APPLICATIONS OF SHORTEST PATH ALGORITHMS TO VLSI LAYOUT PROBLEMS

BY

ASHOK JAGANNATHAN
B.E., Computer Science and Engineering, Regional Engineering College, Trichy, India, 1997

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering and Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2000

Chicago, Illinois

To my parents...

ACKNOWLEDGMENTS

I express my profound gratitude to my advisor Dr.John Lillis for his constant guidance, support and encouragement throughout the course of this work. His friendly behavior with students has made working with him a wonderful experience. As his student, I have learnt a lot of things from him, both technically and as a person.

I also thank my committee members Dr.Shantanu Dutt and Dr.Peter Nelson for spending their precious time on reviewing my thesis and providing valuable comments on this work.

As colleagues in the same lab, I sincerely thank Sung-Woo Hur for all the help he extended without any reservations. I also thank Milos Hrkic for the time he spent on reviewing my thesis. I cannot forget the various technically enriching discussions we had during the course of our work. It was really a pleasure working with both of them.

Of course, I thank all my roommates for making my stay at UIC absolutely unforgettable.

Finally, I am very thankful to my parents, brother and sisters for having given me this opportunity and for constantly backing me on all my pursuits.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

SUMMARY

Due to the natural interpretation of a circuit as a graph model, a lot of research has been done on efficiently applying graph-theoretic algorithms to VLSI layout problems. In this thesis, we study two such applications of classical shortest-path algorithms in the context of VLSI layout synthesis.

The first problem deals with minimizing the wirelength of a given standard-cell layout by re-arranging the cells within a given window in a row. We study the modeling of this problem as a shortest-path problem and apply the A^* algorithm for efficiently finding the shortest path. A powerful network-flow based technique is proposed to find lower bounds on wirelength which is used to accelerate the A^* search.

The second problem deals with minimizing interconnect delay via buffer insertion in the context of a given layout, where there are typically restrictions on where buffers can be placed. We model this problem with a buffer graph and formulate a new problem called *Delay Reduction to Cost Ratio Maximization*, which is aware of the tradeoff between cost and delay. We also propose a fast algorithm for probing the tradeoff curve efficiently.

Our experimental results show the viability of the approaches suggested in this work.

CHAPTER 1

INTRODUCTION

The progressively increasing number of devices in modern high-performance circuits have made the use of CAD tools for layout synthesis inevitable. Moreover, due to the scaling down VLSI process technology, interconnect delay has become the bottleneck in designing modern circuits. Clearly, techniques which consider minimizing the interconnect delay directly (by inserting buffers, sizing wires etc.) or indirectly (by minimizing total wirelength etc.) have become necessary and important. Toward this end, a lot of work has been done in recent years, opening new avenues for potential research.

Due to the natural interpretation of a circuit representation as a graph model, applications of a variety of graph theoretic algorithms have been studied with respect to VLSI layout synthesis. For example, graph partitioning algorithms are extensively used to partition a large circuit into smaller components such that the number of interconnections between components is minimized, which is a very crucial problem in the layout domain. It is important to note that most of the layout problems are **NP-hard**(1) and hence there has been continued interest in exploring new applications of graph-based algorithms to these problems.

In this thesis, we study the applications of the classical *shortest-path algorithm* in a graph in the context of two different VLSI layout problems which are very important during layout optimization of any chip. Abstracts of these two applications are presented below.

1.1 Intra-row Optimization Of Standard-Cell Placement

Minimizing the total wirelength of a standard-cell placement by optimally placing cells within a row is a fundamental problem in CAD. Given a window of n adjacent cells in a row, the problem is to find the optimal arrangement of these cells within the window such that the total wire-length of all nets incident on these cells is minimized. In this work, we study a dynamic programming (DP) based algorithm which finds the minimum wirelength arrangement of cells by incrementally constructing all 2^n subsets of the given cells. The idea is based on the fact that the optimal arrangement of a subset S of placed cells within the window is independent of the arrangement of the unplaced cells (within the window). The DP algorithm implies a configuration graph with 2^n vertices, where each vertex represents a subset of cells within the window. In the traditional DP algorithm, this graph is exhaustively explored level-by-level. To speed up this process, we apply the A^* algorithm(2) to this configuration graph. By using A^* along with efficient lower bounding techniques using network flows, only a small percentage of the vertices are visited and hence the problem size which can be solved effectively is increased.

1.2 Context Aware Buffer Insertion

Buffer insertion has been proven to be a very powerful technique in optimizing interconnect delay. We study the problem of inserting buffers in the context of a given layout with possibly some restrictions on the location of buffers. Due to the presence of such restrictions on where buffers can be placed, it sometimes becomes necessary to detour just to "pick up" a buffer. Due to this reason, it is necessary to perform routing and buffer insertion simultaneously. Also, it is important that such algorithms are aware of the tradeoff between cost (eg. routing cost,

total capacitance etc.) and delay. In this context, we propose the *Delay Reduction to Cost Ratio Maximization* problem (which is similar to the shortest weight-constrained path problem (1)) and propose a fast algorithm for the same. Some interesting properties of the solutions generated by the algorithm have also been studied.

Chapter 2 deals with the linear arrangement problem and its application to standard-cell layout while chapter 3 discusses the problem of buffer insertion in the context of a given layout.

CHAPTER 2

INTRA-ROW OPTIMIZATION OF STANDARD-CELL LAYOUT

2.1 Standard-Cell Design

In a Standard-cell design methodology, the designer is provided with a library of efficiently designed basic logic cells such as NAND gates, Multiplexers, Decoders *etc.* Additionally, all logic cells in the library are implemented so that the height of the blocks is the same while the widths vary. In such a scenario, any logic function on the designer's side can be implemented in terms of a set of basic blocks taken from the standard-cell library. The advantage of such a design methodology is that designs can be completed rapidly. Also, as the layout for the cells from the library is readily available, the layout tool is only concerned about assigning locations to these cells and finding interconnections between these cells. Although the pre-designed cell library considerably reduces the complexity of the design process, there can be as much as hundreds of thousands of cells in a single circuit these days, and hence the physical design process has to be efficiently automated.

The *Standard-cell Placement* problem is related to finding locations for all the cells in the circuit such that no two cells overlap and the placement optimizes some objectives. Most common objectives for the placement problem are timing optimization, chip area minimization, wirelength minimization *etc.* or a composite function of these objectives. Irrespective of the objective function, most of the placement problems belong to the class of **NP-hard** (1) prob-

lems. The placement problem is one of the crucial problems in VLSI physical design, and it has a direct impact on the performance, area, routability and yield of the circuit.

Since all the cells in a standard-cell design have the same height, such a design is typically placed in rows of cells, as shown in Figure 1.

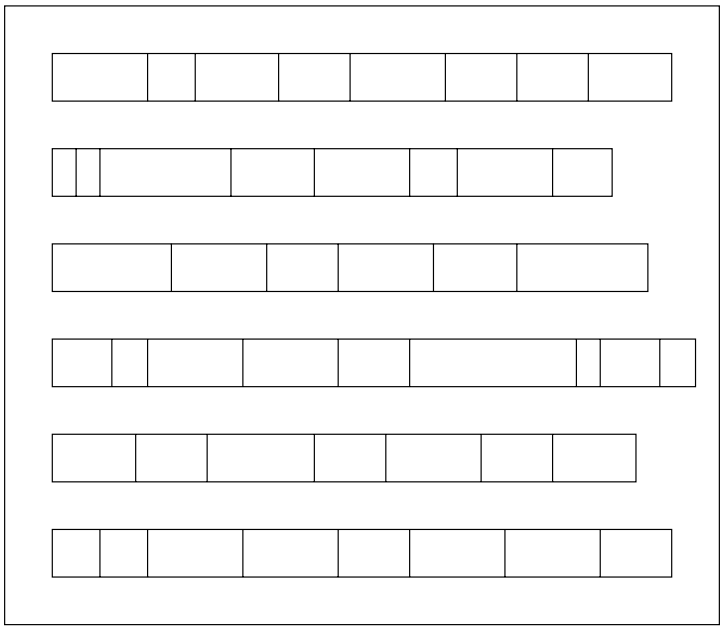

Figure 1: *Illustration of a typical standard cell placement where cells are distributed over a set of rows.*

Once the cells are distributed across rows, a variety of intra-row optimization techniques are applied to further improve the placement by optimizing certain objectives. The focus of this work is also on performing intra-row optimization by finding optimal linear arrangement

of cells within a row, where the cost of any arrangement is the sum of the wirelengths of the nets incident on the cells under consideration.

2.2 Preliminaries

A netlist is a representation of a circuit which specifies the components contained in the circuit along with the respective interconnecting signals. Such a netlist of any circuit can be modeled as a hypergraph $G = (V, E)$, where the vertex set V represents the set of cells (or modules) and the hyperedge set E represents the set of all signal nets that interconnect the cells. Each edge $e \in E$ connects a subset of two or more vertices from V – i.e., $e \subset V$.

Since each cell in the circuit has a non-zero length and width, the pins on each cell can be located anywhere on the cell. However, for simplicity reasons, we approximate the pin locations on any cell to be the center of that cell. Based on such an assumption, there are several methods to estimate the wirelength of any signal net in a given placement of a circuit. Of these, we use the half-perimeter(HP) metric explained below to estimate the wirelength of the nets.

2.2.1 Wirelength Estimation

Basically, the HP-estimator finds the smallest rectangle enclosing all the pins on the net, and takes the sum of the length and width of this rectangle to be an estimate of the wirelength of the corresponding net. Figure 2 shows an example where the HP-estimator is used to find the wirelength of a 6-pin net. The dark lines in the figure show the length and width of the smallest rectangle enclosing all the pins, and the sum of these two values is the estimated wirelength of this net.

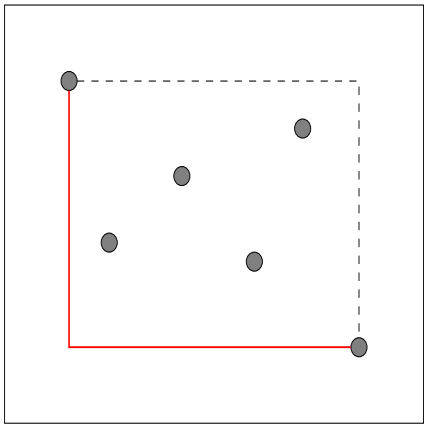

Figure 2: *Estimation of wirelength using Half-Perimeter metric. The wirelength of the net is the sum of the length and width of the smallest rectangle enclosing all the pins in the net as shown by the dark lines.*

The HP-estimator is widely used because of its simplicity in implementation. More importantly, the metric provides an exact estimate of the wirelength for 2-pin and 3-pin nets and a lower bound on the wirelength of nets with four or more pins.

In the following section, we formally introduce the linear arrangement problem in the context of a given standard-cell layout and discuss some previous work relevant to the problem.

2.3 Problem Formulation

Given a standard-cell layout with cells distributed over different rows, the optimal linear arrangement problem can be stated as follows.

Formulation 1 ***Given:*** *A set W of n cells in a row of a given standard-cell placement along with their connectivity information.*

Objective: *Find an optimal linear arrangement of the n cells within the window such that the sum of the wirelength of the nets incident on these cells is minimized.*

To better understand the problem, Figure 3 shows an example of a window of *eight* adjacent cells in a row of a standard-cell placement. As in the figure, the nets incident on the cells within the window may have pins outside the span of the window. Since we are only changing the x-positions of the cell inside the window (by re-arranging them), the y-direction wirelengths of the incident nets remain unaffected. Therefore, it is sufficient to consider only the change in the x-direction wirelength when computing the cost of any arrangement of cells.

Due to the above reason, throughout this work, we consider only variations in the x-dimensional wirelength of signal nets due to any changes in the positions of cells. Before we present our algorithm, we will briefly review some previous work relevant to this problem.

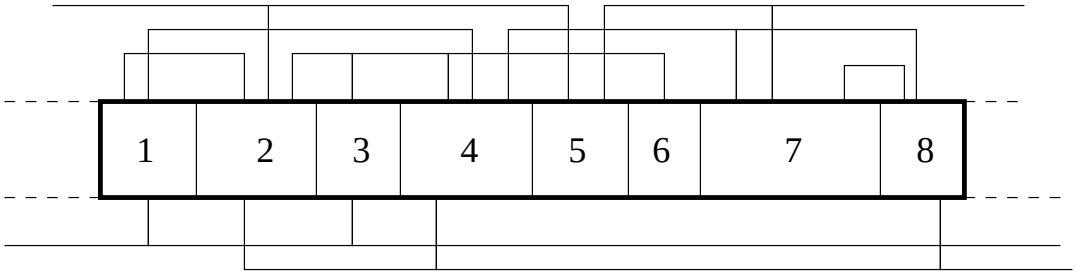

Figure 3: *Example of a window of 8 cells from a single-row of a standard-cell placement.*

2.3.1 Previous Work

The optimal linear arrangement problem naturally lends itself to *enumeration* and *branch-and-bound* based approaches. Enumeration based methods evaluate all $n!$ permutations of the given cells and choose the minimum wirelength arrangement of the cells. However, as stated before, such techniques become extremely slow in practice for even modest problem sizes (windows containing 8 or more cells) (3).

On the other hand, branch-and-bound based methods have been proposed to find the minimum wirelength arrangement of the cells (3). In such a branch-and-bound placer, starting from an empty initial arrangement, cells are added one at a time to the arrangement, and the bounding box of the incident nets are extended to include the the new pin locations introduced by the newly added cell. The efficiency of this approach largely relies on computing, from a given partial arrangement, a lower bound on the wirelength of any completion of this arrangement. Extensions of the current partial arrangement are considered only as long as this lower bound is smaller than the cost of the best complete arrangement seen so far. Though the branch-and-bound algorithms can be efficient depending on the lower bounding technique, they explore all $n!$ permutations in the worst case.

The techniques presented in this work are based on the fact that the optimal arrangement of a subset S of cells in W is independent of the arrangement of the cells in $(W - S)$ – i.e., any prefix of the minimum wirelength arrangement of all the cells in W is the minimum wirelength arrangement of the cells contained in that prefix. This observation was earlier applied to the problem of backboard ordering by Cederbaum (4). In this work, we study this property in the

context of a standard-cell placement and present an algorithm based on a dynamic programming framework to compute the minimum wirelength linear arrangement of the cells in W.

2.4 Optimal Arrangement By Dynamic Programming

Consider any possible linear arrangement L of all n cells. This linear arrangement can be obtained by placing one cell at a time starting from the left boundary of the window till all the n cells are placed. Since each cell v has some signal nets associated with it, each cell contributes some wirelength to the total wirelength of the arrangement L. We now introduce the notion of *incremental wirelength* contributed by any cell to the total wirelength of an existing partial arrangement.

Let $\mathcal{N}$ represent the set of all nets which are incident on at least one cell in the window – i.e., these are exactly those nets whose wirelength is affected by any re-arrangement of the cells within the window. Then, if S is any subset of cells in W, for each $v \in (W - S)$, we define the incremental cost $I(S, v)$ of the cell v with respect to the subset S as the total wirelength of all the nets in $\mathcal{N}$ within the region spanned by the width of cell v, when v is placed immediately to the right of any arrangement of the cells *only* in S.

To clearly illustrate the notion of incremental wirelength, Figure 4 shows an example of a window $W = \{a, b, c, d, e, f, g, h\}$ of eight cells. As shown, x_0 and x_3 mark the left and right boundaries of the window respectively. The subset $S = \{a, b, c, d\}$ of cells has already been placed as shown in the figure. We now want to place the cell e to the right of the cells only in S as shown. The incremental wirelength $I(S, e)$ is exactly the wirelength of the nets in the region between x_1 and x_2 which is the x-region spanned by the width of cell e. The portion of the nets

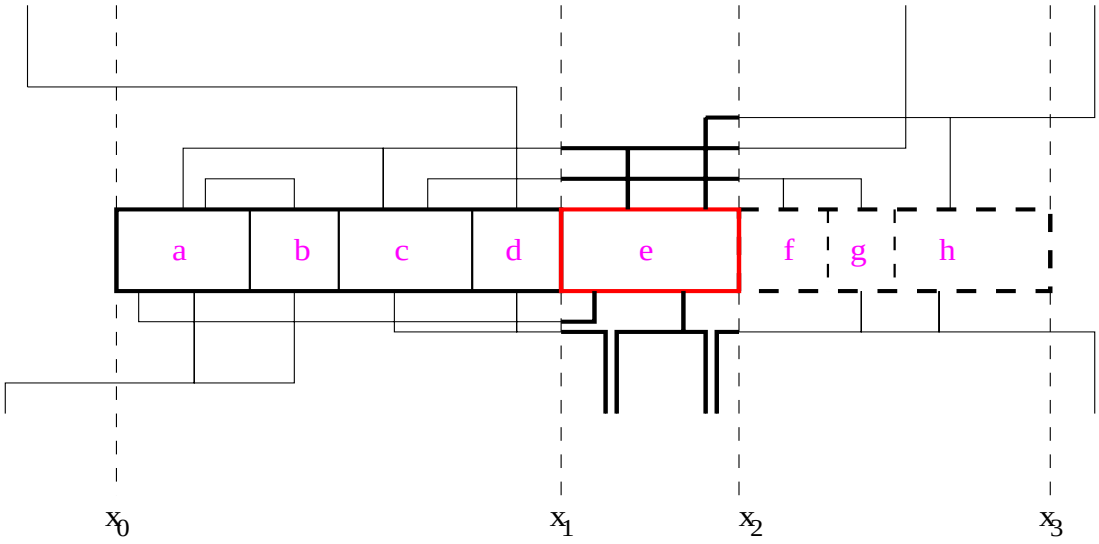

Figure 4: *Illustration of the notion of incremental wirelength of a cell with respect to a subset of cells.*

which contribute to $I(S, e)$ is shown in dark lines in the figure. The sum of the x-dimensional span of all the dark lines is the incremental wirelength for cell e with respect to subset S.

An interesting property of $I(S, v)$ is that it is independent of the arrangement of cells in S and in $(W - S - \{v\})$. It is only dependent on the cells in S and the corresponding x-region that is spanned by the width of cell v. Also, note that if $\emptyset$ represents the empty set, then $I(\emptyset, v)$ gives the incremental wirelength of placing cell v at the left boundary of the window.

Thus, the total wirelength of any linear arrangement L of all n cells can be computed as the sum of the incremental wirelengths contributed when each cell was added. The arrangement for which this sum is a minimum is the solution to the problem.

Based on this incremental wirelength computation, we now present a dynamic programming recurrence for finding the minimum wirelength arrangement of the cells in W.

2.4.1 Dynamic Programming Recurrence

Let $\emptyset$ represent the empty set and S be any subset of cells in W. If $Cost(S)$ denotes the wirelength of an optimal linear arrangement of the cells in S, then the recurrence relation can be stated as follows:

$$Cost(\emptyset) = \quad 0 \text{ and}$$

$$Cost(S) = \quad \min_{v \in S} \quad [Cost(S - \{v\}) + I(S - \{v\}, v)]$$

where $[Cost(S-\{v\})+I(S-\{v\},v)]$ is the total wirelength in placing the cell v immediately to the right of any optimal arrangement of the subset of cells $(S-\{v\})$ – i.e., to find the wirelength of an optimal arrangement of the cells in a set S, consider for each cell $v \in S$, abutting it to the right of an optimal arrangement of the subset $(S - \{v\})$; of all such possibilities, the one with the least wirelength is the solution for an optimal arrangement of S. Our objective then would be to find $Cost(W)$, which is the minimum wirelength of any optimal arrangement of all the n cells.

As an example, consider the set $S = \{a, b\}$ of two cells. Suppose we want to find the optimal arrangement of the cells in S. Following the recurrence relation, we have two choices – i.e.,

$[Cost(\{a\}) + I(\{a\}, b)]$ which is the cost of the arrangement ab and $[Cost(\{b\}) + I(\{b\}, a)]$, which is the cost of the arrangement ba. The one with the least cost of these two choices is the solution. However, to compute these values, we should have computed the minimum cost arrangement of sets $\{a\}$ and $\{b\}$, which is basically the cost of placing cell a or b at the left boundary of the window.

Observe that for any net in $\mathcal{N}$, the dynamic programming recurrence covers only the wirelength that is within the x-range spanned by the entire window – i.e., it does not take into account the wirelength of any net in $\mathcal{N}$ beyond the left and right boundaries of the window. For example, in Figure 4 the portion of the nets to the left of x_0 and right of x_3 is not taken into account by the dynamic programming recurrence. In the following section, we detail the procedure to compute the incremental cost of a cell v with respect to a subset S of cells.

2.4.2 Computing Incremental Cost

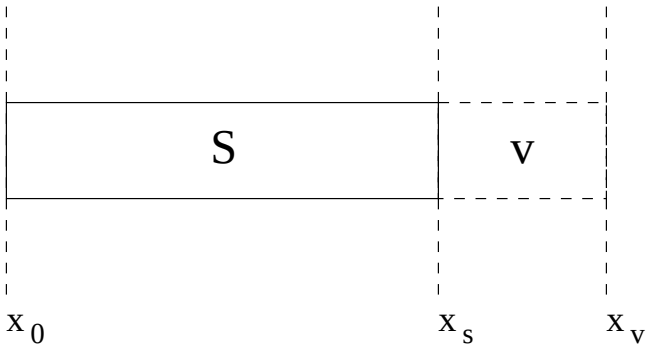

Figure 5: *Illustration of the extension of an arrangement of a subset of cells.*

Suppose we have already constructed a linear arrangement of a subset S of cells in W and we consider expanding S by placing one more cell $v \in (W - S)$ to its right. As shown in Figure 5, let x_0 be the left boundary of the window W and x_s represent the right boundary of the subset S. We will now show how to compute the incremental cost $I(S, v)$ of cell v with respect to S.

Assume that the cell $v \in (W - S)$ is placed to the right of S. If $width(v)$ represents the width of the cell v, then let $x_v = x_s + width(v)$ represent the right boundary of $(S \cup \{v\})$ as shown in Figure 5. As mentioned earlier, let $\mathcal{N}$ represent the set of nets which are incident at least on one cell in W. With respect to S, the subset of nets in $\mathcal{N}$ which contribute to the incremental wirelength of cell v can be classified as follows:

- **Terminating nets (T_v) :** These are the nets in $\mathcal{N}$ which have at least one pin to the left of x_s and have no pins to the right of x_v. Intuitively, these are the set of nets that start before x_s and terminate before x_v.
- **Continuing nets (C_v) :** These are the nets in $\mathcal{N}$ which have pins both to the left of x_s and right of x_v. Basically, these are the nets which start before x_s, continue over the region covered by the width of v and terminate somewhere to the right of x_v.
- **Starting nets (S_v) :** These are the nets in $\mathcal{N}$ which have no pin to the left of x_s, have atleast one pin between x_s and x_v and have at least one pin to the right of x_v. Alternatively, these are the nets which start between x_s and x_v and terminate somewhere to the right of x_v.

- **Starting and Terminating nets (ST_v) :** These are the nets in $\mathcal{N}$ which have all their pins between x_s and x_v – i.e., these are the nets which start as well as terminate between x_s and x_v.

It is important to note that the nets that contribute to $I(S, v)$ may or may not be incident on the cell v. To facilitate the computation of the wirelength of nets not incident on v, as a pre-processing step, for each net in $\mathcal{N}$, we determine the relevant **external pins** – i.e., the left or right extreme pins which are either in a different row or outside the window's span in the same row. The set of external pins is maintained as a sorted list so that it is easy to find the pins that lie within a given x-range.

Therefore, to compute the incremental wirelength $I(S, v)$, it is sufficient to process only those nets which are incident on cell v and those relevant external pins which lie within the x-region spanned by the width of cell v when placed immediately to the right of S. Once we determine the subset of nets which belong to each of the above mentioned types, the incremental wirelength $I(S, v)$ can be computed as stated below.

$$I(S, v) = |C_v| \cdot width(v) + \sum_{e \in ST_v} (r_e - l_e) + \sum_{e \in S_v} (x_v - l_e) + \sum_{e \in T_v} (r_e - x_s)$$

where r_e and l_e are the right and left extreme pins of net e based on the current arrangement of cells in S.

2.4.3 DP-Algorithm

The outline of an algorithm named **DP-Algorithm** based on the dynamic programming recurrence presented earlier is given in Figure 6. The algorithm starts from the left boundary of the window and incrementally computes optimal arrangement of all 2^n subsets until it finds the best arrangement of all n cells. The dynamic programming recurrence to compute the optimal arrangement for any subset is used in line (06) of the algorithm. Since the algorithm generates subsets in increasing sizes, it ensures that no subset is expanded before its optimal arrangement is determined.

DP-Algorithm
Given : A set W of n adjacent cells **Objective :** Min wirelength linear arrangement of cells
// W_i = set of all subsets of size i // x_0 is the left boundary of the window
01. $W_0 \leftarrow \emptyset$ 02. for $i = 1$ to n do 03. begin 04. for each subset $S \in W_i$ do 05. begin 06. $Cost(S) \leftarrow \min_{v \in S}[Cost(S - \{v\}) + I(S - \{v\}, v)]$ 07. end 08. end 09. // all 2^n subsets generated 10. Output $Cost(W)$

Figure 6: *Outline of the dynamic programming approach for finding the optimal linear arrangement of cells.*

It is important to note that when we find the minimum wirelength arrangement of a subset S of cells using the dynamic programming recurrence presented before, we also find the corresponding optimal arrangement of the cells in S. The recurrence ensures that no subset S is expanded – i.e., by adding another cell v from $(W - S)$, until the optimal arrangement for S is determined. The information on the arrangement of the cells in S is actually encoded in the last cell v that was added to S based on the recurrence – i.e., the cell v which corresponds to the minimum cost of S is the last cell added to S and appears as the rightmost cell in the optimal ordering of S. Thus we immediately have the information on the subset $(S - \{v\})$ from which we obtained the optimal arrangement for S, following which we can obtain the linear arrangement of cells which led to this minimum cost of S. Hence, when we reach the set W, we can backtrack based on the last cell added to W to obtain the corresponding linear arrangement of all n cells.

2.4.4 Complexity

The running time of the algorithm depends on the total time spent on computing the incremental wirelengths throughout the algorithm. As stated before, to compute the incremental wirelength $I(S, v)$, it is sufficient to examine all nets incident on cell v and all external pins which lie within the x-region covered by the width of cell v immediately preceded by S. Hence the total running time of the algorithm can be represented as $T_i + T_e$, where T_i is the total time to process incident nets during all incremental wirelength computations and T_e corresponds to the total time spent on external pins during all incremental wirelength computations. These two running times are analyzed separately below.

- **Case 1 (T_i):** This case deals with those nets which are incident on cell v when computing $I(S, v)$. Since any subset S can have all the n cells in the worst case, the time taken to examine all incident nets for the subset S is bounded by

$$\sum_{v \in W} \sum_{e \in N(v)} |e|$$

where $N(v)$ is the set of all nets incident on cell v. However, all the nets e in the above expression correspond to the set of nets $\mathcal{N}$ incident on atleast one cell in the window. Since any pin on net $e \in \mathcal{N}$ is examined at most $|e| - 1$ times (corresponding to examining the net on each of the remaining $|e| - 1$ pins), the total time spent on examining incident nets when computing $I(S, v)$ for any one subset is bounded by

$$\sum_{e \in \mathcal{N}} |e|^2$$

.

Since there are 2^n different subsets that are generated by the algorithm, the total time spent on examining all incident nets for all the subsets is bounded by

$$2^n \times \sum_{e \in \mathcal{N}} |e|^2$$

.

- **Case 2 (T_e):** The second case deals with the external pins which lie within the x-range covered by cell v placed immediately to the right of S. In the worst case, each external pin can be processed for $\begin{pmatrix} n \\ n/2 \end{pmatrix}$ different subsets and since each subset can be generated in at most n ways by adding a complementary cell to an existing subset, the number of times any external pin is processed can be bounded by $n \times \begin{pmatrix} n \\ n/2 \end{pmatrix}$. For the net that is incident on each such external pin, we have to find if the net belongs to T_v or S_v or C_v or ST_v, which requires examination of all pins on the net in the worst case.

  If Ext represents the set of all external pins relevant to the nets in $\mathcal{N}$, then the total time involved in processing all external pin related wirelengths is bounded by

$$n \times \begin{pmatrix} n \\ n/2 \end{pmatrix} \times \sum_{p \in Ext} |e_p|$$

  where e_p is the net incident on the external pin p. This reduces to

$$\sqrt{n} \times 2^n \times \sum_{p \in Ext} |e_p|$$

  since $\begin{pmatrix} n \\ n/2 \end{pmatrix}$ can be approximated to $\frac{2^n}{\sqrt{\pi \cdot n}}$.

Combining the bounds due to both the cases, the total runtime of the DP-Algorithm can be bounded by

$$O(\overbrace{2^n \times \textstyle\sum_{e \in \mathcal{N}} |e|^2}^{T_i} + \overbrace{2^n \times \sqrt{n} \times \textstyle\sum_{p \in Ext} |e_p|}^{T_e})$$

From the discussion, we can see that the running time of the DP-Algorithm depends on the problem instance – i.e., if $\sqrt{n} \geq |e|$ for all $e \in \mathcal{N}$, then T_e dominates the runtime. However, if $\sqrt{n} < |e|$ for all nets $e \in \mathcal{N}$, then the running time of the DP-Algorithm is dominated by T_i.

It is evident that in spite of its exponential complexity, the DP-Algorithm does not evaluate all $n!$ permutations even in the worst case (unlike enumeration or traditional branch-and-bound techniques). Thus, it is quite efficient for solving small problem instances. A more important limitation of this algorithm is memory – since it generates all 2^n subsets og the given n cells, the algorithm consumes a lot of memory as n increases. However, by an alternative graph-theoretic interpretation of the dynamic programming recurrence, we are able to apply powerful lower bounding schemes which effectively reduces the number of subsets expanded. Consequently, we are able to efficiently solve large problem sizes. Such a graph-based representation and the corresponding techniques are explained in the following section.

2.5 Graph-Theoretic Interpretation

As stated before, it is clear that the DP-algorithm finds the optimal arrangement of n cells by generating subsets incrementally – i.e., all subsets of size $(i-1)$ are generated before any subset of size i is generated. This leads to an alternative graph-theoretic interpretation of the DP-algorithm, on which finding an optimal arrangement is equivalent to finding a *shortest-path*

between two vertices in the graph. This abstraction was earlier studied by Cederbaum (4) in the context of finding optimal backboard ordering for minimizing the total interconnection length and was later used by Auer (5) in applying linear placement algorithm for cell generation.

To clearly understand the graph-based representation, we present the construction of a directed, acyclic graph $G = (V, E)$. The vertices $v_i \in V$ of this graph correspond to all the 2^n subsets W_i for all $i = 0, 1, 2, ..., 2^n - 1$ of the set W, including the empty set $\emptyset$ and W itself. The vertex v_0 corresponding to the $\emptyset$ is connected by edges e_{0i} to n vertices $v_i \sim \{w_i\}$, $i = 1, 2, .., n$ corresponding to the singleton subsets containing just one cell. In general, a vertex $v_i \sim W_i$, is connected by an edge e_{ij} to all vertices $v_j \sim W_j$, where $W_j = W_i \cup \{w\}$, $\forall w \in (W - W_i)$. An example of such a graph for a window of three cells $\{a, b, c\}$ is shown in Figure 7.

For a vertex v_k corresponding to a subset W_k with m different cells ($m \leq n$), there are m edges incident into v_k and $(n - m)$ edges incident out of v_k, together making n edges incident on v_k. Therefore, the graph G is a regular graph of degree n and the total number of edges in G is $|E| = (n \cdot 2^n)/2 = n \cdot 2^{n-1}$.

In this graph, $v_0 \sim \emptyset$ will be the source vertex and is the only vertex with n outgoing edges. Similarly, $v_t \sim W$ is the sink vertex and is also the only vertex in the graph with n incoming edges. Along any directed path $v_0 \rightsquigarrow v_t$, the order of the corresponding subsets of cells always increases. Consequently, no directed source-to-sink path can pass through the same vertex twice, and hence G is cycle free. Starting from the source vertex v_0, one can continue along a number of different directed paths, each of which contain exactly n edges, and terminate at the sink v_t.

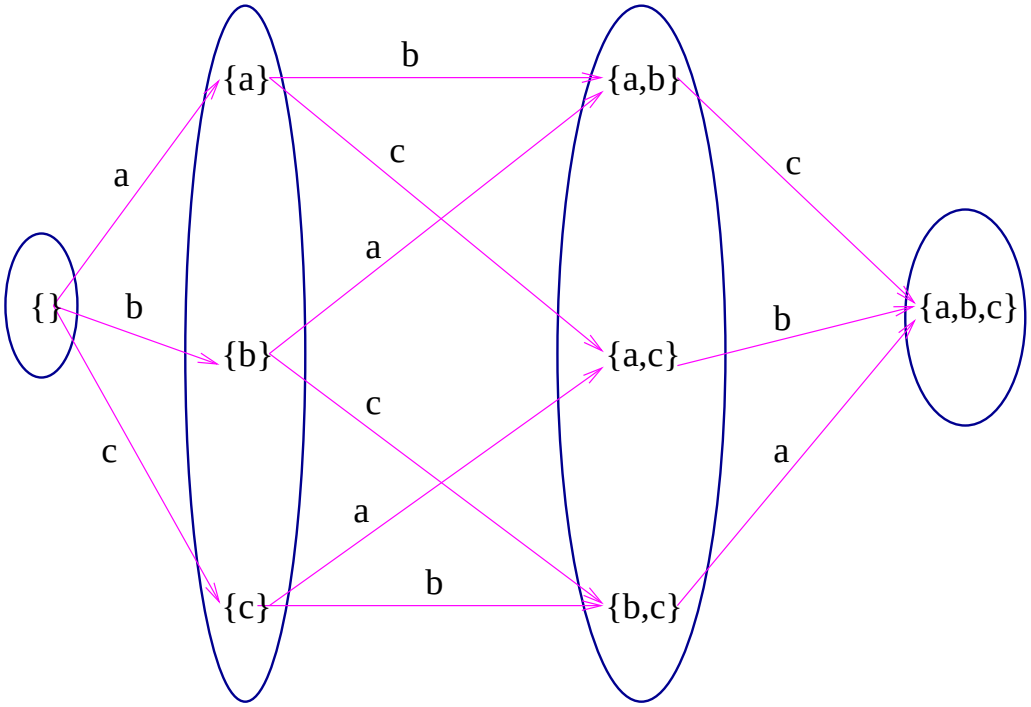

Figure 7: *Illustration of the graph implied by the DP-algorithm for an instance of three cells* $\{a, b, c\}$ *inside the window.*

An important property of the graph is that there exists a one-to-one correspondence between the linear arrangements of the cells in W and the directed source-to-sink paths in the graph G. To observe this, let $L = \{w_{i_1}, w_{i_2}, ..., w_{i_n}\}$ be some linear arrangement of the cells in W. Thus L can be represented by a sequence of subsets of cells as

$$S_L = (\emptyset, \{w_{i_1}\}, \{w_{i_1}, w_{i_2}\}, ..., \{w_{i_1}, w_{i_2}, ..., w_{i_{n-1}}\}, W)$$

which starts with the empty set $\emptyset$, terminates with the complete set of cells W, and whose jth term, $2 \leq j \leq n$, is the subset $\{w_{i_1}, w_{i_2}, ..., w_{i_{j-1}}\}$ formed by the first $j-1$ elements of L.

If the set S_L is projected on the graph G, its image will be a sequence of vertices in G, starting from $v_0 \sim \emptyset$. Since each element in the set S_L but for the first one $\emptyset$, is obtained by appending one complementary cell to the previous set of cells, the image vertices will be joined by edges of G, eventually forming a directed path P_L from v_0 to v_t. Thus, through the medium of the sequence set S_L of cells, the path P_L can be seen as the unique image of the linear arrangement L of cells on the graph G.

Since this argument is valid in both directions, the correspondence between any linear arrangement of cells in W and the directed source-to-sink paths in G is one-to-one.

Also, along any directed source-to-sink path, the out-degrees of the consecutive vertices in G (starting from v_0) follows the sequence n, $n-1$, $n-2$,..., 2, 1. Hence the number of different source-to-sink directed paths in G is equal to

$$(n)(n-1)(n-2)...(2)(1) = n!$$

which is the number of different permutations of the n cells.

To see that the optimal linear arrangement problem is equivalent to finding a shortest-path in this graph, let us introduce the notion of "length" for each edge. Consider any edge e_{ij} directed from vertex $v_i \sim W_i$ to vertex $v_j \sim W_j$. By the construction of the graph, we know that $W_j = W_i \cup \{v\}$ for some cell $v \in (W - W_i)$. Then, the length of the edge e_{ij} is given by

$I(W_i, v)$, which is the incremental cost of placing cell v to the right of any arrangement of cells in W_i. Based on this notion, all the edges edges in E can be annotated with their corresponding length values.

Now, consider any path P in the graph between vertices $v_0 \sim \emptyset$ and $v_t \sim W$. The total length of this path is equal to the sum of the lengths of its constituent edges. However, as discussed before, the path P corresponds to some linear arrangement L of the cells in W. Following the way the edge lengths are computed, it is clear that the length of P is equal to the total wirelength corresponding to arrangement L. Since the optimal linear arrangement problem is to find the linear arrangement with the least wirelength, this translates to finding the path between v_0 and v_t with shortest length – i.e., the shortest path between the two vertices.

It is interesting to note that the configuration graph need not be generated beforehand. In fact, the graph is implicitly constructed level-by-level – i.e. , all vertices (or subsets) at level i are generated before any vertex (or subset) at level $i+1$ is generated (as presented in Figure 6). Similarly, the edges going out of vertices at level i are also generated only during the expansion of vertices (or subsets) at level i by adding a complementary cell to the right of any arrangement of the cells in that subset. However, the optimal linear arrangement cannot be found until all the 2^n vertices are generated.

2.6 Accelerating Shortest-Path Search

The technique presented in this section for speeding-up the search for shortest path is based on the A^* approach (2)(6) which is a goal-oriented search paradigm. The approach relies on

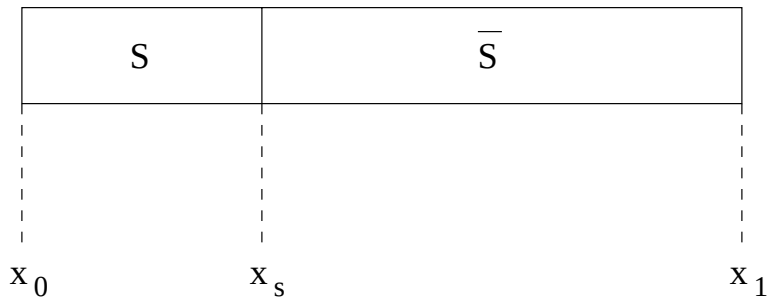

Figure 8: *Illustration of A^* approach.*

computing lower bounds on the wirelength of the "unplaced" cells with respect to any subset S.

Let C_{init} represent the total wirelength of the initial arrangement of the cells in W. As shown in Figure 8, assume that we have already computed the cost of an optimal arrangement of a subset S of cells in the window and we consider expanding S to include more cells from $\overline{S} = W - S$. Let $Cost(S)$ represent the wirelength of an optimal arrangement of cells in S. Note that $Cost(S)$ covers the wirelength of all nets in $\mathcal{N}$ between x_0 and x_s as shown in the Figure 8.

Let $lb(\overline{S})$ represent a lower bound on the wirelength of the nets in $\mathcal{N}$ within the x-region covered by cells in $\overline{S}$ – i.e., the region between x_s and x_1 in Figure 8.

Then $Cost(S) + lb(\overline{S})$ gives a lower bound on the total wirelength of any arrangement of the n cells, whose prefix is an optimal arrangement of cells in S. Let $h(S) = Cost(S) + lb(\overline{S})$ represent this heuristic lower bound.

Then, if $h(S) \geq C_{init}$, it means that any ordering of the cells in W, whose prefix is an optimal ordering of S will not produce a solution with a better cost than the initial cost C_{init}. Hence we do not have to expand the subset S. This way, we can suppress the expansion of many intermediate subsets. This heuristic $h(S)$ also captures the estimated path length from v_0 to v_t which passes through $v_s \sim S$. Therefore, a subset S with a lower value for $h(S)$ means that the path through the corresponding vertex v_s has more chances of being the shortest-path. Hence, among all candidate subsets for expansion at any stage, the subset with lowest h value is expanded first. For this reason, the configuration graph is not necessarily expanded level-by-level.

However, the effectiveness of such a technique largely depends on how tight lower bounds can we compute for the wirelength of the cells in $\overline{S}$. In this section, we present an efficient network-flow based method for computing lower bounds on wirelength, which helps in accelerating the shortest-path search.

2.6.1 Relaxation Based Lower Bound

The lower bounding technique that we use is based on a relaxation based approach as discussed in (7). The idea is to find optimal relaxed locations for all the "unplaced" cells in the given window W and use this information to obtain lower bounds on the wirelength.

Suppose that we want to find a lower bound on the wirelength of any arrangement of a subset S of cells within the window W. The set S of cells inside the window form the set of mobile cells – i.e., these cells can be moved from their current position in the placement. All nets which are incident on any mobile cell are termed as "active nets" E' – i.e., these are the

nets whose lengths are affected if we change the arrangement of the mobile cells. We then determine the set of fixed cells F as follows.

$F = \{v \mid v \in (e_i - S)$, where $e_i \in E'$, and v is either the left or right extreme cell of e_i $\}$.

The primary role of the fixed cells is to aid in establishing a lower bound on the wirelength of nets in E' – i.e., no matter where the mobile cells are placed, the total wirelength of nets in E' is at least as much as that determined by the positions of the relevant fixed cells in F.

When finding the optimal relaxed x-coordinates for the mobile cells, we consider only those fixed cells which affect the x-direction wirelength of the active nets. Since we are only re-arranging cell locations within a row, we will consider only finding relaxed locations along the x-direction.

To find relaxed x-positions for the set of mobile cells, we project the set of fixed cells on the x-axis. Note that we are concerned only about the location of the fixed cells. Hence if more than one fixed cell is projected to the same x-location, only one cell is considered for each associated x-location.

Figure 9 shows an example of a projected sub-circuit on the x-axis. The fixed cells, mobile cells and the corresponding interconnections are shown in the figure.

We now derive a *LP-formulation* for optimally placing the mobile cells. By following a LP-formulation, we are able to model hyper-edges exactly – i.e., there is no need to use a clique model as in other analytical methods. Similarly, the LP-formulation captures true linear wirelength objective.

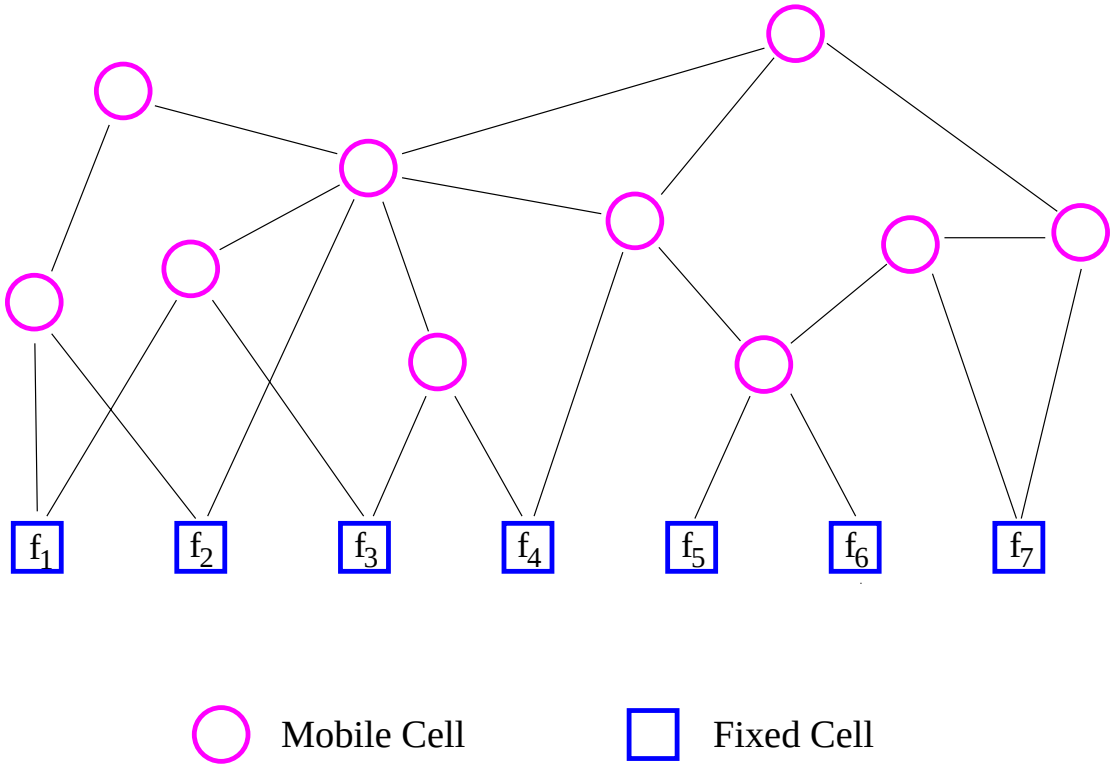

Figure 9: *Example of a sub-circuit projected on the x-axis.*

Such a linear program will produce an x-coordinate x_v for each mobile cell $v \in S$. The LP is of course influenced by the locations of the fixed cells F. Let X_v be the location of a cell $v \in F$ in a given placement. Then the LP relaxation for x-coordinate can be stated as follows.

$$\min \sum_{e \in E'} (r_e - l_e) \quad \text{subject to}$$
$$l_e \leq x_v \leq r_e, \qquad \forall v \in e,$$
$$x_v = X_v, \qquad \forall v \in F$$

The dummy variables r_e and l_e in the formulation give the leftmost and rightmost ends of net e.

This formulation can be solved using any standard LP-solver. However, we use the network flow based algorithm outlined in (7) to solve this problem as this approach has been found to be efficient in practice. The algorithm iteratively finds minimum cuts from left to right which assign mobile cells to bins formed by each of the fixed cells. The resulting relaxed placement is an optimal solution to the LP formulation.

Based on the relaxed x-locations obtained for all the cells in the set S, we can compute the lower bound on the wirelength of the corresponding active nets.

2.6.2 Faster Lower Bound Computation

Though a single run of the network-flow algorithm explained in the previous section is fast in practice, due to the large number of subsets (2^n in the worst case) for which we may want to find

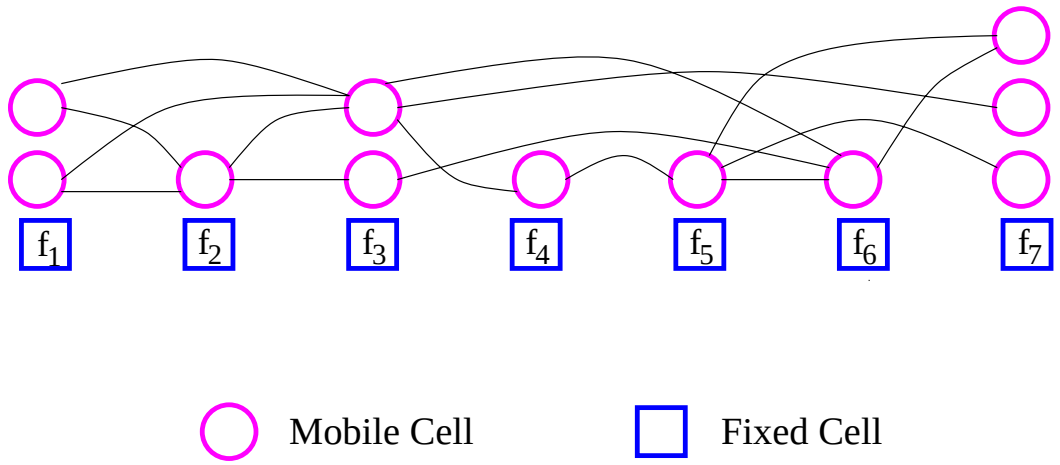

Figure 10: *Instance of an output generated by the network-flow algorithm. Many mobile cells can be associated with a single fixed cell.*

lower bounds, using the network-flow algorithm for each subset turns out be computationally expensive as n increases. In this section, we suggest a faster method for computing lower bounds, which uses information generated by a single run of the network-flow algorithm done as a pre-processing step.

Assume that we choose all the cells in the window W as mobile cells and run the network flow algorithm to find optimal relaxed locations for all mobile cells. Figure 10 shows an example of the output of the network-flow algorithm. As expected, there may be more than one mobile cell associated with each fixed cell.

An alternative interpretation of the output of the network-flow algorithm is that it gives the minimum number of nets cut between every pair of adjacent fixed cells, no matter how the mobile cells are arranged.

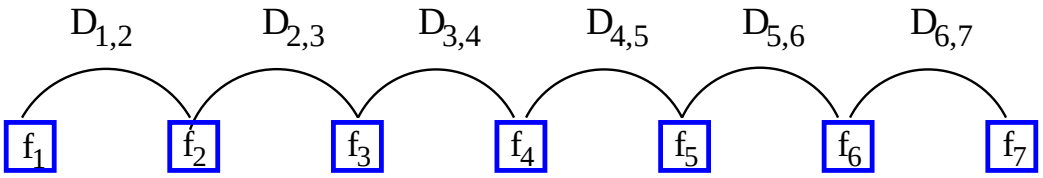

Figure 11: *Computing lower bounds from the information on the minimum number of nets cut between pair of adjacent fixed cells.*

Suppose, as shown in Figure 11, we have information on the minimum number of nets cut between every pair of adjacent fixed cells – i.e., let $D_{i,j}$ represent the minimum number of nets cut between fixed cells f_i and f_j. Then the lower bound on the wirelength of the active nets is given by

$$\sum_{i=2}^{|F|} D_{i-1,i} * (X_{f_i} - X_{f_{i-1}})$$

where X_{f_i} gives the x-location of the fixed cell f_i and $|F|$ is the total number of relevant fixed nodes.

Given any window of cells, this information on the minimum number of nets cut between adjacent fixed cells can be obtained using the network-flow algorithm as a pre-processing step. Once we have obtained this information, computing lower bounds on the wirelength for any subset of cells S can be done quickly as shown below.

As shown in Figure 12, assume that we already have an arrangement of a subset of cells S and we want to compute a lower bound on the wirelength of the nets corresponding to the

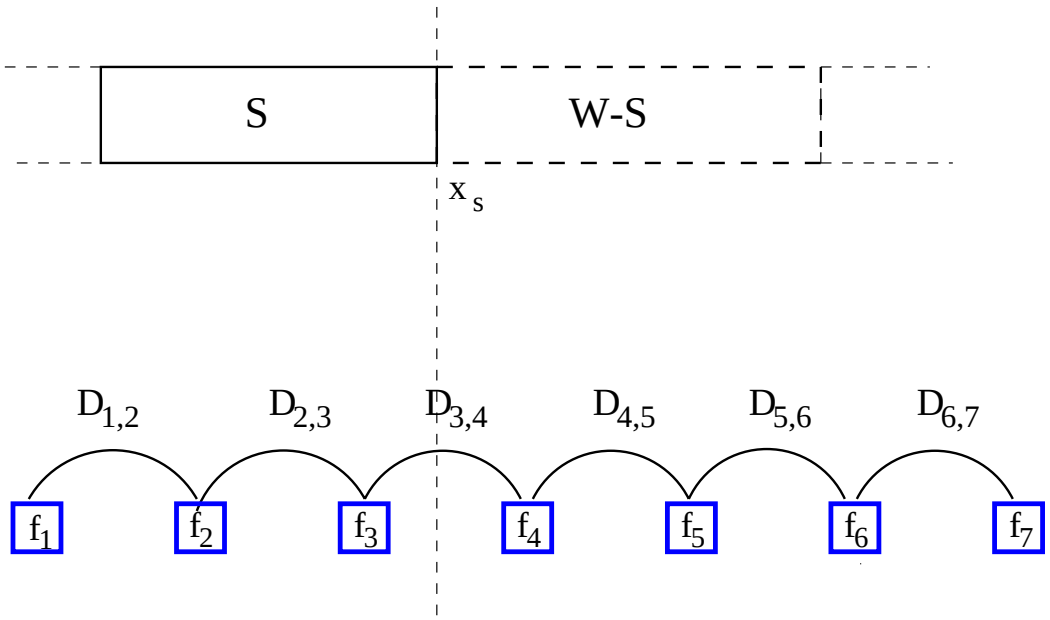

Figure 12: *Computing lower bounds for subsets based on the information obtained from a pre-processing step.*

unplaced cells in $\overline{S} = W - S$. Let x_s be the right boundary of S along the x-direction. Then if f_i and f_j are two adjacent fixed cells such that $X_{f_i} \leq x_s \leq X_{f_j}$, then the lower bound on the wirelength of the unplaced cells is given by

$$lb(\overline{S}) = D_{i,j} * (X_{f_j} - x_s) + \sum_{k=j}^{|F|-1} D_{k,k+1} * (X_{f_{k+1}} - X_{f_k})$$

However, it is to be noted that though this method to find a lower bound is faster than using the network-flow algorithm on $\overline{S}$, it may not be as tight as the lower bound provided

by the network-flow algorithm. Nevertheless, this simplified approach has been proven to be powerful in practice.

2.6.3 SP-Algorithm

An algorithm named **SP-Algorithm** (SP for shortest-path) which is based on the A^* paradigm and uses the above mentioned lower bounding technique is shown in Figure 13. The algorithm uses a priority queue to hold the subsets that are to be expanded. The subsets are prioritized based on the heuristic path length h through the corresponding vertex in the graph – i.e., the vertex with the lowest h value is the first one to be expanded.

Note that this algorithm does not explore the configuration graph level-by-level. Also, it does not generate all 2^n subsets unless necessary; the tighter the lower bound is, the lesser is the number of subsets generated by this algorithm. The SP-Algorithm has improvement both in terms of memory and run-time when compared to the DP-Algorithm, as shown in our experiments results.

2.7 Experimental Results

We have implemented both the dynamic programming based *DP-Algorithm* and the A^* based *SP-Algorithm* in *C*. Table I shows a list of circuits with their characteristics that we used for our experiments. The initial reference placements for these circuits were generated using the placement tool Mongrel(8).

The goal of the experiments was to evaluate the reduction in runtime produced by the SP-Algorithm over the DP-Algorithm for various window sizes. Also, we wanted to see the

SP-Algorithm

Given: A window W of n adjacent cells
Output: Minimum wirelength ordering of the n cells

// Q is a priority queue of sets S
// ordered by $Cost(S) + lb(\overline{S})$.
// C_{init} is the wirelength of the initial
// arrangement of cells in the window W.
// x_0 is the left boundary of the window

```
01. Run network-flow algorithm on W to find
    min-cut information between fixed cells.
02. EnQueue(Q,∅)
03. while (Q ≠ Empty) do
04. begin
05.       S ← DeQueue(Q)
06.       if (|S| == n) goto step 15
07.       for each v ∈ (W − S) do
08.         S' ← (S ∪ {v})
09.         Cost(S') ← Cost(S) + I(S, v)
10.         S'̄ ← (W − S')
11.         Compute lower-bound lb(S'̄)
12.         if Cost(S') + lb(S'̄) < C_init //feasible path
13.            EnQueue(Q,S')
14. end
15. // Shortest path to W == S found
16. Output Cost(S).
```

Figure 13: *Outline of the algorithm using lower bounding to find optimal linear arrangement.*

TABLE I

CIRCUIT CHARACTERISTICS

Circuit	# cells	# nets
industry2	12637	13419
industry3	15433	21940
avq.small	21918	22124
avq.large	25178	25384

tightness of the lower bounding schemes presented in this work in suppressing the unnecessary expansion of intermediate subsets.

Table II shows our experimental results for both the DP-Algorithm and the SP-Algorithm collected over multiple runs for window sizes ranging from 10 to 20. The table lists the percentage of subsets (of 2^n) generated by the SP-Algorithm, the runtimes for the DP and SP-Algorithm, and the reduction in runtime of the SP-Algorithm over the DP-Algorithm. We use the faster lower bound computation explained in Section 2.6.2 for the SP-Algorithm.

It can be seen from Table II that the SP-Algorithm generates only a small percentage of the subsets because of the lower bounding technique applied. Also, the average reduction in runtime of the SP-Algorithm over the DP-Algorithm is around 37.85%, which is attributed to the effectiveness of the lower bounds computed.

TABLE II

EXPERIMENTAL RESULTS

	DP-Algorithm	SP-Algorithm		
#cells	Time(s)	%subsets	Time(s)	Reduction in Time(%)
10	0.162	38.61	0.236	-45.67
11	0.388	35.55	0.420	-8.25
12	0.848	32.87	0.785	7.43
13	2.040	28.66	1.459	28.48
14	8.849	26.41	2.861	67.67
15	10.289	24.00	5.775	43.87
16	21.028	20.84	10.663	49.29
17	55.370	19.42	21.552	61.07
18	119.437	21.18	47.303	60.39
19	345.740	20.65	93.284	73.02
20	702.315	14.54	147.124	79.05
			Average	37.85

2.8 Comments and Conclusion

We studied the problem of min-cost linear arrangement in the context of intra-row optimization of a standard-cell placement. A dynamic programming approach was suggested to solve the problem optimally.

Also, it was shown that the dynamic programming approach implies a configuration graph, where the vertices represent all 2^n subsets of the given cells. Moreover, on this configuration graph, the problem of finding an optimal linear arrangement reduces to that of finding a shortest-path between two vertices.

A powerful network-flow based algorithm was suggested to find lower bounds on the wire-length of any arrangement of a set of cells. Using this lower bounding technique, an algorithm based on the goal oriented A^* framework was presented.

Our experimental results show the tightness of the lower bounds generated by the network-flow based approach. The results also show that by using such tight bounding techniques with the A^* approach, the size of the problem that can be efficiently solved increases.

CHAPTER 3

CONTEXT-AWARE BUFFER INSERTION

3.1 Introduction

In Chapter 2, we presented the application of the shortest path algorithm to find the optimal linear arrangement of cells within a given window in a placement. By using such intra-row optimization techniques, the wirelength of the given placement can be further optimized. Once we have an optimized placement, we perform routing of the signal nets to interconnect the cells in the circuit. At this stage, a variety of interconnect delay optimization techniques are applied to ensure that the signal nets meet the corresponding timing requirements. In this chapter, we study the application of the shortest path algorithm in the context of optimizing interconnect delay which is very crucial in practice.

In the Deep Submicron era, delay optimization for high performance interconnects has become of fundamental interest. In this context, buffer insertion has been proven to be a very powerful technique. Much of the past work (e.g.,(9)) on buffer insertion, while of fundamental interest, has focused on idealized situations where buffers can be inserted at arbitrary positions on the routing area. However, as pointed out in the recent work of Zhou and Wong (10), in practice such optimizations must occur in the context of, for example, a floorplan where there may be pre-placed macro cells which can be routed over, but which preclude the insertion of buffers in that region.

This is illustrated in Figure 14. The point illustrated in the figure is that it now seems critical to consider both the global route of a signal and the buffer insertion problem simultaneously since we may, for instance, have to detour not merely around congested routing regions, but also to "pick up" a buffer if necessary.

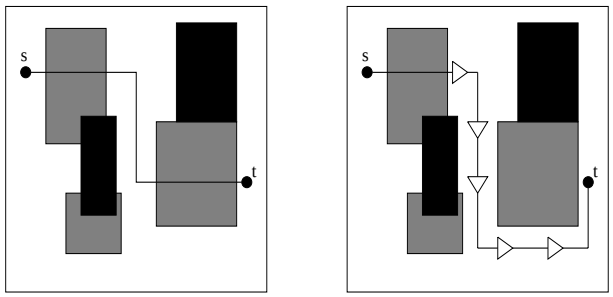

Figure 14: *An illustration of buffer insertion taking pre-placed macro cells into consideration; the black boxes represent portions of the routing area where neither buffering nor wiring is possible; the grey boxes represent areas where wires can be routed, but buffers cannot be inserted.*

This kind of context-aware buffer insertion problem was studied by Zhou and Wong in (10) in the context of the two-pin problem. Their main result was a labeling algorithm which finds the minimum delay buffered source to sink path. They also discussed several natural and more general formulations which capture a cost vs. performance tradeoff (e.g. minimizing congestion subject to a delay constraint). Such formulations were shown to be NP-hard and could in fact be viewed as instances of the classical shortest weight-constrained path problem (1). A

pseudo-polynomial algorithm for such formulations was also presented, which finds the set of all source-to-sink paths that lie on the cost vs. delay tradeoff curve (a similar algorithm appears in (11)).

This thesis also focuses on the two-pin problem with an emphasis on the tradeoffs between cost (e.g., total capacitance) and delay. The ability to capture such tradeoffs is crucial in practice since the cost overhead of "fastest" solutions tends to be excessive.

Toward this end we propose and characterize a new formulation, called the *Delay Reduction to Cost Ratio Maximization* **(DRCR)** problem. Given a set of candidate buffer insertion locations and their candidate connections modeled as a directed graph, and a reference delay value D_{ref}, we wish to maximize the ratio $\frac{D_{ref}-d}{g}$ over all source-to-sink paths, where g and d are the path cost and delay respectively – i.e., we maximize the ratio of the reduction in delay to the corresponding cost.

A nice property of this formulation is that it is completely independent of the cost and delay models used to estimate the cost and delay associated with the candidate edges interconnecting the buffers. For example, we are not restricted to using the total capacitance as the cost and Elmore delay model (12) for the interconnect delay (though for simplicity in our experiments we have used Elmore). By the same token, the cost associated with a particular candidate connection and buffer is also flexible; while total estimated capacitance is a natural measure, heuristic measures relating to congestion and buffer availability are also plausible.

It is suggested that the DRCR is a natural composite objective function capturing the tradeoff between cost and delay. Our main contribution is a fast polynomial time algorithm for

this problem. It is then natural to consider the relation between solutions of the DRCR problem and other formulations (in particular cost minimization subject to a delay constraint). Toward this end the problem is characterized with respect to all source-to-sink paths that lie on the cost vs. delay tradeoff curve (i.e., non-dominated paths). A subset of these paths forms the *Lower Convex Hull* (LCH); the LCH is essentially the points on the lower-left of the convex hull. It is shown that a variant of the algorithm can efficiently identify any point on the LCH. Thus we have a tradeoff: the expense of using the fast algorithm presented is that we are no longer able to identify paths which lie *off* the LCH while a comparatively slow pseudo-polynomial algorithm is able to identify such points. We argue that in practice this is not a major sacrifice since paths off the LCH tend to make less sound engineering choices. Computational experiments show the algorithms to be extremely efficient. Thus we believe the proposed algorithm will become a valuable tool in the early stages of design where buffers must be allocated and topological buffer-to-buffer routes determined.

3.2 Background

The algorithm that we present in this work is independent of the delay models that are used to model the interconnect delay. However, in this section, we briefly review some background material on delay models which are used in our experiments.

3.2.1 Delay Models

Though the graph model we utilize allows any desired technique to estimate the delay from the input of a buffer to the input of the next, we review some of the basic RC delay models for the purpose of discussion.

Let r_0 and c_0 represent the unit length resistance and capacitance respectively of a wire. Then for a wire e of length $l(e)$, the resistance $r(e)$ and the capacitance $c(e)$ are given by

$$r(e) = r_0 \cdot l(e) \quad \text{and} \quad c(e) = c_0 \cdot l(e)$$

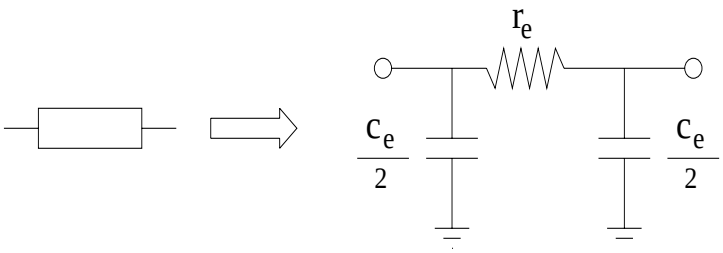

Figure 15: *RC delay modeling for a wire.*

Figure 15 shows the common RC model of a wire. Recall that according to the Elmore delay model (12), the delay D_e of a wire e driving a load C is given by

$$D_e = r(e) \cdot \left(\frac{c(e)}{2} + C \right)$$

Similarly, if b is a buffer with intrinsic delay d_b, output resistance r_b and a loading capacitance C, then the delay of the buffer D_b is given by

$$D_b = d_b + r_b \cdot C$$

3.2.2 Dominance Property

Since paths are characterized by two parameters g and d, they may be compared with a partial order. A path $p : s \rightsquigarrow t$ is said to be non-dominated, if all other paths $p' : s \rightsquigarrow t$, have $g' \geq g$ or $d' \geq d$. – i.e., the set of non-dominated paths are those on the cost vs. delay tradeoff curve intrinsic in the problem.

3.3 Buffer Graph

We model the context-aware buffer insertion problem by a directed graph in which nodes represent buffers and edges represent the candidate connections between buffers. A path in such a graph represents a sequence of buffers inserted by virtue of the nodes on the path. To avoid confusion, we emphasize that the buffer selection is *implicit* in the node – there is no need to explicitly determine the type of buffer inserted at a node; this is determined by the graph itself (see below).

Each edge e is annotated with two labels: g_e is the cost associated with taking the edge (including the routing cost and the cost of the destination buffer) and d_e is the delay from the input of the source buffer to the input of the destination. Figure 16 shows two buffers a and b and their candidate interconnection modeled as a graph, where the cost and the delay of the edges are computed under the Elmore delay model. Recall that the cost and delay of the edge are computed from the input of buffer a to the input of buffer b.

It is often the case that each buffer station has a set of buffers of different sizes to facilitate cascading of buffers within the buffer station for improved performance. An illustration of a buffer station with multiple buffer sizes and its corresponding graph model is shown in Figure

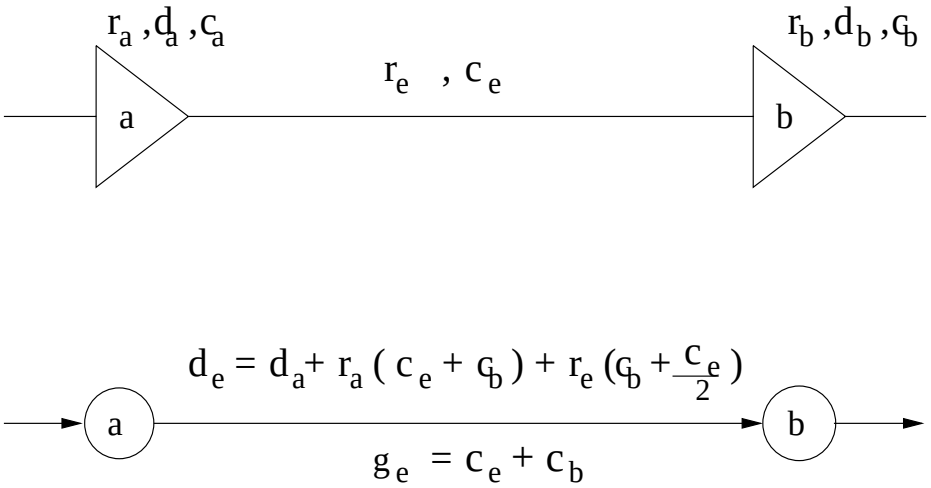

Figure 16: *Illustration of the transformation of a wire connecting two buffers into its corresponding graph model under the Elmore delay model. Note that $c(e)$ and $r(e)$ represent the capacitance and resistance of the wire connecting the buffers a and b while g_e and d_e represent the edge cost and delay in the corresponding graph model.*

Figure 17. Note that buffers within a buffer station are cascaded only in increasing order of their size – i.e., edges go only from smaller buffers to larger ones within the same buffer station. Thus the graph model naturally captures this situation.

Such an abstract graph model has several benefits. Of foremost importance is that it is completely independent of the models used to estimate the delay and cost of any edge in the graph – i.e., any desired means can be used to estimate the input-to-input delay and any cost measure can be applied depending on the situation. For example, the effects of coupling noise and congestion constraints due to pre-routed adjacent nets can be incorporated into the delay and cost measures. Also, more sophisticated interconnect and gate delay models can be used to model the delay associated with every edge. We are not limited for example to using Elmore

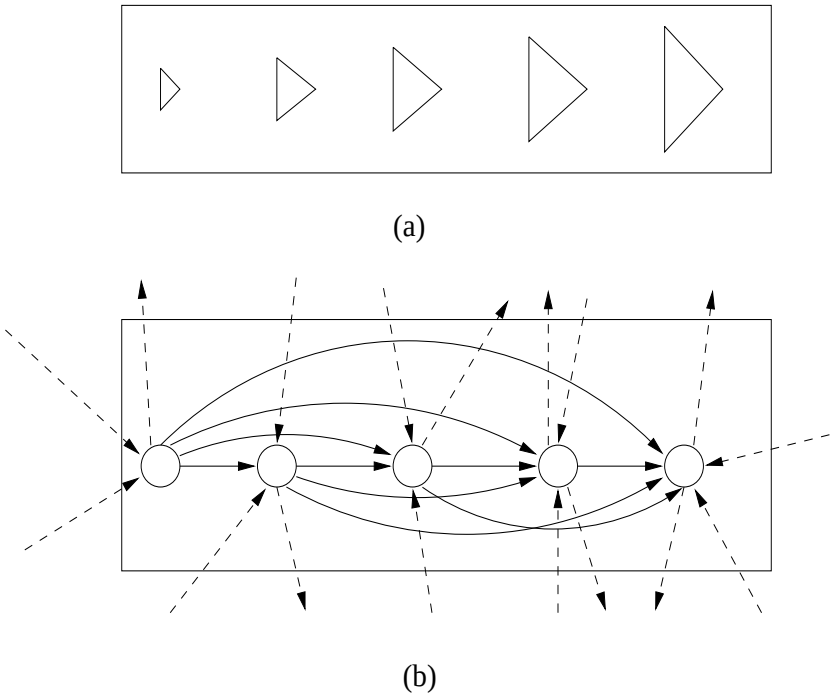

Figure 17: *Illustration of cascading buffers within a buffer station; solid lines represent connections within the buffer station and broken lines represent edges to and from outside the buffer station. (a) A buffer station with buffers of different sizes and (b) the corresponding graph model.*

delay or considering only the total capacitance as our cost measure.[1] Moreover, this graph model points out the intimate relationship between the context-aware buffer insertion problem and the shortest weight-constrained path problem in (1).

---

[1]As presented, we do require that the delay be independent of the previous stage ; however, if this is a serious issue, it can be modeled via a further transformation of the graph (at the expense of a large graph).

As stated earlier, a path in such a graph represents not only the wiring route to be taken, but by virtue of the vertices on the path, the buffers to be inserted. Figure Figure 18 presents a complete example of a set of buffer stations, the corresponding graph model, and two $s \rightsquigarrow t$ paths in the graph.

A naive graph construction method results in a complete graph – i.e., there is one vertex for each size of buffer inside every buffer station and an edge connecting every pair of vertices. However, in practice there is a threshold on the interconnect length beyond which a buffer must be inserted and thus it is sufficient if a buffer is connected to only its neighbors which lie within a specific technology dependent distance. By taking this factor into account during the construction process, the graph size can be reduced considerably.

3.4 Problem Formulations

Given such a graph theoretic interpretation of the problem, the traditional constrained optimization problem can be stated as follows (recall this problem is NP-hard).

Formulation 2 ***Given:*** *A directed graph $G = (V, E)$, where V represents the set of candidate buffer insertion locations, a buffer library B, each $e \in E$ is annotated with a cost g and delay d, a source terminal s with a driving resistance R_s, a sink terminal t with load C_t, and a delay bound d_{spec}.*

Objective: *Find a buffered path connecting s and t such that the total cost of the path is minimized subject to the delay not exceeding d_{spec}.*

The Delay Reduction to Cost Ratio Maximization problem is stated as follows.

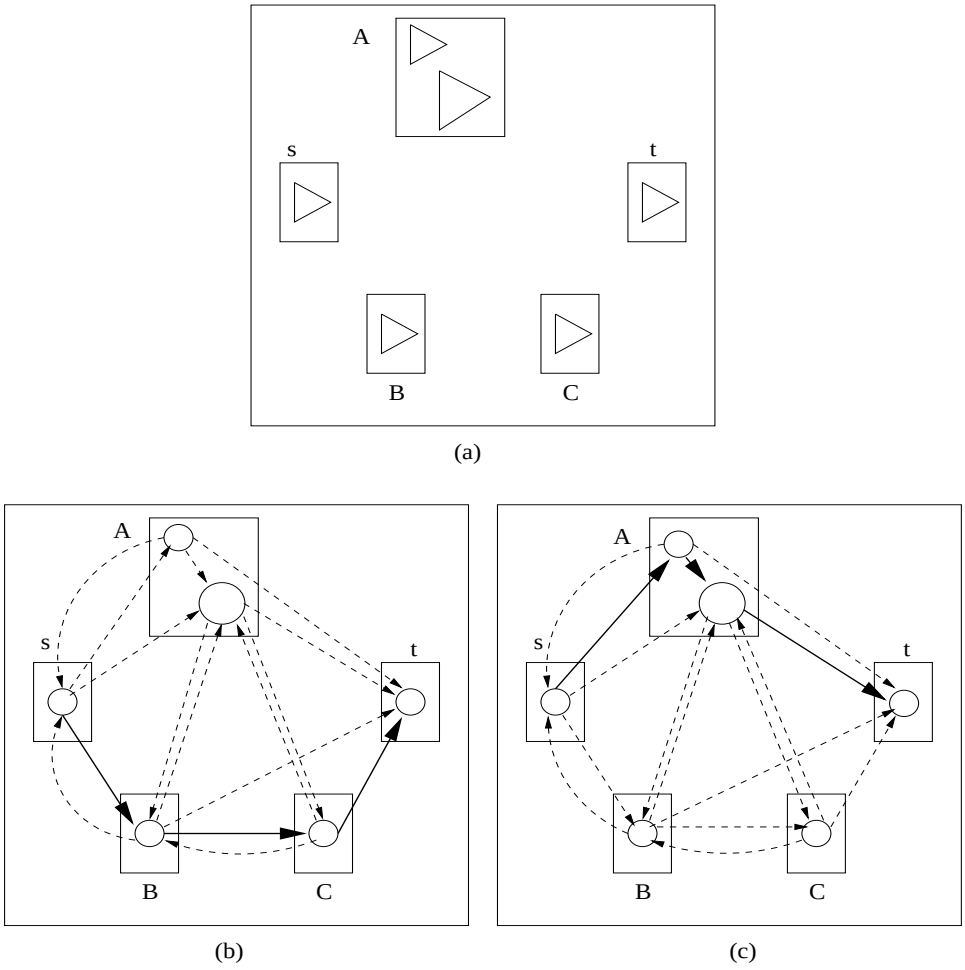

Figure 18: *Illustration of a set of buffer stations modeled as a graph ; all lines represent edges while the solid lines represent source-to-sink paths. (a) instance of a set of buffer stations with finite buffering resources (b) a simple $s \rightsquigarrow t$ path passing though various buffer stations (c) a $s \rightsquigarrow t$ path in which buffers are cascaded at the intermediate buffer station A.*

Formulation 3 ***Given:*** *A directed graph $G = (V, E)$, where V represent buffers, each $e \in E$ is annotated with a cost g and delay d, a source terminal s with driving resistance R_s, a sink terminal t with load C_t and a reference delay D_{ref}.*

Objective: *Find a buffered path $p : s \rightsquigarrow t$ in G such that the ratio $\frac{D_{ref} - \sum_{e \in p} d_e}{\sum_{e \in p} g_e}$ is maximized.*

Note that the selection of the reference delay D_{ref} value will influence the optimal path; this issue is addressed later.

The following subsection outlines a pseudo-polynomial labeling algorithm for solving the constrained optimization problem as in Formulation 1.

3.4.1 Labeling Algorithm

Since the labeling algorithm is not the focus of this work, we will not present the entire pseudo-polynomial algorithm for the constrained optimization problem. We point the reader to (11) and (10). We note however that the main idea is based on maintaining for each vertex u in the graph sets of non-dominated paths $P(u)$. A path is characterized by its cost and delay (g, d) and $(g, d) \in P(u)$ indicates that there exists a path from node u to the sink t (in a bottom-up approach) with cost g and delay d which is not dominated by any other u to t path. These sets are updated in what can be viewed as an extension of Dijkstra's algorithm (13) by examining the solutions at neighboring vertices. At termination, the set $P(s)$ encodes all of the non-dominated paths from s to t. The algorithm is pseudo-polynomial because the the sets $P(u)$ are not bounded in size by a polynomial function of the graph size; rather their size depends on the values of the problem instance (delays and costs).

3.5 DRCR Algorithm

The *Delay Reduction to Cost Ratio maximization* (**DRCR**) problem is the focus of this work. Fortunately, the DRCR problem appears to be computationally easier than the constrained optimization problem while still capturing key cost vs. delay characteristics. We now present a strongly polynomial time algorithm solving DRCR.

Recall that for a given value of the reference delay D_{ref}, our objective is to find a buffered path $p : s \rightsquigarrow t$ in G, such that the ratio $\frac{D_{ref} - \sum_{e \in p} d_e}{\sum_{e \in p} g_e}$ is maximized, i.e., we want to find the path which has maximum delay reduction to cost ratio.

Let R_{max} represent this maximum ratio. Then

$$R_{max} = \frac{D_{ref} - \sum_{e \in p} d_e}{\sum_{e \in p} g_e} \tag{3.1}$$

for some path $p : s \rightsquigarrow t$ in G. We can rearrange the above equation as

$$R_{max} \sum_{e \in p} g_e + \sum_{e \in p} d_e = D_{ref} \tag{3.2}$$

The left hand side of (3.2) can be interpreted as the total length of the path $p : s \rightsquigarrow t$ in G, where all the edges $e \in G$ are relabeled as $w_e = R_{max}\, g_e + d_e$. The idea behind our algorithm is to start with a conjecture I for the value of R_{max}, and iteratively correct the value of I until we find the actual value of R_{max} which satisfies (3.2) corresponding to the given D_{ref}, and also the associated path p,which has this maximum ratio.

Starting with the initial conjecture I, for each edge $e \in G$, we assign edge weights $w_e = I\, g_e + d_e$, where g_e and d_e are the original cost and delay values associated with edge e. With this relabeled graph, we find the *shortest path* p from s to t. Clearly, the length of the path p is given by $\sum w_e = I \sum g_e + \sum d_e$, where $e \in p$.,i.e., the sum of the cost and delay values associated with all the edges in the path. One of the following three situations is possible.

- If the length of the path p is **equal** to D_{ref}, then (3.2) is satisfied and we have the current $I = R_{max}$ and also the corresponding path p, and we are done.
- If the length of the path is **less** than D_{ref}, then we increase the value of I, relabel the graph with the new I value and repeat the algorithm until we reach a value of I for which equation (3.2) is satisfied.
- If the length of the path is **greater** than D_{ref}, then we decrease the value of I, relabel the graph with the new I value and repeat the algorithm until we reach a value of I for which equation (3.2) is satisfied.

Since we do not know how I should be increased or decreased during any iteration of the algorithm, we suggest a binary search technique to optimize the number of iterations (The algorithm employs binary search on the optimal ratio and is similar in spirit to algorithms for the minimum time-to-profit cycle problem presented in (14)).

The idea is to find two values of I, namely I_{low} and I_{high} such that

$$I_{low} \sum_{e \in p_{low}} g_e + \sum_{e \in p_{low}} d_e \leq D_{ref} \tag{3.3}$$

and

$$I_{high} \sum_{e \in p_{high}} g_e + \sum_{e \in p_{high}} d_e \geq D_{ref} \tag{3.4}$$

where p_{low} and p_{high} are the shortest s to t paths in the graphs relabeled with I_{low} and I_{high} respectively.

Identifying I_{low} and I_{high} can be done as follows. For the initial value of the conjecture I, if the length of the shortest path in the relabeled graph is less than D_{ref}, then we repeatedly keep doubling I, till we find the two successive values I_{low} and I_{high} such that equations (3.3) and (3.4) are satisfied. On the other hand, if starting from the initial I, the length of the shortest path is greater than D_{ref}, we keep halving I until we find I_{low} and I_{high}.

Once we find I_{low} and I_{high}, we can do a binary search in the range $[I_{low}..I_{high}]$ to identify the final value of I which maximizes the delay reduction to cost ratio. The **DRCR** algorithm which is based on such a binary search technique is shown in Figure Figure 19.

The correctness of the iterative search algorithm in identifying the path with maximum delay reduction to cost ratio is justified by the following lemma.

Lemma 1 *Given D_{ref}, the iterative search technique finds the path for which*

$$I = \frac{D_{ref} - d}{g}$$

is a maximum, *where g is the cost and d is the delay of the path.*

```
Algorithm DRCR
    Subroutine AssignWeights (G, I)
        For each e ∈ G,
            w_e = I g_e + d_e
    Main Routine
1   Find I_low and I_high
2   I ← (I_low + I_high)/2
3   AssignWeights(G, I)
4   P ← shortest s ⇝ t path in G.
5   while (D_ref ≠ length(P))
6       if (length(P) > D_ref)
7           I_high ← I
8       else
9           I_low ← I
10      endif
11      I ← (I_low + I_high)/2
12      AssignWeights(G, I)
13      P ← shortest s ⇝ t path in G.
14  endwhile
15  return P
```

Figure 19: *DRCR algorithm to solve the ratio maximization problem.*

Proof : Let p be the shortest path between s and t found during any stage of the algorithm. Following assignment of weights to edges, the length of p is given by $I \sum_{e \in p} g_e + \sum_{e \in p} d_e$. At any stage, we need to deal with one of the following three cases:

- **Case 1:** $I \sum_{e \in p} g_e + \sum_{e \in p} d_e < D_{ref}$. Then,

$$I < \frac{D_{ref} - \sum_{e \in p} d_e}{\sum_{e \in p} g_e}.$$

i.e.,the ratio for the path p is clearly better than the conjecture I.

So, I is increased.

- **Case 2:** $I \sum_{e \in p} g_e + \sum_{e \in p} d_e > D_{ref}$.

  Since p is the shortest path, for all other paths p',

$$I \sum_{e \in p'} g_e + \sum_{e \in p'} d_e > D_{ref}.$$

  Also,

$$I > \frac{D_{ref} - \sum_{e \in p'} d_e}{\sum_{e \in p'} g_e} \ \forall\, p'.$$

  So, I is decreased.

- **Case 3:** $I \sum_{e \in p} g_e + \sum_{e \in p} d_e = D_{ref}$.

  Therefore,

$$I = \frac{D_{ref} - \sum_{e \in p} d_e}{\sum_{e \in p} g_e}.$$

  Since p is the shortest path, all other paths p' have

$$I \sum_{e \in p'} g_e + \sum_{e \in p'} d_e \geq D_{ref}.$$

and

$$I \geq \frac{D_{ref} - \sum_{e \in p'} d_e}{\sum_{e \in p'} g_e}.$$

i.e. all those paths p' have smaller ratios.

The algorithm terminates with the maximum ratio path. □.

3.5.1 Complexity

Observation : If D is the delay of the path with minimum cost, then the number of invocations of Dijkstra's algorithm is bounded by $O(\log D)$.

The running time of the **DRCR** algorithm is given by (number of invocations of the shortest path algorithm) $\times$ (time to find the shortest path) *i.e.* $O(\log D \times E \log |V|)$.

3.6 Properties

In this section, we explain some interesting properties of the solutions generated by the DRCR algorithm.

Lemma 2 *For any value of D_{ref}, the optimal ratio solution is not dominated by any other solution.*

Proof: Let (g, d) be the path found by the DRCR algorithm. Suppose there exists a path (g', d') which dominates (g, d) – i.e., $g' < g$ and $d' < d$. Since (g, d) is found by the DRCR algorithm and has the maximum delay reduction to cost ratio,

$$\frac{D_{ref} - d}{g} > \frac{D_{ref} - d'}{g'}.$$

Therefore,

$$\frac{D_{ref}}{g} - \frac{d}{g} > \frac{D_{ref}}{g'} - \frac{d'}{g'}$$

$$\frac{g'}{g}(D_{ref} - d) > D_{ref} - d'$$

Since we assumed that $d' < d$,

$$D_{ref} - d' > D_{ref} - d$$

Hence,

$$\frac{g'}{g}(D_{ref} - d) > D_{ref} - d$$

This is a *contradiction* as $\frac{g'}{g} < 1$ since $g' < g$ $\square$.

Lemma 3 *As the value of D_{ref} decreases, the cost of the corresponding maximum ratio path increases.*

Proof : Let p be the maximum ratio path for some value of D_{ref}, say D. Let (g_p, d_p) be the cost and delay values of this path p respectively.

Suppose p' is the maximum ratio path corresponding to $D_{ref} = (1-\epsilon)D$, for some value $0 < \epsilon < 1$. Let $(g_{p'}, d_{p'})$ be the cost and delay values with respect to path p'.

It follows from our claim in the previous section that p and p' are the non-dominated solutions for the D_{ref} values D and $(1-\epsilon)D$. Therefore,

$$\frac{D - d_p}{g_p} \geq \frac{D - d_{p'}}{g_{p'}}$$

as p is the maximum ratio path for the value of $D_{ref} = D$.

Similarly,

$$\frac{(1-\epsilon)D - d_{p'}}{g_{p'}} \geq \frac{(1-\epsilon)D - d_p}{g_p}$$

as p' is the maximum ratio path for $D_{ref} = (1-\epsilon)D$.

Adding the two inequalities, we get

$$\frac{D - d_p}{g_p} + \frac{(1-\epsilon)D - d_{p'}}{g_{p'}} \geq \frac{D - d_{p'}}{g_{p'}} + \frac{(1-\epsilon)D - d_p}{g_p}$$

With algebraic manipulation of the above inequality, we find that

$$g_p \leq g_{p'} \quad \square.$$

By Lemma 2, we know that the solutions generated by the DRCR algorithm are non-dominated and hence lie on a cost vs. delay tradeoff curve. Consequent to Lemma 3, we see that decreasing D_{ref} moves us right on the cost vs. delay curve; increasing D_{ref} moves us left.

Thus, a natural goal is to characterize the solutions to the DRCR problem and the set of *all* non-dominated paths (i.e., those generated by the labeling algorithm). This relationship is explained in the next subsection.

3.6.1 Lower Convex Hull

Definition : Let $S = \{(g_0, d_0), (g_1, d_1), ..., (g_k, d_k)\}$ be the set of non-dominated s to t paths. We define the *lower convex hull* (LCH) of S as follows.

- The minimum delay solution is on the LCH.
- The minimum cost solution is on the LCH.
- Any point (g_i, d_i) is on the LCH iff $\forall\ (g_l, d_l) \in S$ such that $g_l < g_i$ and $\forall\ (g_g, d_g) \in S$ such that $g_i < g_g$, (g_i, d_i) lie **below** the line segment joining (g_l, d_l) and (g_g, d_g).

An example of a set of non-dominated points appears in Figure Figure 20.

Since the constrained optimization problem is NP-hard, it is clear that our polynomial time algorithm must sacrifice something. What we sacrifice is summarized in the following theorem.

Theorem 1 *There exists a D_{ref} for which a (g, d) path is optimal* **iff** *(g,d) lies on the lower convex hull of the trade-off curve.*

Proof : (i) $\exists D_{ref} \Rightarrow (g, d)$ is on LCH.

Let D be a value of D_{ref} for which this is true and let (g, d) be the corresponding path. This yields

$$\frac{D - d}{g} > \frac{D - d'}{g'} \tag{3.5}$$

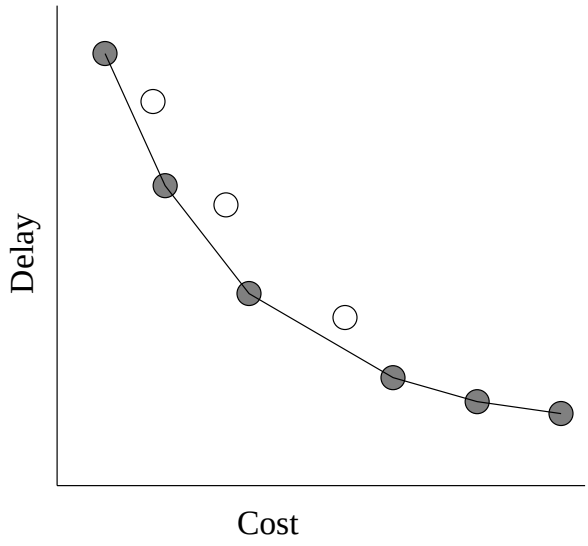

Figure 20: *Illustration of the Lower Convex Hull property. All circles represent the set of all non-dominated paths; the shaded circles represent points on the LCH.*

for all other non-dominated paths (g', d'). If (g, d) is the min-cost or the min-delay solution, then the proof is trivial (these paths are always on the lower convex hull).

Hence we only need to prove that for any pair of solutions (g_1, d_1) and (g_2, d_2) such that $g_1 < g < g_2$ (as in Figure Figure 21),

$$\frac{d - d_1}{g - g_1} < \frac{d_2 - d_1}{g_2 - g_1} \tag{3.6}$$

i.e., the slope of the line joining (g, d) and (g_1, d_1) should be less that the slope of the line joining (g_1, d_1) and (g_2, d_2).

Rearranging inequality (3.6),

$$d < d_1 + (g - g_1)\frac{d_2 - d_1}{g_2 - g_1} \quad \text{and hence} \tag{3.7}$$

$$d < \frac{g_2 d_1 + g d_2 - g d_1 - g_1 d_2}{g_2 - g_1} \tag{3.8}$$

Hence our goal is to prove inequality (3.8) holds.

Since D is the value of D_{ref} for which the path (g, d) has a maximum ratio,

$$\frac{D - d}{g} > \frac{D - d_1}{g_1} \quad \text{and} \tag{3.9}$$

$$\frac{D - d}{g} > \frac{D - d_2}{g_2}. \tag{3.10}$$

Rearranging inequality (3.9), we have

$$d < D - \frac{g}{g_1}D + \frac{g}{g_1}d_1. \tag{3.11}$$

Multiplying (3.11) by g_1,

$$g_1 d - g d_1 < D(g_1 - g).$$

Since the term $(g_1 - g)$ is negative, we have

$$\frac{g_1 d - g d_1}{g_1 - g} > D. \tag{3.12}$$

Rearranging inequality (3.10) and multiplying g_2 yields

$$\frac{g_2 d - g d_2}{g_2 - g} < D. \tag{3.13}$$

Combining (3.12) and (3.13) , we have

$$\frac{g_1 d - g d_1}{g_1 - g} > \frac{g_2 d - g d_2}{g_2 - g}.$$

Multiplying both sides by $(g_1 - g)(g_2 - g)$, which is negative,

$$(g_2 - g)(g_1 d - g d_1) < (g_1 - g)(g_2 d - g d_2).$$

By algebraic manipulation, we see that this reduces to

$$(g_2 - g_1)d < g_2 d_1 + g d_2 - g d_1 - g_1 d_2$$

which is the same as inequality (3.8).

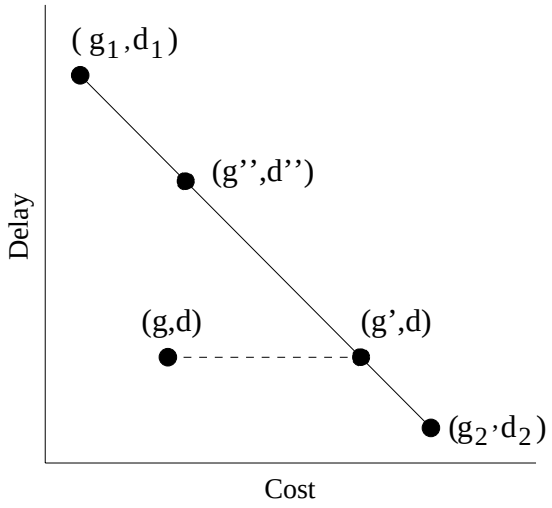

Figure 21: *Figure illustrating the existence of a D_{ref} for every (g, d) on the LCH.*

(ii) To prove the other way –*i.e.*, (g, d) is on the LCH $\Rightarrow$ $\exists D_{ref}$ which has the maximum ratio at (g, d), set

$$D_{ref} = \frac{g_2 d_1 - g_1 d_2}{g_2 - g_1}$$

which makes the ratio

$$r = \frac{D_{ref} - d_1}{g_1} = \frac{D_{ref} - d_2}{g_2}.$$

As in Figure Figure 21, for any point (g'', d'') on the line joining (g_1, d_1) and (g_2, d_2),

$$\frac{D_{ref} - d''}{g''} = r.$$

Consider any point (g', d) which is on the line and has the same d value with (g, d). This yields

$$\frac{D_{ref} - d}{g'} = r \quad \text{and thus}$$

$$\frac{D_{ref} - d}{g} > r \quad \text{since } g < g'$$

which completes the proof. $\Box$.

This theorem can be seen as a generalization of Lemma 2. We argue that sacrificing the solutions not on the LCH is typically not a serious drawback in practice since the solutions which make the most cost-effective use of the available resources are precisely those which lie on the LCH.[1] Further, in practice optimization problems such as buffer insertion tend to have largely convex tradeoffs, so the algorithm fits such applications nicely.

Finally, we note that a variant of the algorithm allows us to explicitly explore the LCH via a similar binary search scheme. The main idea is to note that D_{ref} is really artificial and that for any I, the resulting shortest path is optimal for *some* D_{ref}. In this way we can explore the tradeoff curve by modifying I (increasing it to move left, decreasing to move right) as inferred from Lemma 3. This eliminates the need for another level of binary search and the algorithm retains the same complexity.

---

[1] As an aside, it can be shown that the path which minimizes the cd product lies on the LCH.

3.7 Experimental Results

We implemented the pseudo-polynomial algorithm and our DRCR algorithm in C and tested on different test cases on a $200MHz$ Sun Ultra-Sparc 1. The main objective of our experiments was to evaluate the computational feasibility of the DRCR algorithm and compare it with the pseudo-polynomial algorithm.

Test graphs were generated by randomly placing some macro blocks inside a rectangular routing region. Candidate buffer locations are then chosen randomly from the area that is not covered by the blocks and two buffers are considered as candidate neighbors (i.e., there is an edge between the corresponding nodes) only if they lie within a threshold distance (e.g., $2000\mu m$). The routing regions vary from $1cm \times 1cm$ to $2cm \times 2cm$. From existing literature, we use the following values for the technology parameters : unit length capacitance $c_0 = 0.15fF/\mu m$; unit length resistance $r_0 = 0.12\Omega/\mu m$; driver resistance $R_s = 270\Omega$; loading capacitance $C_t = 50fF$. We use a single buffer with the following parameters and build the buffer library by scaling these parameters: $r_b = 814\Omega$; $c_b = 28fF$; $d_b = 125ps$. We use the Elmore model (12) to compute the delay of the interconnect in our test cases.

Table III shows the characteristics of the graphs that were used for the experiments. The table lists the number of buffer locations in each graph, the different sizes of buffers used, total number of nodes in the graph and the corresponding number of edges.

Table IV shows the running times for the pseudo-polynomial and DRCR algorithms on all the test graphs. The running time shown for the pseudo-polynomial algorithm is for generating all the solutions on the tradeoff curve while for the DRCR, the running time reported is to

TABLE III

GRAPH CHARACTERISTICS

Graph	#locs	#buffers	#nodes	#edges
G1	300	3	900	264114
G2	700	3	2100	456102
G3	700	2	1400	631224
G4	500	6	3000	2750976

find only one solution on the tradeoff curve. However, it is important to note that to generate even one solution with the pseudo-polynomial algorithm, we have to generate the whole curve, which may not be fast in practice. Hence we see the DRCR algorithm as a fast alternative to the pseudo-polynomial algorithm and the computational experiments show the viability of the approach.

TABLE IV

EXPERIMENTAL RESULTS

Graph	Pseudo-Polynomial (seconds)	DRCR (seconds)
G1	79.55	1.1
G2	256.59	1.95
G3	126.54	3.03
G4	305.00	9.89

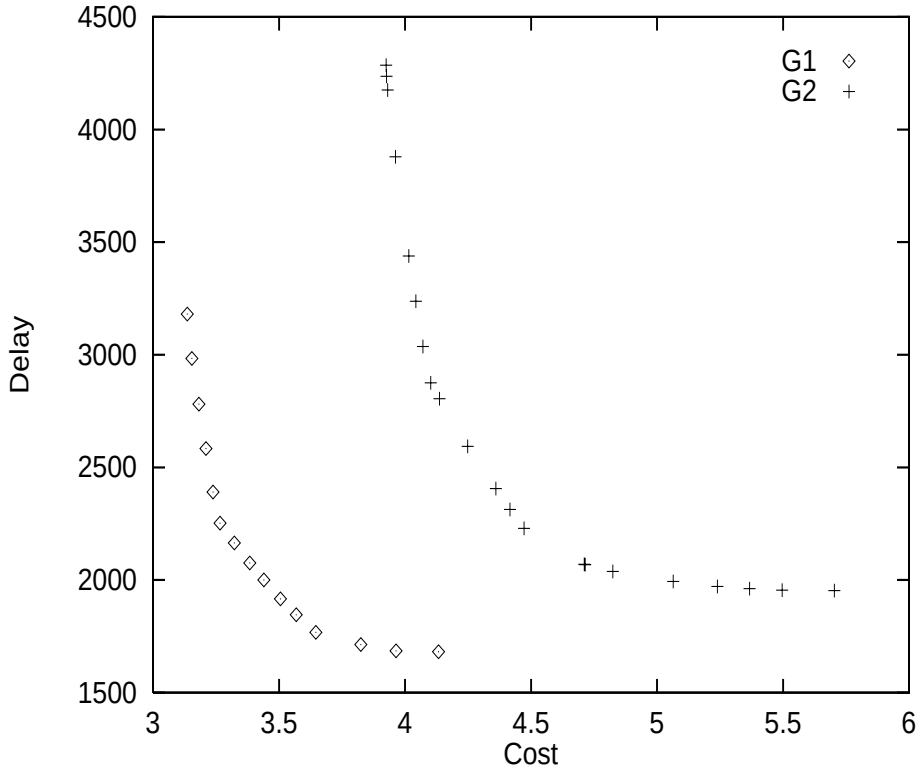

Figure 22: *Examples of LCH of the cost vs. delay trade off curve for two different graphs* G_1 *and* G_2.

Figure Figure 22 shows the plot of the solutions which can be found the DRCR algorithm for two different graphs $G1$ and $G2$. The graph shows the minimum delay solution, the delay of the minimum cost solution, and the solutions that lie on the LCH of the tradeoff curve. Our experiments show that on the average, around 30-40solutions lie on the LCH.

3.8 Comments and Conclusion

In this work, we have studied the problem of buffer insertion in the context of a given layout where there are typically restrictions on the buffer locations.

We have proposed the *Delay Reduction to Cost Ratio Maximization* problem in this context that captures the cost vs. delay tradeoff which is very crucial in practice. We also presented a fast algorithm for solving the problem, which can be considered as an application of the traditional shortest-path algorithm. The experimental results show the practicability of our approach. Hence, we see the DRCR algorithm to be a possible alternative to slow pseudo-polynomial algorithms.

An important note is that we have assumed in this work that the delay of an interconnect between two buffers is independent of the input slew at the source buffer. Though this assumption is not valid in practice, such varying input slews can be taken into account in our graph model at the expense of more vertices – i.e., one vertex for each possible input slew at every buffer input. But using such modeling, it is possible that we capture more realistic and accurate delay modeling.

Though the DRCR problem in this work focuses on the 2-pin nets in a circuit, a possible extension of this work would be to explore if a similar approach is possible for multi-pin nets, which we hope would be of great practical value.

Finally, this algorithm fits problems with largely convex tradeoff (like buffer insertion). Hence we hope that this algorithm has a broad range of practical applications.

CITED LITERATURE

1. Garey, M. R. and Johnson, D. S.: Computers and Intractablility: A Guide to the Theory of NP-Completeness. W.H.Freeman and Company, 1979.

2. Hart, P. E., Nilsson, N. J., and Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transations on Systems, Science and Cybernetics, SCC-4(2):100–107, 1968.

3. A. E. Caldwell, A. B. K. and Markov, I. L.: Optimal Partitioners and End-Case Placers for Standard-Cell Layout. In Proc. of Intl. Symposium on Physical Design, pages 90–96, 1999.

4. Cederbaum, I.: Optimal Backboard Ordering Through The Shortest Path Algorithm. In IEEE Trans. on Circuits and Systems, volume 21, pages 626–634, 1974.

5. E. Auer, W. S. and Sigl, G.: A New Linear Placement Algorithm for Cell Generation. In Proc. of IEEE Intl. Conference on Computer-Aided Design, pages 486–489, 1991.

6. Shirai, Y. and Tsujii, J.: Artificial Intelligence, Concepts, Techniques, and Applications. John Wiley & Sons, 1984.

7. Hur, S.-W. and Lillis, J.: Relaxation and Clustering in a Local Search Framework: Application to Linear Placement. In Technical Report UIC-EECS-99-2, 1999.

8. Hur, S.-W. and Lillis, J.: Hybrid Techniques for Standard-Cell Placement. In To appear in Proc. of International Conference on Computer-Aided Design, Nov, 2000.

9. Chu, C. C. N. and Wong, D. F.: Greedy Wire–Sizing is Linear Time. In Proc. of Intl. Symposium on Physical Design, pages 39–44, 1998.

10. Zhou, H., Wong, D. F., Liu, I.-M., and Aziz, A.: Simultaneous Routing and Buffer Insertion with Restrictions on Buffer Locations. In Proc. of ACM/IEEE Design Automation Conference, pages 96–99, 1999.

11. Joksch, H. C.: The Shortest Route Problem With Constraints. J. Math Anal. Appl., 14:191–197, 1999.

12. Elmore, W. C.: The Transient Response of Damped Linear Network with Particular Regard to Wideband Amplifiers. J. Applied Physics, 19:55–63, 1948.

13. Dijkstra, E. W.: A Discipline of Programming. Prentice-Hall, Englewood Cliffs, N.J., 1976.

14. Lawler, E.: Combinatorial Optimization – Networks and Matroids. Holt, Rinehart and Winston, 1976.

VITA

NAME:	Ashok Jagannathan
EDUCATION:	B.E., Computer Science and Engineering, Regional Engineering College, Trichy, India, 1997.
	M.S., Electrical Engineering and Computer Science, University of Illinois at Chcicago, Illinois, USA, 2000.
TEACHING EXPERIENCE:	Department of EECS, University of Illinois, Chicago, January 2000 – May 2000.
HONORS:	DAC Graduate Scholarship, Design Automation Conference, August 1998 – June 2000.
PROFESSIONAL MEMBERSHIP:	Association for Computing Machinery
PUBLICATIONS:	Ashok Jagannathan, Sung-Woo Hur and John Lillis: A Fast Algorithm for Context-Aware Buffer Insertion, in ACM/IEEE Proc. of Design Automation Conference, June 2000, Los Angeles, CA, pages 368-373.
	Sung-Woo Hur, Ashok Jagannathan and John Lillis: Timing Driven Maze Routing, in ACM Proc.of International Symposium on Physical Design, April 1999, Monterey, CA, pages 208-213.
	Sung-Woo Hur, Ashok Jagannathan and John Lillis: Timing Driven Maze Routing, in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 19, No. 2, pages 234-241, Feb., 2000.