ECO TIMING OPTIMIZATION USING SPARE CELLS AND TECHNOLOGY REMAPPING

Student: Yen-Pin Chen   Advisor: Dr. Yao-Wen Chang

Department of Electrical Engineering
National Taiwan University

Abstract (Chinese)

ECO TIMING OPTIMIZATION USING SPARE CELLS AND TECHNOLOGY REMAPPING

Student: Yen-Pin Chen Advisor: Dr. Yao-Wen Chang

Department of Electrical Engineering
National Taiwan University

Abstract

Spare cells rewiring is a technique used to fix defects or deficiencies after the placement stage. It is traditionally done by manual work but becomes extremely hard nowadays. In this thesis, we propose the first spare cells selection algorithm consisting of two phases to optimize timing of the circuit by rewiring spare cells. In the first phase, we apply gate sizing and buffer insertion to all timing violated paths to fix timing violations. In the second phase we further fix timing violations by extracting timing critical parts and apply technology remapping to them. Experimental results based on five industrial benchmarks show that our algorithm can fix up to 99.82% of the total negative slack, and the runtime is very short. The experimental results show that our algorithm is efficient and effective.

Acknowledgements

First, I would like to thank my thesis advisor, Dr. Yao-Wen Chang, for his guidance in the past two years. His tireless instruction leads to my achievement. I have also learned lots abilities about technical researching, specialized presentation, academic discussion and survey methodology. He also provides plenty of suggestions and encouragement when I was pursuing my master degree. The success of this thesis is totally owing to him.

Second, I want to thank Mr. Jyh-Herng Wang and Mr. Wang-Jin Chen. They propose this research topic and help me very much.

Third, I also want to express my appreciation to all members of the Electronic Design Automation Laboratory, including Tsung-Yi Ho, Tai-Chen Chen, Ping-Hung Yuh, Tung-Chieh Chen, Zhe-Wei Jiang, Chen-Feng Chang, and I-Jye Lin.

Finally, I'm going to thank my parents and friends for concerning themselves with me these days. I cannot overcome the difficulty without their encouragement.

Dedicate this thesis to my parents!

Jia-Wei Fang

National Taiwan University

July 2005

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we introduce the ECO timing optimization problem and the spare cells rewiring technique. Related work and our contributions are also included in this chapter, and the organization of this thesis is in the last part.

1.1 Using Spare Cells to Optimize Timing

ECO (Engineering Change Order) is usually performed during the chip implementation cycle. If engineers need to change only a small portion of the netlist in a very short time, running the traditional back-end design flow to the whole netlist is very time-consuming. The most efficient way is to change the netlist locally without affecting other parts of the chip. Using spare cells is a good choice for this purpose because rewiring the circuit by spare cells can change the netlist without changing the chip placement. Engineers do not need to run placement tools to place the netlist after the re-wiring process. Since timing closure is hard to be achieved in today's nanometer designs and engineers have to run the back-end design flow many times to meet timing constraints, using spare cells to do netlist changes can save a lot of time and effort. Besides, if masks are already produced before netlist change, rewiring the netlist using spare cells only needs the masks of the routing layers to be re-produced. This will save a large amount of production cost because masks are quite expensive in the nanometer process.

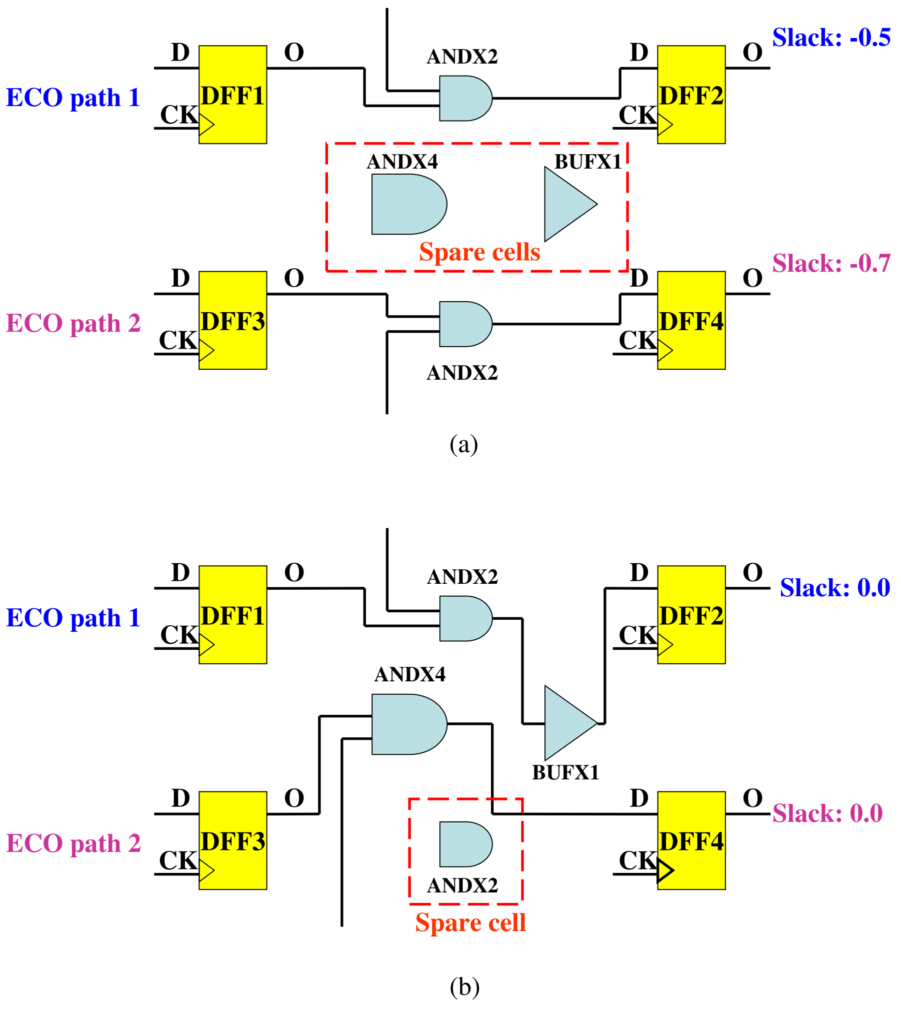


Figure 1.1: (a) ECO paths before rewiring. (b) ECO paths after rewiring.

Although spare cells rewiring is a very effective ECO technique, using it to fix timing deficiencies is getting tougher and tougher nowadays. This is because the local netlist change cannot consider its effect on the circuit timing and makes the circuit fail to achieve timing closure. Additionally, increasing of the gate count of chip designs also makes the problem substantially harder. Thus we need an efficient algorithm to deal with the problem of timing optimization by spare cells.

Figure 1.1 shows an instance of timing optimization by rewiring spare cells. The AND gate ANDX4 and BUFFER gate BUFX1 are spare cells and not connected to any path. Gate DFF1, gate DFF2, gate DFF3, and gate DFF4 are D flip-flops. They are start points and end points of path 1 and path 2. Arrival times of DFF2 and DFF4 are larger than the clock cycle, and the timing of path 1 and path 2 needs to be fixed by ECO. Thus we call these two timing violated paths ECO paths. We can improve the timing of ECO path 1 by inserting the adjacent BUFFER gate BUFX1 in that path to help driving the load. To fix the timing of ECO path 2, we can use the AND gate ANDX4 instead of the AND gate ANDX2 on ECO path 2 because ANDX4 has a larger driving capability. After the sizing and buffering operations, both ECO paths meet the timing constraints. The AND gate ANDX2 is now released from the netlist and becomes a spare cell.

Spare cells are designed for further design changes, and are evenly placed on the chip layout. The type and number of spare cells vary from different chip characteristics, and are usually determined empirically. The number of spare cells is usually small compared to other standard cells. Thus using spare cells to perform ECO operations needs to consider the resource sharing problem. Figure 1.2 by [5] shows the spare cells placement. The spare cells are plotted as white points, and they are spread throughout the standard cell area for good performance.

1.2 Previous Work

Spare cells selection is a very common technique in industrial designs, and it was traditionally done by manual labor. Since the gate count of a chip design is increasing and timing closure is more and more difficult to be achieved, this problem becomes so tough that it can no longer be solved by manual ways.

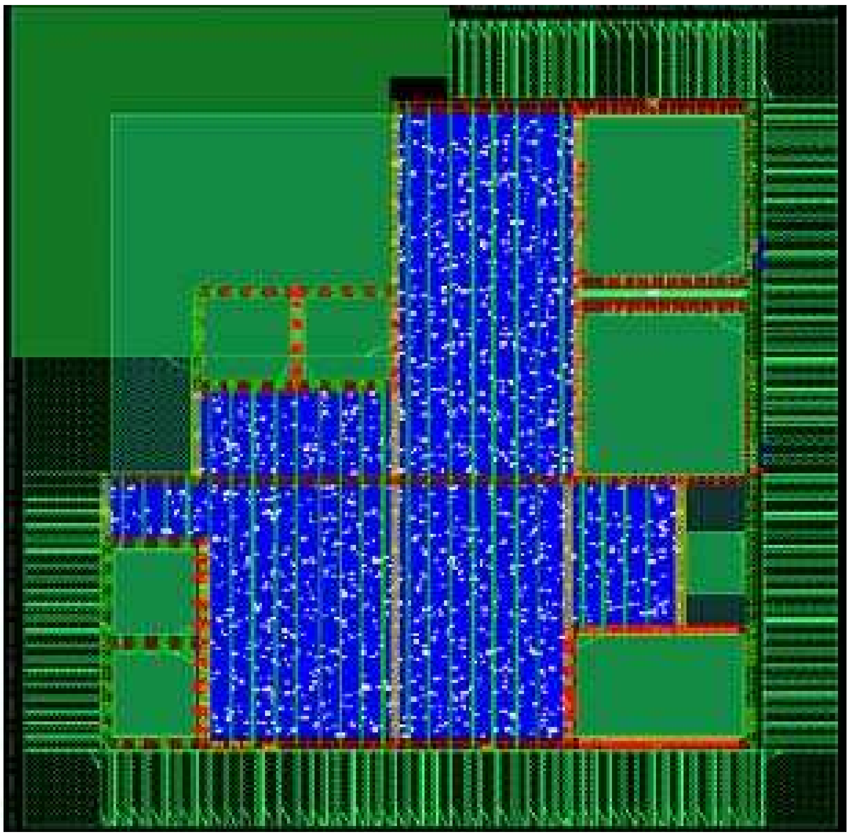


Figure 1.2: Spare cells placement.

To the best knowledge of the authors, there is still no published literature for the ECO timing optimization by spare cells. There are two topics related to this problem: (1) buffer insertion and gate sizing problem, and (2) physical and logic co-synthesis.

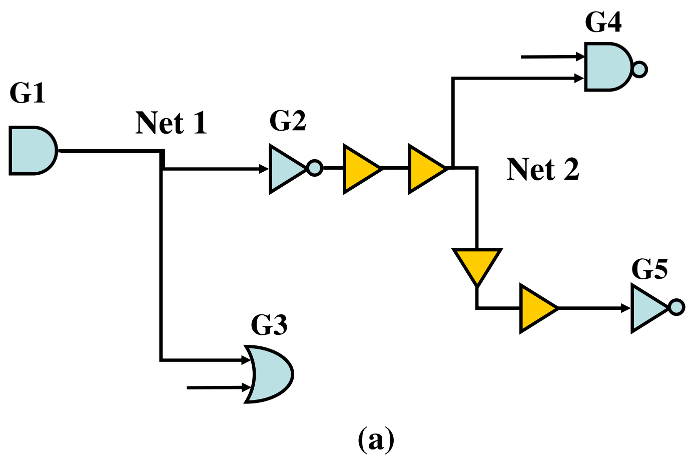
1.2.1 Buffer Insertion and Gate Sizing

Buffer insertion is a well-known technique for timing optimization. It can not only reduce the path delay but also eliminate signal noise. Authors of [6] proposed a dynamic programming method for the buffer insertion problem. When the candidate buffer locations of a signal net are known, the dynamic programming method (VGDP) can find the maximum timing slack solution in quadratic time. Many works are proposed based on this method, and they can be grouped into three categories: (1) net based buffer insertion algorithms, (2) network based buffer insertion algorithms, and (3) path based buffer insertion algorithms.

Net based buffer insertion algorithms find optimal buffer solutions in each signal net. Buffer insertion is performed on each net individually. This method is simple and previous works have shown excellent theoretical results. Authors of [27] reduced the timing complexity from $O(n^2)$ of [6] to $O(n \log n)$, where n is the number of candidate buffer locations. In [7] they further proposed a method to insert buffers into nets of m sinks in $O(mn)$ time. Although this method can get a good result for a single net, it lacks the global information of the whole circuit. Figure 1.3 shows an example. There are two nets, Net 1 and Net 2, along a critical path of the circuit. If we perform buffer insertion to Net 2 first and use 4 buffers, then we do not need to insert buffers into Net 1 because the timing constraints are already met. However, if we insert one buffer into both Net 1 and Net 2, we can get a solution meeting the constraints by using fewer buffers. Net-based buffer insertion methods usually result in sub-optimal solutions because of over-consuming buffer resource.

Network based buffer insertion algorithms [14] [15] use a directed acyclic graph (DAG) to represent the circuit, and apply Lagrangian relaxation to translate the timing constraints to costs of the objective function. This method can find a global buffering solution but usually has a large run time, especially for industrial benchmarks.

Path-Based buffer insertion method [28] considers buffering and gate sizing from a global view. Nets on a timing violated path are merged, and the whole path looks like a big routing tree. Then buffering and sizing operations are performed to the path like van Ginneken's algorithm.



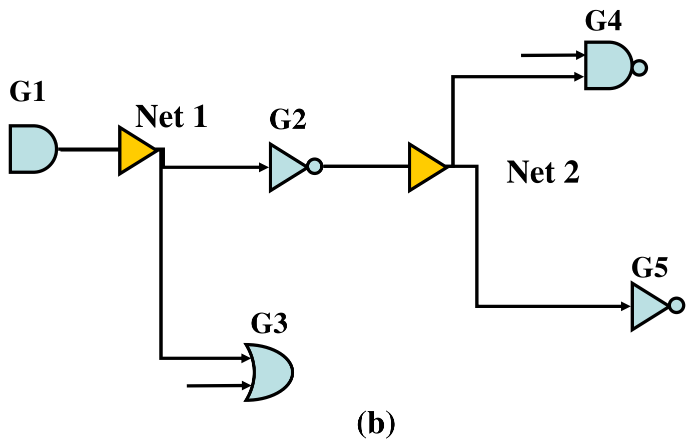


Figure 1.3: (a) Net-based buffer insertion may be trapped in a local optimal solution. (b) A better buffering solution.

1.2.2 Physical and Logic Co-Synthesis

Logic synthesis and placement are two important stages of the IC design flow. Logic synthesis tools translate functions into logic gates, and placement tools place them on the chip layout. Since the logic synthesis tools do not have the information of exact positions of gates, they cannot fully optimize the circuits in the right way. On the other hand, placement tools have limited flexibility to optimize because they cannot change the circuit netlist. Thus combining the two stages has been an

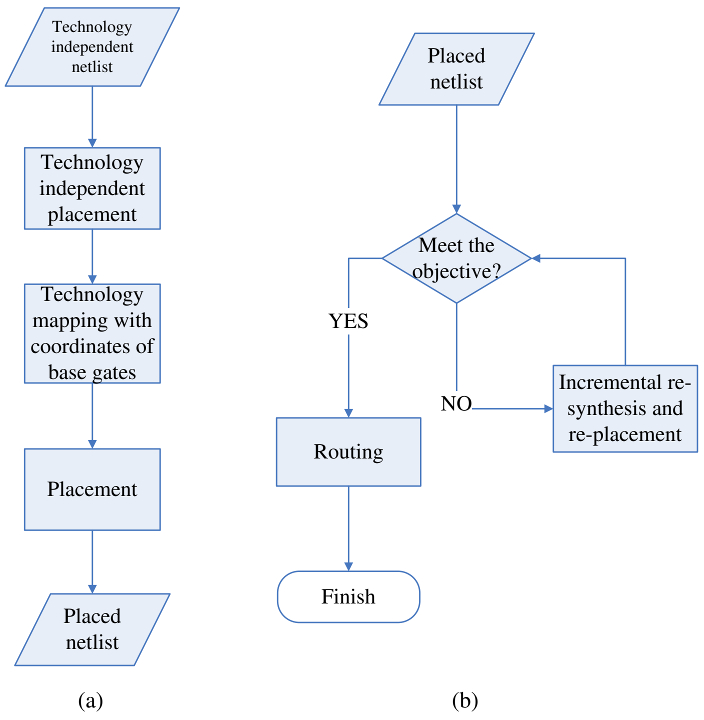

Technology independent netlist
Technology independent placement
Technology mapping with coordinates of base gates
Placement
Placed netlist
(a)
Placed netlist
Meet the objective?
YES
NO
Routing
Incremental re-synthesis and re-placement
Finish
(b)

Figure 1.4: (a) Layout driven logic synthesis flow. (b) Local netlist transformation flow.

attractive topic in recent years. Previous works can be grouped into two categories: (1) layout driven logic synthesis, and (2) local netlist transformation.

Layout driven logic synthesis methods [11], [12], [23], and [25] generate an initial placement of the technology-independent netlist first, and then optimize the netlist using the coordinates given by the initial placement. The accuracy of these methods is not guaranteed because the final placement is likely to be much different from the initial placement. The design flow of layout driven logic synthesis is shown in Figure 1.4 (a).

Local netlist transformation methods [18] [22] work on a placed netlist. They focus on different objectives such as timing or power. Critical parts of the placed netlist are extracted, re-synthesized according to the target objective, and then re-placed. This method preserves the existing placement as much as possible. Local netlist transformation flow is shown in Figure 1.4 (b).

1.3 Our Contributions

To our best knowledge, this thesis is the first work for the ECO timing optimization by spare cells rewiring. The major difference between this problem and the traditional buffer insertion problem is the cost metric. The traditional buffer insertion problem is to insert buffers at some candidate locations. These candidate locations are not related to the placement and the placement after buffering may overlap. The cost of buffering is known, such as area overhead or buffer delay. The ECO timing optimization problem considers re-wiring the netlist with spare cells, and all spare cells are the candidate buffering/sizing locations for each net/gate. Since every spare cell becomes a normal standard cell if it is rewired to the netlist, the candidate buffering locations of each net vary along the optimization process. Thus the buffering/sizing cost is dynamic, making this problem much harder than the traditional buffer insertion problems. We propose a dynamic programming method considering such dynamic cost, and so we call our algorithm "Dynamic Cost Programming (DCP)".

Our spare cells selection algorithm consists of two phases. The first phase is buffer insertion and gate sizing. We iteratively perform buffer insertion and gate sizing simultaneously to the ECO paths. This loop terminates until all timing violations of ECO paths are fixed or every timing violated ECO path cannot be further optimized. We also propose several heuristics to reduce the solution size during dynamic cost programming.

The second phase is technology remapping. We extract timing critical parts of the ECO paths and remap it using spare cells. From our proposed optimization flow, our method can be smoothly integrated into commercial a design flow. Experimental results show that our algorithm can fix almost all of the timing deficiencies in a short CPU time.

1.4 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 2 gives the preliminaries of this thesis and the formulation of the ECO timing optimization problem. In Chapter 3, we detail our algorithm, including the buffer insertion, gate sizing and technology remapping. Chapter 4 shows the experimental results. Finally, conclusions and the future work are given in Chapter 5.

Chapter 2

Preliminaries

In this chapter, we introduce the dynamic programming framework for buffer insertion [6],and the path-based buffer insertion algorithm [28]. The timing model used in this thesis is also detailed here.

2.1 The Timing Model

We apply the Synopsys .lib timing model to evaluate the circuit timing. Although the wire length is the commonest cost metric during placement, we can use a more accurate timing model because the placement is fixed and all cells' locations are known. The Synopsys .lib timing model combines wire loading and gate input capacitance with gate driving loading. The coordinate of a gate g_i is denoted as $p_i(x_i, y_i)$. The relation between fanout wirelength of a gate g_i and its corresponding capacitance loading value C_{wi} is shown below:

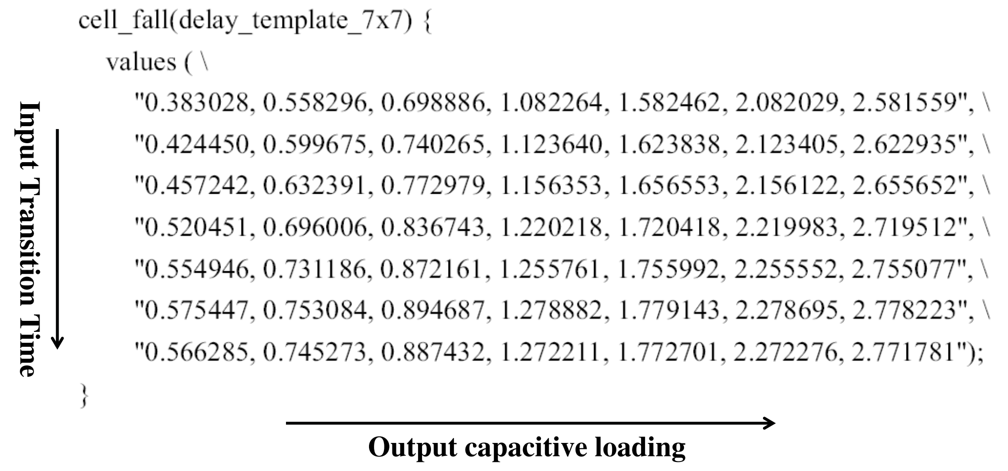
$$C_{wi} = \sum_{g_j \in fanouts\ of\ g_i} (|x_i - x_j| + |y_i - y_j|) \times \phi. \tag{2.1}$$

ϕ is the amount of capacitance from per unit wirelength. Then the capacitance loading of a gate g_i can be defined as:

$$C_i = C_{wi} + (output\ pin\ capacitance\ of\ g_i)$$

$$+ \sum_{g_j \in fanout\ of\ g_i} (input\ pin\ capacitance\ of\ g_j). \tag{2.2}$$

The delay and output transition time of a gate are functions of its input transition time and output driving capacitance, and the functions are characterized by lookup tables. An example of the lookup table for a gate's delay when the output signal is falling is shown in Figure 2.1.



```
cell_fall(delay_template_7x7) {
  values ( \
    "0.383028, 0.558296, 0.698886, 1.082264, 1.582462, 2.082029, 2.581559", \
    "0.424450, 0.599675, 0.740265, 1.123640, 1.623838, 2.123405, 2.622935", \
    "0.457242, 0.632391, 0.772979, 1.156353, 1.656553, 2.156122, 2.655652", \
    "0.520451, 0.696006, 0.836743, 1.220218, 1.720418, 2.219983, 2.719512", \
    "0.554946, 0.731186, 0.872161, 1.255761, 1.755992, 2.255552, 2.755077", \
    "0.575447, 0.753084, 0.894687, 1.278882, 1.779143, 2.278695, 2.778223", \
    "0.566285, 0.745273, 0.887432, 1.272211, 1.772701, 2.272276, 2.771781");
}
```

Figure 2.1: Lookup table example.

It is obvious that if a gate and its fanouts are far away, the gate must have a large delay due to its large capacitance loading. The timing path delay is the sum of delays of all gates on the timing path.

There are several properties of this timing model:

1. Loading domination: For a gate g_i, the effect of the output loading to the gate delay is much larger than that of the input transition time.

2. Shielding: If two gates, g_i and g_j, on the same ECO path are not directly connected, and g_i is former in the path than g_j, then gate sizing to g_j does

not affect the gate delay of g_i. On the other hand, gate sizing to g_i has only a little effect to g_i because of loading domination.

The first property is summarized from the technology data empirically. This is important because during our optimization process we only know one factor among the output loading and the input slope when calculating the gate delay. We assume a value to the input slope while we have an exact output loading value to get an approximated gate delay. The delay calculated in this way is more accurate than the case that output loading is an assumed value and the input slope is known.

The second property means that changing one part of the ECO path by gate sizing and buffering does little effect on timing of the unchanged part. It facilitates us to judge one operation by its local delay effect other than its global delay effect. This will speed up the algorithm greatly. Figure 2.2 shows a conceptional example.

2.2 Dynamic Programming Framework

A signal net consists of one source (g_s) and several sinks (g_t). Given the candidate buffer locations of a net, authors of [6] proposed a dynamic programming method to get an slack optimum buffering solution.

All possible assignment of buffers are called pair options. Every pair option is evaluated by the capacitance loading seen from the upstream and the required arrival time (RAT). A pair option p_i is dominated by another pair option p_j if the required arrival time of p_i is smaller than that of p_j and the capacitance loading of p_i is larger than that of p_j.

The method starts from the sinks and ends at the source. At each candidate location new pair options are added. If two sub-trees merge, pair options of both sub-trees are also merged. Since dominated pair options are canceled during the

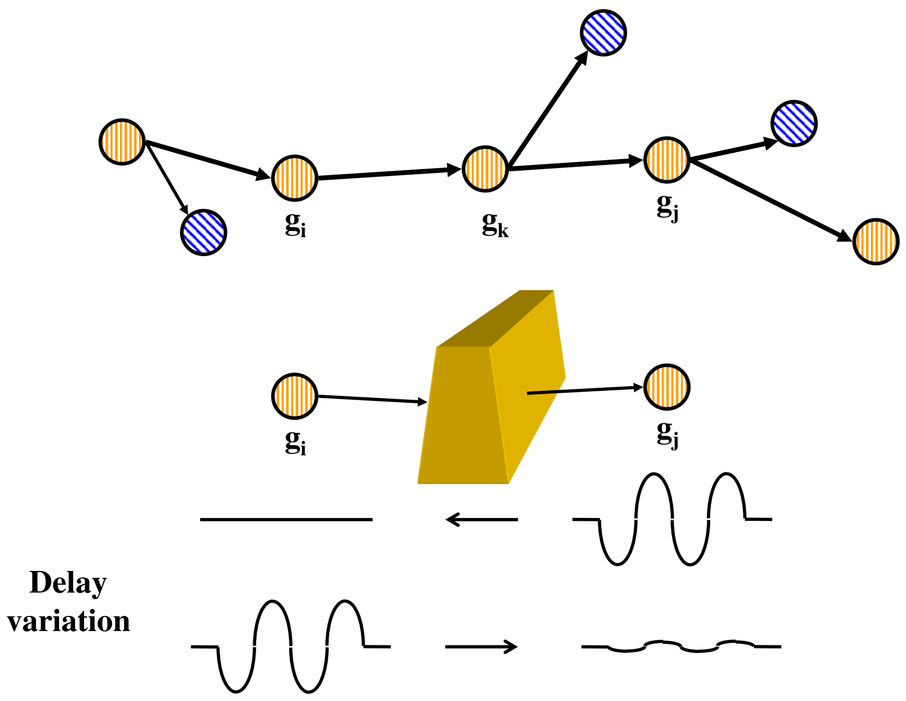


Figure 2.2: The gate g_k serves as a barrier and mitigates the delay variation of g_i and g_j due due to changes of the other gate.

buffering process, the dynamic programming method guarantees that every reserved pair option of a tree is slack optimal under the loading value.

Figure 2.3 is an buffering example. From the sinks to the source we calculate all possible buffering assignment and store non-dominated assignments as pair options at every candidate location and tree branch. The set of pair options at the root of the tree are the slack optimal buffer assignments under their loading value. The whole algorithm is shown in Figure 2.4.

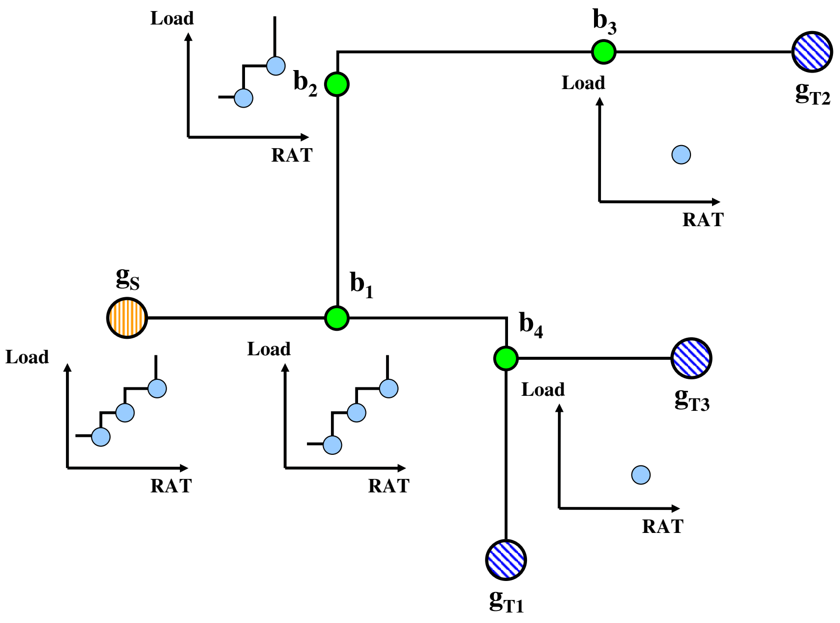


Figure 2.3: Solutions of the dynamic programming framework.

2.3 Path-based Buffer Insertion

While the algorithm in [6] is to insert buffers into one net each time, the path-based buffer insertion method in [28] considers insert buffers to nets of a path concurrently.

This method is done by five steps:

1. Identify k most critical paths of the circuit by STA.

2. Cut those k paths into k' distinct paths.

3. Every path is viewed as a big routing tree, and gates on the path are treated as special type buffering locations that special buffers must be inserted into.

4. The special type buffers have the same input/output characteristics as the corresponding gates.

```
Algorithm: VGDP(k)
1 begin if isleaf(k)
2           then Z=pair options of k
3         else
4           begin
5             Z1=VGDP(left(k));
6             Z2=VGDP(right(k));
7             Z=Z1⋃Z2; (merge pair options of two sub-trees)
8             add wire delays to pair options in Z;
8             update pair options in Z by inserting wires;
9             add buffer option to Z;
10          end
11 end
```

Figure 2.4: Dynamic programming framework.

5. The van Ginneken's algorithm is applied to each routing tree sequentially.

Figure 2.5 illustrates the steps. The two most critical paths are $P_1\{g_1, g_4, g_7, g_{11}, g_{12}, g_{14}\}$ and $P_2\{g_1, g_4, g_7, g_8, g_{10}, g_{12}, g_{14}\}$. We cut the two paths into two distinct paths $P_1'\{g_1, g_4, g_7, g_{11}, g_{12}, g_{14}\}$ and $P_2'\{g_8, g_{10}\}$. They are shown in Figure 2.5 (b). In Figure 2.5 (c), nets of each distinct path are merged into a routing tree and the gates on the routing tree are treated as special type buffering locations which must be inserted buffers later.

During the van Ginneken's algorithm for each routing tree, we force the software to insert buffers at every special buffering location. These buffering operations are equivalent to the gate sizing operations to the gates corresponding to special buffering locations. Then we can apply gate sizing and buffer insertion simultaneously.

Since the timing paths may be cut into routing trees during the step 2, the timing information of every routing tree are not known to each other during

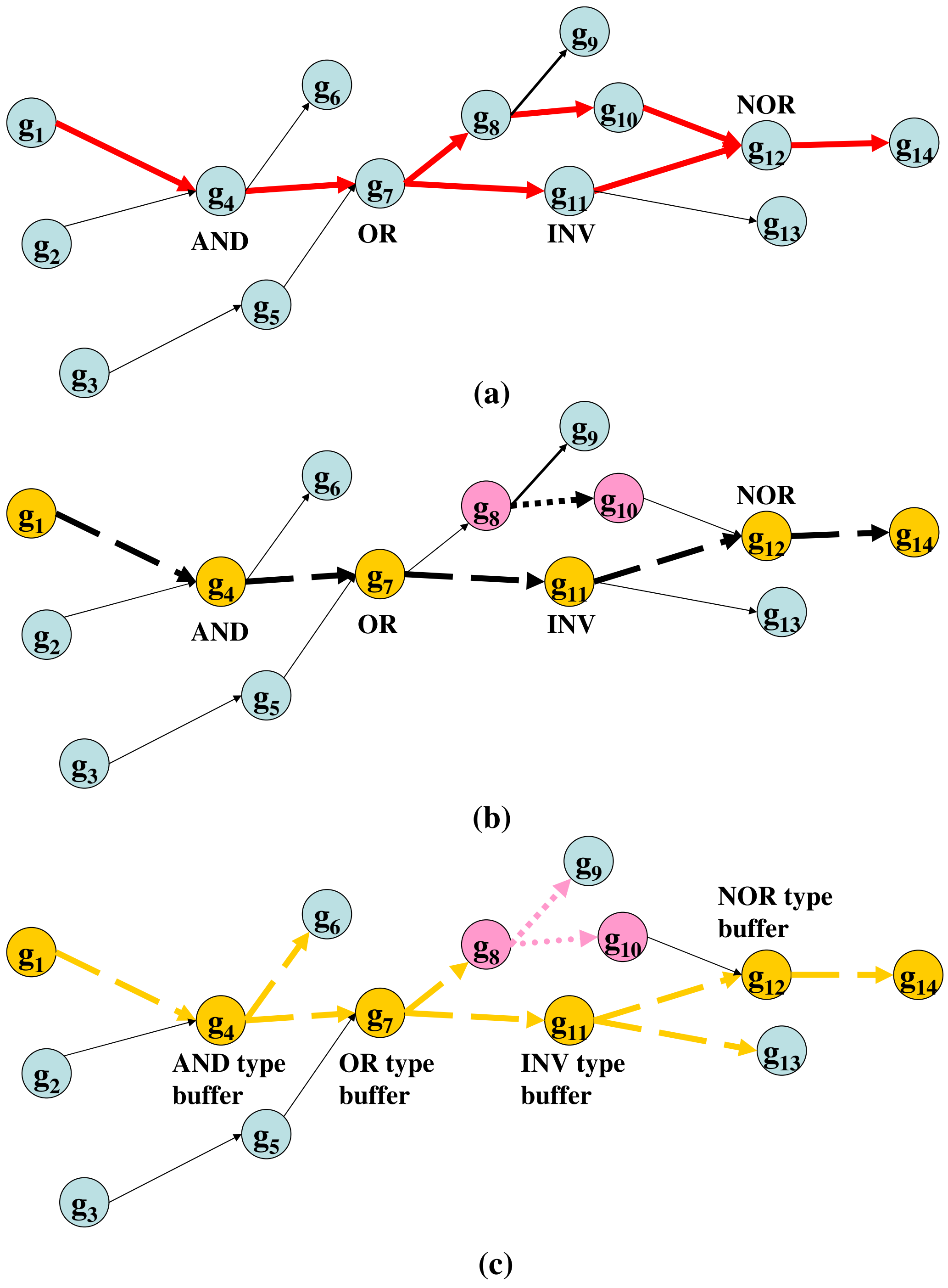


Figure 2.5: (a) A combinational circuit. (b) Two distinct paths. (c) Two merged routing trees. Gates on the path are viewed as special buffer locations.

optimization. In Figure 2.5 (c), if we optimize the path P_1' first, we can find a buffering solution making P_1' meet timing constraints. However, the path P_2 may still violate the timing constraints even if we choose a slack optimum buffering assignment to P_2'. This is because the sub-paths $\{g_1, g_4, g_7\}$ and $\{g_{12}, g_{14}\}$ are not considered with regard to P_2' during optimization to P_1', and the buffering assignments of these sub-paths are not good enough to P_2'.

The authors proposed a method to solve the above problem. Since the slack of the slack optimum buffering to the path must be larger than or equal to zero, we can distribute the positive slack as "useful slack" to each net along the path. It means that we do not need to choose a slack optimum buffer assignment in every sub-path. This idea will control the delay of every sub-path within a reasonable value. It can reduce the number of inserted buffers because the buffering efficiency decreases when we insert more buffers to a net to further reduce the delay. The idea is shown in Figure 2.6.

2.4 Problem Formulation

In this section, we introduce the notations used in this thesis and the problem formulation. A timing path is defined as (1) a path from one primary input to one primary output, (2) a path from one primary input to one D flip-flop input pin, (3) a path from one D flip-flop output pin to one primary output, and (4) a path between one D flip-flop output pin and one D flip-flop input pin. An ECO path is a timing path which violates the timing constraints and we are going to fix it by spare cell rewiring. We denote the start point of a ECO path as the D flip-flop or the primary input at the beginning of the ECO path. We also denote the end point of a ECO path as the D flip-flop or the primary output at the end of the ECO path. Figure 2.7 (a) shows the modeling of the ECO path. Let N be the set of nets of the

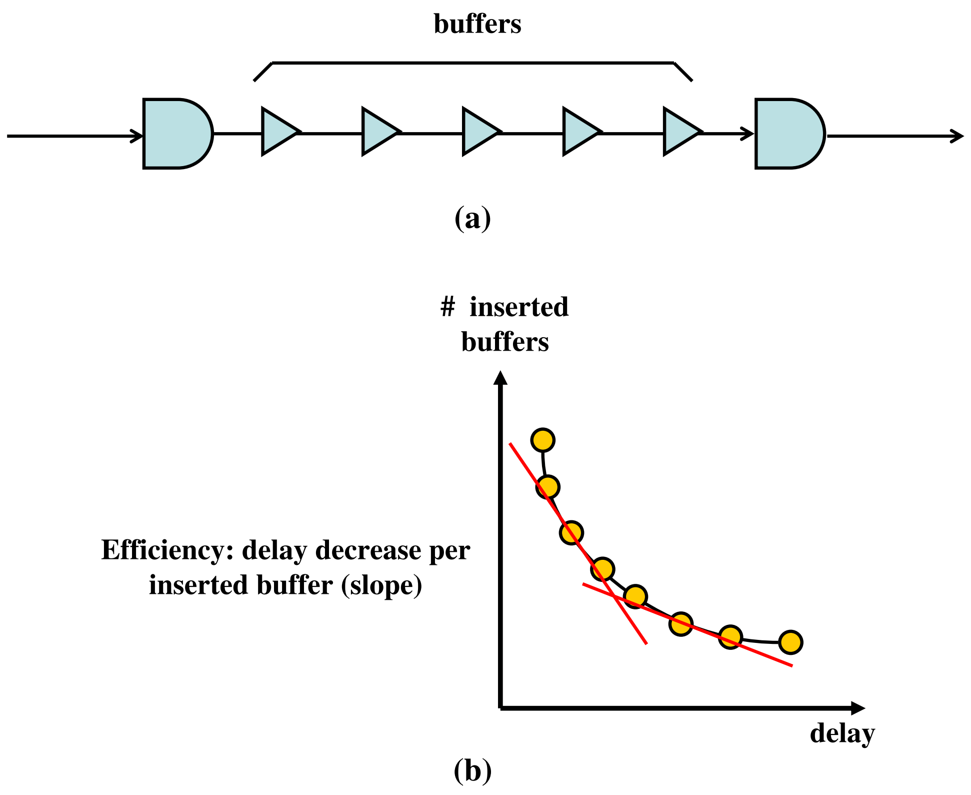


Figure 2.6: (a) Inserting buffers into a net. (b) The efficiency decreases if more buffers are inserted.

netlist and N^E be set of nets of the ECO paths. Let G be the set of all standard cells. We denote G^E as the cells on the ECO paths, and G^S as the spare cells. The coordinate of a gate g_i is denoted as $p_i(x_i, y_i)$. We define the buffering and sizing operations below.

Definition 2.1 *A buffering operation is to insert a buffer type spare cell g_i^S into a net n_j^E in the ECO paths. A gate sizing operation is to exchange a spare cell g_i^S with a gate g_j^E in the ECO paths by rewiring.*

Definition 2.2 *The delay of a gate g_i is $delay(g_i)$ while the delay of g_i after sizing or buffering operations is $delay'(g_i)$.*

During our optimization process, we generate many solutions. A solution can be formally defined as follows:

Definition 2.3 *The target gate of a solution is a gate on the ECO path and this gate is being considered to be sized or buffered.*

Definition 2.4 *The scope of a solution is a sub-path between the ECO path end point and the target gate. The delay of the scope is the sum of delays of the gates in the scope.*

Definition 2.5 *A solution is a set of gate sizing and buffering operations to gates and nets in its scope. The cost of a solution is the sum of delays of gates in the scope if operations of the solution are performed.*

Figure 2.7 (b) shows a solution S_1 corresponding to the ECO path shown in Figure 2.7 (a). Solution S_1 consists of one buffering operation and one sizing operation. The scope of S_1 is the sub-path between g_2^E and g_5^E, and the cost of S_1 is the sum of delays of g_2^E, g_1^E, g_4^E, g_3^E, and g_5^E. Another solution S_2 is shown in Figure 2.7 (c) with the same scope, the cost of S_2 is the sum of delays of gates in the scope after inserting two buffers.

At the end of the optimization process, we get a set of solutions which scope is the whole ECO path and the cost is the ECO path delay. We choose a solution meeting timing constraints that uses minimum number of buffers as our final solution. Operations of the final solution are performed to the ECO path and STA is re-run to update the timing information.

Based on the definition above, the ECO timing optimization problem can be formally defined as the follows:

Problem 2.1 *Given a netlist after ECO process, the ECO timing optimization problem is to re-wire the netlist using spare cells G_S so that the netlist meets the timing constraints, and the functionality of the netlist cannot be unchanged.*

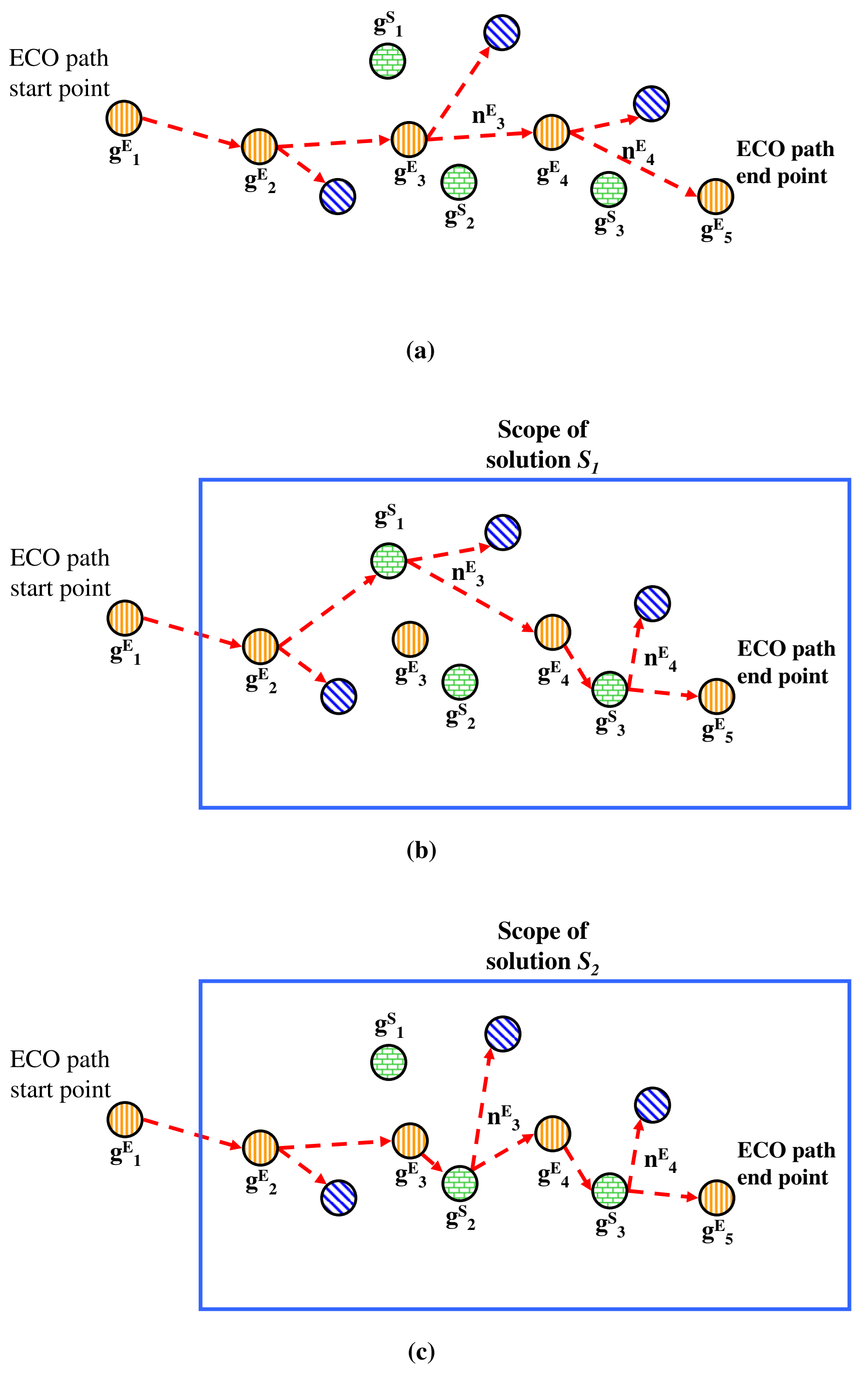


Figure 2.7: (a) The model of ECO paths. (b) Solution S_1. (c) Solution S_2.

Chapter 3

The Spare Cells Selection Algorithm

In this chapter, we present our spare cells selection algorithm for the ECO timing optimization problem. First we give the overview of our algorithm and the optimization flow. Then we detail the methods used in each phase.

3.1 Algorithm Overview

We propose a two-phase flow to solve the ECO timing optimization problem. The input information is a placed netlist with some timing violations. We use static timing analysis (STA) to identify those timing violated paths as ECO paths. Then we enter the first phase.

In the first phase we iteratively choose the ECO path with the largest negative slack and optimize it by gate sizing and buffering operations. Since ECO paths are usually overlapped, we update netlist and timing information of all ECO paths after optimization to one of them. If the processed path is not improved after optimization, we put it into a denied list. It means that this path can not be improved under the current netlist and the condition of spare cells. Then we choose the ECO path whose negative slack is the largest among all ECO paths not in the denied list and optimize it. Whenever timing of an ECO path is improved, we clear the denied list to make paths in the denied list be able to be optimized again. This is because the optimization of one ECO path may change the spare cells resource or modify path structure of some paths in the denied list. This criterion prevents us from continuously trying to optimize an ECO path which can not be further improved under current situation. This loop continues until all ECO paths meet the timing constraints or all remained timing violated ECO paths can not be improved any more.

After the first phase, we fix most timing violations. If there are still violations left, we forward those information to the second phase. In the second phase, we identify timing critical parts of the netlist. Those parts are extracted from the netlist and remapped using spare cells. The remapping process stopped when all timing violations are fixed or no more paths can be optimized. After the second phase, we write the optimized netlist to a DEF file, and the program terminates. The optimization flow is shown in Figure 3.1.

3.2 Gate Sizing and Buffer Insertion

In this section, we present the Dynamic Cost Programming (DCP) algorithm which uses the gate sizing and buffering operations to optimize a path. This algorithm is named DCP because it is based on the dynamic programming framework and considers dynamic cost. Figure 3.2 shows an overview of the DCP algorithm.

3.2.1 Delay Cost Calculation

In our optimization process, we have two timing calculation operations. The first one is the Static Timing Analysis (STA), which is a well known technique. The second one is applied during dynamic cost programming, and is different from STA because it only calculates the timing of a small region. We detail the second timing calculation operation in the following paragraphs.

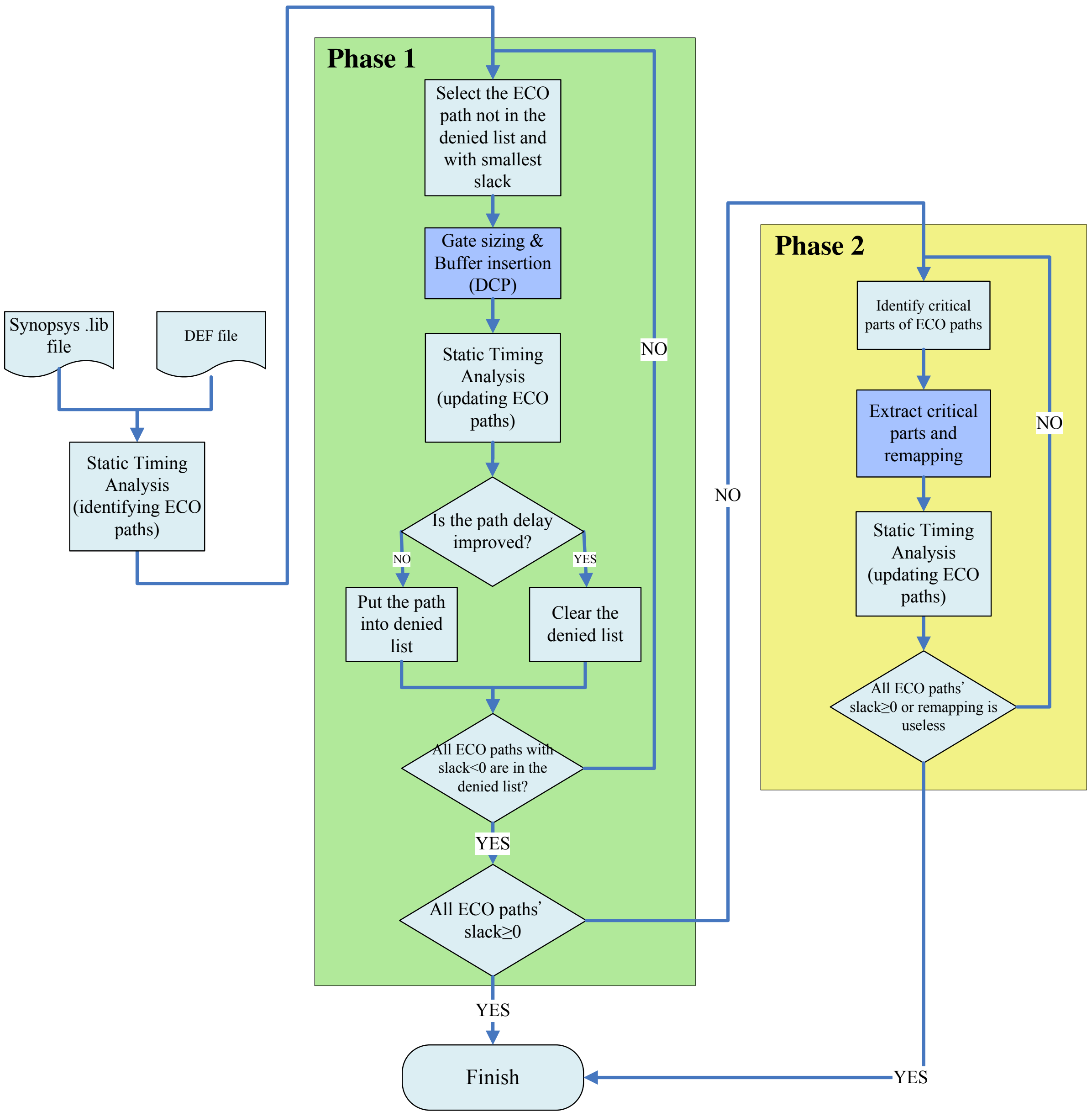

Phase 1
Select the ECO path not in the denied list and with smallest slack
Gate sizing & Buffer insertion (DCP)
Static Timing Analysis (updating ECO paths)
Is the path delay improved?
NO
YES
Put the path into denied list
Clear the denied list
All ECO paths with slack<0 are in the denied list?
NO
YES
All ECO paths' slack≥0
YES
NO
Synopsys .lib file
DEF file
Static Timing Analysis (identifying ECO paths)
Phase 2
Identify critical parts of ECO paths
Extract critical parts and remapping
Static Timing Analysis (updating ECO paths)
All ECO paths' slack≥0 or remapping is useless
NO
YES
Finish

Figure 3.1: The ECO timing optimization flow.

From the Synopsys .lib timing model, we have two observations for our ECO timing optimization problem:

1. The buffering to the net n_i^E changes the delay of the source gate of n_i^E, g_i^E,

```
**Algorithm: Dynamic Cost Programming**($P$, $G^S$,$N^E$,$G^E$)
$P$: the ECO path to be optimized;
$G^S$: set of all spare cells;
$N^E\{n_1^E, ..., n_{M-1}^E\}$: set of all nets on the target ECO path;
$G^E\{g_1^E, ..., g_M^E\}$: set of all cells on the target ECO path;
$M$: size of $G^E$
$S_i^N$: set of solutions stored for buffering net $n_i^E$
$S_i^G$: set of solutions stored for sizing gate $g_i^E$
begin
  Merge $G^E$ and fanouts of $\{g_1^E, ..., g_{M-1}^E\}$ into a routing tree
  for $i = M - 1 \rightarrow 2$
    for all buffer type spare cells $\{g_i^S\}$ in the bounding box of $n_i^E$
      apply buffer insertion to $n_i^E$ using $g_i^S$ based on $S_{i+1}^G$
      store the assignment in $S_i^N$ if delay'($g_i^E$)+delay'($g_i^S$)< delay($g_i^E$)
    prune the solutions in $S_i^N$
    for all spare cells $\{g_i^S\}$ with type same as $g_i^E$ in the bounding polygon of $g_i^E$
      apply gate sizing to $g_i^E$ using $g_i^S$ based on $S_i^N$
      store the assignment in $S_i^G$ if delay'($g_{i-1}^E$)<delay($g_{i-1}^E$) & delay'($g_{i-1}^E$)+delay'($g_i^S$)<delay($g_{i-1}^E$)+delay($g_i^E$)
    prune the solutions in $S_i^G$
  for all buffer type spare cells $\{g_i^S\}$ in the bounding box of $n_1^E$
    apply buffer insertion to $n_1^E$ using $g_i^S$ based on $S_2^G$
    store the assignment in $S_1^N$ if delay'($g_1^E$)+delay'($g_i^S$)<delay($g_1^E$)
  choose the solution in $S_1^N$ that meets the timing constraints and uses fewest buffers, and rewire the netlist according to the solution
end
```

Figure 3.2: Overview of the DCP algorithm.

while other gates are little affected or not affected. Thus the effect of buffering to the timing is the delay change of g_i^E and the delay increase of the inserted buffer.

2. The sizing to the gate g_i^E changes the delay of the fanin gates of g_i^E, $\{g_j^E\}$, while other gates are little affected or not affected. Thus the effect of sizing

g_i^E to the timing is the delay change of $\{g_j^E\}$ and the sized gate.

Based on the description above, we can evaluate the effect of buffering and sizing by calculating delay of the changed part of the path without applying STA to the whole path.

It is important that the delay value calculated in the second operation is an approximated value. As described in Chapter 2.1, the gate delay is a function of the input transition time and the output loading. Since the dynamic programming method optimizes the path along one direction, we can not know both factors of a gate at the same time. Thus we apply dynamic programming method from the end point of the ECO path to the start point of the ECO path. During sizing and buffering we calculate the gate delay as cost with a known output loading value and assume the input slope to be the calculated input slope of the gate which was calculated by STA before the optimization to this path,

If we apply dynamic programming from the start point to the end point of the ECO path, the gate delay is calculated with a known input slope value while the output loading is an assumed value. Gate delays calculated in this way have larger error because estimation error of the output loading causes larger delay variation than that of the input slope due to loading dominatio0n property.

3.2.2 Sizing and Buffering Operations

Here we describe the overall process of the DCP algorithm.

Given an ECO path to be optimized, we merge the nets of the ECO path into a big routing tree. Fanouts of gates along the path are also merged into the tree because they affect the loading of the ECO path. M is the number of gates along the ECO path. From start point to end point, gates on the ECO path are numbered as $g_i^E, i = 1 \sim M$, and nets of the path are numbered as $n_i^E, i = 1 \sim M - 1$.

We start applying buffering to the net n_{M-1}^E. We try to insert one buffer in the neighborhood into n_{M-1}^E and calculate the approximated delay of the driving gate g_{M-1}^E of the net n_{M-1}^E. Each possible buffering assignment is a solution whose scope is the sub-path consisting of g_{M-1}^E and g_M^E (the end point), and we store it if sum of delay of the gates in its scope is smaller than the case without buffering. At this time we only estimate the effect of buffering n_{M-1}^E without actually inserting a buffer to the net. Figure 3.3 shows the result of buffering n_{M-1}^E, $(M = 5)$.

Since we only store the buffering solutions that reduces the delay of g_{M-1}^E, timing of paths that are overlapped with the ECO path by g_{M-1}^E will not be worse than the case without buffering. This property is very important because we must guarantee no timing change to unprocessed paths (ex: $\{g_1^E, g_2^E, g_3^E, g_4^E, g_8\}$ in Figure 3.3) when optimizing ECO paths.

Additionally, since spare cells assignment at early stage will affect the assignment at latter stage, the cost of buffering and sizing is dynamic. Thus we must store a set of solutions instead of only the best one at every operation.

After buffering to n_{M-1}^E, we apply gate sizing to g_{M-1}^E using nearby spare cells with the same type as g_{M-1}^E. Timing and loading information of every solution stored in n_{M-1}^E is considered to generate new solutions for g_{M-1}^E. For one sizing solution, if (1) sum of new delays of g_{M-2}^E and the sizing spare cell is smaller than the delay sum of g_{M-2}^E and g_{M-2}^E , and (2) delay of g_{M-2}^E after sizing is smaller than the delay before sizing, we store this solution for g_{M-1}^E.

With the second criterion like buffering n_{M-1}^E, timing of paths overlapped with the ECO path by g_{M-2}^E is no worse than the case with no sizing. Figure 3.4 shows the solutions of sizing g_{M-1}^E, $(M = 5)$.

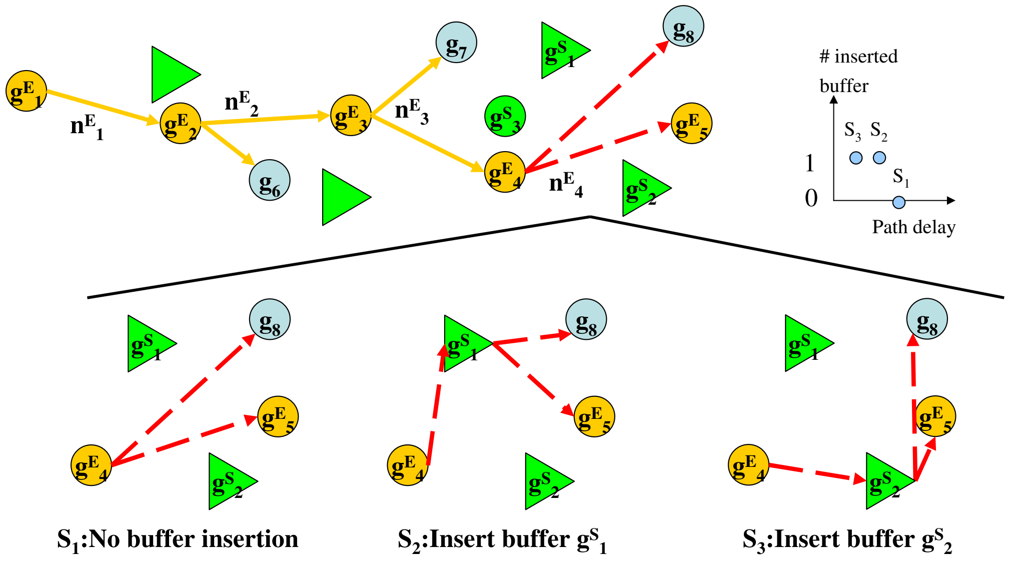
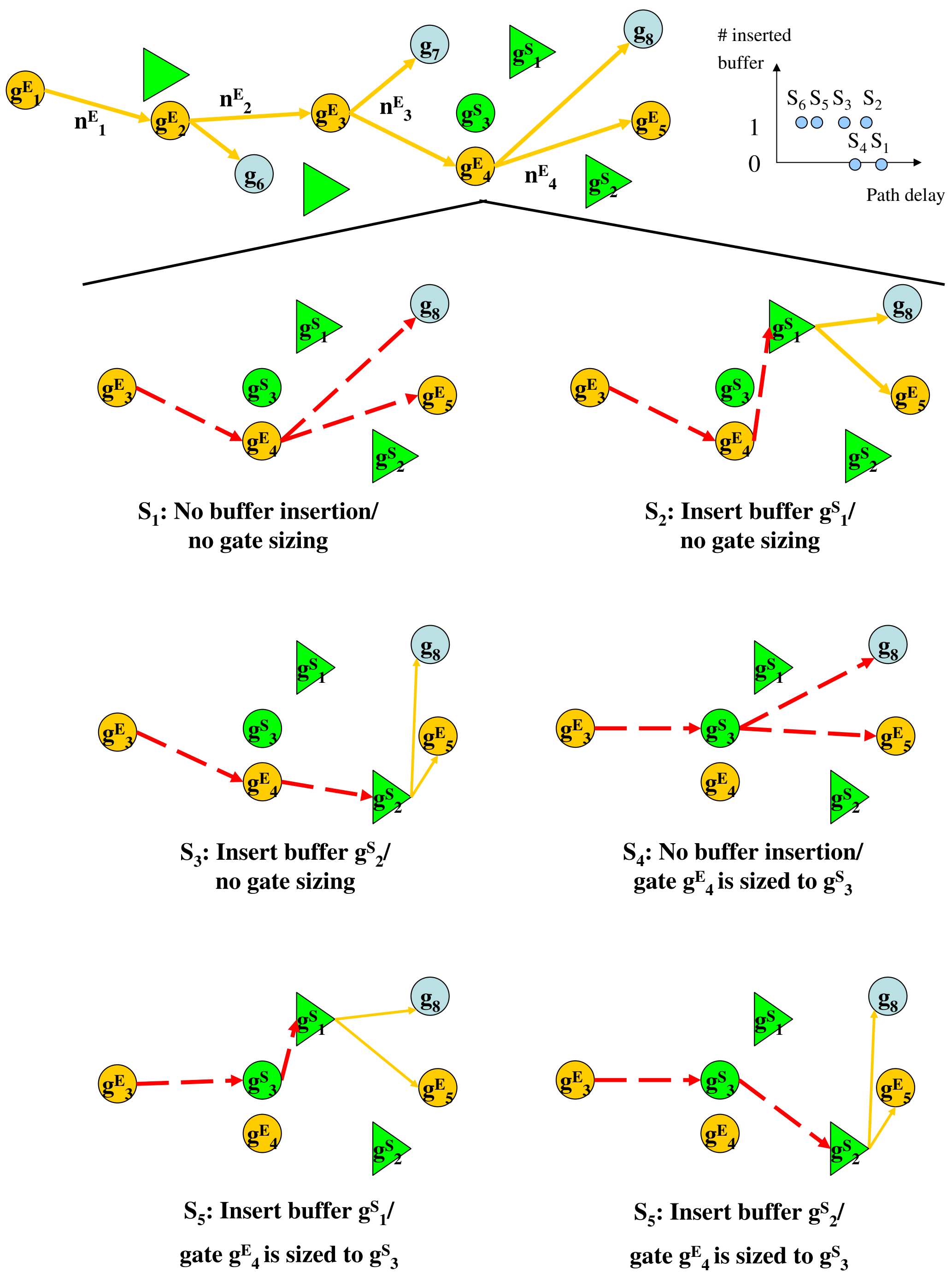


Figure 3.3: The solutions for buffering n_4^E.

Then we apply buffer insertion to n_{M-2}^E. After buffering we size g_{M-2}^E. We recursively apply buffer insertion to n_{M-i}^E, $i = 1 \sim M-1$ and gate sizing to g_{M-i}^E, $i = 1 \sim M-2$ one after the other until the start point is reached. During buffering n_i^E, we consider sizing solutions of the driven gate, g_{i+1}^E, to generate new solutions. We also calculate solutions of sizing g_i^E based on buffering solutions of the driven net n_i^E.

The DCP algorithm starts from the buffering to n_{M-1}^E and stops at the buffering to n_1^E. Sizing operations to g_1^E and g_M^E are not considered because they will influence the timing of non-ECO paths. Among the buffering solutions stored in n_1^E, we choose the one that makes the ECO path meet the timing constraints and use fewest buffers as the final solution of the ECO path. We rewire the netlist according to the operations of the final solution, and STA is run to update the circuit timing.



gE1
nE1
gE2
nE2
gE3
nE3
g7
g6
gS3
gS1
gE4
nE4
gS2
g8
gE5
# inserted
buffer
S6 S5 S3 S2
1
0
S4 S1
Path delay
S1: No buffer insertion/ no gate sizing
S2: Insert buffer gS1/ no gate sizing
S3: Insert buffer gS2/ no gate sizing
S4: No buffer insertion/ gate gE4 is sized to gS3
S5: Insert buffer gS1/ gate gE4 is sized to gS3
S5: Insert buffer gS2/ gate gE4 is sized to gS3

Figure 3.4: The solutions for sizing g_4^E.

3.2.3 Bounding Box for Choosing Spare Cells

When buffering to a net and sizing to a gate, we need to use spare cells as resource. Since the amount of spare cells is large, exhaustive search for every possible assignment is not efficient. We propose a heuristic to greatly reduce the number of assignment during gate sizing and buffer insertion.

For the case of buffer insertion, we consider inserting a buffer into the net n_i^E along the ECO path with driving gate g_i^E and driven gates $\{g_j^E\}$. We generate a square bounding box for selection. The box is centered at g_i^E and the width and height of the box is defined as below:

$$width = \sum_{g_j^E \in fanouts\ of\ g_i^E} distance(g_i^E, g_j^E) \tag{3.1}$$

We choose buffer type spare cells in the bounding box as candidate buffer locations for the net n_i^E. Buffer type spare cells outside the box is not considered because they can not improve the delay of the net. The reason is described below.

If we assume that pin capacitance is much smaller than wire loading capacitance, it is negligible when calculating the output loading. We also neglect the effect of input transition time to gate delay due to the loading domination property. If a buffer type spare cell g_i^S is outside the bounding box and inserted into n_i^E, g_i^E must drive a larger load than the case with no buffering. Thus gate delay of g_i^E increases because of larger loading. Since gate delay of g_i^S is larger than zero, the sub-path delay (sum of gate delays of g_i^E and g_i^S) is larger than the original case that no buffer is inserted. This means the buffer insertion does not help the path delay. Based on the discussion, we have the following lemma.

Lemma 3.1 *Given a net n_i^E with source g_i^E and sinks $\{g_j^E\}$ to be buffered, we can consider only buffer type spare cells located in the bounding box. The bounding box is square and centered at g_i^E with width $= \sum_{g_j^E \in fanouts\ of\ g_i^E} distance(g_i^E, g_j^E)$.*

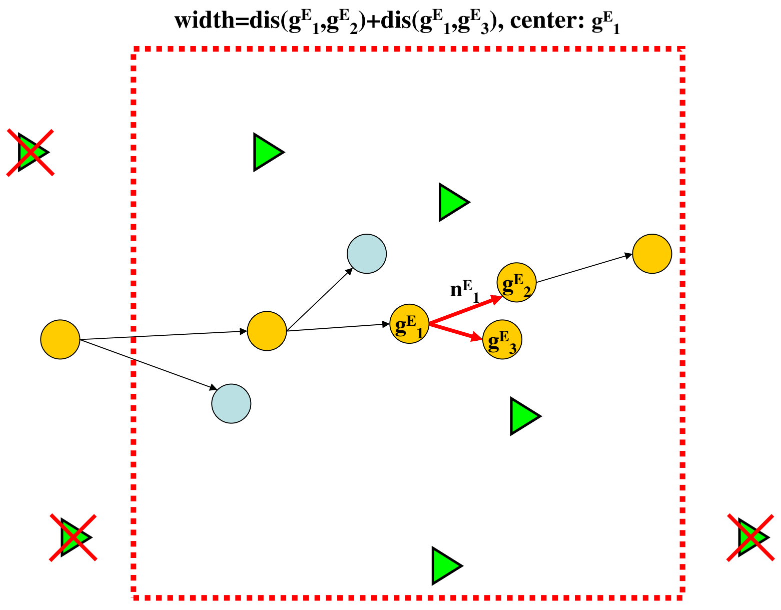
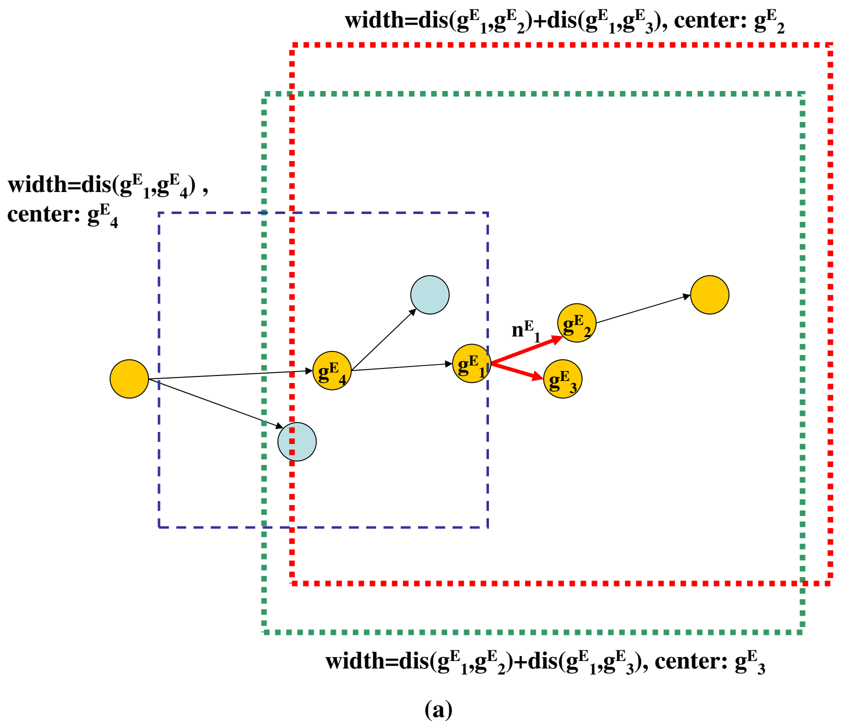

width=dis(g^E_1,g^E_2)+dis(g^E_1,g^E_3), center: g^E_1
n^E_1
g^E_1
g^E_2
g^E_3

Figure 3.5: The bounding boxes for N_1^E to reduce buffer assignment.

For the case of gate sizing, we have a similar conclusion for reducing spare cell assignment. Figure 3.6 shows an example of bounding polygon.

Lemma 3.2 *Given a gate g_i^E with fanins $\{g_j^E\}$ and fanouts $\{g_k^E\}$ to be sized, we can consider only spare cells with the same type as g_i^E and located in the bounding polygon. The bounding polygon is the union of a set of square bounding boxes of $\{g_j^E\}$ and $\{g_k^E\}$. A bounding boxes of g_j^E is centered at g_j^E with width $= distance(g_i^E, g_j^E)$. A bounding box of g_k^E is centered at g_k^E with width $= \sum_{g_k^E \in fanouts\ of\ g_i^E} distance(g_i^E, g_k^E)$.*



width=dis(gE1,gE2)+dis(gE1,gE3), center: gE2
width=dis(gE1,gE4) , center: gE4
nE1
gE1
gE2
gE3
gE4
width=dis(gE1,gE2)+dis(gE1,gE3), center: gE3
(a)

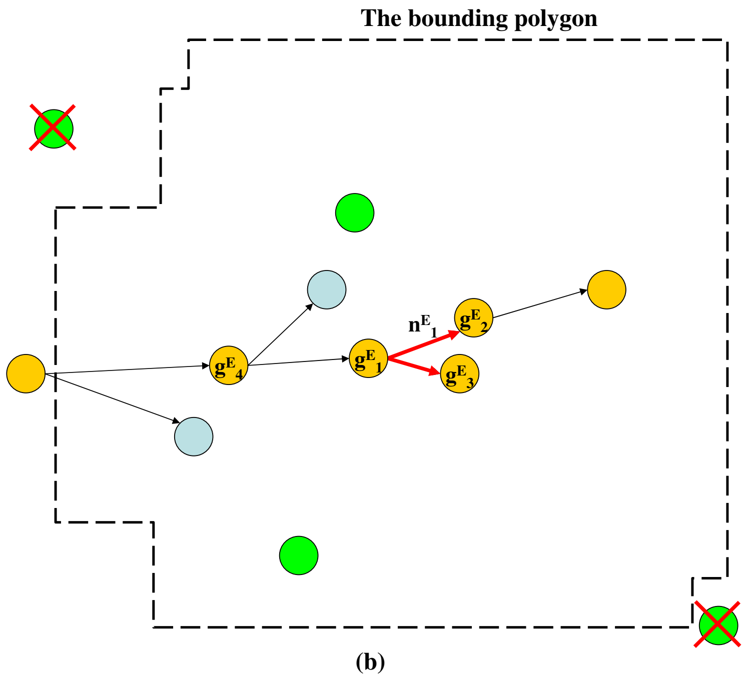


Figure 3.6: (a) Bounding boxes of fanins and fanouts. (b) The union of bounding boxes. Spare cells outside the polygon are unconsidered to size g_1^E.

3.2.4 Solution Control

Although we carefully delete many redundant spare cells assignments during DCP by bounding box and bounding polygon method, number of feasible spare cells assignment is still too large. To speed up the algorithm, we propose another heuristic to control the number of solutions during DCP.

When applying sizing/buffering operation, we generate a set of solutions based on solutions of previous net/gate. The generated solutions can be pruned by the two criteria:

1. The number of used buffers.
2. The delay of the scope.

Number of sized gates is not counted in the first criterion because gate sizing operation changes a cell in G^E with a spare cell in g^S and does not reduce the number of available spare cells. Thus we prefer to use gate sizing operations to fix timing rather than buffer insertion operations. If a solution S_i uses more buffers than another solution S_j but delay of S_i is larger than S_j, we can delete S_i because it is dominated by S_j.

After deleting dominated solutions, we can further prune the solutions. Solutions are grouped into classes according to the number of used buffers, and we keep at most k solutions for each class. k is a user-defined parameter and can be modified to trade solution quality with runtime.

It is important that both heuristics can not guarantee that the optimum solution will never be deleted, but in general we can delete a lot of sub-optimal solutions and keep the optimum one. Figure 3.7 illustrates the pruning idea.

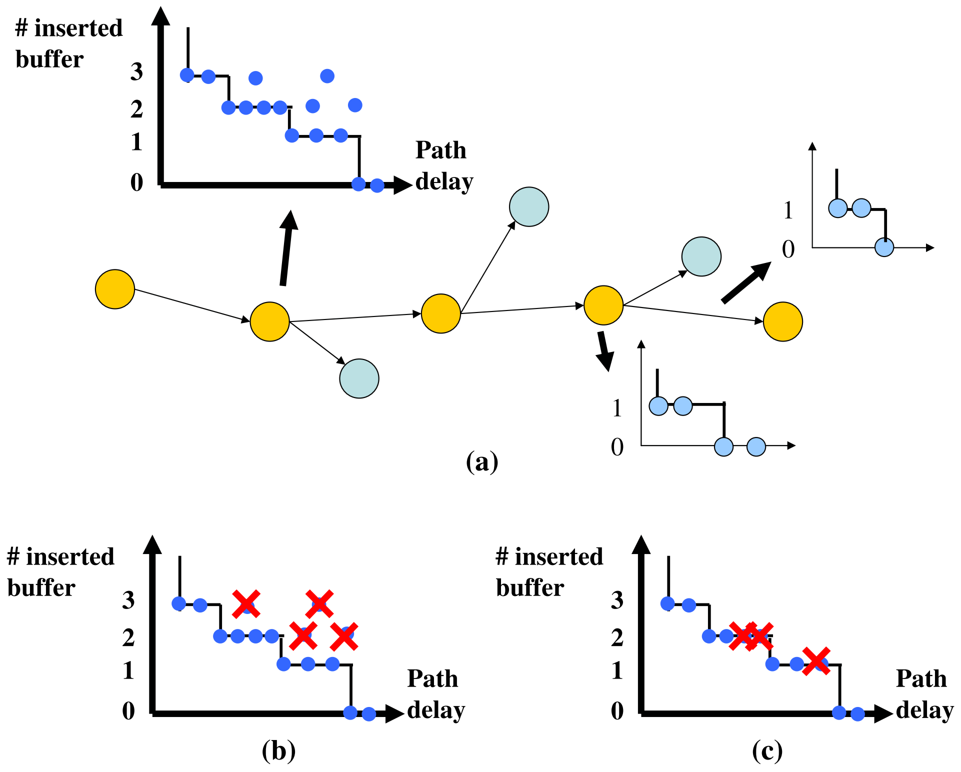


Figure 3.7: (a) A set of solutions. (b) Delete the dominated solutions. (c) Keep at most k solutions for every solution class. (k=2)

3.3 Technology Remapping

After DCP, we fix timing violations by technology remapping method. This method is to deal with the case that cannot be solved by gate sizing and buffer insertion. For the example shown in Figure 3.8, there is an AND gate g_1^E driving a large loading but there is no BUFFER and AND type spare cells near g_1^E. We can use a NAND type spare cell g_1^S and an INVERTER type spare cell g_2^S to replace g_1^E and separate the loading.

The placement driven technology mapping methods in Chapter 1.2.2 first place the base gates. Then they map the base gates according to the coordinates

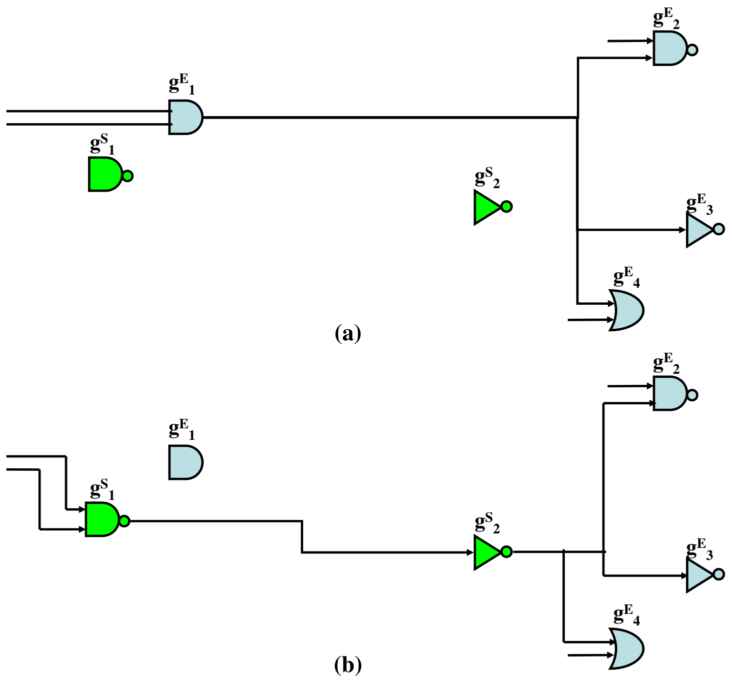


Figure 3.8: (a) An AND gate driving a large loading. (b) Map the AND gate to a NAND gate and an INVERTER gate.

of the initial placement. Similar to the methods above, we first calculate ideal coordinates of the base gates, and map them using this information. The remapping method has following four steps:

1. Identify critical parts of the netlist and extract them from the netlist. We denote the extracted gates as G^M.
2. Decompose gates in G^M into base gates G^B (NANDs and INVERTERs).
3. Calculate ideal locations of base gates of G^B.
4. Map G^B. The mapping cost is related to their ideal locations.

We will detail the methods of calculating ideal locations and technology mapping in the following subsections.

3.3.1 Ideal locations

We know from [1] that optimal buffering to a line is to insert buffers with equal distance, and the distance is $\sqrt{\frac{2R_bC_b}{RC}}$. If we want to map a path that locations of the input pins and output pins are known and fixed, it is intuitive that we map the gates along the path in a way that they are evenly located between input pins and output pins. Since the wire delay is proportional to square of the wirelength, distributing wire loading evenly between the gates reduces the total delay along the path. Furthermore, the buffering after mapping is easier because no gate drives a large loading. Figure 3.9 shows this concept.

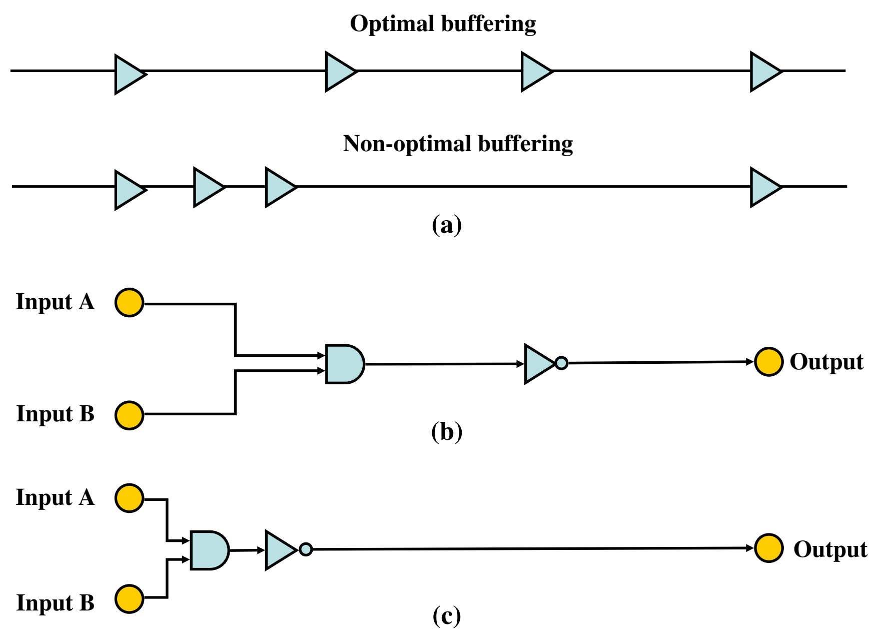


Figure 3.9: (a) Optimal buffering and Non-optimal buffering. (b) Distribute the gates between input pins and output pins evenly. (c) Gates are not placed evenly. The inverter needs to drive a large loading.

Given a part of netlist to be mapped and locations of input pins and output pins, we calculate the ideal mapping positions of base gates by:

1. For every paths from one input pin to one output pin, calculate the candidate locations of base gates along the path as equal distance between the input pin and the output pin.

2. If a base gate has more than one candidate locations, average these values to get the final ideal location of the base gate.

An example of calculating ideal locations is shown in Figure 3.10.

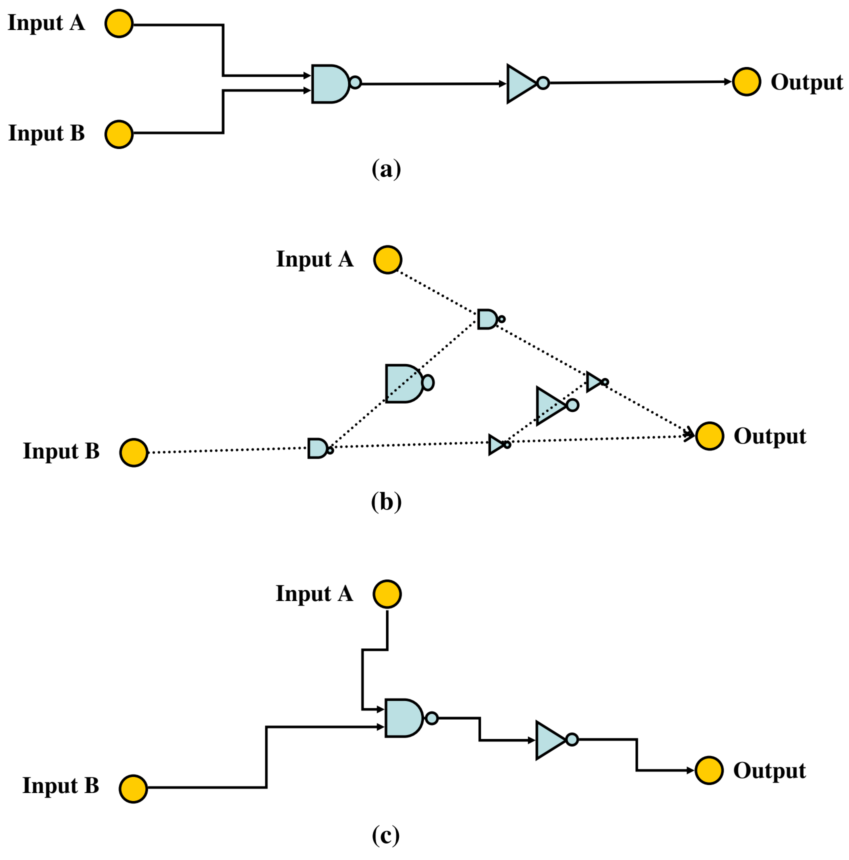


Figure 3.10: (a) Subject graph of the netlist without location information. (b) When locations of inputs and outputs are known, calculate the ideal locations of the base gates. (c) The resulted placement if the base gates are placed at their ideal locations.

3.3.2 Mapping

Our mapping algorithm uses dynamic programming method [9]. After decomposing the extracted netlist and calculating ideal locations of the base gates, we cut the network into a forest. Then we map each tree by the following cost function:

$$Cost(g_i) = \sum_{g_j \in fanins\ of\ g_i} (Cost(g_j) - distance(g_i, g_j)) + distance(g_i, fanout\ of\ g_i) \quad (3.2)$$

The locations of already mapped gates are real locations of spare cells, while locations of the unmapped base gates are their ideal locations.

After remapping a part of the netlist, we apply STA to update the circuit timing. We apply the remapping process until no timing violations left.

3.4 Time Complexity Analysis

In this section we analyze the timing complexity of phase 1 of our spare cells selection algorithm.

There are P ECO paths of the netlist. Total gate count is V, and the number of spare cells is N. If we can finish phase 1 in L iterations, and we keep at most k solutions during each sizing and buffering operation, then the timing complexity of each sizing and buffering operation is $O(kN)$. If an ECO path has at most M gates, the complexity of the DCP algorithm is $O(kMN)$. We apply DCP and STA once in an iteration, and complexity of STA is $O(V)$. Hence the timing complexity of phase 1 is $O((kGN + V)L)$.

3.5 Summary

We propose an algorithm which consists of gate sizing, buffer insertion, and technology remapping to optimize the circuit timing. The whole spare cells selection algorithm is illustrated in Figure 3.11.

```
Algorithm: Spare Cells Selection($G$, $G^S$, $N$)
$N$: set of all nets of the netlist;
$G$: set of all cells;
$P$: set of all ECO paths;
$L$: denied list;
$G^S$: set of all spare cells;
$N^E$: set of all nets on the ECO paths;
$G^E$: set of all cells on the ECO paths;
1 begin
2   apply STA to identify all ECO paths $P$ and $G^E$, $N^E$
3   While $P$ is not empty (phase 1)
4       $p_i$= the most critical path in $P$;
5       apply DCP to $p_i$;
6       apply STA to update all ECO paths $P$ and circuit timing;
7       if delay of $p_i \leq$ clock cycle
8           delete $p_i$ from $P$;
9       if delay of $p_i$ is improved
10          put all paths in $L$ to $P$ and clear $L$;
11      else
12          delete $p_i$ from $P$ and put it into $L$;
13  identify critical parts of the paths in $L$ and remap them (phase 2);
14  apply STA to update the circuit timing;
15 end
```

Figure 3.11: Overview of the spare cells selection algorithm.

Chapter 4

Experiment Results

We implemented our algorithm in the C++ programming language on a 3.2GHz Linux workstation with 3 GB memory. The benchmark circuits Case 1, Case 2, Case 3, Case 4, and Case 5, are real industry designs.

In Table 4.1, "Case name" denotes the names of circuits, "Gate count" denotes the number of gates of the circuit, "# Spare cells" denotes number of spare cells in the circuit, "# ECO path" denotes the number of timing violated paths in the unoptimized circuit, "TNS" denotes the total negative slack, and "Max # gates" denotes the largest number of gates along the path among all ECO paths.

The experimental results are shown in Table 4.2. We report the number of ECO paths left, the total negative slack after optimization, the CPU time, and the memory usage. The timing of the circuit is checked by PrimeTime.

We plot the chip layout through Figure 4.1 to Figure 4.5 for better visualization. Figure 4.2 (a) shows all timing violated paths (ECO paths) in the Case 2, while spare cells are plotted as points. Several gates on the ECO paths are misplaced and cause long wires. This is identical to our analysis that a large wire loading results in a large gate delay. The ECO paths after optimization by our algorithm are shown in Figure 4.2 (b). Those paths are more compact because misplaced gates on the paths are rewired by spare cells. Additionally, since we prefer gate sizing operations to buffer insertion operations when optimizing timing, number of gate sizing operations are larger than that of buffer insertion operations.

In Chapter 3.4, our timing complexity is $O((kMN+V)L)$. The term kMN is dominated by V if the number of spare cells and the number of gates along ECO paths are much smaller compared to the gate count. Since STA is applied to the whole netlist in every ireration, runtime is proportional to the gate count. Figure 4.6 shows the values of Case 1, Case 2, Case 3 and it confirms our analysis. The Case 4 is special that the ECO paths are seriously misplaced, and we need a longer time to fix them. Runtime of the Case 5 is much longer than other cases because the number of spare cells is much larger than other cases and even makes the term kMN dominate the term V.

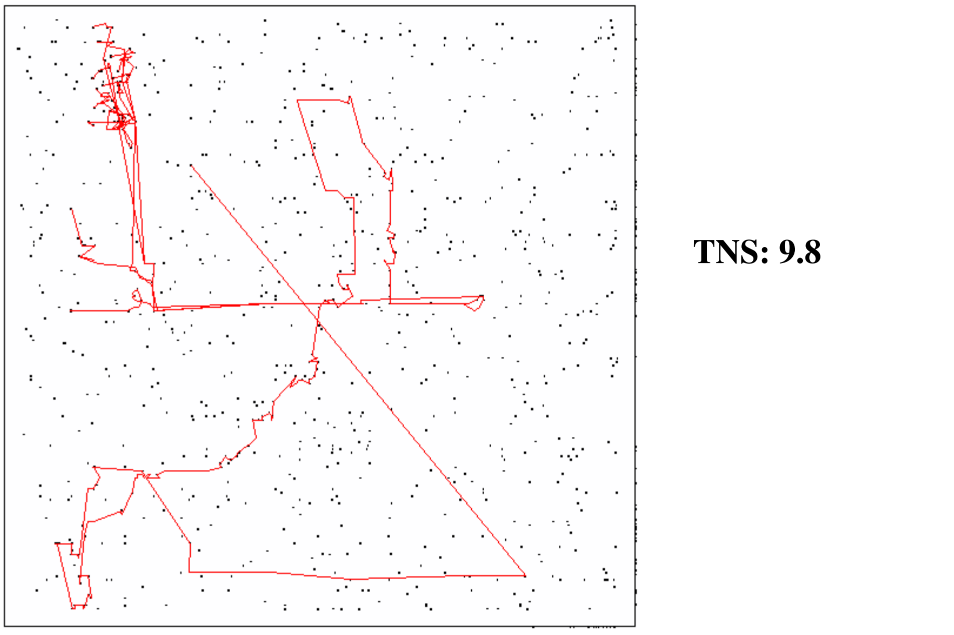
The experimental results demonstrate the effectiveness of our spare cells selection algorithm for the ECO timing optimization problem.

Case name	**Gate count**	**# Spare cells**	**# ECO path**	**TNS (ns)**	**Max # gates**
Case 1	28927	860	16	9.8	164
Case 2	200504	860	80	312	178
Case 3	91107	860	27	319	173
Case 4	18932	860	22	70	85
Case 5	38011	8600	137	161	72

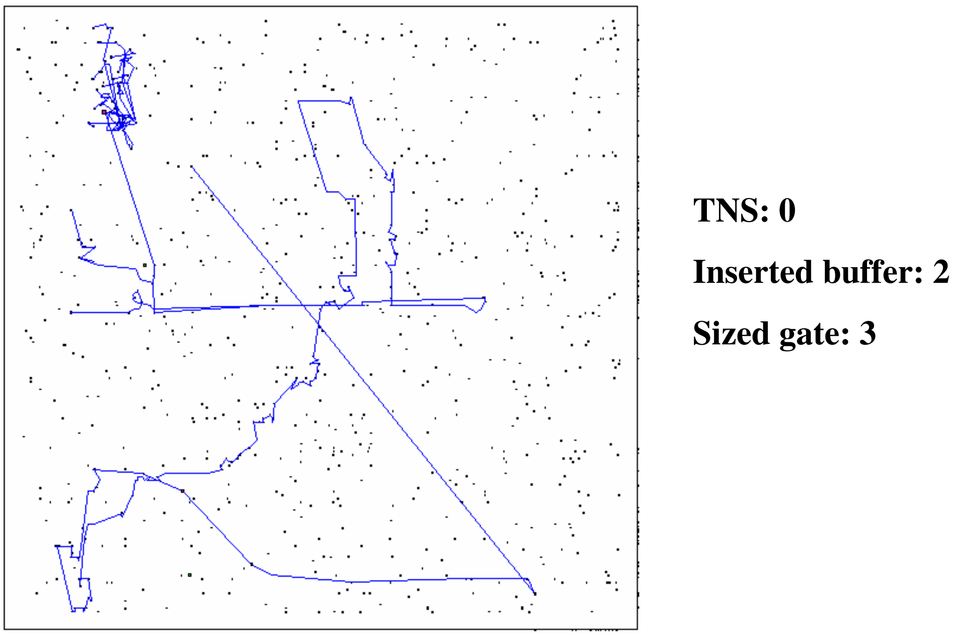
Table 4.1: Statistics of the test cases.

	Original			**DCP**				
Case name	**Gate count**	**# ECO path**	**TNS (ns)**	**# left ECO path**	**TNS (ns)**	**Run time (s)**	**Memory (MB)**	**Comp. rate**
Case 1	28927	16	9.8	0	0	7.7	36	100%
Case 2	200504	80	312	0	0	37	177	100%
Case 3	91107	27	319	0	0	17.4	78	100%
Case 4	18932	22	69.5	3	0.57	30	288	99.20%
Case 5	38011	137	161	0	0	2125.5	84	100%
avg								**99.84%**

Table 4.2: Results of ECO timing optimization.

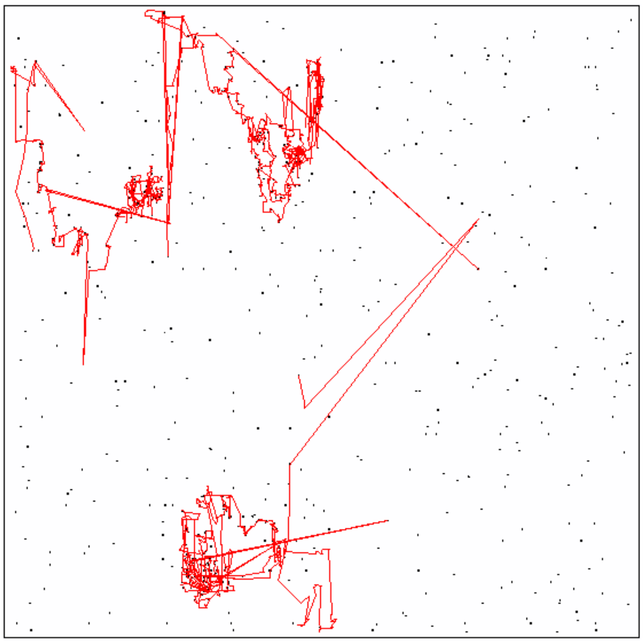


(a)

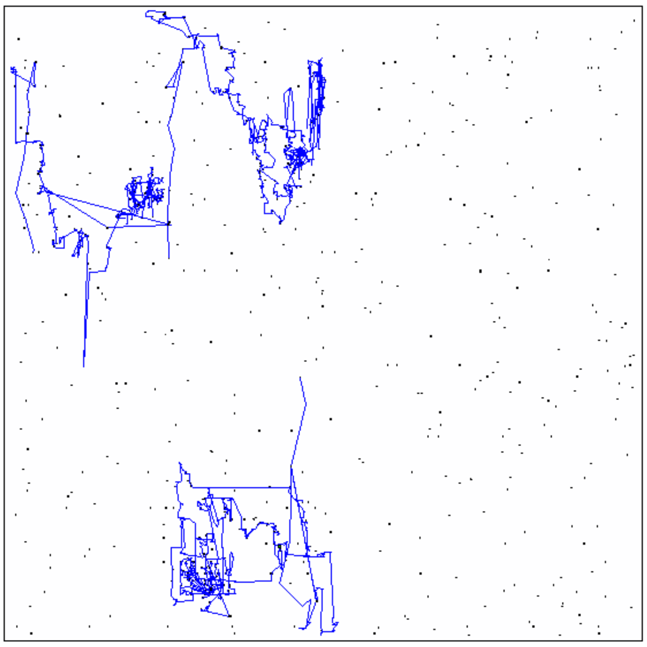


(b)

Figure 4.1: (a) ECO paths of Case 1 before optimization. (b) ECO paths of Case 1 after optimization.

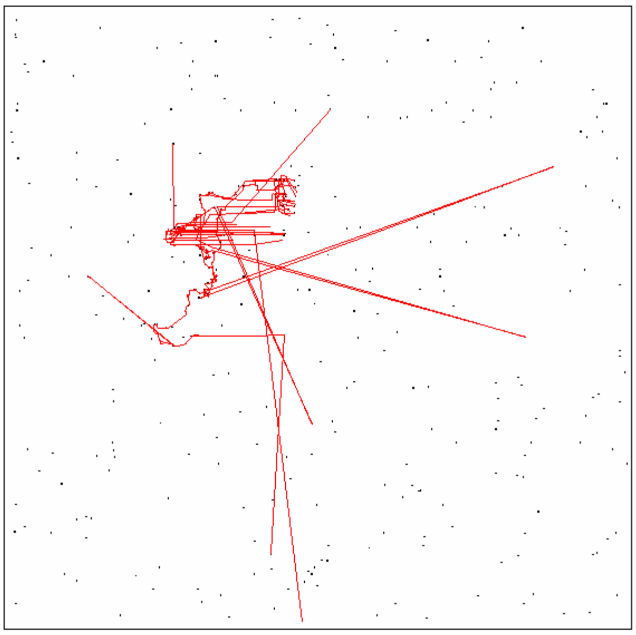


(a)

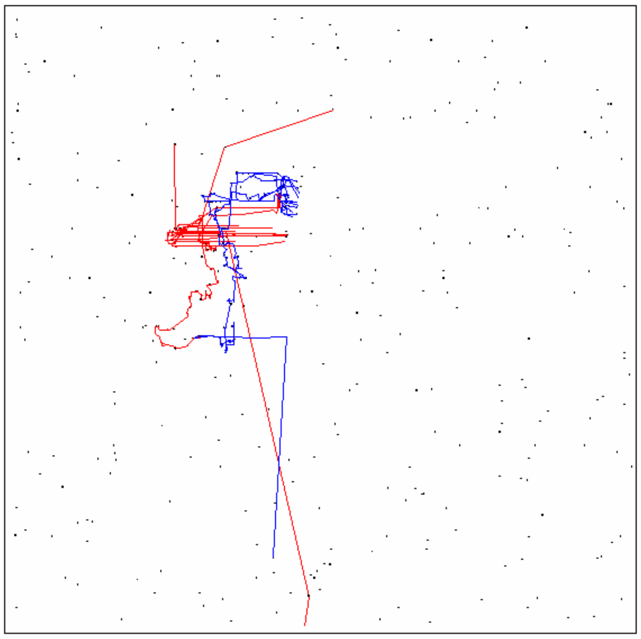


(b)

Figure 4.2: (a) ECO paths of Case 2 before optimization. (b) ECO paths of Case 2 after optimization.

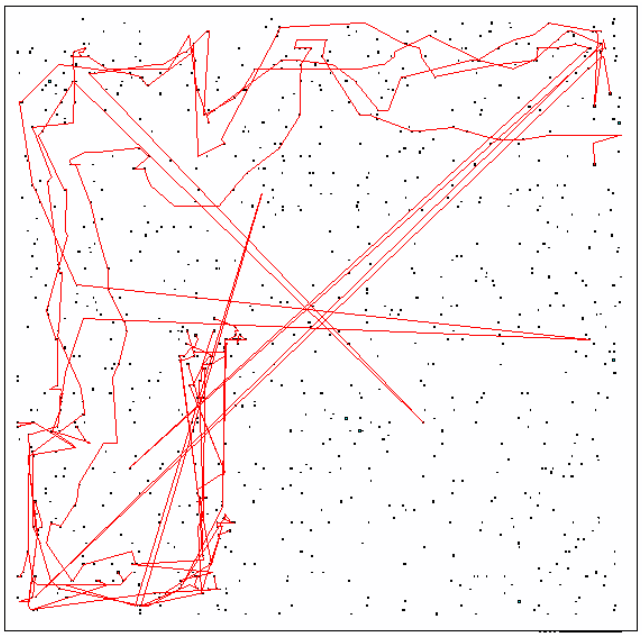


(a)



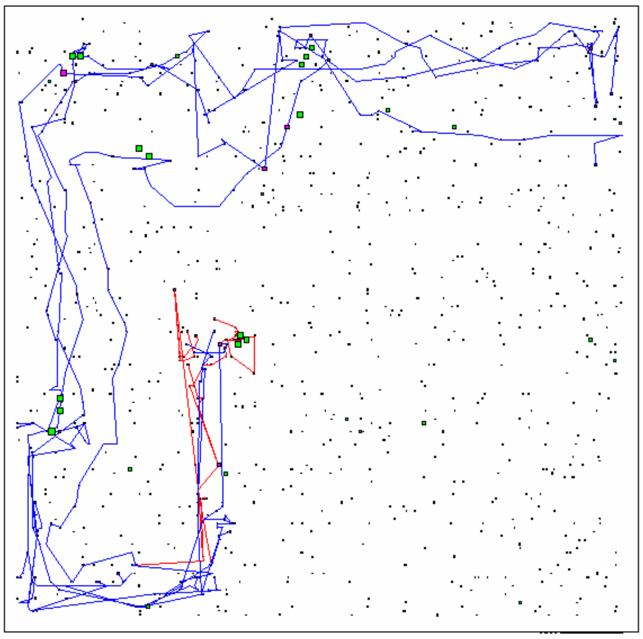
(b)

Figure 4.3: (a) ECO paths of Case 3 before optimization. (b) ECO paths of Case 3 after optimization.



TNS: 70

(a)



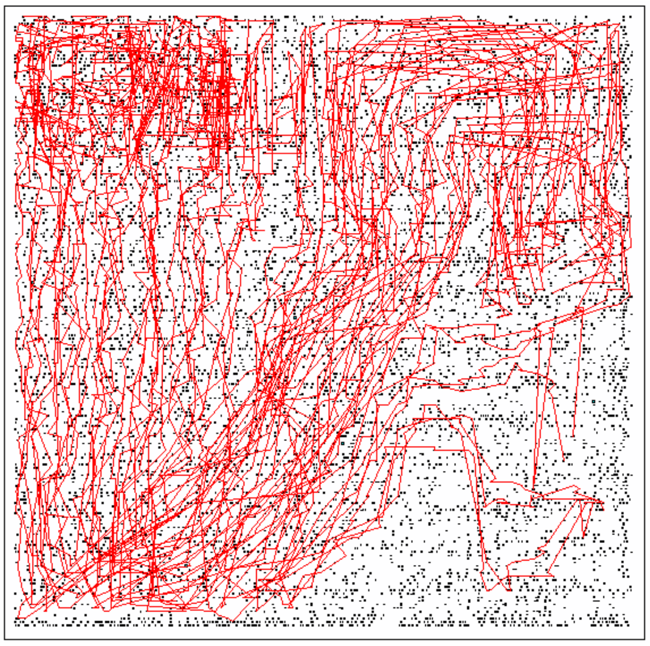
TNS: 0.57

Inserted buffer: 3

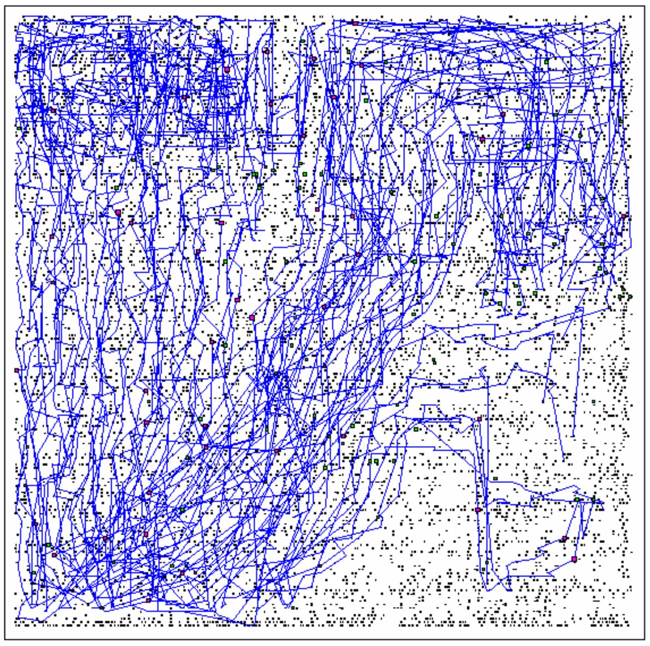
Sized gate: 25

(b)

Figure 4.4: (a) ECO paths of Case 4 before optimization. (b) ECO paths of Case 4 after optimization.



(a)



(b)

Figure 4.5: (a) ECO paths of Case 5 before optimization. (b) ECO paths of Case 5 after optimization.

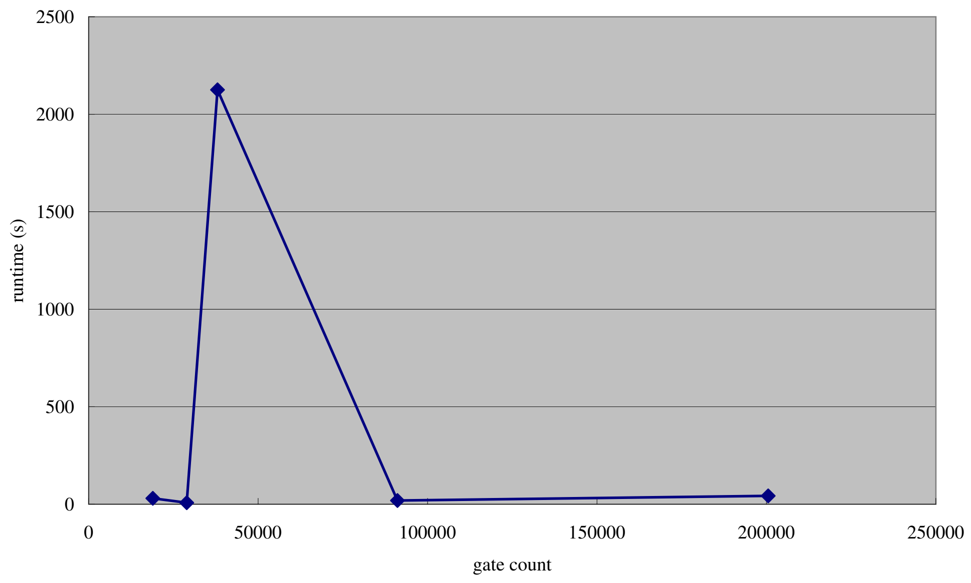


Figure 4.6: Runtime versus gate count for all benchmarks.

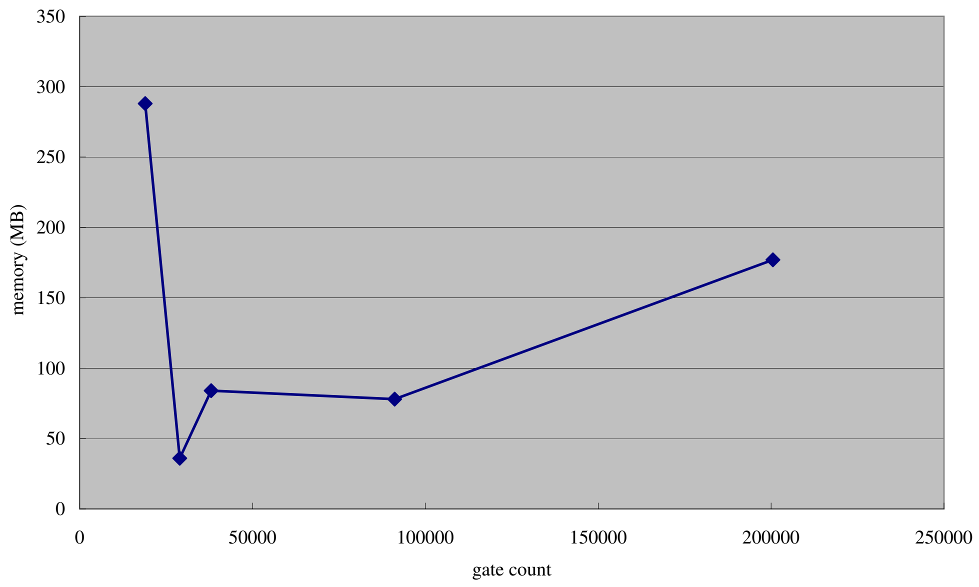


Figure 4.7: Memory usage versus gate count for all benchmarks.

Chapter 5

Conclusion and Future Work

In this thesis, we propose a spare cells selection algorithm to fix circuit timing after placement. The algorithm consists of gate sizing, buffer insertion, and technology remapping operations. The gate sizing and buffer insertion operations change the netlist gate by gate, while the technology remapping fixes timing violations in a more global view. Experimental results show that our algorithm can fix almost all of the timing violations in a much shorter time than the manual method.

We leave the gates not on the ECO paths unchanged in our method to reduce the problem size. A more general way of timing optimization using spare cells is replacing gates on non-ECO paths with spare cells and using those replaced gates as spare cells to optimize ECO paths. We will extend our work in this way later.

The functional change is a more difficult work than the timing optimization. Since the changed functions may be very complex and timing issues of the changed netlist also need to be considered, we have to carefully cope with logic and physical co-optimization. This is the direction we plan to advance our research in the future.

Bibliography

[1] C. J. Alpert, J. Hu, S. S. Sapatnekar, and C. N. Sze, “Accurate Estimation of Global Buffer Delay within a Floorplan,” in *Proceeding of International Conference on Computer Aided Design*, pp. 1140-1146, 2004.

[2] S.-C. Chang, C.-T. Hsieh, and K.-C. Wu, “Re-synthesis for Delay Variation Tolerance,” in *Proceeding of Design Automation Conference*, pp. 814-819, 2004.

[3] K. Chaudhary and M. Pedram, “A Near Optimal Algorithm for Technology Mapping Minimizing Area under Delay Constraints,” in *Proceeding of Design Automation Conference*, pp. 492-498, 1992.

[4] J. Cong and Y. Ding, “An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-table Based FPGA Designs,” in *Proceeding of International Conference on Computer Aided Design*, pp. 48-53, 1992.

[5] Faraday Technology Corporation, http://www.faraday-tech.com/index.html

[6] L. P. P P. van Ginneken. “Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay,” in *Proceeding of International Symposium on Circuits and Systems*, pp. 865-868, 1990.

[7] Z. Li, W. Shi, “An $O(mn)$ Time Algorithm for Optimal Buffer Insertion of Nets With m Sinks,” in *Proceeding of Asia and South Pacific Design Automation Conference*, pp. 320-325, 2006.

[8] D.-J. Jongeneel, Y. Watanbe, R. K. Brayton, and R. Otten, "Area and Search Space Control for Technology Mapping," in *Proceeding of Design Automation Conference*, pp. 86-91, 2000.

[9] K.Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching," in Proceeding of Design Automation Conference, pp. 617-623, 1987.

[10] Y. Kukimoto, R. K. Brayton, and P. Sawkar, "Delay-optimal Technology Mapping by DAG Covering," in *Proceeding of Design Automation Conference*, pp. 348-351, 1998.

[11] T. Kutzschebauch and L. Stok, "Congestion Aware Layout Driven Logic Synthesis," in *Proceeding of International Conference on Computer Aided Design*, pp. 216-223, 2001.

[12] T. Kutzschebauch and L. Stok, "Layout Driven Decomposition with Congestion Consideration," in *Proceeding of Design Automation and Test in Europe*, pp.672-676, 2002.

[13] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic Decomposition during Technology Mapping," In *Proceeding of International Conference on Computer Aided Design*, pp. 264-271, 1995.

[14] I.-M. Liu, A. Aziz, D.F. Wong, and H. Zhou, "An Efficient Buffer Insertion Algorithm for Large Networks Based on Lagrangian Relaxation," In *Proceeding of International Conference on Computer Design*, pp. 614-621, 1999.

[15] I.-M. Liu, A. Aziz, and D.F. Wong, "Meeting Delay Constraints in DSM by Minimal Repeater Insertion," In *Proceeding of Design Automation and Test in Europe*, pp. 436-440, 2000.

[16] Q. Liu and M. Marek-Sadowska, "Pre-layout Wire Length and Congestion Estimation," In *Proceeding of Design Automation Conference*, pp. 582-587, 2004.

[17] Q. Liu and M. Marek-Sadowska, "Technology Mapping: Wire Length Prediction-based Technology Mapping and Fanout Optimization," in *Proceeding of International Symposium on Physical Design*, pp. 145-151, 2005.

[18] J. Lou, W. Chen, and M. Pedram, "Concurrent Logic Restructuring and Placement for Timing Closure", in *Proceeding of International Conference on Computer Aided Design*, pp. 31-36, 1999.

[19] A. Lu, G. Stenz, and F. M. Johannes, "Technology Mapping for Minimizing Gate and Routing Area," in *Proceeding of Design Automation and Test in Europe*, pp. 664-669, 1998.

[20] Y. Matsunaga, "On Accelerating Pattern Matching for Technology Mapping," in *Proceeding of International Conference on Computer Aided Design*, pp. 118-122, 1998.

[21] A. Mishchenko, X. Wang, and T. Kam, "A New Enhanced Constructive Decomposition and Mapping Algorithm," in *Proceeding of Design Automation Conference*, pp. 143-148, 2003.

[22] M. Murofushi, T. Ishioka, M. Murakata and T. Mitsuhashi, "Layout Driven Re-synthesis for Low Power Consumption LSIs", in *Proceeding of Design Automation Conference*, pp. 666-669, 1997.

[23] D. Pandini, L. T. Pileggi, and A. J. Strojwas, "Understanding and Addressing the Impact of Wiring Congestion during Technology Mapping," in *Proceeding of International Symposium on Physical Design*, pp. 131-136, 2002.

[24] D. Pandini, L. Pileggi, and A. Strojwas, "Congestion-aware Logic Synthesis," in *Proceeding of Design Automation and Test in Europe*, pp. 664-671, 2002.

[25] M. Pedram and N. Bhat, "Layout Driven Technology Mapping," in *Proceeding of Design Automation Conference*, pp. 99-105, 1991.

[26] R. S. Shelar, P. Saxena, X. Wang, and S. S. Sapatnekar, "Technology Mapping: An Efficient Technology Mapping Algorithm Targeting Routing Congestion under Delay Constraints," in *Proceeding of International Symposium on Physical Design*, pp. 137-144, 2005.

[27] W. Shi and Z. Li, "An O(nlogn) Time Algorithm for Optimal Buffer Insertion," in *Proceeding of Design Automation Conference*, pp. 580-585, 2003.

[28] C. N. Sze, C. J. Alpert, J. Hu, and W. Shi, "Path Based Buffer Insertion," in *Proceeding of Design Automation Conference*, pp. 509-514, 2005.

[29] M. Zhao and S. S. Sapatnekar, "A New Structural Pattern Matching Algorithm for Technology Mapping," in *Proceeding of Design Automation Conference*, pp. 371-376, 2001.