Placement"

و قد أختصر بالمسمى "**TDGAP**".

درجـة الماجسـتير في العلـــوم

جامعة الملك فهد للبترول والمعادن
الظهران، المملكة العربية السعودية

يونية، ١٩٩٤

Chapter 1

Introduction

The design of current VLSI circuits is of such high complexity that working with the full design on the detailed level is not practical. The design process of an integrated circuit (IC) can be divided into three sub-tasks so that it is easy to manage. First, system design which includes the description of the system in a structural form. Second, logic design which includes gate level description, logic assignment, and mapping the design into the cell library. Third, physical design which includes placement, routing, mask generation, and fabrication. Figure 1.1 represents a description of the three levels of the design process on an IC. This thesis is concerned with the placement problem presented at the physical stage.

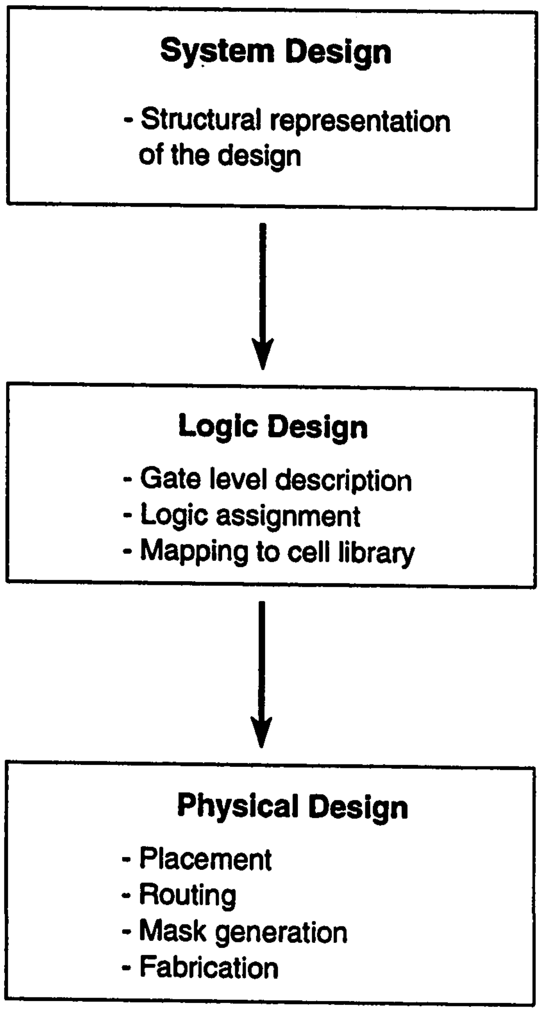

Figure 1.1: A three levels representation of the IC design process.

1.1 The Placement Problem

The placement problem for VLSI design can be defined as the process of assigning the modules or circuit elements to suitable physical locations on a layout surface. Suitable locations are those locations that minimize given objective functions, subject to certain constraints imposed by, for example, the designer or the implementation process. The most popular measure of the quality of the placement is the total wirelength [SY94].

1.2 Standard-cell Design

Designing large and dense VLSI circuits using the full custom design style is very tedious. Therefore, semi-custom approaches are used to ease the automatic mapping of logic level descriptions to mask level. These approaches include gate array, standard-cell, and general cell design styles. With the help of computer aided design tools it is possible to automate the entire layout procedure that follows the logic design step in VLSI design. Applying standard-cell design style has made this automation possible, combined with efficient software packages for automatic placement and routing. Figure 1.2 shows a standard-cell layout within a chip.

Standard-cells are logic modules with a predefined internal layout. They have

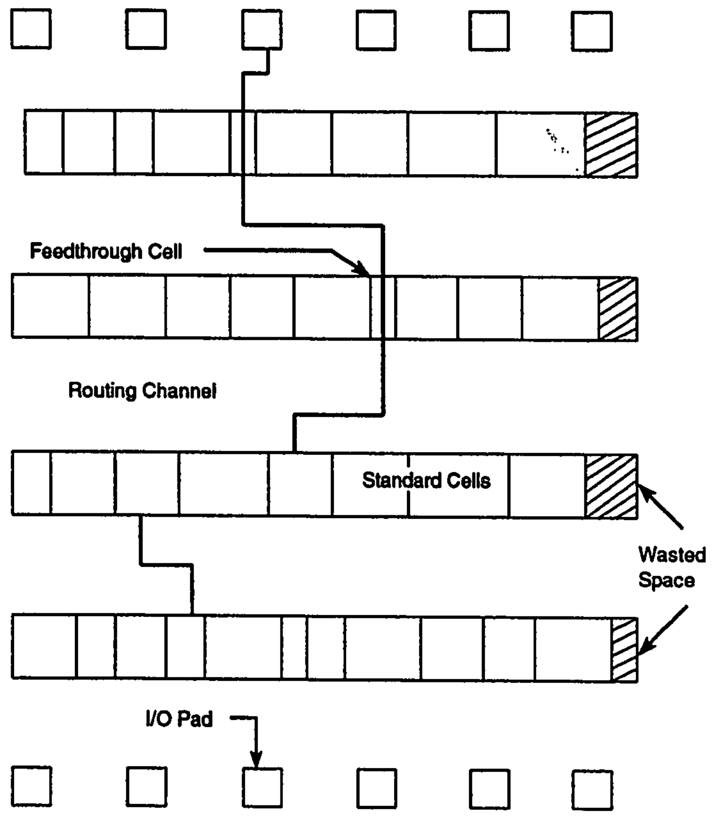

Figure 1.2: Standard-cell layout.

a fixed height but different widths, depending on the functionality of the modules. They are laid out in rows, with routing channels or spaces between rows used for laying out the interconnections between the chip components. Standard-cells are usually designed such that the power and ground interconnects run horizontally through the top and the bottom of the cells. When the cells are placed adjacent to each other, these interconnects form a connected track in each row. The cell's input and output terminals or pins are accessible along the top or bottom edge (or both). To connect them, interconnects are spread through the routing channels. In order for connections to go from one row to another, they either run through vertical wiring channels at the edges of the chip or through feed-through paths which may be contained in existing cells, or require new cells to carry the connections [Bra87]. These feed-through cells are standard-cells with interconnects running through them vertically.

Layouts using standard-cell design style do not allow easy incorporation of design requirements such as special cell or pad placement or use of special macrocells such as ROMs and PLAs. In addition, the restriction of cell placement to rows can lead to long interconnection distances and use of extra silicon area [PG78].

On the other hand, layouts using standard-cell design style are more structured. Furthermore, cells designed using this style have relatively high performance compared to gate array design style. Regarding CAD tools, there are many layout programs for practical use, which are based on the standard-cell design style [Oht86].

1.3 Cost Functions for Placement

Due to advances in device sizing, small sizes of gates have reduced their switching delay and driving power. Low driving strength of gates has made the propagation delays of the interconnects relatively more dominant. So far, on an average, these interconnect delays have been responsible for roughly half of the clock cycle.

In the past, the main objectives of most placement algorithms were to minimize the total chip area and the total estimated wire length for all the nets without regard to performance aspects. However, in today's dense CMOS FET technologies and high performance designs, typically 50-70 % of a circuit's delay is attributable to inter-circuit wiring capacitance [DNA+90][SS90]. Large wiring capacitances, besides degrading the overall performance of the circuit, usually lead to long path problems. Large interconnect capacitances are usually caused by a "bad" placement. Therefore, to avoid having placement-related long path problems, there is a great demand to make the placement procedure aware of the timing requirements of the circuit. This measure may be needed even at the cost of an increase in the total wire length. A circuit may, also, exhibit short path timing problems. However, these are easy to be corrected by the insertion of buffers or by adding metal into the paths in question [SS90][You90].

In addition to the circuit timing constraints, the placement procedure should still be constrained by geometric aspects of the problem; that is,

1. the cells should not overlap,
2. they should lie within the boundaries of the chip, and
3. standard-cells should be restricted to rows in predetermined locations.

Prior to the layout step of the design process, the actual propagation delays of the paths are not known. Therefore, timing problems that may arise from the long paths after layout are difficult to adjust.

Three ways have been suggested to solve long path timing problems. The first approach consists of making some changes to the design, for example, reducing the load on some circuit elements of a certain path. The second approach performs transistor sizing. The third approach consists of reducing the interconnect delays to a minimum by imposing timing constraints on the nets and (or) paths. This approach will make the circuit faster without changing the design [You90].

To make the placement process timing driven, timing constraints should be imposed and included into the placement objective(s) as well. For these objectives, a cost function is defined. The goal of the placement algorithm is to determine a placement with minimum possible cost.

Chapter 2

Literature Review

2.1 Classification of Placement Techniques

The placement problem is NP complete. Therefore, it can not be solved exactly in polynomial time [Don80]. For example, for a placement of n modules, the number of possible permutations of the placement can be as large as $n!$ (in the case where all modules are identical) [Bra87]. This method is, therefore, impossible to use for circuits with any reasonable number of cells. Because of this exponential nature of the problem, exact (globally optimal) solutions can not be achieved in a reasonable time. Therefore, heuristics must be used to obtain a good placement which meets a given objective within a reasonable amount of time. The two most important placement metrics are routability and timing constraints [Bra87].

Two basic approaches were applied in earlier works for the cell placement problem: constructive placement and iterative improvement [SM91][SS90]. Constructive placement approaches build a placement in a piecewise manner, where they start with a seed cell, then remaining cells are constructively selected and placed one (or a group) at a time until all cells are placed. Iterative improvement algorithms start with an initial placement and repeatedly modify it in search for a cost reduction. This iterative search stops when a given termination condition is met [SM91].

In a constructive placement algorithm a seed cell is selected and placed in the chip layout area. Then other cells are chosen one or group at a time, (for example, in order of their connectivity to the already placed cells), and are placed at an empty location near the placed cells, such that the cost function is minimized. Such algorithms are generally very fast, but may result in poor layouts. Because of their speed, they are now used for generating initial placements for iterative improvements algorithms where the value of the objective function gets substantially improved [SS90]. Constructive placement algorithm is probably the hardest type of algorithm to implement [Bra87].

An iterative improvement algorithm generally enhances the present placement in small steps [Bra87]. In general, it produces good layout solutions but need tremendous amount of computation time for checking all the constraints for any new point in the solution space [SS90]. The simplest strategy for iterative improvement is to interchange randomly selected pairs of cells and accept the interchange if it results in a reduction in cost. The algorithm continues until there is no more improvement during a given large number of trials. One way to improve this algorithm is to repeat the process many times with different initial configurations in the hope to get a good solution in one of the trials [SM91].

Another way to classify placement algorithms is whether they are deterministic or probabilistic. Deterministic algorithms work on the basis of fixed connectivity rules or formulas, or determine the placement by solving simultaneous equations. These algorithms will always generate the same layout for a certain placement problem [PG78]. However, probabilistic (or non deterministic) algorithms work by randomly testing configurations and may generate a different final layout each time they are run. Constructive algorithms are usually deterministic, whereas iterative improvement algorithms are usually probabilistic [SM91].

2.2 Placement Algorithms

In general, there are many classes of VLSI module placement algorithms. We will briefly introduce five classes of placement algorithms which are widely used. These are: simulated annealing, mincut, force-directed, numerical optimization, and genetic algorithms [SM91].

Simulated annealing is among the most popular heuristics. It is one of the best algorithms available in terms of placement quality, but it takes an excessive amount of computation time. It is derived from the analogy of the process of annealing, or the attainment of ordered placement of atoms in a metal during slow cooling from a high temperature [KGV83].

Mincut algorithms rank second in terms of placement quality but would probably be the best in terms of cost/performance ratio, since they are much faster than simulated annealing. These algorithms are based on a simple principle: cells that are densely connected to each other ought to be placed close together. This can be achieved by repeated partitioning of the given network. The main objective of the partitioning operation is to minimize the nets cut between partitions. Partitions are assigned to different areas on the layout surface. By applying this operation repeatedly on the newly obtained partitions the wire length is minimized [Bre77].

Force-directed algorithms operate on the physical analogy of masses connected by springs, where the system would tend to come to rest in the minimum energy state, with minimum tension on the springs. In the context of the placement problem the forces are proportional to the square of the wirelength. Force-directed algorithms have been around since the 1960s and were among the first algorithms to be used for placement (mainly printed circuit board placement in those days) [FCW67]. A rich variety of implementations have been developed over the years, including constructive (equation solving) methods for determining a minimum-energy configuration from scratch and two types of iterative techniques. The first consists of selecting modules one at a time and determining an ideal location for them from force considerations. The second consists of random/exhaustive pairwise interchange, with acceptance of the good moves and rejection of the bad moves, on the basis of force considerations [HWA76].

Placement is an optimization problem, and methods such as Simplex, Quadratic Programming, and the Penalty Function Method have traditionally been used for various linear and nonlinear optimization problems. Further, the placement problem can also be formulated in terms of the quadratic assignment problem, which can be solved by the eigenvalue method. Accordingly, several papers that use these techniques have been discussed under the category of numerical optimization techniques. The common feature of all these techniques is that they do not constrain the modules to rigid points or rows, hence they are more applicable to macro blocks than to standard-cells or gate arrays, although the solution generated by numerical techniques can be further processed to map the modules to the nearest grid points [Hal70].

The last class of algorithms are genetic algorithms, which although invented in the 1960s [Hol75], were not used for placement until 1986 [Gre85] [Gre87]. The genetic algorithm is an efficient search and optimization technique for problems with a large and varied search space, as well as problems with multi objective functions.

The placement problem is represented in the form of a genetic code. This is a major deviation from conventional placement algorithms that directly apply transformation to the physical layout. The genetic algorithm processes a set of alternative placements together and creates a new placement for trial by combining sub-placements from two parent placements. This causes the inheritance and accumulation of good sub-placements from one generation to the next. It also causes the mixing of the good features of several different placements that are being optimized simultaneously. Thus, the search through the solution space is inherently parallel. This parallelism of the genetic algorithm can, however, be a potential problem in terms of excessive memory size and CPU time, and unless a clever representation scheme is devised to represent the physical placement as a genetic code, the algorithm may prove ineffective [SM91]. Table 2.1 is an approximate comparison of the performance of the algorithms discussed here.

Algorithm	Result quality	Speed
Simulated annealing	Near optimal	Very slow
Genetic algorithm	Near optimal	Very slow
Force directed	Medium to good	Slow to medium
Numerical optimization	Medium to good	Slow to medium
Mincut	Good	Medium
Clustering and other constructive placement	Poor to Medium	Fast

Table 2.1: A rough comparison of the performance of placement algorithms.

2.3 Timing Driven Placement

For the placement problem many approaches have been reported which optimize not only for wirability, but also for timing. The circuit timing performance is determined by the maximum clock rate that gives a correct functional response. However, the maximum clock rate is bounded by the longest path delay present in the circuit. The path delay is the time needed by a signal propagating along a path between two input/output pads or storage elements [ID90].

One of the earlier approaches to include timing aspects of the design with the placement procedure consisted of, first finishing the placement, then performing timing analysis. Individual cells were allowed to change their positions whenever timing verification indicated that some of the paths did not meet their timing constraints [DNA+90].

One possible way to include timing information into the placement stage is to transform the path-oriented timing constraints into constraints (bounds) on the nets. This is based on the path margin, called SLACK, which is the difference between the required arrival time and the actual arrival time at the sink of the path. The constraint of a net is calculated in terms of the criticality of the paths involved. The cells that are connected with tightly constrained nets are placed closer together so as to satisfy the net bounds and therefore the timing constraints [ID90], [JKMS87], [D+87], [HNY87], [OI+86].

More recent strategies share the idea of limiting net capacitance. In one technique, the placement program is directed to individually shorten those nets contained in critical paths as determined by timing analysis. However, because placement algorithms based on capacitance limits optimize nets without regard to paths (which are combinations of nets), they do not address the fact that timing is not determined by the behavior of particular nets, but rather by paths [DNA+90]. In other words, the performance of a design tends to be path-oriented in nature and the minimization of total net length may not lead to the improvement of the circuit's performance [ID90]. The authors of [DNA+90] use the approach of simulated annealing [SSV85], [VK83]. Simulated annealing generally results in placements with good wirelength and performance.

Another recent approach, also described in [DNA+90] uses a hierarchical technique to solve this problem. It uses capacitive weights based on timing analysis which are recalculated at each level of the hierarchy. In [JK89], a system called Allegro is developed based on a hierarchical technique. In this approach, linear programming is used to dynamically optimize the performance of the chip during placement. It has been tested on Sea-of-Gate designs

In [SCK92], a system called RITUAL is developed to find an approximate solution to the problem by using mathematical techniques and heuristics based on Lagrangian relaxation. It is solved in two phases: continuous and discrete. An average improvement of 8% to 30% in the interconnect delay were reported compared to the results of an industry standard simulated annealing based placement package TimberWolf 5.6 [SL88].

2.4 Placement Routability

The general routing problem is an NP-hard problem [Bra87]. Therefore, the complex task of routing all nets of the circuit can be reduced by first performing what is called loose or global routing, followed by detailed routing. The purpose of the global routing, also known as topological routing, is to decompose a large problem into smaller manageable ones. This is done by first assigning nets to routing regions called channels. Initial assignment of nets to channels helps in reducing wire lengths.

Furthermore, the global router can be used to check the routability of the circuit before subjecting it to time consuming detailed routing.

Conventional routing sequences for standard-cell design would be: (1) channel definition, (2) global routing, (3) channel (detailed) routing [Bra87].

In standard-cell placement, global routing can be used to determine the required separation between the cells rows in order to ensure routability of the chip. Channels between the rows are used to connect the cells. However, vertical wiring is needed to connect cells placed in non-adjacent rows. This is done by inserting some feed-through cells. Having to insert feed-through cells complicates the placements, because it is difficult to know exactly where a feed-through will be needed. Therefore, the placer has to effectively anticipate what the router will be doing. Furthermore, the insertion of feed-through cells will perturb the placement which may cause cells to change rows [Bra87]. This may happen if there is a restriction on the maximum row length.

Chapter 3

Modeling and Constraints

3.1 Problem Definition

In this work we address the problem of timing driven placement. Given a VLSI design, with no macro blocks, consisting of modules (cells) with predefined input and output terminals and interconnected in a predefined way, the task is to design and implement a placement algorithm to place the modules such that the placement solution does not violate predefined timing constraints. Furthermore, the standard-cell design style is adopted in our work. Satisfying timing constraints may in turn affect the overall wire length of the chip and cause it to increase. Therefore, a careful representation of the objective function is required so as to balance these two important constraints. In addition, routability will be taken into consideration so that the final layout of the design is routable. The main objective of our placement algorithm is to reduce the wiring delays associated with critical paths by minimizing their wire length. The objective function adopted is discussed in detail in Section 5.7.

3.2 Inputs and Outputs

The standard-cell design style is adopted in our system. The system takes the following as inputs:

- A netlist describing the interconnections between the terminals of the modules of the circuit. Two input formats are accepted by our system, AHPL [MS86] and VPNR [MCN90] netlists. A brief description is introduced in the next section.
- The aspect ratio of the chip, that is the ratio of height and width. This gives the user some control on the final layout shape.
- A standard-cell library that includes the cell's description consisting of their delay parameters, dimensions (h, w), and the locations of their terminal points. The cells used in our design are in 2μ SCMOS technology obtained from the *OASIS* SCMOS standard-cell library [MCN90].

- Timing information that includes the predicted K most-critical paths and their SLACKs, the nets covered by those K most-critical paths. This information is obtained from a pre-placement timing analysis of the circuit. Furthermore, the placement program is interfaced with the *OASIS* system [MCN90].

The output of the program is a list of the physical x-y location of each cell in VPNR format. The generated solution is reported with the remaining $SLACK$, if any, and its estimated overall wirelength. The generated solution can then be fed to the detailed router of the *OASIS* layout system. This detailed router uses the left-edge algorithm [HS71] [PDS76]. Following detailed routing, the mask layout of the circuit is viewed using the *MAGIC* layout system [MAS+90]. *MAGIC* allows viewing and extraction of the circuit for validation. An overview of the design steps is given in Figure 3.1.

3.3 AHPL and VPNR Formats

AHPL and VPNR netlist formats are two different ways to represent a net list of a certain circuit. The AHPL [MS86] net list is contained in two main files: the gate list and the I/O list. The gate list file includes the gate number, gate type, input link, and output link respectively. An example of such file is given in Figure 3.2. The I/O list file includes a link number, gate(1) number, gate(2) number, and next

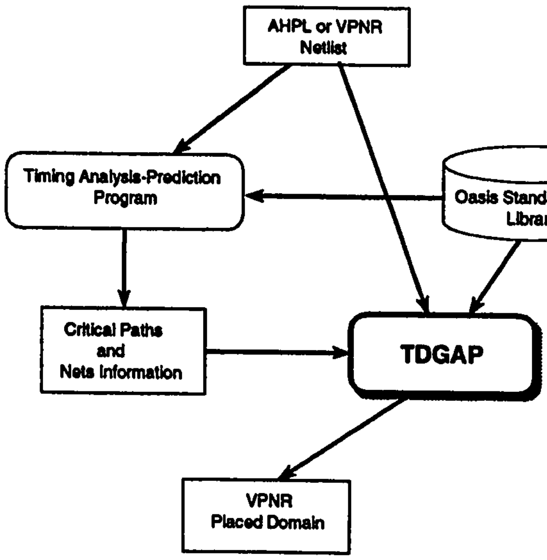
AHPL or VPNR Netlist
Timing Analysis-Prediction Program
Oasis Standard Cell Library
Critical Paths and Nets Information
TDGAP
VPNR Placed Domain

Figure 3.1: A representation of the design process.

link number. An example of such file is given in Figure 3.3. For example, the gate list of Figure 3.2 has the gate number '123' which is of type '4401' (type '4401' is a four inputs AND gate) and it has an input link number '700' and output link number '1200'. From Figure 3.3, the link number '700' leads to four gates '20', '30', '22', and '34', which means that gate number '123' has four inputs coming from four gates, namely '20', '30', '22', and '34'. The link '1200' leads to one gate '124', which means that gate number '123' is feeding only one gate, namely '124'.

The VPNR format has two possible variants of a domain description, placed and unplaced. A sample file of the unplaced domain is given in Figure 3.4. The first line begins with the keyword *domain begin*, followed by the domain name, *example*, and other domain attributes. The other two lines starting with the keyword *profile*

gate-number	gate-type	in-link	out-link
.	.	.	.
.	.	.	.
20	4402	300	350
22	4403	120	150
30	4404	10	20
34	4203	60	65
.	.	.	.
.	.	.	.
123	4401	700	1200
124	4050	200	456
.	.	.	.
.	.	.	.

Figure 3.2: An example of an AHPL gate list file.

link	gate1-number	gate2-number	next-link
.	.	.	.
.	.	.	.
700	20	30	400
400	22	34	0
.	.	.	.
.	.	.	.
1200	124	0	0
.	.	.	.
.	.	.	.

Figure 3.3: An example of an AHPL I/O list file.

```
domain begin example lib=scmos2.0
profile top (0,0) (0,0) ;
profile bot (0,0) (0,0) ;
iolist
a T: (0,100) pintype=pi
b T: (0,100) pintype=pi
nand B: (0,100) pintype=po
;
row 1
ai2s INSI1 (a,b,q1)
i4s INSI2 (q1,nand)
;
domain end example
```

Figure 3.4: An example of an unplaced domain of a VPNR file.

describe the domain shape. The keyword *iolist* indicates the beginning of the input and output signals. This part, is then ended with a semicolon. After the semicolon, the keyword *row* followed by the row number starts the section describing the domain interface. The row description ends with a semicolon, followed by the key word *domain end*, followed by the domain name. For the unplaced domain, there is always exactly one row of cells. On the other hand, for the placed domain, there are more than one row description of the cells. More details can be found in [MCN90].

3.4 Timing Issues

The performance of today's integrated circuits is largely dominated by the interconnects between the modules. In this work, we assume that wiring between the cells is performed on two metal layers. All vertical wires run on one layer using Metal2, while all horizontal wires run on another layer using Metal1. Metal wires introduce large capacitance that load the logic modules and significantly change the timing properties of the design. This significant role of interconnects emphasizes the need to reconsider traditional physical design tools so that they are performance driven. In the following sections we present some important timing concepts and introduce the timing model used in our design for making the placement driven by the timing constraints of the design.

3.4.1 Timing Analysis Concepts

A path is defined as the sequence of circuit modules that a signal must travel through from a starting point to an ending point. In synchronous circuits, four types of paths are defined, depending on the type of the starting and ending points. These are: a primary input to a sequential cell, a sequential cell to a sequential cell, a sequential cell to a primary output, and a primary input to a primary output. An illustration of the path concept is shown in Figure 3.5.

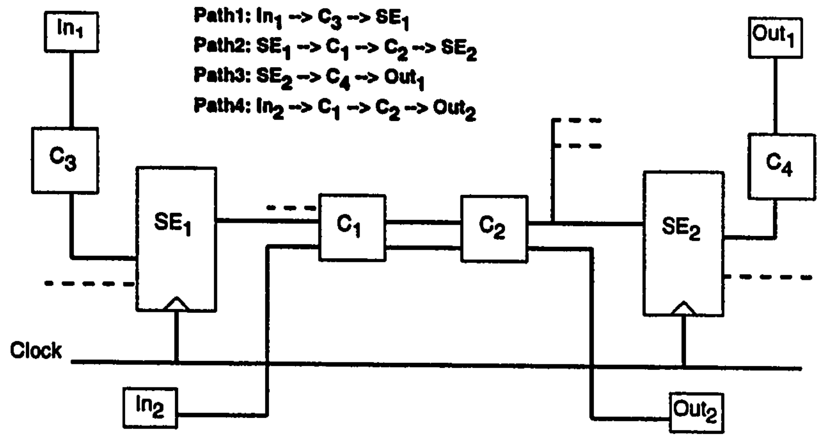
Path1: In1 --> C3 --> SE1
Path2: SE1 --> C1 --> C2 --> SE2
Path3: SE2 --> C4 --> Out1
Path4: In2 --> C1 --> C2 --> Out2
In1
C3
SE1
C1
C2
SE2
C4
Out1
Clock
In2
Out2

Figure 3.5: Four types of paths are defined in synchronous designs.

A short path problem occurs when a signal arrives at its destination too early. On the other hand, a long path problem occurs when a signal arrives at its destination too late. The problems caused by long paths are more difficult to adjust after the design process has been completed, than those caused by short paths. For example, for the circuit to operate with the same predefined clock period, a short path problem can be easily solved by inserting some buffer logic along the short path. However, for a long path problem, more complicated solutions are used as described in Section 1.3. Because circuit performance is highly depended on long paths, the major objective in our design is directed toward solving long paths.

Let $S(i)$ be the $SLACK$ of path i, given by

$$S(i) = LRAT - T(cells) - T(nets) \tag{3.1}$$

where,

LRAT = *Latest Required Arrival Time* of a signal at its sink,

$T(cells)$ = the switching delay of the cells on that path,

$T(nets)$ = the interconnect delays of the nets on that path.

Then, for a given chip to have a correct performance, it is a necessary condition that $S(i) \geq 0$ for all the paths in the circuit. The path whose $SLACK$ value is most negative is defined as the most critical path in the design. Therefore, to achieve a circuit with a correct performance the minimum $SLACK$ has to be maximized by minimizing critical path delay.

3.4.2 Timing Model

In this section, the delay model used in our design is presented. This delay model is called the linear model. It assumes that for a given net, when a signal starts from its source, the sinks will be equally charged at the same time. So, the time needed for the signal to reach any of the sinks is equal. For a given path π, the total delay is computed as follows:

$$Delay(\pi) = \sum_{cell \in \pi} Delay(cell) + \sum_{net \in \pi} Delay(net) \tag{3.2}$$

Before we proceed with a description of the net and cell delay computations, we first need to introduce the following notations. Let,

ID_i = Interconnect Delay of net i,

CD_i = Overall Cell Delay of cell i,

BD_i = Base Delay of cell i,

ACL_i= Total Load Capacitance on the input pins driven by net i,

LF_i = Load Factor of cell i,

C_i = Interconnect Capacitance of net i,

R_i = Interconnect Resistance of net i,

then, the interconnect delay along net i, which is driven by cell i, is computed as follows:

$$ID_i = LF_i \times C_i + C_i \times R_i + R_i \times ACL_i \tag{3.3}$$

The switching delay of a cell i, ignoring the interconnect capacitance is,

$$CD_i = BD_i + LF_i \times ACL_i \tag{3.4}$$

For any net n, the interconnect capacitance C_n and the interconnect resistance R_n are computed as follows:

$$C_n = Area\_capacitance + Fringe\_capacitance \tag{3.5}$$

where,

$$Area\_capacitance = (C_{m1} \times Lm1 + C_{m2} \times Lm2) \times w \tag{3.6}$$

$$Fringe\_capacitance = 2 \times ((w + Lm1) \times Cf_{m1} + (w + Lm2) \times Cf_{m2}) \tag{3.7}$$

$$R_n = \frac{(R_{m1} \times Lm1 + R_{m2} \times Lm2)}{w} \tag{3.8}$$

where,

C_{m1} = Plate Capacitance/Area of Metal1,

Cf_{m1} = Fringe Capacitance/unit length of perimeter of Metal1,

C_{m2} = Plate Capacitance/Area of Metal2,

Cf_{m2} = Fringe Capacitance/unit length of perimeter of Metal2,

R_{m1} = Sheet Resistance of Metal1,

R_{m2} = Sheet Resistance of Metal2,

$Lm1$ = Length of Metal1,

$Lm2$ = Length of Metal2,

w = Width of Metal1 or Metal2 interconnects.

For example, for a given path π as in Figure 3.6, the total delay on that path is computed as follows,

$$Delay(\pi) = Cells\_Delay_{\pi} + Interconnect\_Delay_{\pi} \tag{3.9}$$

where,

$$Cells\_Delay_{\pi} = CD_{SE_1} + CD_{C_1} + CD_{C_4} + CD_{C_5} + CD_{SE_2} \tag{3.10}$$

$$Interconnect\_Delay_{\pi} = ID_{SE_1} + ID_{C_1} + ID_{C_4} + ID_{C_5} \tag{3.11}$$

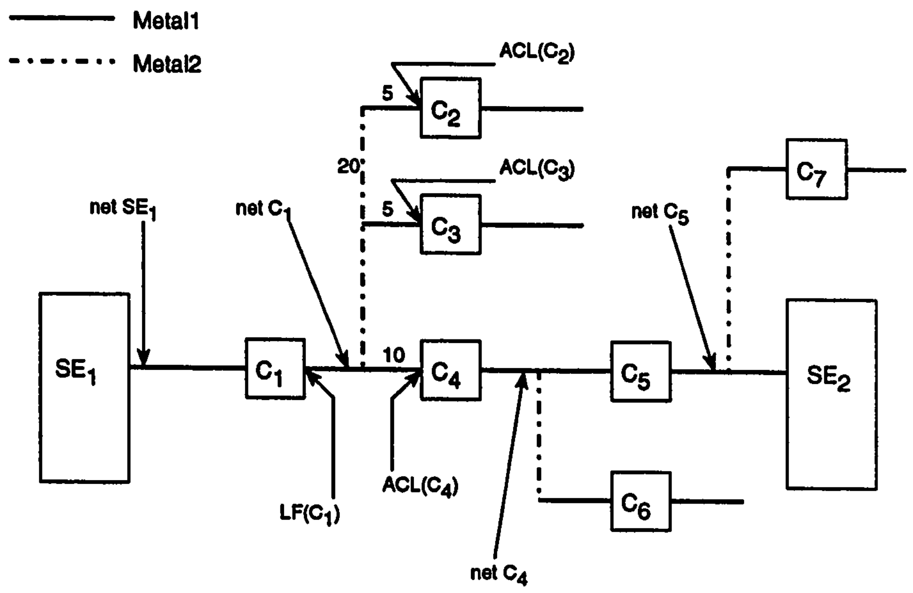

Figure 3.6: Linear Delay Model of a path. Path $\pi : SE_1 \rightarrow C_1 \rightarrow C_4 \rightarrow C_5 \rightarrow SE_2$.

then, for example, the value of the interconnect delay of net C_1 shown in Figure 3.6 is computed as follows:

$$ID_{C_1} = LF_{C_1} \times C_{C_1} + C_{C_1} \times R_{C_1} + R_{C_1} \times ACL_{C_1} \tag{3.12}$$

where,

$$ACL_{C_1} = ACL(C_2) + ACL(C_3) + ACL(C_4) \tag{3.13}$$

Based on our assumption that Metal1 is used for horizontal wires and Metal2 is used for vertical wires, then for net C_1 we have, $L_{m1} = 10 + 5 + 5$, and $L_{m2} = 20$. Therefore, the interconnect capacitance and the interconnect resistance are,

$$C_{C_1} = (C_{m1}\times(10+5+5)+C_{m2}\times 20)\times w+2\times((w+10+5+5)\times Cf_{m1}+(w+20)\times Cf_{m2}) \tag{3.14}$$

$$R_{C_1} = \frac{(R_{m1} \times (10+5+5) + R_{m2} \times 20)}{w} \tag{3.15}$$

To complete the computation of Equations 3.12, 3.13, 3.14, and 3.15 let,

$ACL(C_2) = 0.076\, pF,$

$ACL(C_3) = 0.068\, pF,$

$ACL(C_4) = 0.083\, pF,$

$C_{m1} \quad = 0.26 \times 10^{-4}\, pF/\mu^2,$

$Cf_{m1} \quad = 0.82\, pF/\mu,$

$C_{m2} \quad = 0.15 \times 10^{-4}\, pF/\mu^2,$

$Cf_{m2} \quad = 0.85\, pF/\mu,$

$R_{m1} \quad = 0.060\, \Omega/square,$

$R_{m2} \quad = 0.033\, \Omega/square,$

$w \quad = 3\, \mu,$

then the values of Equations 3.13, 3.14, and 3.15 are:

$ACL_{C_1} = 0.227\, pF$

$C_{C_1} \quad = 0.0246 + 0.007682 = 0.032282\, pF$

$R_{C_1} \quad = 0.62\, \Omega$

then if $LF_{C_1} = 4.13\, ns/pF$ the value of Equation 3.12 is:

$ID_{C_1} = 0.13332 + 0.00002 + 0.0001407 = 0.1335\, ns.$

From the above example and from experimental results, we observed that the dominating term in Equation 3.3 is $(LF_i \times C_i)$. The delay due to the value of $(LF_i \times C_i)$ was big compared to the delay due to the value of the other two terms. The contact resistance that connects the two layers of interconnects running horizontally and vertically, has been neglected because the delay due to its value was small compared to the delay due to the other variables in Equation 3.3.

The computation of C_i has included two terms, one for the plate (area) capacitance and the other is for the fringe capacitance. For circuits with size of about 250 cells, it has been shown that the fringe capacitance dominates the plate capacitance. For example, using the above values of C_{m1}, C_{m2}, Cf_{m1}, Cf_{m2}, and w for a two pins net i with $Lm1 = 1000\,\mu$ and $Lm2 = 1000\,\mu$ we find that the values of the $Area\_capacitance$ and the $Fringe\_capacitance$ are:

$$Area\_capacitance = (0.26 \times 10^{-4} \times 1000 + 0.15 \times 10^{-4} \times 1000) \times 3 = 0.123\, pF$$

$$Fringe\_capacitance = 2\times((3+1000)\times 0.82\times 10^{-4}+(3+1000)\times 0.85\times 10^{-4}) = 0.335\, pF$$

From this example we can note that the value of the $Fringe\_capacitance$ is about 63% more than that of the $Area\_capacitance$.

3.4.3 Prediction of the Critical Paths

From past layouts of circuits with similar complexity, the average and standard deviation of net length are estimated for each type of net (2 pin-, 3 pin-,..., n pin-nets). These are converted to capacitances for the particular technology of the design at hand.

To simplify the explanation, we shall assume that all cells are single output cells. Let $\pi = [c_1, c_2, \cdots, c_i, \cdots, c_{m+1}]$ be a particular path from cell c_1 to cell c_{m+1}. Under the assumption that the nets are statistically independent, the expected delay on path π can be expressed as given in Equation 3.2.

Similarly, the delay variance on path π can also be computed as follows,

$$S_\pi^2 = \sum_{i=1}^{k}(LF_i^2 \cdot S_i^2) \tag{3.16}$$

where, S_i^2 is the variance on the capacitance of net i as estimated from past designs.

Let $T_{\max}$ be the estimated delay of the longest path in the design, that is,

$$T_{\max} = \max_\pi\{T_\pi\} \tag{3.17}$$

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

$$T_\pi + \alpha \cdot S_\pi \geq T_{\max} \tag{3.18}$$

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. We will refer to the number of *α-critical* paths as the K most critical paths. Reasonable values of α are ≤ 3 or 4.

The algorithm used to enumerate all paths which satisfy Equation 3.18 is investigated in [AF]. It is a variation of the algorithm reported in [AS85]. The prediction strategy described in this section achieved a 100% hit ratio with all test cases that we experimented with.[1] Nevertheless, one has to be careful as to when such a strategy is possible. In our case, the variances on net lengths were relatively much smaller than the overall path interconnect length. However, it may happen that for design styles other than the standard-cell design style (used in this work) and very dense and large designs the net length variances would be unacceptably large. For example, such a phenomenon has been observed in [YSS92] for very dense CMOS sea-of-gates chips.

Our placement program then computes Equation 3.1 for each of the K reported critical paths. These values are compared with those reported by the timing analysis program to verify the correctness of the critical paths so they are free from long path problems. This means that the operation of our program is very sensitive to the reported K most critical paths. Therefore, the accuracy of the resulting layout will heavily depend on the effectiveness of the delay model used by the timing analysis

[1]The critical paths enumerated after placement were a proper subset of the predicted paths.

program and the correctness of its reported timing information which is supplied to the placement procedure.

3.5 Wirelength Model

The performance of a certain placement algorithm can be affected by the way it estimates the wirelength of its nets. In our work, the semi-perimeter method is used with the assumption that all the routing is done using the Manhattan geometry in which routing channels are either horizontal or vertical. The semi-perimeter estimate is one of the most popular and fast methods used for estimating the wirelength of a net. This method starts by finding the smallest bounding rectangle that encloses all the pins connected to a net, (see Figure 3.7). Then, the estimated wirelength of the net is one half the perimeter of its bounding rectangle.

This method works very well for nets with two or three pins, which is the case for most nets in practical circuits. However, for nets with more than three pins, this method under-estimates the wirelength. To improve the accuracy of this estimation, an inflation factor of 20% has been included for nets with more than three pins. This value for the inflation factor was chosen based on experiments. Using this inflation factor has resulted in a better estimate for the wirelength. It should be mentioned that all pins on the cells used in our work are accessible from both top and bottom

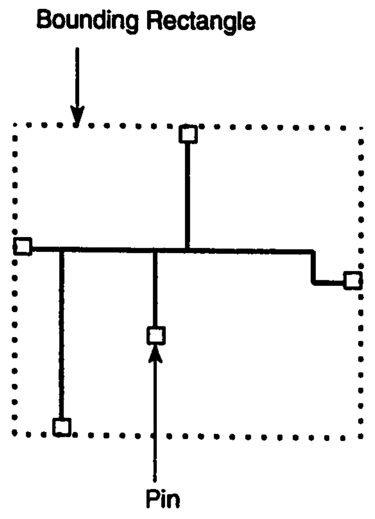

Figure 3.7: Bounding rectangle for net with five pins.

sides. This helps in connecting nets with pins on different rows. For example, if we apply the semi-perimeter method on the nets with different sizes as shown in Figure 3.8, the resulting wirelength estimate will be:

For the two-pin net in Figure 3.8(a):

$$Actual = 5 + 8 = 13$$

$$Estimated = 5 + 8 = 13$$

For the three-pin net in Figure 3.8(b):

$$Actual = 3 + 8 + 4 = 15$$

$$Estimated = 7 + 8 = 15$$

For the four-pin net in Figure 3.8(c):

$$Actual = 8 + 3 + 9 = 20$$

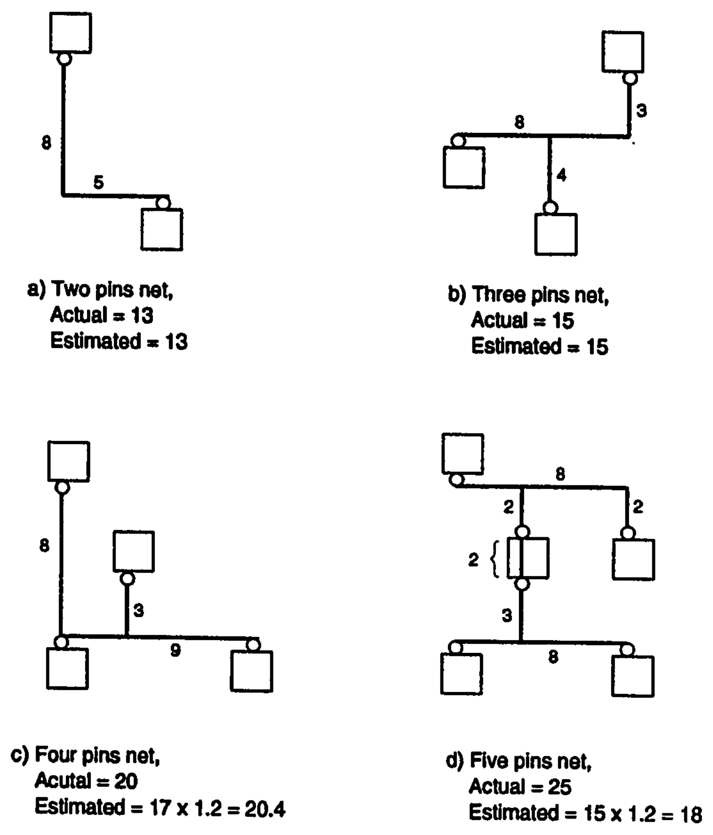

Figure 3.8: The effect of semi-perimeter estimate on different net sizes.

$$Estimated = 8 + 9 = 17$$

$$Estimated\ with\ 20\%\ inflation = (8 + 9) \times 1.20 = 20.40$$

For the five-pin net in Figure 3.8(d):

$$Actual = 8 + 2 + 2 + 2 + 3 + 8 = 25$$

$$Estimated = 8 + 7 = 15$$

$$Estimated\ with\ 20\%\ inflation = (8 + 7) \times 1.20 = 18$$

To estimate the width of a routing channel, we have used the vertical constraint graph of that channel as a measure for estimating its width. The length of the longest path in that graph represents a lower bound on the number of tracks required for that channel. For example, to estimate the width of a channel, say c, we first build the vertical constraint graph, then find the length of the longest path in that graph, say Min_tracks. After that, the estimated value of the width of channel c, $Channel\_width(c)$ in microns is computed as follows:

$$Channel\_width(c) = Min\_tracks \times Track\_width \times Channel\_factor$$

where,

$Channel\_factor = 8.5$,

$Track\_width = 3\mu$.

The value of the $Channel\_factor$ is determined based on some analysis done on complete layouts (placed and routed) which have been placed by our program and routed by $OASIS$. The analysis involved collecting some data of channels width after routing. For each channel, the actual width after routing is found from the layout. Then, the actual width is divided by the $Track\_width$ (3μ) and by the Min_tracks of the current channel. The result of this division is what we call the $local\ factor$. By averaging the collected data of the $local\ factors$ we have found that the $Channel\_factor$ is about 8.5. This value accounts for the separation between routing tracks, which is determined by the design rules[2], and for those extra tracks that are not included in the vertical constraint graph.

3.6 Placement of Main I/O Pins

The main input and output pins of the circuit which will be connected to the I/O pads of the chip are assigned to the periphery of the chip. In our design, only the top and bottom sides of the chip are used for placing the main I/O pins. Fixing some of those I/O pins to the top, and others to the bottom, affects the solution in a bad direction compared to having those I/O pins free to move between top and bottom sides. This is because there might be some I/O pins on one side, for example top side, that are connected to cells that are near the bottom side. In this case it would be better for these I/O pins to be placed at the bottom side. Therefore, in our program we have implemented a procedure that finds the best side for placing each I/O pin.

[2]In our design, the minimum separation between two tracks of Metal1 is 4 μ.

	Area_capacitance $(10^{-4}pF/\mu^2)$			*Fringe_capacitance* $(10^{-4}pF/\mu)$		
	Min	Typ	Max	Min	Typ	Max
Metal1	0.21	0.23	0.26	0.75	0.79	0.82
Metal2	0.13	0.14	0.15	0.78	0.81	0.85

Table 3.1: Values of the Area and Fringe Capacitance of Metal1 and Metal2.

	Sheet_resistance $(ohms/square)$		
	Min	Typ	Max
Metal1	0.050	0.055	0.060
Metal2	0.022	0.028	0.033

Table 3.2: Values of the Sheet Resistance of Metal1 and Metal2.

3.7 The Technology Used

Cells in the standard-cell design style have been adopted in our work. For these cells we have used a 2μ p-well CMOS technology. The parameters of these cell, such as timing characteristics and dimensions are given in [MCN90]. Furthermore, it is assumed that routing is done using Metal1 for horizontal tracks, and Metal2 for vertical tracks. The values of the capacitance and resistance of these metal types are obtained from an industrial manual [Orb92][3]. These values are given in Tables 3.1 and 3.2. In our program we have used the worst case 'Max' value of these numbers.

[3]These values are dated July 1992.

Chapter 4

Genetic Algorithm and the Placement Problem

The Genetic Algorithm (GA) was introduced first by John Holland in 1975 at the University of Michigan. Since then it has found a lot of interest among researchers in different fields of science and engineering. Table 4.1 lists some of the early applications of GA. The rest of this chapter presents GA basics and its application to the placement problem.

Year	Investigators	Description
	Computer Science	
1985	Rendell	GA search for game evaluation function
1986	Cohoon and Paris	GA for Placement
1987	Raghavan and Agarwal	Adaptive document clustering using GA
1990	Shahookar and Mazumder	GA for Placement
	Engineering & Operations Research	
1985	Grefenstette and Fitzpatrick	Test of simple genetic algorithm with noisy functions
1985	Schaffer	Multiobjective optimization using GAs with subpopulations
	Image Processing & Pattern Recognition	
1985	Gillies	Search for image feature detectors via GA
1987	Stadnyk	Explicit pattern class recognition using partial matching

Table 4.1: Genetic algorithm applications in search and optimization.

4.1 The General Genetic Algorithm

In this section, we give a brief overview of the Genetic Algorithm. For more detailed information, we refer the reader to [Gol89]. GA is an exploratory procedure based on the process of natural selection and natural genetics. GA has four distinctive features [Fre93], [Gol89]:

1. GAs work with the representations of solutions, rather than the solutions themselves.

2. With GA, the search in the solution space is inherently parallel, as opposed to a point by point search. It can search many zones of the search space at the same time. This characteristic has made GA less likely to be trapped in local minima.

3. GA does not use derivatives or other auxiliary information. It only requires values of the objective function associated with individual solutions.

4. Transitions in GA are based on probabilistic rules, not deterministic rules. This gives GA the broad search scheme which helps it to get out of local minima.

For GAs to work correctly, two basic assumptions are made:

- the fitness value of an individual is a correct estimate of its ability to solve the problem, and
- the integration of individuals allows the construction of improved offsprings.

```
Algorithm (Genetic_Algorithm)
    $N_p$= Population Size.
    $N_g$= Number of Generations.
    $N_o$= Number of Offsprings.
    $\mathcal{P}_c$= Crossover Probability.
    $\mathcal{P}_\mu$= 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 $N_o$
                    (*CHOICE*)
                $(x, y) \leftarrow$ Choose_parents
                    (*CROSSOVER*)
                With probability $\mathcal{P}_c$ Apply
                    Offspring[$j$] $\leftarrow$ Crossover$(x, y)$
                    (*MUTATION*)
                With probability $\mathcal{P}_\mu$ Apply
                    Mutation(Population[$j$])
                Evaluate Fitness(Offspring[$j$])
            EndFor
                    (*SELECTION*)
            Population $\leftarrow$ Select(Population, offspring, $N_p$)
        EndFor
    Return highest scoring individual in Population
    End
```

Figure 4.1: A simple Genetic algorithm.

A high level algorithmic description of a basic genetic algorithm is given in Figure 4.1. To start a GA, a clever representation of the solutions is developed. Then the GA starts with an initial set of solutions called population. Population is made of individuals which are also called chromosomes. These chromosomes are made of genes. On each iteration, the individuals of the current population are evaluated using some measure which is called the *fitness* value. The survival of potential individuals is determined by their fitness value. On the other hand, individuals with low fitness values are more likely to die off. After the fitness evaluation, a number of operators are applied on the current population which will generate a new population that preserves the old but good characteristics. Four major operators are applied in each iteration. These are, *choice*, *crossover*, *selection*, and *mutation*. More information about genetic algorithms can be found in [Gol89].

4.1.1 Choice

The choice operator is a preparatory step for the crossover operation. It chooses two parents (individuals) at a time to participate in the creation of a new offspring. In its simplest form, for a population P, a parent $s \in P$ with a cost value $cost(s)$ is chosen for crossover with probability Z_s defined as follows:

$$Z_s = \frac{w_s}{\sum_{i \in P} w_i}$$

where,

$$w_s = \frac{max\{cost(t) \mid t \in P\}}{cost(s)}$$

This scheme selects individuals with low cost values at a higher probability than those with high cost values. It is called the ***roulette wheel*** scheme and is based on the concept of stochastic sampling with replacement. Several researchers have examined a number of other schemes, aimed at the minimization of the stochastic errors related to the roulette wheel scheme. Brindle A. has examined several schemes in her doctoral dissertation (1981) titled *"Genetic algorithms for function optimization"* [Gol89]. Stochastic remainder without replacement is one of these schemes which has been proven by Booker L. B. in his doctoral dissertation (1982) titled *"Intelligent behavior as an adoption to the task environment"* [Gol89], to be superior over the expected value. Because of the superiority of this scheme, it has been adopted in our GA implementation. The details of this scheme will be discussed in Section 5.3.

4.1.2 Crossover

Crossover is the main genetic operator. It is performed on the chosen parents with a certain probability $\mathcal{P}_c$. In other words, after the parents are chosen using the choice operator a random number between 0 and 1 is generated. Then, if this number

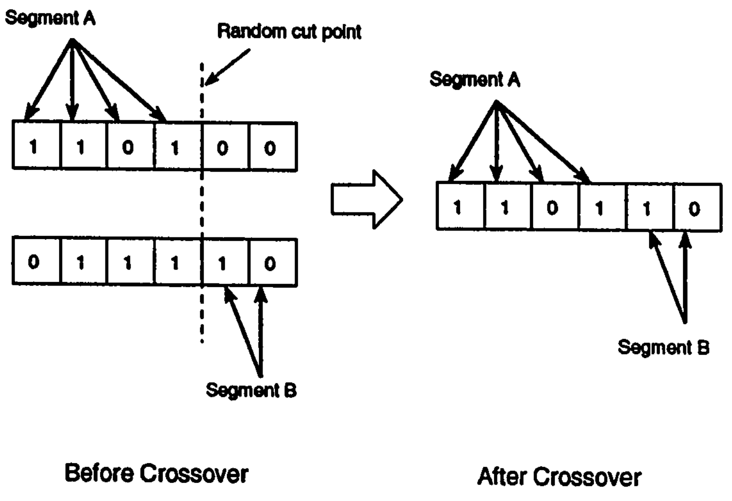

Figure 4.2: A simple crossover operation.

is greater than or equal to $\mathcal{P}_c$ crossover is performed, otherwise, it is not. The operation of crossover results in the generation of an offspring for some pairs of parents. The generated offspring inherits some characteristics (good or/and bad) from both parents in a way similar to natural evolution. An example of this operation is depicted in Figure 4.2. In this example we assume that each individual consists of a string of 0s and 1s. Given two strings (parents), a random cut point is chosen. Then the offspring is generated by combining the segment of one parent to the left of the cut point (segment A) with the segment of the other parent to the right of the cut point (segment B). This example shows a simple crossover operation. More sophisticated crossover operations are usually used in GAs. The overall progress of a GA is highly dependent on the choice of the crossover operation.

4.1.3 Selection

This operation is responsible for the selection of individuals for the new generation. It operates on the combined set of parents and offsprings. Many variations of this function have been used by researchers. Three selection methods are most commonly used. These are: competitive, random, and stochastic.

In competitive selection, only the P fittest individuals are selected, where P is the population size. In random selection, the individuals are selected at random with uniform probability. Finally, in stochastic selection, an individual is selected with probability proportional to its fitness. It is similar to the roulette wheel described earlier for the choice of parents for crossover.

4.1.4 Mutation

Mutation is the process of injecting new information in the population. It is a secondary operation which happens occasionally (with small probability $\mathcal{P}_\mu$) to protect against the loss of important information. Furthermore, it helps in introducing some variations in the solutions in order to avoid getting trapped in local minima.

The most popular mutation method is pairwise interchange of two genes that are selected at random as shown in Figure 4.3. In this example, the gene c is swapped

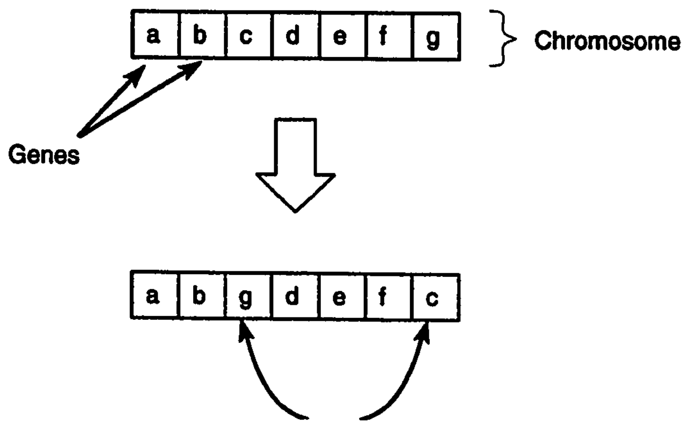

Figure 4.3: The mutation operation using a simple pairwise interchange.

by mutation with the gene g.

4.1.5 GA Parameters

Many parameters, such as the crossover and mutation probabilities, are used with GA to control its operations. The best values for these parameters are problem dependent. However, it is possible to follow some general guidelines. First, the population size should not be too small compared to the size of the search space, otherwise, it will be difficult to efficiently search the entire space. Second, a low mutation probability is recommended, so as to avoid the disturbance of the steady improvement resulting from crossover. In general, it has been found that for the placement problem a population size of 30 individuals, a crossover probability of 60%, and a mutation probability of 3% are good starting values.

4.2 Placement by Genetic Algorithm

The GA is one of the many heuristic techniques that were applied to the placement problem. The reported applications of GA to placement aimed mainly at obtaining a placement configuration with minimum total wirelength. In this section, we give an overview of some of the proposed GAs for solving the placement problem including a brief discussion of the crossover and mutation operators that are applied.

4.2.1 Genie

Genie is a genetic placement algorithm developed by Cohoon and Paris [CP87]. The authors did not mention what design style was used with Genie. The following is a description of some of the applied functions and their results.

1. *Initial population construction.* Three constructors are considered. These are: (1) cells are placed randomly, (2) cells are clustered based on their nets; the cells are linearly ordered then folded to generate the rows. (3) same as (2) but cells are placed in row major style. This is shown in Figure 4.4. Their observations have indicated that a mixed initial population of size 50 (75% using the third method and 25% using the first method) resulted in a good initial mean population score while having a satisfactory amount of diversity.

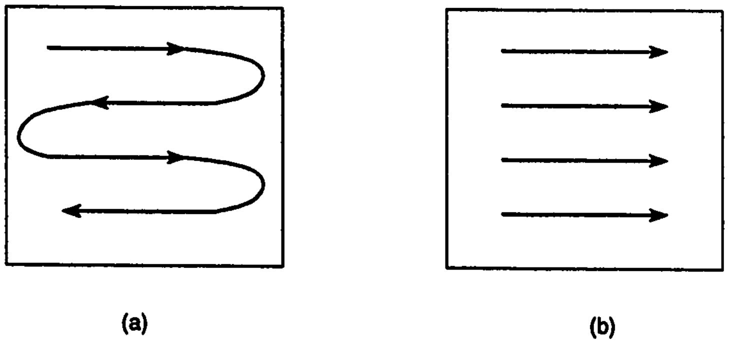

Figure 4.4: Order of slot assignment. (a) Constructor of type 2, (b) constructor of type 3.

2. *Scoring function.* The fitness of a placement is computed using a scoring function. The applied scoring function uses the traditional semiperimeter for each net. Furthermore, this function encourages a more uniform distribution of the wiring by penalizing all channels that have a wiring density more than one standard deviation above the mean.

3. *Choice function.* Four choice functions are considered. These are: (1) choose the best scoring parent and a random parent, (2) choose both parents randomly, (3) select parents with probability proportional to their fitness, (4) same as (3), but only those parents with above average fitness are considered. Functions (2) and (3) produced the best results.

4. *Crossover operator.* Two types of crossover are used. These are: (1) select a random module e_s and bring its four neighbors in parent 1 into neighboring slots in parent 2; the neighboring locations of e_s in parent 2 are shifted outward

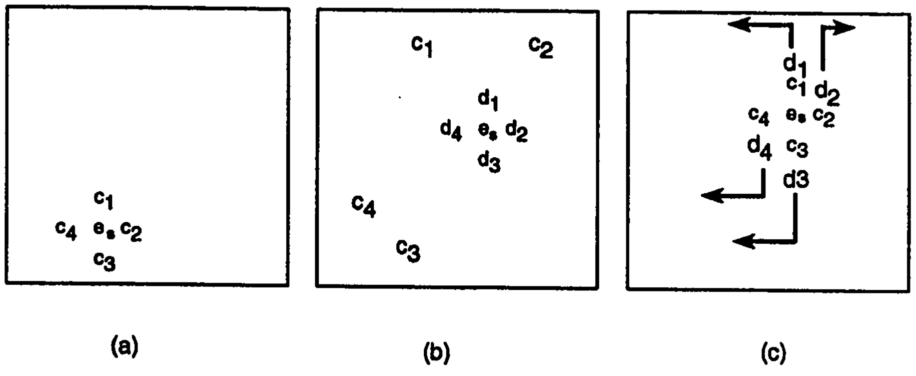

Figure 4.5: Crossover operator 1. (a) Passing parent, (b) target parent, (c) offspring.

by one location at a time in a chain move until a vacant location is found as in Figure 4.5, (2) select a $k \times k$ square from parent 1, where k is a random number with mean 3 and variance 1, and copy it to parent 2. The modules that are in the square of parent 2, but not in the square of parent 1, are moved to the locations of the modules that are in the square of parent 1, but not in the square of parent 2, (see Figure 4.6). Experimental results favor the second operator.

5. *Selection function.* Three functions were considered. These are: (1) select the best individual and $p-1$ other individuals randomly, where p is the population size, (2) select p individuals randomly, (3) select individuals with probability proportional to their fitness. The results favor the second and third functions depending on the characteristics of the placement instances.

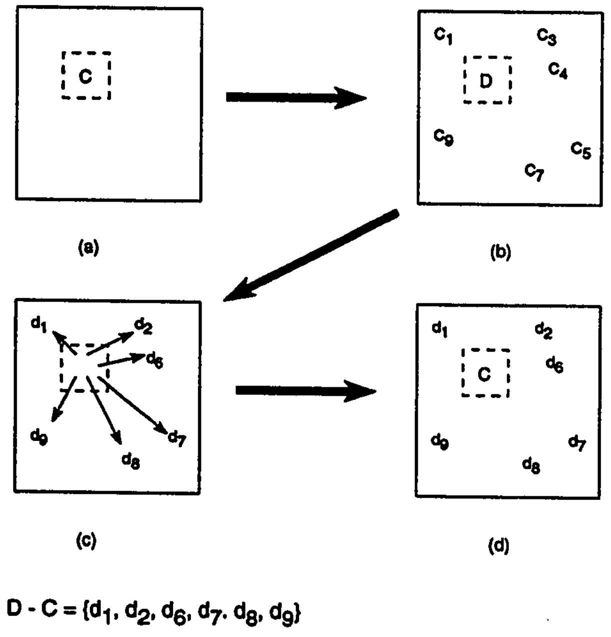

Figure 4.6: Crossover operator 2. (a) Parent 1, (b) parent 2, (c) copy to parent 2 (d) offspring.

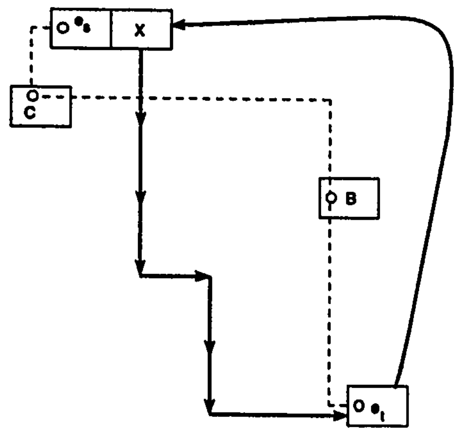

Figure 4.7: Mutation operator in Genie.

6. *Mutation operator.* Two operators were investigated. These are: (1) perform a random series of interchanges, (2) select a module e_s on a net Z, then bring a module e_t that is connected to e_s and farthest from e_s to a closer location. The displaced modules are shifted one location at a time until a vacant location is found, (see Figure 4.7). Experimental results favor the second operator.

4.2.2 GASP

GASP is a Genetic Algorithm for Standard-cell Placement developed by Shahookar and Mazumder [SM90]. The following is a description of some of the applied functions and their results.

1. *Initial population construction.* The only method that was used is to construct the population randomly. Their observation showed that a population size of 24 individuals gave the best performance. However, the authors did not mention whether the population size is dependent on design size or not.

2. *Scoring function.* The reciprocal of the wirelength was used as a measure for computing the fitness of individuals.

3. *Choice function.* Parents are selected from the population with a probability proportional to their fitness.

4. *Crossover operator.* Three crossover operators were investigated. These are: (1) order crossover, (2) partially mapped crossover (PMX), (3) cycle crossover. Either PMX or cycle crossover performed best. More details about these types of operators can be found in [SM90] [SM91].

5. *Selection function.* Three selection functions were considered. These are: (1) competitive selection. In this method, the best of the parents and offsprings are selected, (2) random selection, (3) select the best individual and the rest at random. Competitive selection gave the best results compared to the other methods.

6. *Mutation operator.* The mutation operator consisted of a simple random pair-wise interchange of the modules.

Chapter 5

Timing Driven Genetic Algorithm for Placement (TDGAP)

In our work, we implemented a timing driven placement program using the genetic algorithm. It is the first time that a GA is used for timing (performance) driven placement. All previous works have used GAs for placement with wirelength as an objective function. In our work a two dimensional (multi-objective) score (cost) function has been applied. The applied score function has included the timing performance of the circuit and the total wirelength as a measure of its goodness. Moreover, in this work we assume standard-cell design style. However, the same work can be applied to gate array design style with minor modifications.

Four major issues have been taken care of in our design:

1. *Timing constraints*, in which the delays of the critical paths should be smaller than a certain predefined value as reported by the timing analysis program.

2. *Geometric fit*, in which cells are placed at legitimate locations within the circuit's boundary without overlapping. The circuit's boundary or the shape of the layout can be controlled via an aspect ratio variable, which is user specified. Because cells are not allowed to overlap, the aspect ratio of the sides of the layout can be changed. To maintain this ratio the layout is subjected to a row equator procedure. This procedure tries to make all rows of equal length.

3. *Routability*, in which sufficient routing space is left to properly interconnect the cells without affecting the performance of the circuit. This is done by using a procedure for estimating the channel density of the routing channels based on the vertical constraint graph approach.

4. *Total wirelength*, a certain weight has been given for the total wirelength of the layout. This is to give a chance for those solutions that are best in terms of total wirelength, but within a small difference from the best solution with respect to timing to contribute in the evolvement toward the final solution.

5.1 Approach Overview

5.1.1 General Algorithm

The implementation of our TDGAP follows the general structure of the GA shown in Figure 5.1. In our implementation we have followed a different structure then the basic GA shown in Figure 4.1. In the basic GA the operation of mutation is applied on the generated offsprings. Then, the new generation is constructed by selecting its candidates from a set combining the offsprings (that might have been mutated) and the parents. However, in TDGAP, the selection for the next generation is done on the set of the offsprings (without mutation) and the parents. Then, the mutation operation is applied on the new constructed generation. This sequence has proven to be better than the originally proposed GA. A discussion of the results are given in Section 5.8.

```
Algorithm (TDGAP)
    $N_p$= Population Size.
    $N_g$= Number of Generations.
    $N_o$= Number of Offsprings.
    $\mathcal{P}_c$= Crossover Probability.
    $\mathcal{P}_\mu$= 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 $N_o$
                        (*CHOICE*)
                $(x, y) \leftarrow Choose\_parents$
                    (*CROSSOVER*)
                With probability $\mathcal{P}_c$ Apply
                    Offspring[$j$] $\leftarrow Crossover(x, y)$
            EndFor
                (*SELECTION*)
            Population $\leftarrow$ $Select$(Population, offspring, $N_p$)
            For $k = 1$ to $N_p$
                    (*MUTATION*)
                With probability $\mathcal{P}_\mu$ Apply
                    $Mutation$(Population[$k$])
            EndFor
            For $m = 1$ to $N_p$
                Evaluate $Fitness$(Population[$m$])
            EndFor
        EndFor
    Return highest scoring individual in Population
    End
```

Figure 5.1: A TDGAP simple representation.

Given an initial solution, TDGAP applies a repeated modification and search in the solution space using its operators and functions. The process of modification and search is repeated until all the timing constraints or some terminating criteria are met. Timing constraints refer to critical paths in the circuit whose delays must be below or within certain bounds for the circuit to perform correctly. This means, that the SLACK values of the critical paths should be positive.

The final placement of a design reported by TDGAP is the best one over the whole run time with respect to timing aspects first then with respect to total wire-length. While running TDGAP there is a procedure that keeps track of the best individual since the starting time and saves it. When TDGAP terminates, this procedure reports the saved individual (placement solution) in VPNR format. After that, the *OASIS* system is used to complete the generation of the layout until the mask level.

5.1.2 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. The encoding scheme for n rows solution is shown

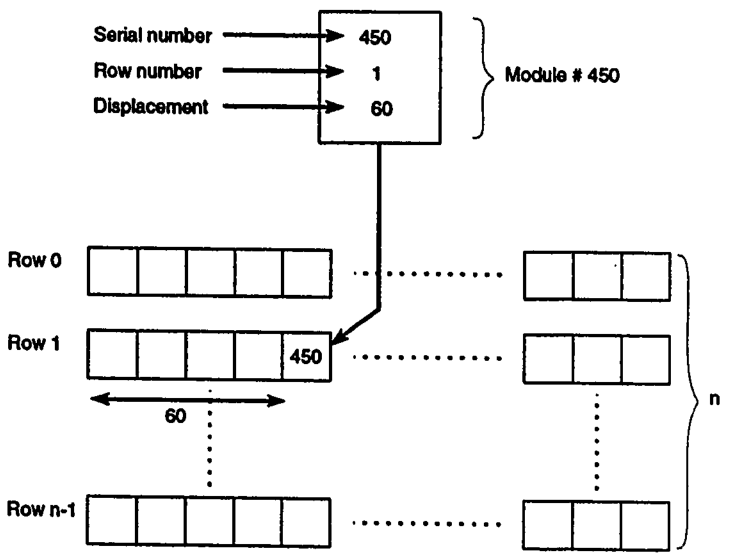

Figure 5.2: Encoding scheme of an individual.

in Figure 5.2. Next the major operators and functions of TDGAP are discussed, then some comments on the other parts of the program are given.

5.2 Initial Population Constructors

Initial solution construction is very critical to GAs. Five initial population constructors $IPC_1, IPC_2, IPC_3, IPC_4$ and IPC_5 are investigated. All of these constructors are dependent on a random number generator, so they can be applied to generate an initial population of any desired size. These are:

- Constructor IPC_1 selects modules at random and places them in rows. The qualities of the generated solutions using this constructor are unpredictable.

- Constructor IPC_2 attempts to cluster cells (modules) affecting the same path to improve the score of the initial population. Based on the linear delay model presented in Section 3.4.2, the cells affecting the same path are defined to be all cells on the nets that are on that path, (see Figure 5.3). It is known that when the solutions in the initial population are all of good quality, GA tends to get quickly trapped into a local minima. We suspect that this constructor might suffer from such a phenomenon.

- Constructor IPC_3 is similar to constructor IPC_2. The major difference between IPC_2 and IPC_3 is in the way they start placing the cells on the layout surface. For a layout of n rows, IPC_2 places cells, left to right, starting from first row, where as in IPC_3 cells are placed starting from middle row and proceeding outward. In both types, cells are ordered in a row-major fashion as shown in Figure 4.4(b).

- Constructor IPC_4 is a variant combination between IPC_1 and IPC_3. For example, IPC_4 can be made of 25% of IPC_1 and 75% of IPC_3. This constructor has an average fitness value that is better than IPC_1 but lower than IPC_3. This can help in avoiding the trap of local minima.

- Constructor IPC_5 is similar to constructor IPC_4. However, IPC_5 has one of its solutions built constructively based on mincut partitioning algorithm that is used with $OASIS$ placer [MCN90], where cells are partitioned into subgroups such that the total wirelength is minimized. Algorithms based on mincut partitioning are well known for their good results in generating layouts with minimum chip area. Therefore, a solution based on this algorithm is constructed to aid the performance of our TDGAP. Using IPC_5 three types of solution qualities are presented. First type is random, the second is good in timing aspects, and the third is good in chip area.

Population size has great effect on the quality of solutions and the total run time of TDGAP. Therefore, we have run different test cases for different population sizes. Furthermore, we have tested the effect of having the population size being fixed or changed during run time. As we will see later, run time is greatly affected by the population size. The results of selector functions IPC_1, IPC_2, IPC_3, IPC_4, and IPC_5 and the effect of the population size are discussed in Section 5.8.1 and Section 5.8.2.

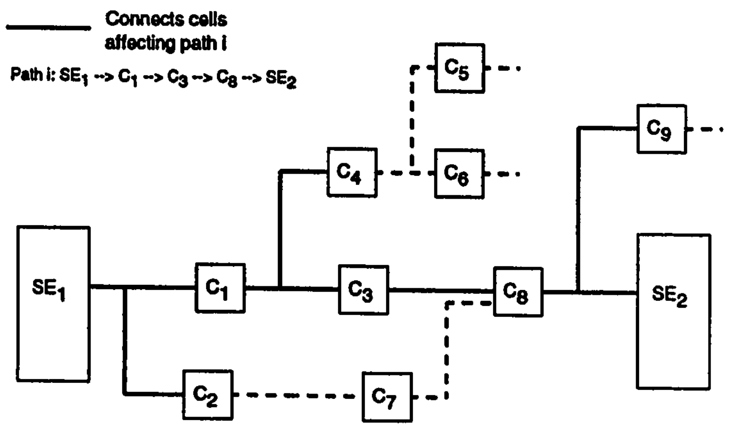
Connects cells affecting path i
Path i: SE1 --> C1 --> C3 --> C8 --> SE2
SE1
SE2
C1
C2
C3
C4
C5
C6
C7
C8
C9

Figure 5.3: Cells affecting path i are: C_1, C_2, C_3, C_4, C_8, C_9, and SE_2.

5.3 Choice Function (Stochastic Remainder without Replacement)

As mentioned in Section 4.1.1, the stochastic remainder without replacement scheme has been proven to be superior over the expected value scheme. As a result, this scheme has been adopted as a choice function in our TDGAP.

The implementation of the choice function involves two procedures. First, a pre-select procedure, prepares a list, say ℓ, of parents that are possible to contribute in the crossover operation. Then a second procedure, called parent selection, selects a parent from the list ℓ for crossover.

In the pre-select procedure, the list ℓ is built based on the concept of stochastic remainder without replacement. First, let the value of the expected count of an

individual$_i$	*score*(i)	*exp_count*(i)	sure copies in ℓ	probabilistic copies in ℓ
1	37	0.78	0	0.78
2	52	1.10	1	0.10
3	60	1.27	1	0.27
4	21	0.45	0	0.45
5	98	2.08	2	0.08
6	15	0.32	0	0.32

Table 5.1: An example to show how to compute the expected count of individuals in a certain population P. AVG(*score*(i))=47.17, $\mid P \mid= 6$.

individual i in population P be *exp_count*(i). This value is computed as follows:

$$exp\_count(i) = \frac{score(i)}{\overline{score}}$$

where,

$score(i)$ =score value of individual i, and

$\overline{score} = \frac{1}{|P|} \times \sum_{i \in P} score(i)$.

In stochastic remainder without replacement, for each individual i the expected individual count value is computed and the integer part is assigned. An individual represents a placement configuration. For example, Table 5.1 shows how to compute the expected count of individuals in a certain generation. From this table, *exp_count*(5) = 2.08 which means that we have to assign two copies of individual 5 in the list ℓ. After that, the fractional parts of the expected number values are treated as probabilities. For each individual, Bernoulli trials are carried out using the fractional parts as success probabilities. This operation is repeated until the list ℓ is full. For example, let the expected number of copies of an individual i be $exp\_count(i) = 1.5$, then this individual will receive a single copy surely in the list ℓ and another with probability 0.5.

After the pre-select procedure, the parent selection procedure operates on the generated list ℓ. Each time a parent is needed for a crossover operation, the parent selection procedure randomly selects it from the list ℓ. Then, the selected parent is removed from the list. In case we have two identical parents, we run the selection again so as to get two different parents for the crossover operation.

5.4 Crossover

Crossover is the dominant genetic operator. For the two parents selected by the choice function, crossover is applied to generate an offspring. Two types of crossover operators Ψ_1 and Ψ_2, are considered in our TDGAP. The two types use information passing from one parent to the other. However, they are different in the way they pass the information. Both operators are aimed at improving the timing aspects of the reported K most critical paths. They try to pass the information about some of satisfied paths (paths with no timing problems) from one generation to another. Operator Ψ_1 does this by maintaining the same locations of the cells affecting these satisfied paths. Operator Ψ_2, however, does it by keeping the cells affecting these satisfied paths within a certain boundary.

Let α_x and α_y be respectively, the passing and target parents, and CP be the set of the K most critical paths of the circuit. Operator Ψ_1 operates in the following way. An identical copy (α_o) of α_y is made. A critical path cp_s is selected from CP according to a criterion that will be explained later. Then, the set β of cells affecting cp_s are identified. The goal of Ψ_1 is to reconfigure offspring α_o such that the cells in β occupy the same locations as if they were in α_x, the passing parent. This operation may overwrite some of the modules in α_o. A collision resolution technique is used to resolve these conflicts and which works as follows. Suppose a module $e_i \in \beta$ is to be assigned to location loc_j which is occupied by module e_j. This conflict is resolved by swapping the locations of the two modules. For example, e_i is assigned to loc_j and e_j to loc_i where loc_i and loc_j are locations of e_i and e_j in α_y, respectively. The steps of Ψ_1 operation are illustrated in Figure 5.4. In this figure, for example, the set $\beta = \{c_1, c_2, c_3, c_4\}$. Then, the cells of β in α_o are moved to the same location as if they were in α_x. As shown in Figure 5.4(c), cell c_1 is moved to the location of cell d_1. To avoid overwriting cell d_1, cells c_1 and d_1 are swapped with each other.

Operator Ψ_2 operates in the following manner. It starts like Ψ_1 by making a duplicate (α_o) from α_y, selecting a critical path cp_s from CP, and identifying the set β of cells affecting cp_s. The size and location of the smallest bounding window ω_x that encloses the cells of β in α_x is determined. The location of ω_x is also determined

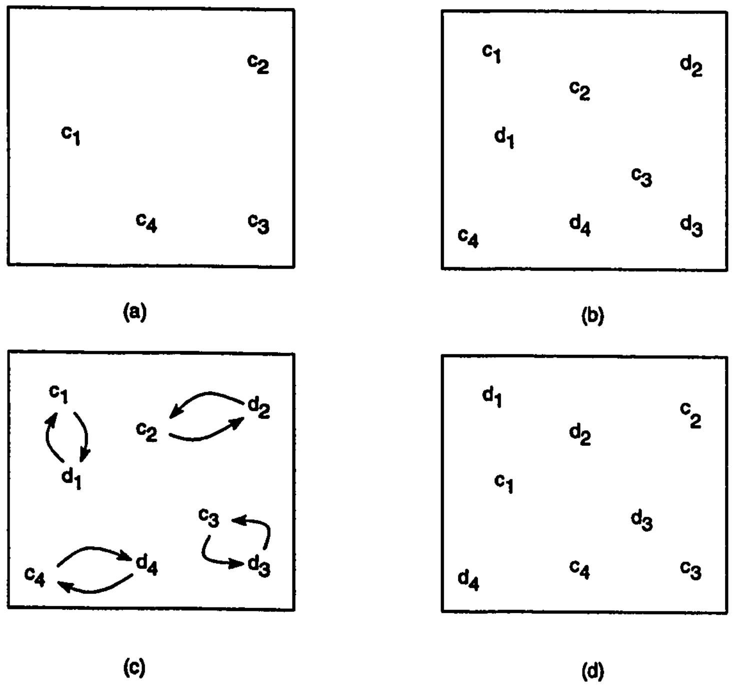

Figure 5.4: Crossover operator Ψ_1. $\beta = \{c_1, c_2, c_3, c_4\}$. (a) Parent 1 (passing parent), (b) parent 2 (target parent), (c) information passing, (d) offspring.

for parent α_y, call it ω_y. Comparing the contents of these two windows, three sets can be defined. These are,

$\sigma =$ cells $\in \beta$, and $\notin \omega_y$,

$\rho =$ cells $\in \omega_x$, and $\notin \omega_y$, and $\notin \beta$,

$\eta =$ cells $\in \omega_x$, and $\in \omega_y$.

The operator Ψ_2 then, tries to reconfigure α_o as follows. It first defines a window ω_o in α_o of the same size and location of ω_x. After that, it scans the contents of ω_o cell by cell in a row-based fashion. Then, for each scanned cell e_i, operator Ψ_2 works according to the algorithm shown in Figure 5.5. A graphical representation of the operation of Ψ_2 is depicted in Figure 5.6. A discussion of the crossover operators Ψ_1 and Ψ_2 is given in Section 5.8.3.

Selection of Critical path cp_s

The operation of selecting a critical path cp_s from CP is done in the following manner. For every solution in the layout, there are two lists associated with it. One is used to keep a list of the critical paths that are within their timing bounds. In other words, it is a list of those critical paths that do not have long path problems ($SLACK \geq 0$). The second list is used to keep the remaining critical paths which still have long path problems (they have exceeded their timing bounds). Based on

```
Algorithm (Reconfigure)
Stop=0
Repeat
 If $e_i \in \eta$ (where $e_i$ is a cell $\in \omega_o$) 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 the modules $e_i$ 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 the modules of $e_i$ 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 5.5: An algorithm used by Ψ_2 to reconfigure an offspring α_o.

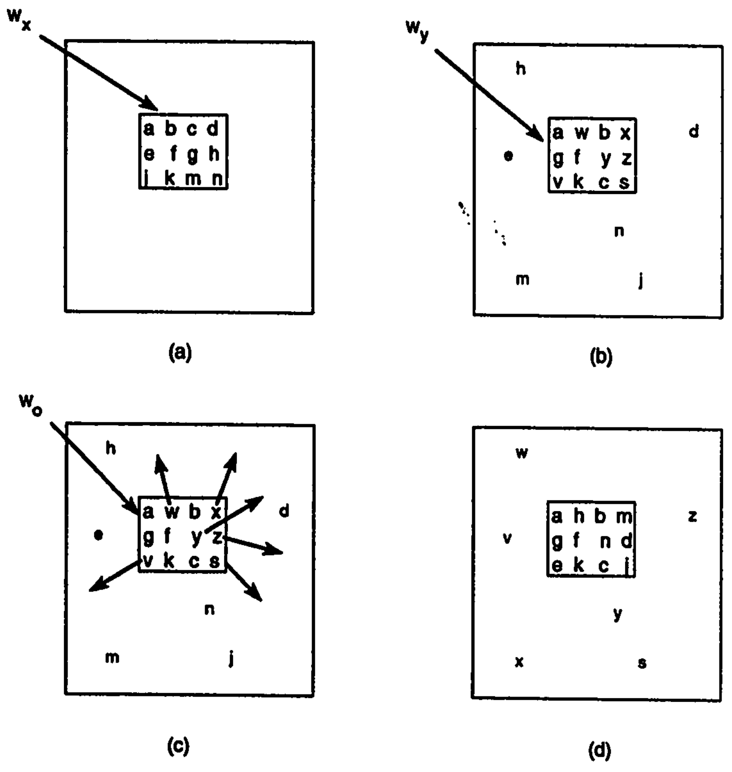

Figure 5.6: Crossover operator Ψ_2. β = {a,h,m,n}, σ = {h,m,n}, ρ = {d,e,j}, η = {a,b,c,f,k,g}. (a) Parent 1 (passing parent), (b) parent 2 (target parent), (c) information passing, (d) offspring.

these two lists, cp_s is selected as explained in the algorithm shown in Figure 5.7. The process of selecting a cp_s is aimed toward the generation of an offspring with better timing characteristics. The idea of selecting a cp_s is based on the evaluation of the following conditions in sequence:

1. if target parent has no timing problems, then select a cp_s at random,
2. else, if there is a cp_s that has a long path timing problem in target parent, but is problem free in source parent then select it,
3. else, if there is a cp_s such that it is with timing problems in both target and source parents, but with better timing values in source parent than in target parent then select it,
4. else, select a cp_s at random.

5.5 Selection of the Next Generation

Four selector functions ρ_1, ρ_2, ρ_3, and ρ_4 are considered for determining which individuals to use as part of the next generation. In other words, the selector function determines the survival of current individuals in a given population. Let:

$$\mathcal{P} = Population$$

```
Algorithm (Select Critical Path)
$Total\_CP = |CP|$
$\ell_1^{source} = \{cp_s \mid cp_s \in CP$ and $(SLACK(cp_s), source) \geq 0, \forall\, 0 < s \leq Total\_CP\}$
$\ell_2^{source} = \{cp_s \mid cp_s \in CP$ and $(SLACK(cp_s), source) < 0, \forall\, 0 < s \leq Total\_CP\}$
$\ell_3^{target} = \{cp_s \mid cp_s \in CP$ and $(SLACK(cp_s), target) < 0, \forall\, 0 < s \leq Total\_CP\}$
$loop_1 = 1$, $loop_2 = 1$
If $(\ell_3^{target} \not\in \emptyset)$
 For all $cp_s \in \ell_3^{target}$ Do Unmark($cp_s$)
 While ($loop_1 = 1$)
  If ($\exists\, cp_s \in \ell_3^{target} \mid cp_s$ is not marked) Then
   Mark($cp_s$)
    If ($cp_s \in \ell_1^{source}$)
     select this $cp_s$, $loop_1 = 0$
    EndIf
  Else
   $loop_1 = 0$
    For all $cp_s \in \ell_3^{target}$ Do Unmark($cp_s$)
    While ($loop_2 = 1$)
     If ($\exists\, cp_s \in \ell_3^{target} \mid cp_s$ is not marked) Then
      Mark($cp_s$)
       If ($cp_s \in \ell_2^{source}$ and $(SLACK(cp_s), source) > (SLACK(cp_s), target)$) Then
        select this $cp_s$, $loop_2 = 0$
       EndIf
     Else
      $loop_2 = 0$
      $cp_s$=random(1,$Total\_CP$)
     EndElse
   EndWhile($loop_2$)
  EndElse
 EndWhile($loop_1$)
Else
 $cp_s$=random(1,$Total\_CP$)
End
```

Figure 5.7: An algorithm used by Ψ_1 and Ψ_2 to select cp_s from CP.

$$J = \mathcal{P} \cup Offspring$$

then these selectors are:

- Selector ρ_1 selects from J the best scoring individual and $p-1$ other individuals at random, where $p = \mid \mathcal{P} \mid$. By this way, the best individual is always remembered and advanced to the next generation where it may pass some of its good characteristics to the new offsprings.
- Selector ρ_2 selects the best 10% of p and the rest are selected at random. With this selector, a good number of good individuals are selected for survival. This may lead to the case where most individuals in some later generation are all alike and then being trapped into local minima.
- Selector ρ_3 selects all p individuals from J at random. With this selector, results are unpredictable.
- Selector ρ_4 selects individuals on a competitive basis with each individual α_j having a probability $Prob(\alpha_j)$ to be selected, where

  $$Prob(\alpha_j) = \frac{score(\alpha_j, J)}{\sum_{i=1}^{i=q} score(\alpha_i, J)}$$

  where,

  $$q = \mid J \mid$$

Using this selector, individuals have a good chance to be trapped into local minima. This might happen because individuals with low fitness values die off with a fast rate.

The results of the selector functions ρ_1, ρ_2, ρ_3, and ρ_4 are given in Section 5.8.4.

5.6 Mutation

Two mutation operators μ_1 and μ_2 are investigated. Operator μ_1 is targeted toward improving the timing of the placement. On the other hand, operator μ_2 is targeted toward improving the wirelength of the placement. To aid the operation of the mutation operators, some analysis on the nets for each individual is made. This analysis includes the computation of the center of mass for every net.

For a given net i with m modules, the term *center of mass* of net i is defined as a pair of $(x_{center}, y_{center})_i$, where, x_{center} and y_{center} are defined as follows:

$$x_{center} = \frac{\sum_{k=1}^{k=m} x\_coordinate\ of\ module_k}{m}$$
$$y_{center} = \frac{\sum_{k=1}^{k=m} y\_coordinate\ of\ module_k}{m}$$

In TDGAP the mutation operation is applied on all individuals in the newly constructed generation except for the best individual. In other word, we do not allow the best individual to be mutated. This is because if it is allowed to mutate the best individual then there is a chance of losing it. Losing the best individual can weaken the population and cause a slower convergence. A discussion of the results are given in Section 5.8.5.

Operator μ_1 works as follows. It starts by randomly selecting a critical path cp_s from those paths with long path problems. Then, it selects randomly a module e_s that is affecting the performance of the selected critical path. After that, a module e_j at the location of the center of mass of the net $Net\ s$ that module e_s belongs to is determined. Module e_j may or may not belong to the net $Net\ s$. Then, the two modules e_j and e_s are swapped (pairwise interchanged) with each other, thus improving the $SLACK$ value of cp_s. This is done once for every individual that is subjected to the mutation operation. This operation is depicted in Figure 5.8.

Operator μ_2 operates in a similar manner as operator μ_1. It starts by selecting at random a two-pin net, where none of these two pins is an I/O pad. After that, one of these two modules is chosen to be swapped with a module at the location of the center of mass of the selected net as in μ_1. The operation of μ_2 is depicted in Figure 5.9.

The reason behind conditioning that the selected net should be a two-pin net,

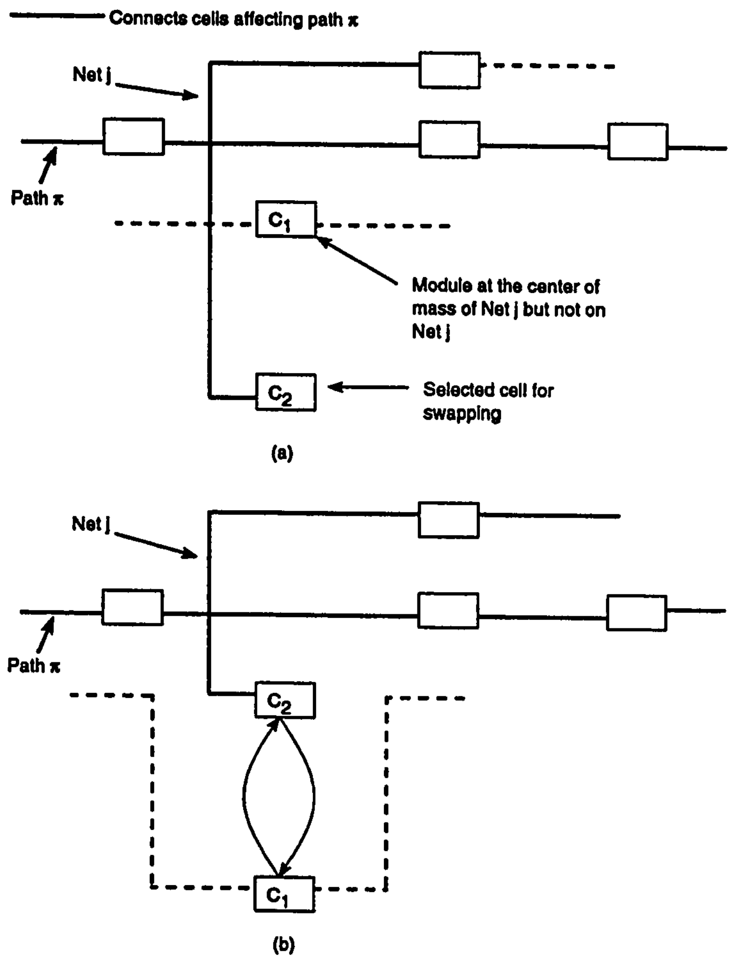
Connects cells affecting path π
Net j
Path π
C_1
Module at the center of mass of Net j but not on Net j
C_2
Selected cell for swapping
(a)
Net j
Path π
C_2
C_1
(b)

Figure 5.8: The operation of mutation operator μ_1. (a) Before mutation, (b) after mutation.

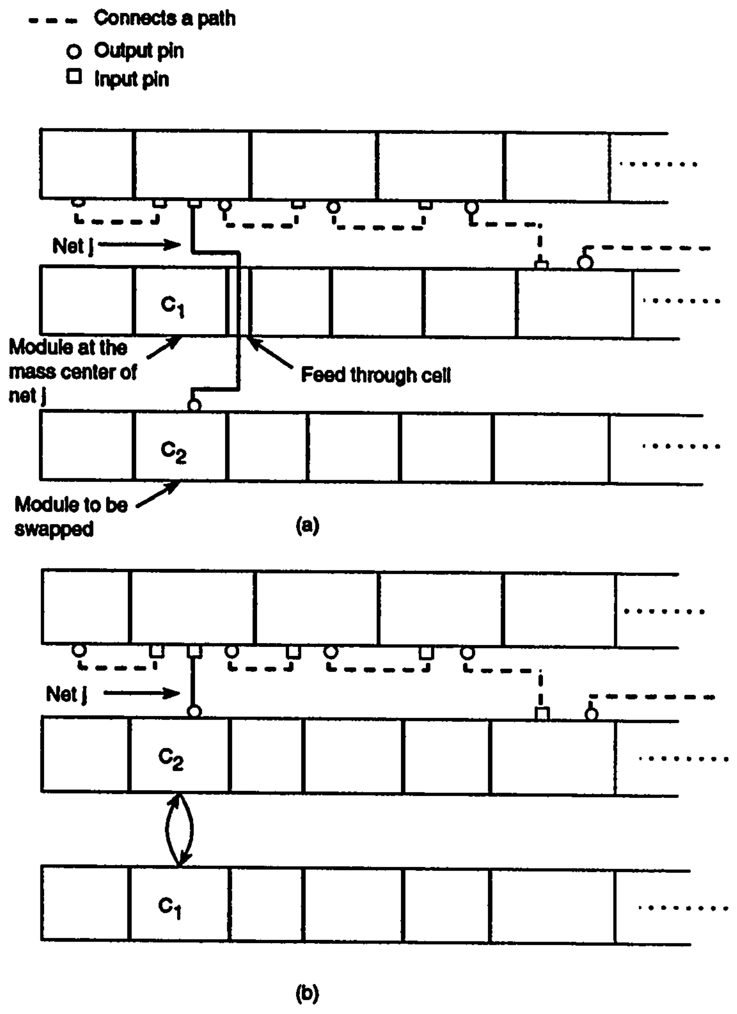

Figure 5.9: The operation of mutation operator μ_2. (a) Before mutation, (b) after mutation.

is that from the analysis of our generated layouts, it has been noticed that most of the two-pin nets that are on critical paths have their modules separated apart with large distances. As a result, this big separation tends to introduce more feed-through cells and increase the total wirelength. This effect is illustrated in Figure 5.9. Furthermore, the reason as to why the selected net should have none of its pins as an I/O pad is because of the following. If a two-pin net has one of its pins as an I/O pad, then the other pin is connected to a cell, say e_i, which is connected to other nets. Moving cell e_i tends to disturb the solution much more than moving the I/O pad between the top and bottom sides of the layout. Due to this reason, we have included a separate procedure that takes care of moving the I/O pads between the top and the bottom sides of the layout, so as to improve the scoring function. In our TDGAP, operator μ_2 is applied on 10% of the total number of nets. A discussion on the results of operators μ_1 and μ_2 is given in Section 5.8.5

5.7 Scoring Function

The scoring function (objective function) is a combination of three terms. These three terms are directed toward the improvement of the circuit performance and total wirelength. Therefore, it is a two-dimensional objective function. The scoring function is a way to determine which individual is fitter than the other. For two individuals i and j in a certain population, if $Score(i) > Score(j)$ then individual i is fitter than individual j. Let n be the population size and CP the set of critical paths. The score of a given individual i is computed as follows:

$$Score(i) = \frac{worst\_slack(i)}{S} \times w_1 + \frac{relax(i)}{R} \times w_2 + \frac{Total\_wirelength(i)}{W} \times w_3 \quad (5.1)$$

where,

$Total\_wirelength(i)$ = the total wirelength of individual i

$worst\_slack(i)$ = Slack value of the worst critical path $\times(-1)$

$relax(i)$ = $\sum_{j \in CP} Slack(path(j))$ and $Slack(path(j)) \geq 0$

S = $max_{j=1...n} worst\_slack(j)$

R = $max_{j=1...n} relax(j)$

W = $max_{j=1...n} Total\_wirelength(j)$

w_1, w_2, and w_3 = different weights assigned for each term

The values of $worst\_slack(i)$, $relax(i)$, and $Total\_wirelength(i)$ are used after they are linearly scaled. Originally, each of these values has a different range of results from others, and thus, they are not comparable. In order to make them comparable and use them in one equation as given in Equation 5.1, we have to make their values all fall in the same range. This is done by linearly scaling them. Another need for scaling is that some of these values are either very near each other or very far from each other. Which means that their variance value is either too low or too high respectively. Therefore, by scaling these values we are able to have them with good variance value. For example, to linearly scale the $Total\_wirelength(i)$ we do the following:

$$Total\_wirelength(i) = New\_AVG - \left(\frac{Total\_wirelength(i) - Old\_AVG}{Old\_STD} \times New\_STD\right)$$

where,

$Old\_AVG = \frac{\sum_{j=0}^{j<n} Total\_wirelength(j)}{n}$,

Old_STD = standard deviation of $Total\_wirelength(j)$, $0 < j \leq n$,

New_AVG= the new required average,

New_STD= the new required standard deviation.

For example, Table 5.2 shows the steps of scaling the values of $worst\_slack(i)$, $relax(i)$, and $Total\_wirelength(i)$ for each of the six individuals in a given population. From this table, the values of S, R, and W are:

$S = 61.85$

$R = 59.85$

$W = 64.70$

	Before Scaling			*After Scaling*		
Pop (i)	*worst_ slack*(i)	*relax* (i)	*Total_ wirelength*(i)	*worst_ slack*(i)	*relax* (i)	*Total_ wirelength*(i)
1	10	3	100	33.38	57.66	54.20
2	3	10	200	58.29	42.34	33.20
3	2	15	150	61.85	31.40	43.70
4	4	4	90	54.73	55.47	56.30
5	5	2	130	51.17	59.85	47.90
6	8	5	50	40.50	53.28	64.70
AVG	5.33	6.5	120	50	50	50
STD	2.81	4.57	47.61	10	10	10
MAX	10	15	200	61.85	59.85	64.70

Table 5.2: An example to show the steps of scaling the values of $worst\_slack(i)$, $relax(i)$, and $Total\_wirelength(i)$ in certain population.

then, the score value of individual 5, for example, $Score(5)$ is computed as follows:

$$Score(5) = \frac{51.17}{61.85} \times w_1 + \frac{59.85}{59.85} \times w_2 + \frac{47.90}{64.70} \times w_3$$

Let $w_1 = 0.70$, $w_2 = 0.05$, and $w_3 = 0.25$ then,

$$Score(5) = 0.83 \times 0.70 + 1.0 \times 0.05 + 0.74 \times 0.25 = 0.816$$

For a given individual i, $relax(i)$ is the summation of the $SLACK$ values of all satisfied critical paths (paths with no long path problems). This value gives a measure of the amount by which these paths can be made longer and still be satisfied. Making some of these paths longer may give other unsatisfied critical paths (paths with long path problems) the chance to become shorter.

By changing the values of w_1, w_2, and w_3, results generated by TDGAP can be different in their quality. For example, if $w_1 > w_3$ then TDGAP favors solutions that are better in terms of timing aspects more than those which are better in term of total wirelength. In the implementation of TDGAP, we have chosen $w_1 = 0.70$, $w_2 = 0.05$, and $w_3 = 0.25$.

5.8 Results

In this section, we present the results related to the implementation of TDGAP along with some comparisons of the generated layouts with others that are generated using the *OASIS* placer. The *OASIS* placer is based on mincut partitioning algorithm that does not incorporate timing aspects. In addition, some other general observations are made during the process of development of TDGAP.

All the figures are given in two domains. First, in the time domain, where the graphs are plotted with timing information on the y-axis vs. the number of generations (iterations) on the x-axis. The timing information is given as the negative of the worst slack in ns. Second, in the space domain, where the graphs are similar to those in the time domain, but with the y-axis representing the total wirelength in μ. For each domain one graph is given to represent the performance of the best solution found over all generations. The data of the graphs is taken for every 50 generations. The data for every 50 generations is averaged and considered as one point in the plot. The tuning of the parameters of TDGAP were conducted using the circuit 'CRC16' which will be described later. For this circuit we have used a clock period of 15 *ns*.

Mutation Prior to or after Selection?

The basic GA shown in Figure 4.1 is one of possible GAs. It applies the mutation operation on the generated offsprings. Then, it selects the candidates of the new generation from a set combining the offsprings and the parents. However, the TDGAP shown in Figure 5.1 follows a different structure. In TDGAP the mutation operation is applied on the new generation after it has been constructed from the set of parents and offsprings produced by the crossover operation. The sequence used in TDGAP has been proven to be better than the one applied in the basic GA. Figures 5.10 and 5.11 show the performance of these two sequences in both the time and the space domains for circuit 'CRC16' which will be described later. A summary of the initial and final values of the best solution with these two sequences in both domains is given in Tables 5.3, and 5.4.

The superior performance of TDGAP over the basic GA can be explained as follows. Although the mutation operation is applied with a small probability in

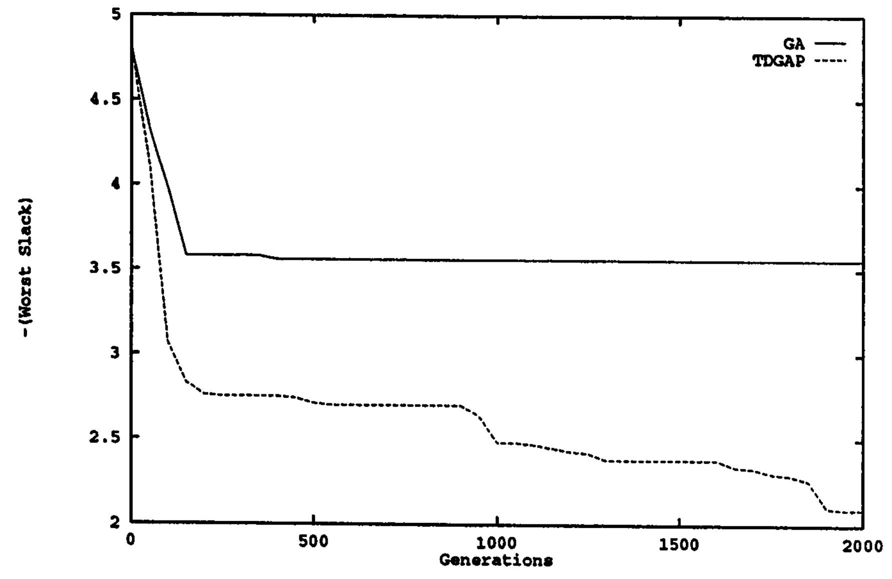

Figure 5.10: The performance of the best solution with the basic GA and TDGAP in time domain

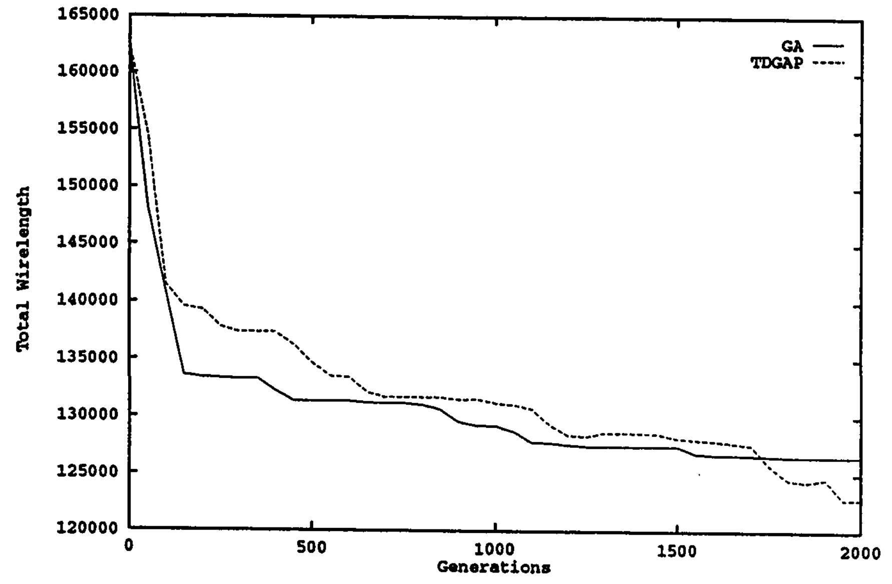

Figure 5.11: The performance of the best solution with the basic GA and TDGAP in space domain.

Algorithm	Basic GA	TDGAP
Initial	4.82	4.82
Final	3.56	2.09
Speed-up %	6.8	16.0

Table 5.3: A summary of the initial and final values of the best solution with basic GA and TDGAP in the time domain. Values are given in *ns*.

Algorithm	Basic GA	TDGAP
Initial	162614.4	162614.4
Final	126556.6	122870.0
Decrease %	22.2	24.4

Table 5.4: A summary of the initial and final values of the best solution with basic GA and TDGAP in the space domain. Values are given in microns.

both structures, it can not be ignored. It is the process that helps in escaping from a local minima that the solutions might be trapped in. The sequence applied in the basic GA reduces the effect of the mutation operation. This is because the mutation operation may change solutions to a better or worse situation. Then, because of the behavior of most selection functions, solutions that are changed by the mutation operation may not be included into the next generation. This approach weakens the effect of the mutation operation and possibly traps the search procedure into a local minima. On the other hand, the sequence applied in the TDGAP tries to advance the effect of the mutation operation completely to the next generation. This way, solutions are less likely to be trapped into local minima.

5.8.1 Comparison of Initial Population Constructors

The choice of constructor function affects both the quality of the solution and the number of generations needed to generate a good solution. In this work we experimented with five constructor functions described in Section 5.2. Figures 5.12 and 5.13 show a comparison between these functions in both the time and space domains. Solutions constructed by IPC_1 tend to have poor starting values in both the time and space domains. Which means that most of the solutions are not fit enough to inherit good features to the next generations. This leads the solutions to be trapped into a local minima with merely no improvement in the time domain and some improvements in the space domain. Although the improvement in the space domain is good, the final value was much higher than the final values obtained with other constructors. In contrast, solutions constructed by IPC_2 are good enough that they are similar to each other. This caused the search procedure to get trapped into a local minima very fast. The results were poor in both domains. Constructor IPC_3 tends to have slower convergence toward good solutions. Constructor IPC_4 is the best in time domain, and constructor IPC_5 is the best in space domain. However, in some runs the results using IPC_5 were better than those using IPC_4 in both domains.

Tables 5.5 and 5.6 summarize the initial and final values of the best solution

Constructor	IPC_1	IPC_2	IPC_3	IPC_4	IPC_5
Initial	8.45	4.75	5.47	4.82	5.48
Final	4.13	4.40	3.02	2.09	3.47
Speed-up %	22.6	1.8	13.6	16.0	10.9

Table 5.5: A summary of the initial and final values of the best solution with different constructors in the time domain. Values are given in *ns*.

with different constructors in both the time and space domains. From these tables we note the increase in the total wirelength of the solutions initially constructed by IPC_5. This is because constructor IPC_5 included a very good solution which favors the total wirelength side of the score function. However, we note that it was not equally good in the time domain compared to other constructors. Therefore, for IPC_5 to gain some improvement in the time domain it had to lose some improvement in the space domain. Because of the superior behavior of IPC_4 and IPC_5, other constructors were discarded. Hence, only IPC_4 or IPC_5 is used in order to tune the remaining important parameters of TDGAP.

The score of the initial solution should neither be too high nor too low. In the first case, the GA will be trapped easily in a local minima, where in the second case, the GA will have a slow rate of convergence toward homogeneity. Therefore, for constructor IPC_4 we have chosen it to be 25% of IPC_1 and 75% of IPC_3. We observed that these values are a good starting point for our test circuits.

The value of the population size can be varied from problem to another. However,

Constructor	IPC_1	IPC_2	IPC_3	IPC_4	IPC_5
Initial	228306.8	154063.2	151513.8	162614.4	87786.3
Final	158183.0	140080.6	121198.8	122870.0	106766.5
Decrease %	30.7	9.1	20.0	24.4	-21.6

Table 5.6: A summary of the initial and final values of the best solution with different constructors in the space domain. Values are given in microns.

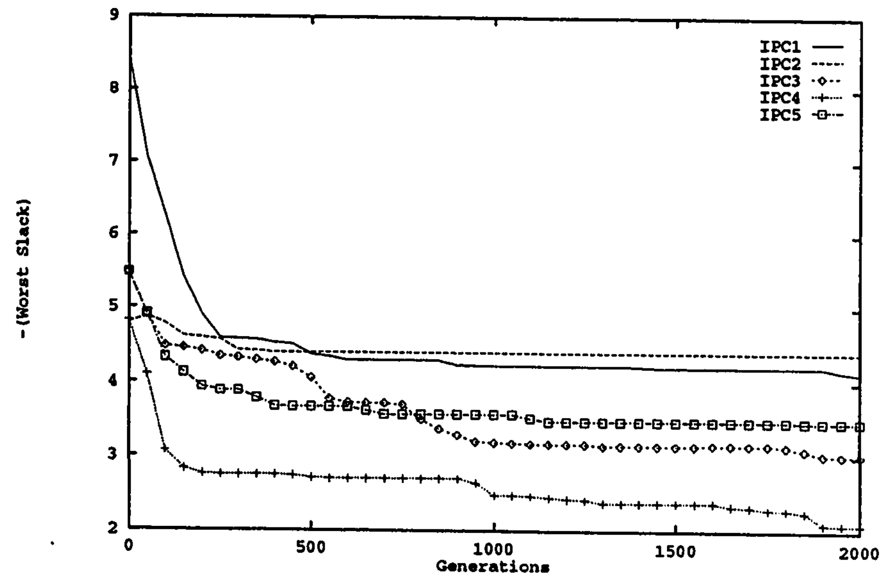

Figure 5.12: The performance of the best solution with constructor functions IPC_1, IPC_2, IPC_3, IPC_4, and IPC_5 in time domain.

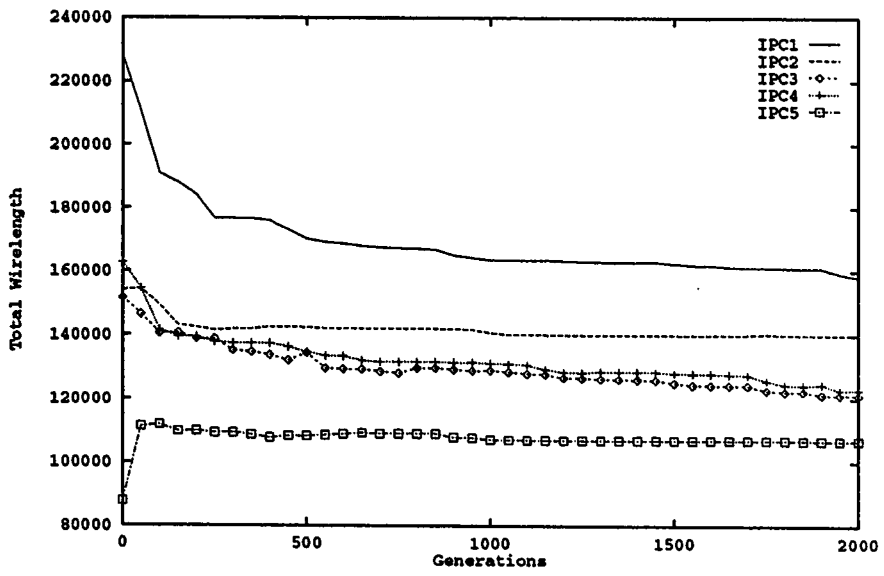

Figure 5.13: The performance of the best solution with constructor functions IPC_1, IPC_2, IPC_3, IPC_4, and IPC_5 in space domain.

Population size	16	20	24	30	34
Initial	6.16	5.36	4.82	4.29	6.40
Final	4.26	3.36	2.09	1.70	2.16
Speed-up %	9.9	10.9	16.0	15.5	24.7

Table 5.7: A summary of the initial and final values of the best solution with different population sizes in the time domain. Values are given in ns.

from our observations, it has been found that a population size of about 24 is best for all circuits we experimented with. The effect of different population sizes is shown in Figures 5.14 and 5.15. A summary of the initial and final values of the best solution with different population sizes in both domains is given in Tables 5.7, and 5.8.

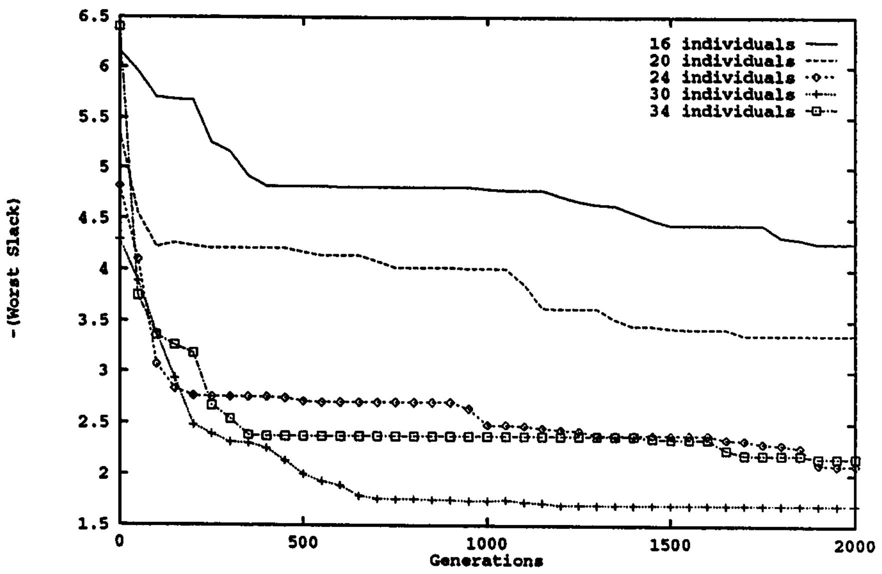

Figure 5.14: The performance of the best solution with different population sizes in the time domain.

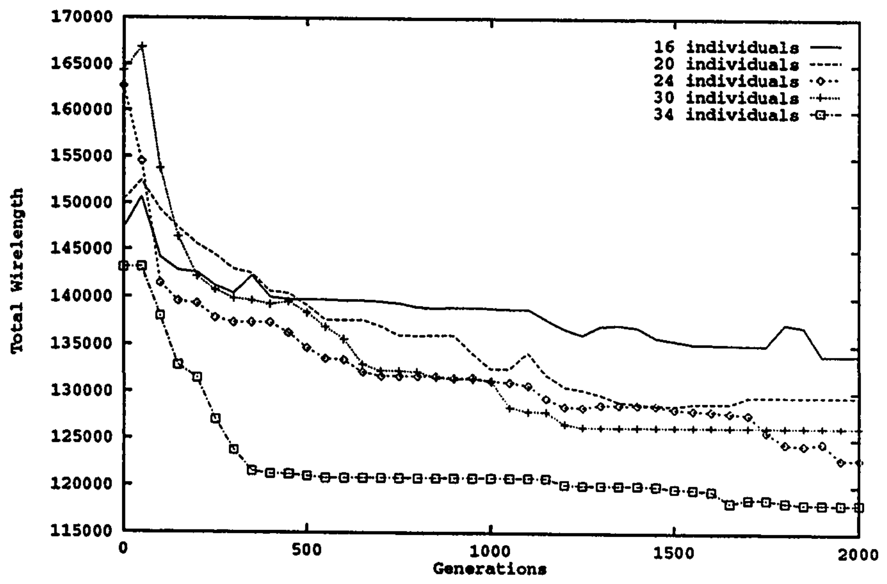

Figure 5.15: The performance of the best solution with different population sizes in the space domain.

Population size	16	20	24	30	34
Initial	147234.6	150332.9	162614.4	164351.2	143136.0
Final	133962.7	129553.1	122870.0	126254.5	118112.0
Decrease %	9.0	13.8	24.4	23.2	17.5

Table 5.8: A summary of the initial and final values of the best solution with different population sizes in the space domain. Values are given in microns.

5.8.2 TDGAP with Dynamic Population Size

Population size has great effect on the total run time of TDGAP. To view this effect we have ran TDGAP on a certain design with two cases. In the first case, we ran it with the population size fixed to 30 individuals. In the second case, we ran it with dynamic population size where it started with a population size of 30 individuals and as the search progress, the population size was reduced according to a certain reduction procedure.

The reduction procedure determines when to reduce the population size and by how much. It works as follows. The performance of the best solution is checked periodically every *Reduction_Period.* If no improvement of at least 3% in the last *Reduction_Period* with respect to timing aspects of the design, the population size is reduced by 20%. This reduction is effective as long as the population size is not less than half of the original size. Each time the population size is checked, the *Reduction_Period* is reduced by a *Period_Factor*. This is done because the convergence rate of TDGAP in the early generations is higher than that of later generations (i.e. the improvement in later generations is slower than for early generations). Thus, the reduction procedure monitors the performance after shorter periods in those generations where the improvement is too slow. For our test case, we have chosen the *Reduction_Period* to be 5000 generations and the *Period_Factor* to be 5%.

Figures 5.16 and 5.17 shows the performance of the best solution with dynamic and fixed population size in both domains. Both cases were ran for 300,000 generations. The total run time for the case of fixed population size was about 59 (Hours), and for the case of dynamic population size was about 32 (Hours). A summary of the initial and final values of the best solution with dynamic and fixed population size in both domains is given in Tables 5.9, and 5.10. From these tables we note that the quality of the results in both cases were of comparable quality. However, the total run time for the case of dynamic population size was much less than the case of fixed population size. That means with dynamic population size, a saving of about 46% in total run time was obtained for almost the same quality of results as with the case of fixed population size.

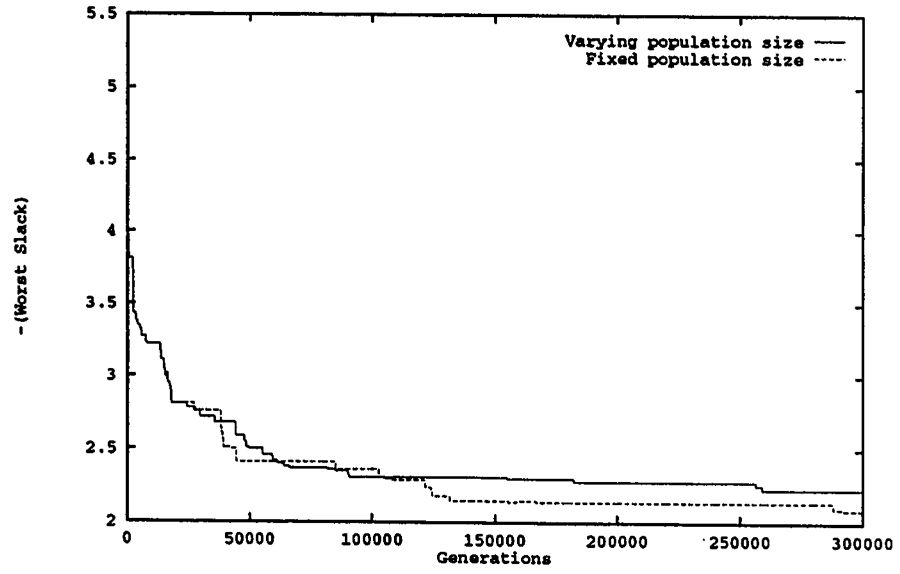

Figure 5.16: The performance of the best solution with dynamic and fixed population size in the time domain.

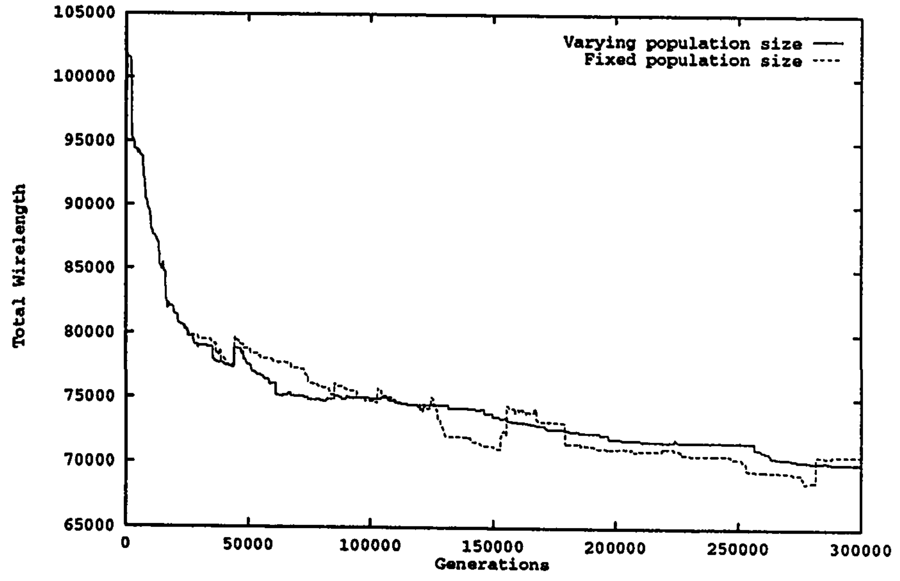

Figure 5.17: The performance of the best solution with dynamic and fixed population size in the space domain.

Population size	Dynamic	Fixed
Initial	5.48	5.48
Final	2.23	2.09
Speed-up %	18.9	19.8
Total run-time (Hrs)	32	59

Table 5.9: A summary of the initial and final values of the best solution with dynamic and fixed population size in the time domain. Values are given in *ns*.

Population size	Dynamic	Fixed
Initial	87786.3	87786.3
Final	69959.1	70580.8
Decrease %	20.3	19.6
Total run-time (Hrs)	32	59

Table 5.10: A summary of the initial and final values of the best solution with dynamic and fixed population size in the space domain. Values are given in microns.

5.8.3 Comparison of Crossover Operators Ψ_1 and Ψ_2

Both crossover operators performed well. However, Ψ_1 performed better than Ψ_2. The performance of both operators is compared in Figures 5.18 and 5.19. Crossover operator Ψ_1 exhibited faster convergence rate and resulted in better solutions than Ψ_2 in both domains.

The sizable difference between the performance of crossover operators Ψ_1 and Ψ_2 can be explained as follows. Operator Ψ_2 tends to disturb the solutions much more than operator Ψ_1. This is because the window size which controls the operation of Ψ_2 is dependent on the positions of cells affecting a chosen critical path. That means

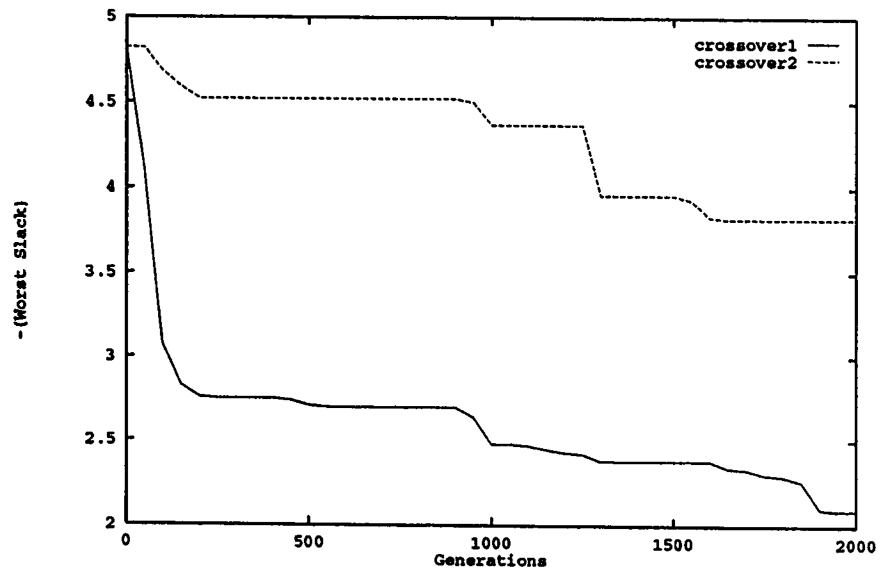

Figure 5.18: The performance of the best solution with crossover1 (Ψ_1) and crossover2 (Ψ_2) in the time domain.

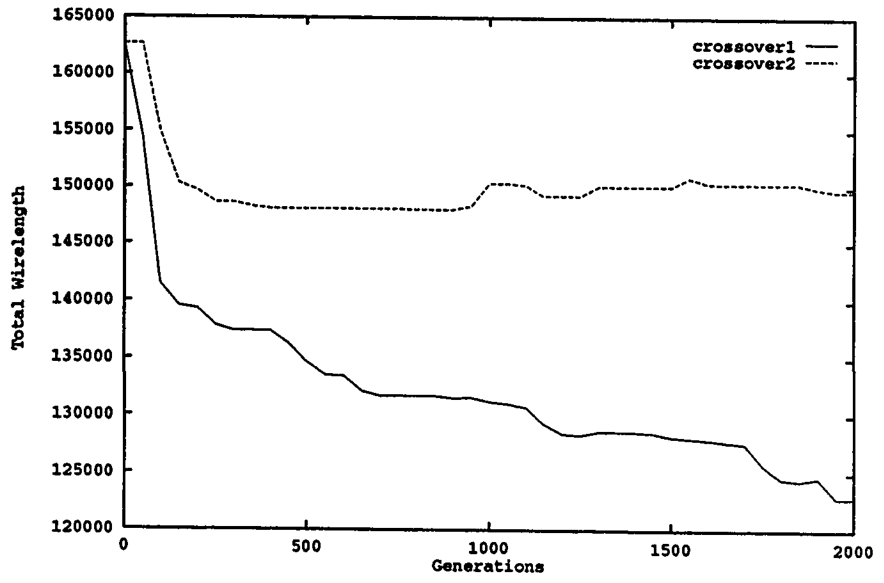

Figure 5.19: The performance of the best solution with crossover1 (Ψ_1) and crossover2 (Ψ_2) in the space domain.

Crossover Operator	Ψ_1	Ψ_2
Initial	4.82	4.82
Final	2.09	3.82
Speed-up %	16.0	5.3

Table 5.11: A summary of the initial and final values of the best solution with different crossover operators in the time domain. Values are given in *ns*.

Crossover Operator	Ψ_1	Ψ_2
Initial	162614.4	162614.4
Final	122870.0	149724.5
Decrease %	24.4	7.9

Table 5.12: A summary of the initial and final values of the best solution with different crossover operators in the space domain. Values are given in microns.

if two cells affecting the chosen critical path are far apart then the window size will be proportionally larger. Having the window size large means that there is a higher chance of disturbing more cells contained in the window where they may not be in direct relation to the chosen critical path. By contrast, in Ψ_1 the disturbance is limited to those cells that are really affecting the chosen critical path. As a result, crossover operator Ψ_1 is chosen as the crossover operator in TDGAP. Tables 5.11, and 5.12 summarize the initial and final values of the best solution with different crossover operators in both domains.

The value of the crossover probability can be varied from problem to another. The effect of different crossover probabilities is shown in Figures 5.20 and 5.21. From these figures, we note that, as we increase the probability of crossover, the solutions

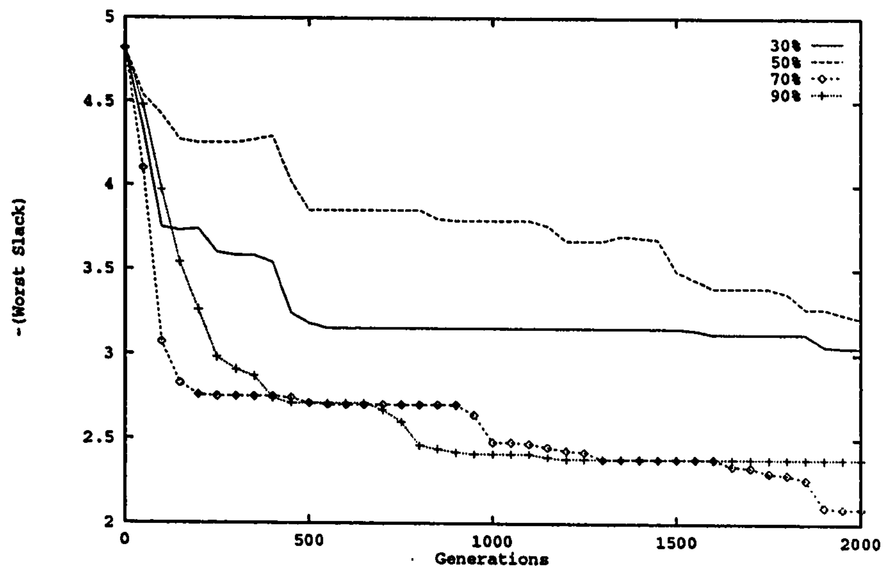

Figure 5.20: The performance of the best solution with different crossover probabilities in the time domain.

tend to converge faster toward a good solution. However, after some high value of crossover probability the convergence rate of the solutions will be too fast which may lead to some local minima that is hard to escape from. Therefore, from these observations, it has been found that a crossover probability between 0.5 and 0.7 is a good choice for our test circuits. A summary of the initial and final values of the best solution with different crossover probabilities in both domains is given in Tables 5.13, and 5.14.

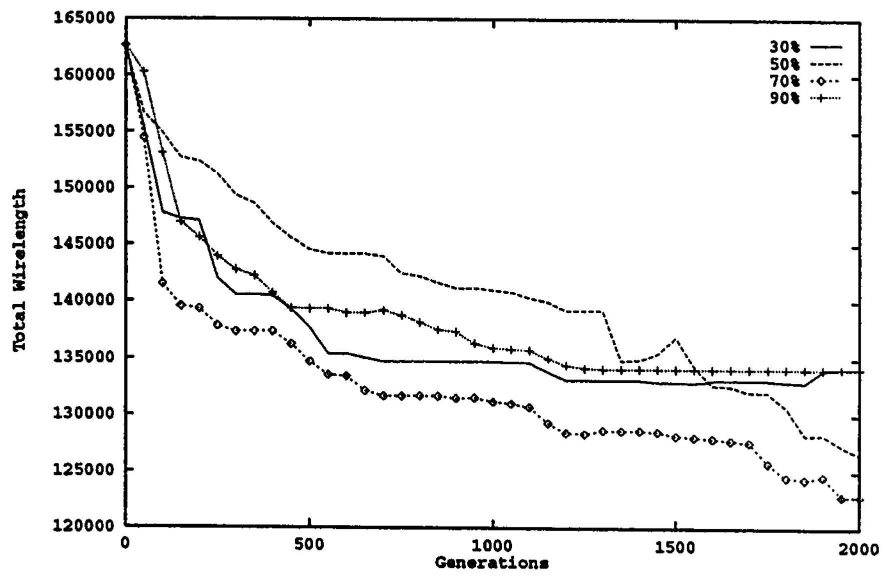

Figure 5.21: The performance of the best solution with different crossover probabilities in the space domain.

Crossover Probability	30%	50%	70%	90%
Initial	4.82	4.82	4.82	4.82
Final	3.04	3.21	2.09	2.38
Speed-up %	9.9	8.8	16.0	14.0

Table 5.13: A summary of the initial and final values of the best solution with different crossover probabilities in the time domain. Values are given in *ns*.

Crossover Probability	30%	50%	70%	90%
Initial	162614.4	162614.4	162614.4	162614.4
Final	134110.8	126537.4	122870.0	134105.0
Decrease %	17.5	22.2	24.4	17.5

Table 5.14: A summary of the initial and final values of the best solution with different crossover probabilities in the space domain. Values are given in microns.

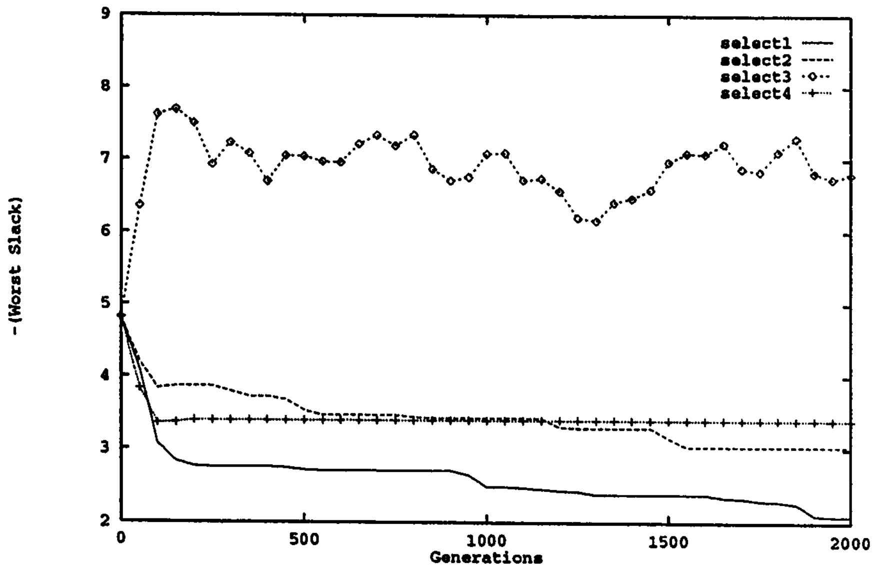

Figure 5.22: The performance of the best solution with selector functions select1 (ρ_1), select2 (ρ_2), select3 (ρ_3), and select4 (ρ_4) in the time domain.

5.8.4 Comparisons of Selector Functions ρ_1, ρ_2, ρ_3, and ρ_4

The selection operation is the gateway to the solutions of the next generation. At this stage some of the solutions are allowed to move to the next generation and some are not allowed. So it is very important to have a good selection function that can help the evolution process converge toward the desired solutions.

Figures 5.22 and 5.23 show the effect of different selector functions in both domains. The behavior of both selector functions ρ_2 and ρ_4 is similar. But selector ρ_2 performed slightly better than ρ_4 because of the limited freedom in selecting some of the individuals at random. Both selectors ρ_2 and ρ_4 favor individuals with high

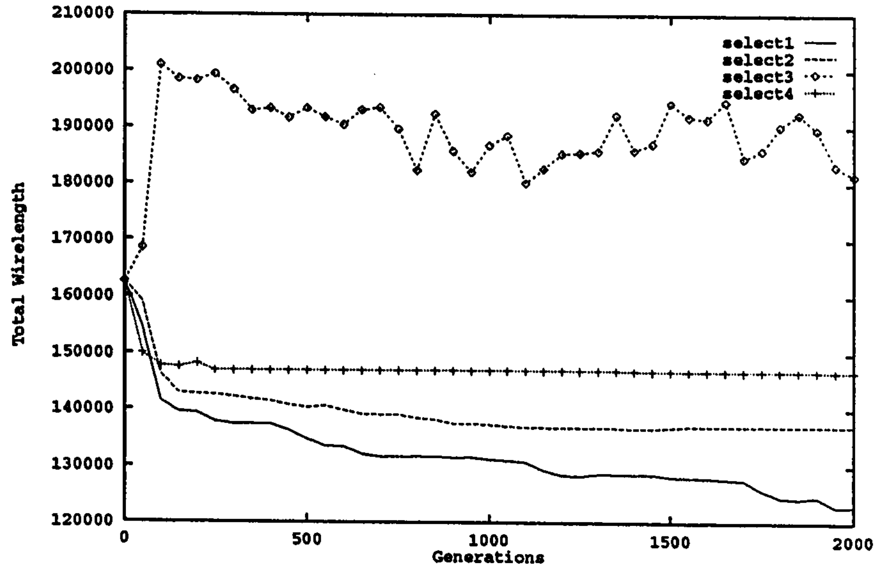

Figure 5.23: The performance of the best solution with selector functions select1 (ρ_1), select2 (ρ_2), select3 (ρ_3), and select4 (ρ_4) in the space domain.

fitness values, which made these selectors more likely to be trapped into a local minima. For example, from the figures of the selector functions we note that selector ρ_4 got trapped into a local minima which it could not escape from.

Selector ρ_3 was the worst among the proposed selectors. This is because selector ρ_3 has almost no control over individuals to be selected. This has caused an unstable performance behavior of the solutions in the next generations. By contrast, selector ρ_1 performed the best among other selectors. The behavior of ρ_1 gave chance for all individuals to be selected at random, but keeping the best of them always as a candidate for the next generation. This behavior saved the search from getting trapped into a local minima as with selector ρ_3, while controlling the evolution

Selector Function	ρ_1	ρ_2	ρ_3	ρ_4
Initial	4.82	4.82	4.82	4.82
Final	2.09	3.03	6.80	3.39
Speed-up %	16.0	9.9	-9.1	7.8

Table 5.15: A summary of the initial and final values of the best solution with different selector functions in the time domain. Values are given in ns.

Selector Function	ρ_1	ρ_2	ρ_3	ρ_4
Initial	162614.4	162614.4	162614.4	162614.4
Final	122870.0	137020.1	181487.1	146663.5
Decrease %	24.4	15.7	-11.6	9.8

Table 5.16: A summary of the initial and final values of the best solution with different selector functions in the space domain. Values are given in microns.

process so as to generate good solutions as with selector ρ_2.

A summary of the initial and final values of the best solution with different selector functions in both domains is given in Tables 5.15, and 5.16.

5.8.5 Comparison of Mutation Operators μ_1 and μ_2

The design of TDGAP has involved a two dimensional objective function that is required to be satisfied. One objective is to improve the timing aspect of the circuit. The other objective is to minimize the overall total wirelength. Therefore, it was a necessary condition to consider both objectives while developing the mutation

Mutation Operator	μ_1	μ_2	$\mu_1 \cup \mu_2$
Initial	4.82	4.82	4.82
Final	1.42	3.17	2.09
Speed-up %	20.7	9.1	16.0

Table 5.17: A summary of the initial and final values of the best solution with different mutation operators in the time domain. Values are given in ns.

operation in TDGAP. To satisfy this condition we have developed two mutation operators μ_1 and μ_2. Operator μ_1 favors the improvement of the timing aspects of the circuit. Operator μ_2 favors the minimization of the total wirelength. The final mutation operator used in TDGAP is a combination of these two operators.

The effect of the different mutation types is shown in Figures 5.24 and 5.25. From these figures, we note the preference of each mutation operator. For example, the improvement in the timing aspects with μ_1 was more than that with μ_2. The combination of these two operators resulted in a good timing improvement that is near the one resulted with μ_1 alone. Also, the combination resulted in a smaller total wirelength than the one obtained with either of the other two operators when individually applied. A summary of the initial and final values of the best solution with the different mutation operators in both domains is given in Tables 5.17, and 5.18.

The effect of allowing the mutation operation to be applied on the best individual or not is shown in Figures 5.26 and 5.27. From these figures it is clear that

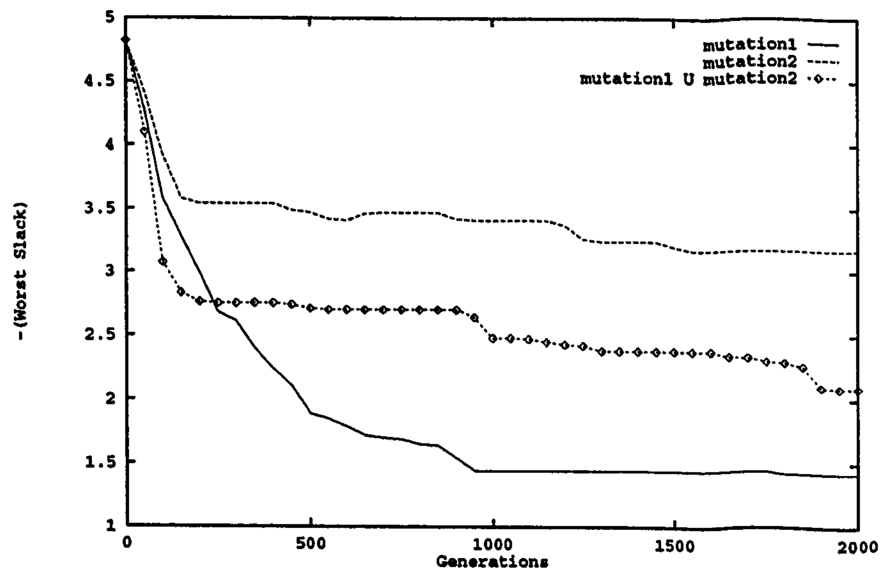

Figure 5.24: The performance of the best solution with mutation1 (μ_1), mutation2 (μ_2), and mutation1 ∪ mutation2 ($\mu_1 \cup \mu_2$) in the time domain.

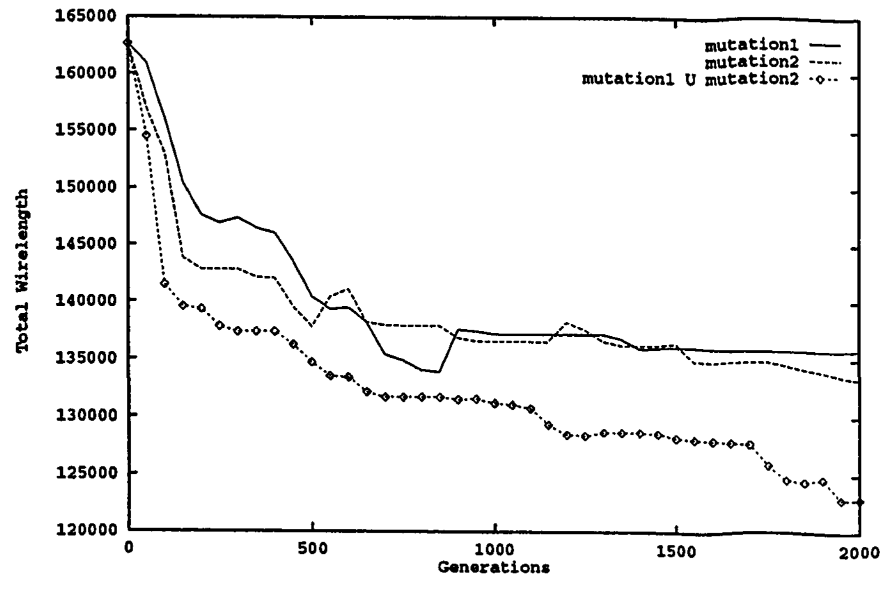

Figure 5.25: The performance of the best solution with mutation1 (μ_1), mutation2 (μ_2), and mutation1 ∪ mutation2 ($\mu_1 \cup \mu_2$) in the space domain.

Mutation Operator	μ_1	μ_2	$\mu_1 \cup \mu_2$
Initial	162614.4	162614.4	162614.4
Final	135827.8	133298.1	122870.0
Decrease %	16.5	18.0	24.4

Table 5.18: A summary of the initial and final values of the best solution with different mutation operators in the space domain. Values are given in microns.

disallowing the best individual from being mutated is better then allowing it. Allowing the best individual to mutate will make TDGAP losses some of its history by losing some of its good individuals. On the other hand, in the case of disallowing the mutation of the best individual, a good deal of the past history is stored in the best individual which is passed to the next generation. If the next generation results in a better individual than the old best then this better individual is considered as a new best. In this case, the search is more guided toward improving the timing aspects and the total wirelength of the circuit.

The value of the mutation probability can be varied from problem to another. However, from our observations, it has been found that a mutation probability around 0.1 is a good choice for our problem. The effect of different mutation probabilities is shown in Figures 5.28 and 5.29. From these figures we note that having high probability such as 15% or 20% disturbed the solutions a lot which made them lose their homogeneity and slowed the rate of convergence. On the other hand, a lower probability such as 5% did not allow the mutation operation to have any real

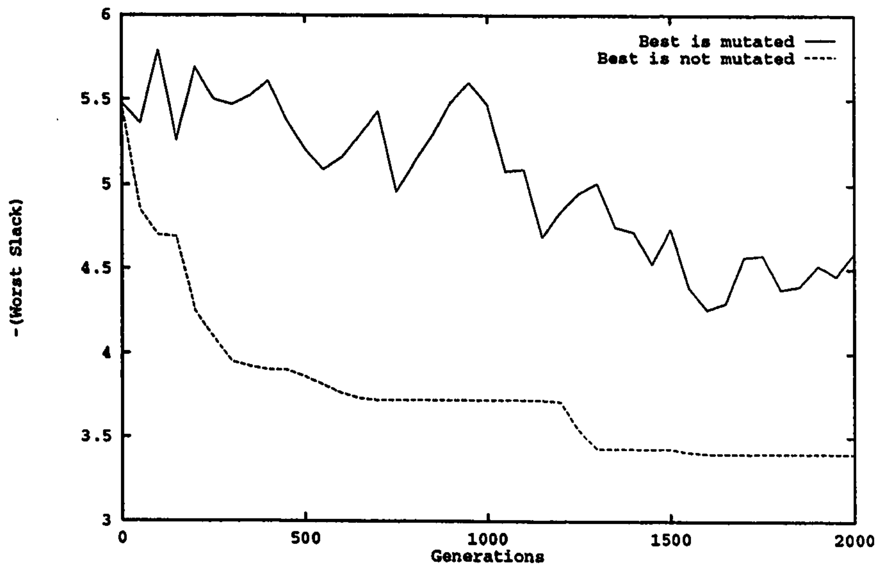

Figure 5.26: The performance of the best solution in case of allowing or disallowing the best individual from being mutated in the time domain.

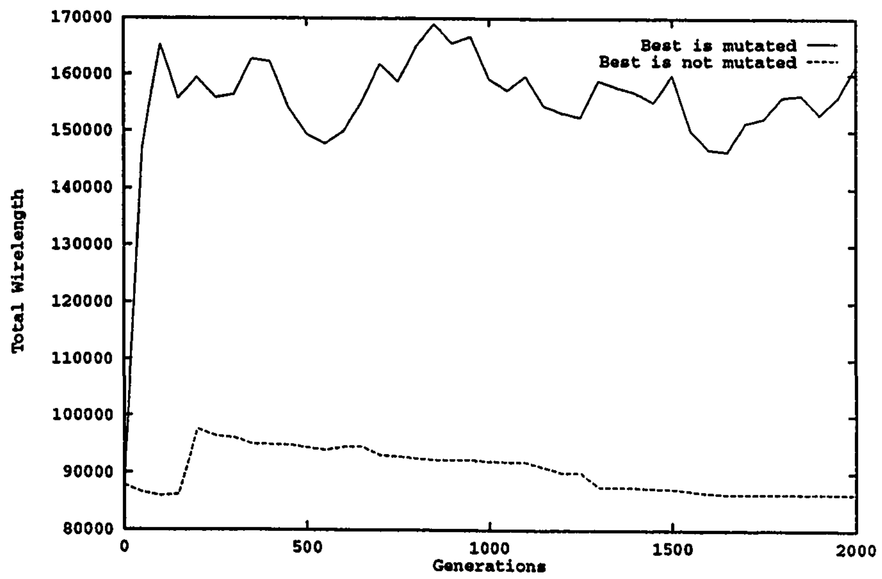

Figure 5.27: The performance of the best solution in case of allowing or disallowing the best individual from being mutated in the space domain.

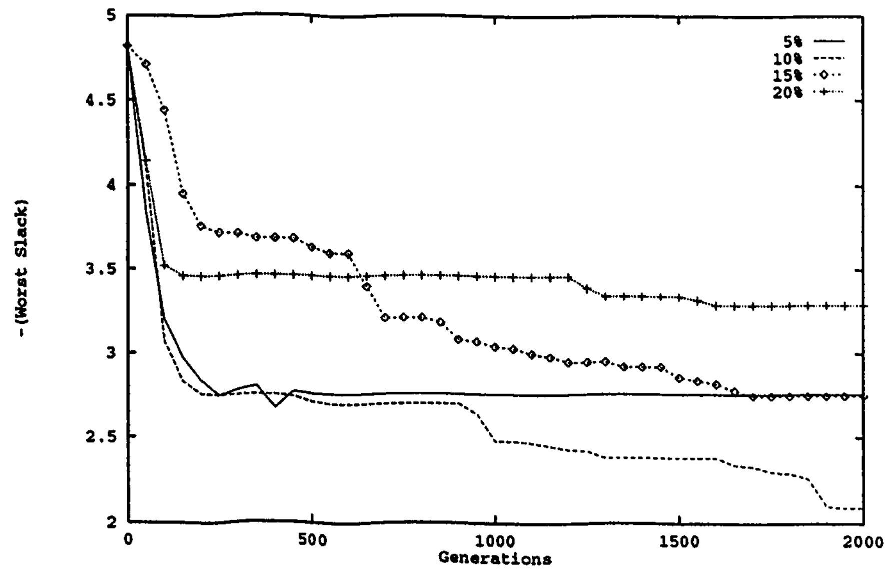

Figure 5.28: The performance of the best solution with different mutation probabilities in the time domain.

Mutation Probability	5%	10%	15%	20%
Initial	4.82	4.82	4.82	4.82
Final	2.76	2.09	2.75	3.29
Speed-up %	11.6	16.0	11.7	8.4

Table 5.19: A summary of the initial and final values of the best solution with different mutation probabilities in the time domain. Values are given in *ns*.

effect. The solutions have been trapped into a local minima. A summary of the initial and final values of the best solution with different mutation probabilities in both domains is given in Tables 5.19, and 5.20.

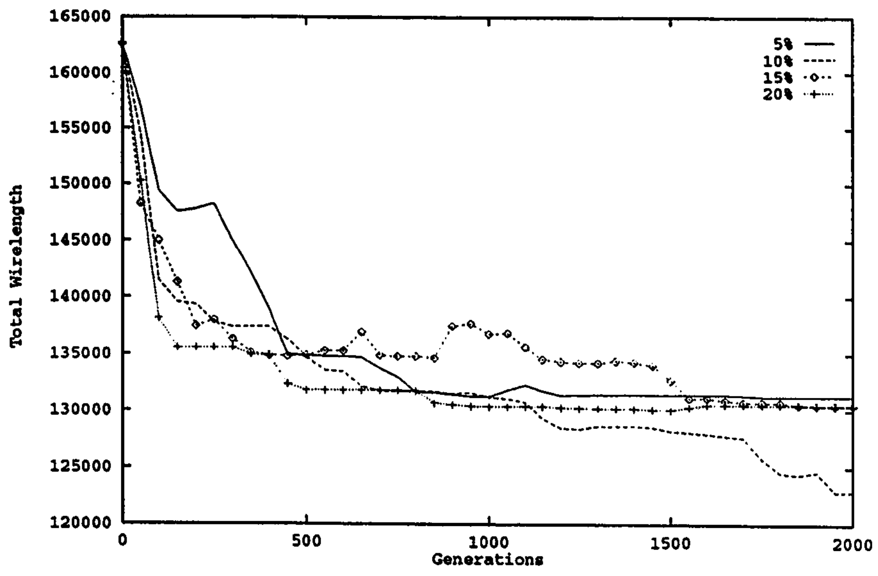

Figure 5.29: The performance of the best solution with different mutation probabilities in the space domain.

Mutation Probability	5%	10%	15%	20%
Initial	162614.4	162614.4	162614.4	162614.4
Final	131281.6	122870.0	130484.5	130464.5
Decrease %	19.3	24.4	19.8	19.8

Table 5.20: A summary of the initial and final values of the best solution with different mutation probabilities in the space domain. Values are given in microns.

Circuit	Format	Bench-mark	Number of cells	Applied clock period (ns)	K reported CP	Number of rows
Ckl	AHPL	No	209	21	200	8
CRC16	AHPL	No	209	18	330	9
Highway	VPNR	No	56	20	14	4
Fract	VPNR	Yes	149	38	368	6
Struct	VPNR	Yes	1952	140	500	22

Table 5.21: Characteristics of the circuits used. CP=critical paths

5.8.6 Discussions

We experimented with several circuits. Some of these are Benchmark circuits (in VPNR format). The characteristics of the circuits used are given in Table 5.21. The function of each circuit given in this table is as follows:

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 simple traffic light controller.
4. **Fract**: A fractional multiplier. The description of this circuit is given in [N$^+$75],
5. **Struct**: A 16-bit multiplier (pure combinational circuit).

A summary of the initial and final values of the best solution with the used circuits in both domains is given in Tables 5.22, and 5.23. In these tables we have

Circuit	*Run time* (Hrs)	*TDGAP*		*OASIS Placer*
		SLACK (ns)	Speed-up %	*SLACK* (ns)
Ckl	11	0.52	8.1	-1.14
CRC16	16	0.61	17.7	-2.47
Highway	6	0.51	11.4	-1.72
Fract	10	0.31	6.5	-2.14
Struct	22	-1.2	0.5	-1.89

Table 5.22: The final $SLACK$ values of five different circuits placed by TDGAP and $OASIS$ placer that is using mincut partitioning algorithm.

made a comparison between the performance of TDGAP and the $OASIS$ placer. The slack values given in Table 5.22 are obtained after the placement phase, but before routing is done. To obtain these values we have developed and used a timing evaluator that checks for timing violations based on the reported K critical paths. The placements obtained by $OASIS$ and TDGAP were evaluated with respect to timing as well as overall wirelength. Improvements up to 17.7 % were obtained with respect to clock speed-up.

The area values given in Table 5.23 are obtained after completing the routing phase and generating the layout. The *Magic* layout editor has been used to view the layouts of the circuits and get their actual height and width. The improvements achieved by TDGAP with respect to timing aspects have resulted in some increase in the overall area. The increase in area is between 1.4% and 9.1%. An example of a placement produced by TDGAP using circuit 'CRC16' is given in Figure 5.30.

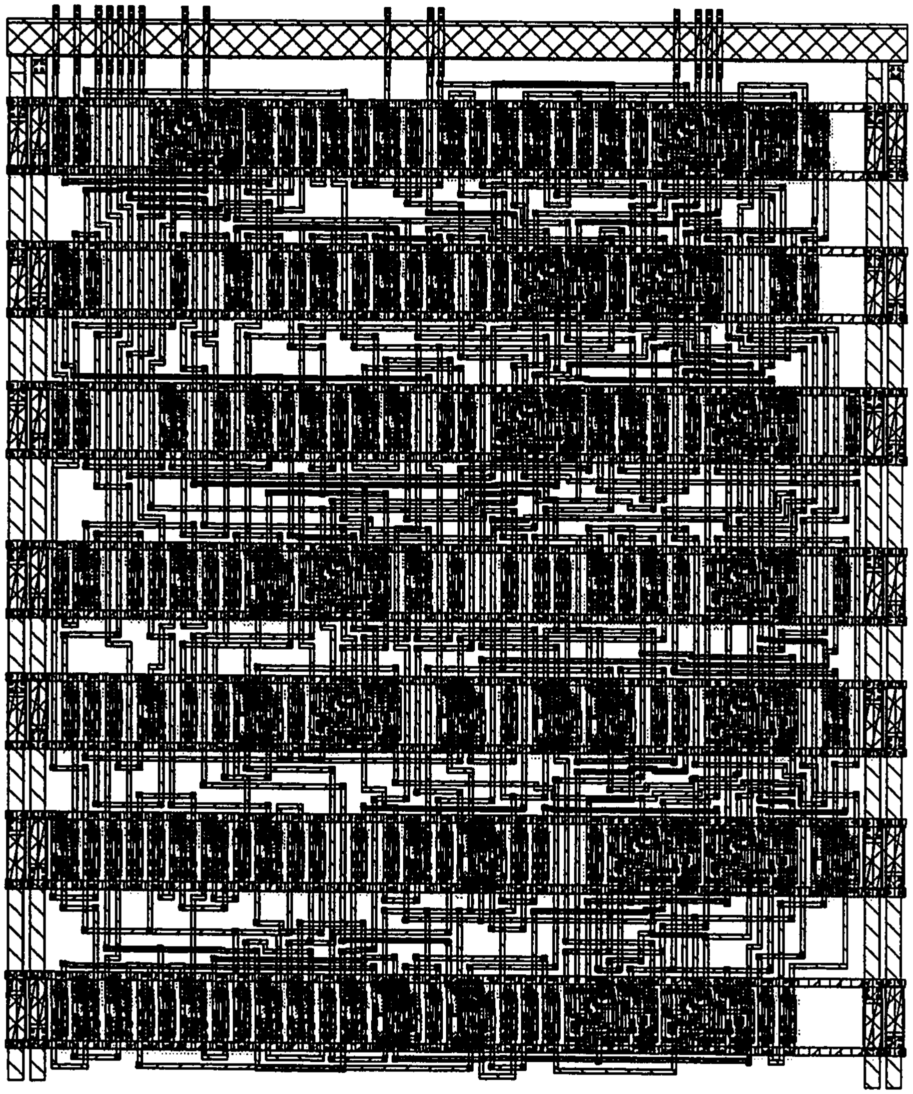

Figure 5.30: A layout example placed using TDGAP.

Circuit	*Run time* (Hrs)	*TDGAP*			*OASIS Placer*	
		Inserted *FTS*	Area (μ^2)	Increase in Area %	Inserted *FTS*	Area (μ^2)
Ckl	11	3	928 ×1016	9.1	13	923 × 936
CRC16	16	10	1131 × 968	5.5	38	1072 × 968
Highway	6	11	478 × 496	6.2	7	465 × 480
Fract	10	44	798 × 824	5.9	50	768 × 808
Struct	22	986	3046 × 2880	1.4	635	3019 × 2864

Table 5.23: The final layout sizes of five different circuits placed by TDGAP and *OASIS* placer that is using mincut partitioning algorithm. Area values are given in terms of Height × Width. *FTS*=Feedthroughs.

TDGAP is implemented in C language. Experiments were performed on a 64-bit DEC Alpha workstation that is running OSF/1 operating system at the speed of 110 MIPS. As an example, Figures 5.31 and 5.32 show the performance of TDGAP in the first 100,000 generations with circuit 'CRC16' in the time and space domains for both the average and best values.

The performance of TDGAP for long run times is shown in Tables 5.24 and 5.25. The values are given with respect to the initial value. From these tables we note that good results were achieved after approximately 100,000 generations for all circuits except for circuit 'Struct', where it was after approximately 5,000 generations. Although the results were changing very slowly during later generations, they were improving most of the time.

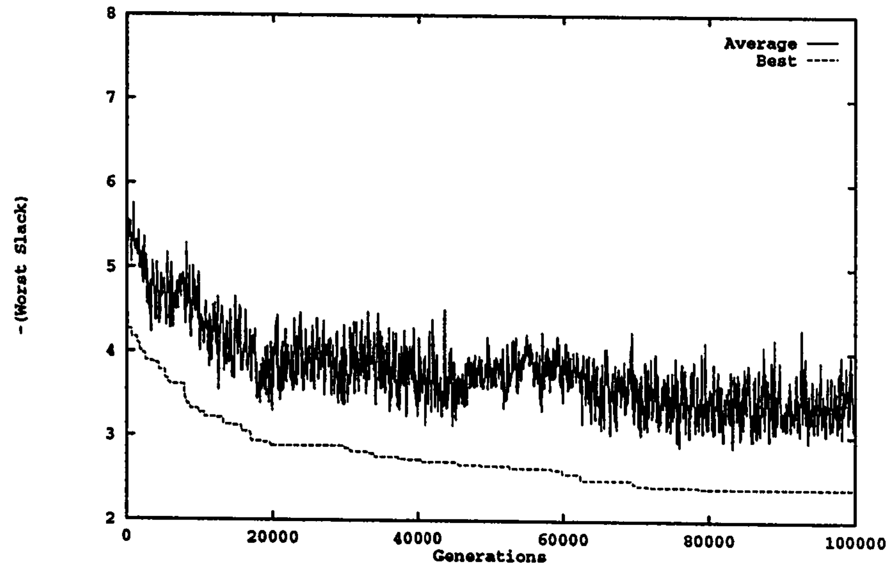

Figure 5.31: The performance of the average and best solution in circuit 'CRC16' in the time domain.

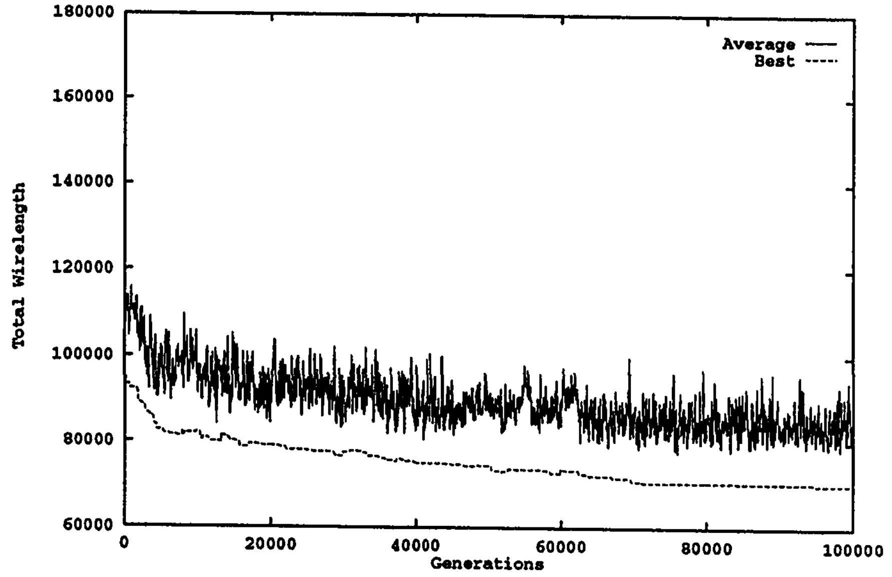

Figure 5.32: The performance of the average and best solution in circuit 'CRC16' in the space domain.

Circuit	*Initial values*		*After 100,000*		*After 200,000*		*After 300,000*	
	S	W	S	W	S	W	S	W
Ckl	1	1	0.22	0.83	0.22	0.81	0.22	0.80
CRC16	1	1	0.44	0.80	0.43	0.78	0.43	0.77
Highway	1	1	0.66	0.73	0.61	0.73	0.61	0.73
Fract	1	1	0.55	0.83	0.52	0.82	0.48	0.82

Table 5.24: The performance of TDGAP after long run times with respect to the initial values. S= $SLACK$, W=Estimated Total Wirelength.

Circuit	*Initial values*		*After 5,000*		*After 10,000*		*After 15,000*	
	S	W	S	W	S	W	S	W
Struct	1	1	0.96	0.81	0.95	0.82	0.94	0.82

Table 5.25: The performance of TDGAP after long run time for circuit 'Struct' with respect to the initial values. S= $SLACK$, W=Estimated Total Wirelength.

Many parameters affect the speed of TDGAP. Among these are the design size and the number of reported critical paths. Table 5.26 shows how the run time of TDGAP is an increasing function of both the design size and the number of reported K critical paths.

To summarize, based on the experiments and above discussions, certain parameters, functions and operators are found to perform better than others. These are summarized in Figures 5.33 and 5.34 and Table 5.27.

Circuit	Run time(Hrs)	Number of cells	K reported critical paths
Ckl	11	209	200
CRC16	16	209	330
Highway	3	56	14
Fract	10	149	368
Struct	22	1952	500

Table 5.26: The run time of TDGAP is a function of the design size and the reported K critical paths. The reported run times were after the first 100,000 generations except for circuit 'Struct', where the reported run time was after the first 5000 generations.

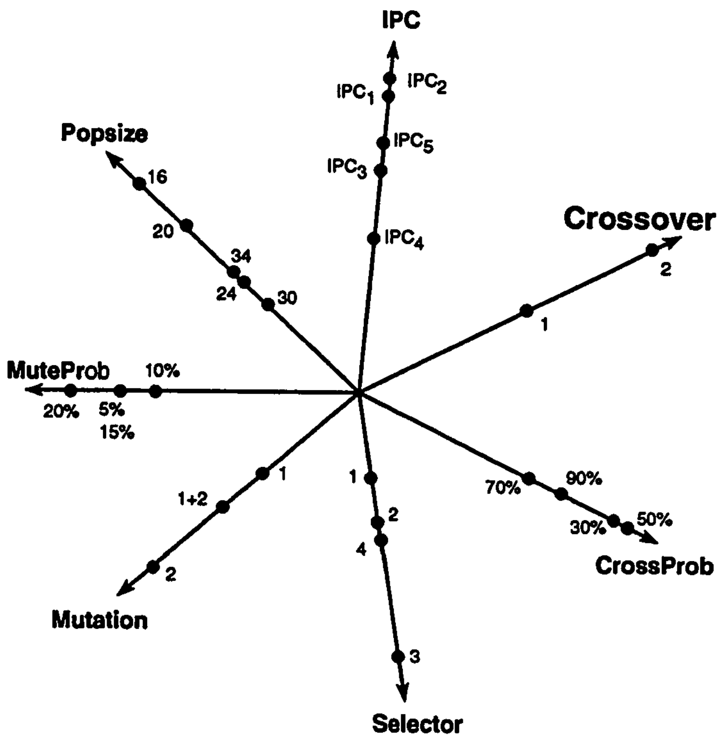

Figure 5.33: A summary of the performance of the parameters, functions and operators of TDGAP in time domain. The nearer the point to the center the better it is.

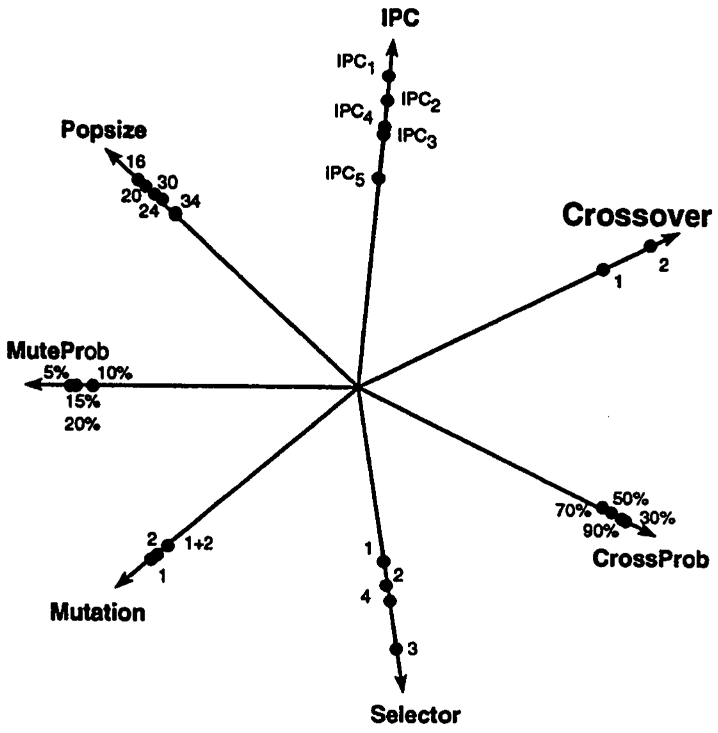

Figure 5.34: A summary of the performance of the parameters, functions and operators of TDGAP in space domain. The nearer the point to the center the better it is.

Constructor function	IPC_5
Crossover operator	Ψ_1
Mutation operator	$\mu_1 \cup \mu_2$
Selector function	ρ_1
Population size	24
Crossover probability	0.5 - 0.7
Mutation probability	0.1

Table 5.27: A summary of certain parameters, functions and operators that are found to perform better than others.

Chapter 6

Conclusion

In this thesis, we have addressed the problem of timing driven placement for standard-cell designs. To solve this problem, we have designed and implemented a Timing Driven Genetic Algorithm for Placement (TDGAP). TDGAP is the first genetic algorithm that is performance driven for standard-cell design style. It is has two main objectives. First, maximize the timing performance. Second, minimize the overall wirelength. A complete discussion of the motivations and the results of our work are presented.

In Chapter 1, the general placement problem is defined. Furthermore, the standard-cell design style which has been adopted in our work is introduced. The most popular cost function applied to the placement problem is the total wirelength.

However, there is a great demand to include the circuit performance in the score function. In Section 1.3 the main motivation behind including the circuit performance in the score function is discussed.

In Chapter 2, a survey on previous works in the field of placement is introduced. In this chapter, the two major approaches of the placement problem (iterative, constructive) are discussed. Also, five classes of placement algorithms are discussed. These are simulated annealing, mincut, force-directed, numerical optimization, and genetic algorithms. The routing problem is an important issue that the placer should take into account. In Section 2.4 the global routing is briefly discussed.

In Chapter 3, the modeling problem and its constraints are discussed. This chapter includes the problem definition, the inputs and outputs, and the AHPL and VPNR formats. In Section 3.4 some timing issues are presented. These issues include the linear delay model which has been adopted in our work. This model is discussed in Section 3.4.2. Furthermore, in Section 3.5, the semi-perimeter estimation method is discussed. This method suffers from the problem of underestimating the wirelength of large net sizes. Therefore, we have improved this method by including an inflation factor in our estimate for certain net sizes.

In Chapter 4, the general genetic algorithm (GA) is introduced. This has included the main operations used in GA which are: choice, crossover, selection, and mutation. In Section 4.2 two GAs (Genie and GASP) for the placement problem are introduced. An overview on Genie, which is a genetic placement algorithm developed by Cohoon and Paris in 1986, and GASP, which is a genetic algorithm for standard-cell placement developed by Shahookar and Mazumder in 1990, is given in Section 4.2. This has included the types of operations they have applied and their results.

In Chapter 5, a Timing Driven Genetic Algorithm for Placement (TDGAP) is discussed. This is the major part of our work. In this chapter, a detailed discussion of the implementation of TDGAP is given. Many different functions and operators related to TDGAP are designed and tested. These include:

- Five Initial Population Constructors:
  - IPC_1: Random.
  - IPC_2: Cluster cells affecting the same path, start row=0.
  - IPC_3: Same as IPC2, but start row=$\frac{n}{2} - 1$.
  - IPC_4: 25% IPC1 + 75% IPC3.
  - IPC_5: Same as IPC4, but include a solution by mincut partitioning.
- Two Crossover operators:
  - Ψ_1: Maintain the same locations of these cells affecting a satisfied path.

  - Ψ_2: Keep the cells affecting a satisfied path within a certain boundary.

- Four Selector functions:
  - ρ_1: Best and rest at random.
  - ρ_2: Best 10% and rest at random.
  - ρ_3: All at random.
  - ρ_4: Competitive basis, each individual has a certain probability.

- Two Mutation operators:
  - μ_1: Directed to improve the timing aspects.
  - μ_2: Directed to improve the total wirelength.

Many experiments were run to determine the best parameters of TDGAP, for example, best population size, best crossover operator, and best crossover probability. In Section 5.8, discussions and comments on the results of the experiments are given.

Iterative algorithms such as TDGAP take long time to produce good results compared to constructive algorithms. Therefore, to reduce the total run time we have tested the case of dynamic population size where we reduced the population size as the run goes on. Preliminary results of TDGAP showed a speed up of about 2 in the case of dynamic population size over the case of fixed population size.

Finally, improvements up to 17.7 % were obtained with respect to clock speed-up of the tested examples with a slight increase in area between 1.4% and 9.1% compared to a non timing driven placer that is based on the mincut partitioning algorithm.

As a continuation of this work we propose the following future works:

- Extend TDGAP to accept general cell design style.
- Fine tuning TDGAP parameters in the case of dynamic population size with more test cases.
- Design TDGAP such that it dynamically selects the best parameters while running.
- Design and implementation of a timing driven router that can be integrated with TDGAP.

Appendix A

Using OASIS System with TDGAP

OASIS system [MCN90] provides the user with some control variables where he can run *OASIS* in different modes. For the purpose of completing the generation of the physical layout from the placement produced by TDGAP and comparing the results of TDGAP with an area driven placement tool, we have used the placer and the router that are implemented in the *OASIS* system. Here we will introduce the commands that have been used with the *OASIS* system with brief explanation.

A.1 The Placer of OASIS

The placement program used with *OASIS* is based on mincut partitioning algorithm. The objective of the placer is the minimization of the total wirelength. It accepts its input from a file describing the circuit which is written in VPNR format. This input file is an unplaced domain of the circuit modules. It has an extension of '.vpnr', and must be in a sub-directory called 'layout'. The placement program is called *cplrtnew* which generates a placed domain of the circuit and outputs the result in a file that has '.scan' as an extension. To obtain the placed domain of the circuit the following line command is used from a directory which has the 'layout' sub-directory:

```
oasis scandata rows=Number_of_Rows
```

The word 'scandata' indicates the goal of the run. It commands the *OASIS* system to stop after generating the file '*File_Name*.scan'. The variable '*Number_of_Rows*' is an integer indicating the total number of rows to be used for placing the cells by *cplrtnew* program.

The two most important files produced are '*File_Name*.placer' and '*File_Name*.scan'. These two files are written in the 'layout' sub-directory. They are needed to complete the design process and produce the routed layout of the circuit.

A.2 The Router of OASIS

Two routing programs are applied. One is the global router. The second is the detailed router. The detailed router program used with *OASIS* is based on the left-edge routing algorithm. The global router program is called 'dtglrtnew' and the detailed router program is called 'mcroute'. To use these programs and obtain the final routed layout in *MAGIC* format the following line command is used from a directory which has the 'layout' sub-directory that includes both files '*File_Name*.placer' and '*File_Name*.scan':

oasis magicfiles rows=*Number_of_Rows* powerargs=""

The goal here is to reach the completed and routed layout files written in *MAGIC* format. The option 'powerargs=""' is an argument for the power router program. It commands the power router program to include all the Power and Ground rails in the final layout. For this variable to work it needs the variable 'rows=' to be specified.

The final output files are those files that describe the completed layout of the circuit at the mask level. They can be found in a sub-directory called '*File_Name*.magic' which is under the 'layout' sub-directory.

Bibliography

[AF] K. Al-Farra. Timing-driven floorplanning. MS. Thesis, Dept. of COE, KFUPM, Dhahran. In preparation.

[AS85] T. Asano and S. Sato. Long path enumeration algorithms for timing verification on large digital systems. *Graph theory with applications to algorithms and computer sciences, John Wiley*, pages 25–35, 1985.

[Bra87] H. Nelson Brady. Automatic placement and routing techniques for gate array and standard cell designs. *Proc. of The IEEE, Vol. 75, No. 6, June*, pages 797–806, 1987.

[Bre77] M. A. Breuer. A class of min-cut placement algorithms. *Proc. 14th Design Automation Conference*, pages 284–290, 1977.

[CP87] James P. Cohoon and William D. Paris. Genetic placement. *IEEE Transactions on Computer-Aided Design*, 6:956–964, November 1987.

[D+87] W. M. Dai et al. BEAR: a new building-block layout system. *Proc. IEEE International Conference on CAD*, pages 34–38, 1987.

[DNA+90] Wilm E. Donath, Reini J. Norman, Bhuwan K. Agrawal, Stephen E. Bello, Sang Young Han, Jerome M. Kurtzberg, Paul Lowy, and Roger I. McMillan. Timing driven placement using complete path delays. *27th ACM/IEEE Design Automation Conference*, pages 84–89, 1990.

[Don80] W. Donath. Complexity theory and design automation. *Proc. 17th Design Automation Conference*, pages 412–419, 1980.

[FCW67] C. J. Fisk, D. L. Caskey, and L. E. West. Automated circuit card etching layout. *Proc. IEEE*, November 1967.

[Fre93] James F. Frenzel. Genetic algorithms. *IEEE Potentials*, pages 21–24, October 1993.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, Inc., 1989.

[Gre85] J. J. Grefenstette. Ed. 1985. *Proc. of an International Conference on Genetic Algorithms and their Applications. Pittsburgh, Penn*, 1985.

[Gre87] J. J. Grefenstette. Ed. 1987. *Proc. of the 2nd International Conference on Genetic Algorithms and their Applications. Cambridge, Mass*, 1987.

[Hal70] K. M. Hall. An r-dimensional quadratic placement algorithm. *Manage. Sci.*, pages 219–229, November 1970.

[HNY87] P. S. Hauge, R. Nair, and E. J. Yoffa. Circuit placement for predictable performance. *Proc. IEEE International Conference on CAD*, pages 88–91, 1987.

[Hol75] J. Holland. *Adaption in natural and artificial systems.* University of Michigan Press, Ann Arbor, Mich, 1975.

[HS71] A. Hashimoto and J. Stevens. Wire routing by optimization channel assignment within large apertures. *Proc. of 8th Design Automation Conference*, pages 155–169, 1971.

[HWA76] M. Hanan, P. K. Wolf, and B. J. Agule. A study of placement techniques. *Journal of Design Automation and Fault-Tolerant Computing*, pages 28–61, October 1976.

[ID90] Lin Ichiang and David H. C. Du. Performance-driven constructive placement. *27th ACM/IEEE Design Automation Conference*, pages 103–106, 1990.

[JK89] Michael A. B. Jackson and Ernest S. Kuh. Performance-driven placement of cell based IC's. *26th ACM/IEEE Design Automation Conference*, pages 370–375, 1989.

[JKMS87] M. A. B. Jackson, E.S. Kuh, and M. Marek-Sadowska. Timing-driven routing for building block layout. *Proc. IEEE International Symposium on circuits and Systems*, pages 518–519, 1987.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, pages 671–680, May 1983.

[MAS$^+$90] Robert N. Mayo, Michael H. Arnold, Walter S. Scott, Don Stark, and Gordon T. Hamachi. *1990 DECWRL/Livermore Magic Release.* Western Research Laboratory, Palo Alto, California, September 1990. CMOS 2.0μ P-well parameters.

[MCN90] MCNC Group. *OASIS 2.0 Reference Manual*, 1990.

[MS86] M. Masud and Sadiq M. Sait. Universal AHPL - A language for VLSI design automation. *IEEE Circuits and Devices Magazine*, 2:8–13, September 1986.

[N+75] H. Troy Nagle et al. *Intro to Computer Logic.* Prentice Hall, 1975. page 461.

[Oht86] T. Ohtsuki. *Advances in CAD for VLSI: Layout Design and Verification*, volume 4. North-Holland, 1986.

[OI+86] Y. Ogawa, T. Ishii, et al. Efficient placement algorithms optimizing delay for high speed ECL masterslice LSI's. *Proc. 23rd Design Automation Conference*, pages 404–410, 1986.

[Orb92] Orbit Semiconductor Inc., Sunnyvale, California. *Foresight Manual*, July 1992.

[PDS76] A. Perskey, D. Deutch, and D. Schweikert. A system for the directed automatic design of LSI circuits. *Proc. of 13th Design Automation Conference*, June 1976.

[PG78] B. T. Preas and C. W. Gwyn. Methods for hierarchical automatic layout of custom LSI circuit masks. *Proc. 15th Design Automation Conference*, pages 206–212, 1978.

[SCK92] 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.

[SL88] C. Sechen and K. W. Lee. An improved simulated annealing algorithm for row-based placement. *Proc. IEEE Int. Conf. Computer-Aided Design*, pages 478–481, November 1988.

[SM90] K. Shahookar and P. Mazumder. A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Transaction on Computer Aided Design*, pages 500–511, May 1990.

[SM91] K. Shahookar and P. Mazumder. VLSI cell placement techniques. *ACM Computing Surveys, Vol. 23, No. 2 June*, pages 143–220, 1991.

[SS90] Suphachai Sutanthavibul and Eugene Shragowitz. An adaptive timing-driven layout for high speed VLSI. *27th ACM/IEEE Design Automation Conference*, pages 90–95, 1990.

[SSV85] C. Sechen and A. Sangiovanni-Vincentelli. The timberwolf placement and routing package. *IEEE Journal of Solid-State Circuits*, 20:510–522, April 1985.

[SY94] Sadiq M. Sait and Habib Youssef. *VLSI Design Automation: Theory and Practice.* Mc-Graw Hill Book Co., Europe, 1994. In press.

[VK83] M. Vecci and S. Kirkpatrick. Global wiring by simulated annealing. *IEEE Transaction on CAD*, pages 215–222, October 1983.

[You90] Habib Youssef. Timing analysis of cell based VLSI designs. *Computer and Information Sciences*, January 1990. PhD thesis, University of Minnesota.

[YSS92] H. Youssef, E. Shragowitz, and S. Suthanthavibul. Prelayout timing analysis of cell-based VLSI designs. *Computer Aided Design*, 24(7):367–379, July 1992.

Vita

- Khaled Muhammad Walid Nassar
- Born in 1969 Riyadh, Saudi Arabia.
- Received a Bachelor of Science degree in Computer Engineering in 1991 from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.
- Received a Master of Science degree in Computer Engineering in 1994 from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.