A Hybrid Dynamic/Quadratic Programming Algorithm for Interconnect Tree Optimization

Yu-Yen Mo and Chris Chu
Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, 50011
moy@iastate.edu and cnchu@iastate.edu

ABSTRACT

In this paper, we present an algorithm for delay minimization of interconnect trees by simultaneous buffer insertion/sizing and wire sizing. The algorithm integrates the quadratic programming approach to handle a wire branch [1] into the dynamic programming framework [2]. Our experimental results show that our hybrid dynamic/quadratic programming algorithm is faster, more accurate, and uses much less memory than the pure dynamic programming approach.

1. INTRODUCTION

Interconnect optimization of VLSI circuits has received more and more attention in recent years. This is mainly because of the increasing operation frequency, the shrinking geometry of new technologies, and the increasing area of VLSI circuits. According to the SIA roadmap [3], interconnect delay has become the dominating factor in determining the system performance of the 0.25 μm technology. Even with the help of copper and low dielectric constant (κ) materials, interconnect delay is still likely to dominate the chip performance beyond 0.18 μm technology. Therefore, we can expect the significance of interconnect delay to rapidly increase in near future. Buffer insertion, buffer sizing and wire sizing are effective techniques to reduce interconnect delay. This paper presents an algorithm for delay minimization of interconnect trees by simultaneous buffer insertion, buffer sizing and wire sizing.

Lillis, Cheng and Lin [2] presented a dynamic programming (***DP***) algorithm for the interconnect tree problem. Their algorithm is a generalization of the dynamic programming algorithm for buffer insertion by van Ginneken [4]. The algorithm was later extended to handle power dissipation and incorporate signal slew into the buffer delay model [5]. However, the dynamic programming algorithms in [2] and [5] are slow and memory intensive. The reason is that in order to obtain an accurate solution, the algorithms need to divide the wires into short segments. This results in a large number of wire segments. For each wire segment, the set of all possible solutions for the whole downstream subtree has to be computed and stored. Hence, it takes a lot of time and memory to handle and store the solutions for all segments. To solve this shortcoming, Alpert and Devgan [6] tried to reduce the run time by a wire segmenting technique. The idea is to trade off run time with solution quality by using a coarser wire segmentation. Lai and Wong [7] tried to reduce the memory usage by a recomputation technique. Their idea is to trade memory with run time by recomputing instead of storing values.

The algorithm presented in this paper is accurate, fast and economical in memory usage. The algorithm combines the dynamic programming framework in [2] and the quadratic programming (***QP***) approach for interconnect optimization of a wire by Chu and Wong [1]. As shown in [1], the problem of simultaneous buffer insertion and wire sizing for a wire can be formulated as a convex quadratic program, and the convex quadratic program can be solved extremely efficiently by the active set method. In this paper, we show that instead of being divided into numerous segments, each wire branch can be handled as a whole by a similar approach in [1]. Therefore, the set of possible solutions of a wire branch can be found in a much shorter time, and only one set per wire needs to be stored. To handle the tree structure (i.e., to combine the sets of solutions of adjacent wires together), dynamic programming is used. We use the name *Dynamic/Quadratic Programming (**DQP**)* to refer to our hybrid algorithm.

In addition, we present a constant reusing technique to speedup the solving of quadratic programs. To process the edges of the tree, a large number of quadratic programs need to be solved. These quadratic programs are of the same form, except for the difference in some parameters such as the downstream capacitance and the wire length. We show that many constant values computed in one quadratic program can be stored and reused by other quadratic programs. This technique increases the memory usage just moderately but reduces the run time dramatically.

In this paper, the Elmore delay model [8] is used for delay calculations. A wire segment is modeled as a π-type model as shown in Figure 1.

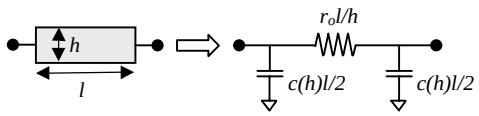

Figure 1. *The model of a wire segment of length l and width h by a π-type RC circuit. r_0 is the unit wire resistance. c(h) is the wire capacitance per unit length for a segment of width h. In this paper, we assume that c(h) is an increasing function.*

A buffer is modeled as a switch-level RC circuit as shown in Figure 2.

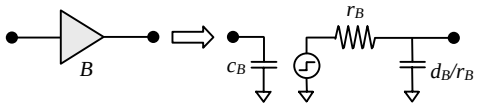

Figure 2. *The model of a buffer of size B by a switch-level RC circuit. c_B, r_B, and d_B are the input capacitance, output resistance and the intrinsic delay.*

The remainder of this paper is organized as follows. In Section 2, we present the basic idea of the dynamic programming approach in [2]. In Section 3, we present the quadratic programming formulation for a wire in [1]. In Section 4, we present our hybrid dynamic/quadratic programming algorithm. We explain the modification needed in order to integrate the ***QP*** formulation into the ***DP*** framework. We also present the active set method and the constant reusing technique to solve the quadratic programs. In Section 5, we present the experimental results. We then conclude the paper in Section 6.

2. DP APPROACH

In this section, we outline the dynamic programming (***DP***) technique for interconnect tree optimization in [2] and [5]. The algorithms adapt a bottom-up dynamic programming approach to minimize the maximum delay among all paths from source to sinks.

The basic idea of the dynamic programming technique is to build a new set of solutions for each segment based on the solution sets of its subtrees by traversing the tree structure in a bottom-up fashion. In their algorithms, instead of computing a single solution for each subtree, a set of solutions where each member contains a downstream capacitance and delay time pair (c, t) is computed and kept. The reason for doing so is that the optimal (c, t) combination for delay minimization cannot be determined without the upstream resistance.

If there are two downstream branches for a node, each downstream branch will have a set of (c, t) pairs. These two sets can be combined into a single set and pruned according to the pair values. The pruned list will have c in increasing order and t in decreasing order in their algorithm. If there are more than two downstream branches for a certain node, this node can be broken into several two downstream branch nodes with zero length in between.

The main drawback of the pure dynamic programming is that in order to obtain a solution with good quality, each wire has to be divided into many small segments. This increases run time and memory usage drastically.

3. QP APPROACH

In this section, we outline the quadratic programming (***QP***) formulation of interconnect optimization for a single wire in [1].

We illustrate the idea by first considering wire sizing alone. The extension to handle buffers is easy and will be presented afterwards. [1] showed that the optimal wire shape can be described by a non-increasing function. Therefore, the wire sizing (*WS*) problem can be formulated as in Figure 3. Given wire length L, driver resistance R_D, load capacitance C_L, a set $H=\{h_1,...,h_n\}$ of n choices of wire width such that $h_1>...>h_n$, *WS* is to determine the segment lengths $l_1,...,l_n$ such that the delay from source to sink is minimized. Note that this approach does not divided the wire into numerous segments. The number of segments is equal to the number of choices of segment width n, which is usually a small number.

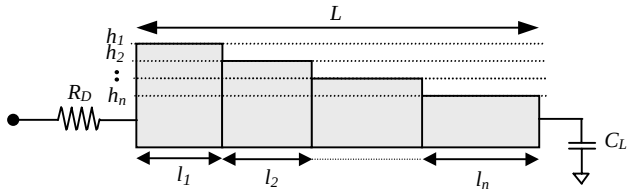

Figure 3. *The wire sizing (WS) problem for a single interconnect wire.*

The problem *WS* can be formulated as follows. Let $c_i = c(h_i)$ for $1 \le i \le n$. The Elmore delay D for this wire is

$$\begin{aligned} D &= R_D\left(c_1 l_1 + c_2 l_2 + \cdots + c_n l_n + C_L\right) \\ &+ \frac{r_0 l_1}{h_1}\left(\frac{c_1 l_1}{2} + c_2 l_2 + \cdots + c_n l_n + C_L\right) \\ &+ \frac{r_0 l_2}{h_2}\left(\frac{c_2 l_2}{2} + c_3 l_3 + \cdots + c_n l_n + C_L\right) \\ &\vdots \\ &+ \frac{r_0 l_n}{h_n}\left(\frac{c_n l_n}{2} + C_L\right) \\ &= \frac{1}{2}\boldsymbol{l}^T \Phi \boldsymbol{l} + \rho^T \boldsymbol{l} + R_D C_L \end{aligned}$$

where

$$\Phi = \begin{pmatrix} c_1 r_0/h_1 & c_2 r_0/h_1 & c_3 r_0/h_1 & \cdots & c_n r_0/h_1 \\ c_2 r_0/h_1 & c_2 r_0/h_2 & c_3 r_0/h_2 & \cdots & c_n r_0/h_2 \\ c_3 r_0/h_1 & c_3 r_0/h_2 & c_3 r_0/h_3 & \cdots & c_n r_0/h_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_n r_0/h_1 & c_n r_0/h_2 & c_n r_0/h_3 & \cdots & c_n r_0/h_n \end{pmatrix},$$

$$\rho = \begin{pmatrix} R_D c_1 + C_L r_0/h_1 \\ R_D c_2 + C_L r_0/h_2 \\ R_D c_3 + C_L r_0/h_3 \\ \vdots \\ R_D c_n + C_L r_0/h_n \end{pmatrix}, \text{ and } \boldsymbol{l} = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \\ \vdots \\ l_n \end{pmatrix}$$

Therefore, *WS* can be written as the following quadratic program:

$$\begin{array}{lll} \textbf{CQP}: & \text{Minimize} & 1/2\boldsymbol{l}^T\boldsymbol{\Phi}\boldsymbol{l} + \boldsymbol{\rho}^T\boldsymbol{l} \\ & \text{Subject to} & l_1 + \ldots + l_n = L \\ & & l_i \ge 0 \text{ for } 1 \le i \le n \end{array} \quad (1)$$

It was proved in [1] that the Hessian matrix $\boldsymbol{\Phi}$ of ***CQP*** is positive definite. Hence ***CQP*** is convex and polynomial-time solvable. Furthermore, [1] also proved that $\boldsymbol{\Phi}^{-1}$ is tri-diagonal. In general, convex quadratic programs can be solved efficiently by a classical technique called active set method [9]. By making use of the property that $\boldsymbol{\Phi}^{-1}$ is tri-diagonal, each iteration of active set method only takes O(n) time. As a result, [1] presented an optimal algorithm to solve ***CQP*** which runs in O(n^2) time in practice. Since n is usually a small number, the algorithm is extremely efficient in practice.

The extension of simultaneous buffer insertion and wire sizing, as shown in [1], is straightforward. The corresponding quadratic program has exactly the same form as ***CQP*** above. The corresponding Hessian matrix is a block diagonal matrix such that each block is just the matrix $\boldsymbol{\Phi}$ for *WS* above. Hence it is positive definite and has a tri-diagonal inverse. Therefore, the corresponding quadratic program can also be solved optimally by

an active set method based O(mn^2) time algorithm, where m is the number of buffers inserted.

Notice that the driver resistance R_D is assumed to be known in the ***QP*** formulation above. Therefore, it cannot be integrated into the ***DP*** framework directly since the upstream resistance is not known during the bottom-up dynamic programming traversal. The modification needed is presented in the Section 4.2.

4. HYBRID DYNAMIC / QUADRATIC PROGRAMMING (*DQP*) ALGORITHM

In this section, we introduce our hybrid dynamic/quadratic programming (***DQP***) algorithm. We integrate the ***QP*** approach into the ***DP*** framework in order to reduce run time and memory usage. Instead of being divided into numerous small segments, each wire in the interconnect tree is handled as a whole by the ***QP*** approach. In Section 4.1, we first present a modified ***QP*** formulation so that it can be integrated into the ***DP*** formulation. In Section 4.2, we present the active set method to solve the ***QP*** problem in Section 4.1. In Section 4.3, we present the modified dynamic programming framework and the ***DQP*** algorithm. In Section 4.4, we introduce the constant reusing technique to speedup the solving of quadratic programs.

4.1 Modified Convex Quadratic Program (*MCQP*)

In this subsection, we modify the quadratic program ***CQP*** in Section 3 so that it can be integrated into the ***DP*** framework. We call the resulting quadratic program the modified convex quadratic program (***MCQP***). It is a building block of the ***DQP*** algorithm to handle a single wire branch.

The main difference between ***MCQP*** and ***CQP*** is that we ignore the driver resistance R_D in ***CQP*** and we include the upstream capacitance c_U into our formulation. First, consider the following new wire sizing problem (*WS'*) for a wire branch as shown in Figure 4. The wire length L, load capacitance c_D, and the set $H=\{h_1,...,h_n\}$ of wire width choices are given as before. In addition, the delay time at the downstream node t_D (i.e., the delay time of the subtree at this node) and the capacitance seen from the upstream node of the branch c_U are also given. The objective is to minimize the upstream delay time t_U by changing the segment lengths $l_1,...,l_n$. In other words, given a list of (c_D, t_D) pairs at the downstream node, ***MCQP*** can be used to find a list of (c_U, t_U) pairs at the upstream node. This is similar to what the ***DP*** approach does.

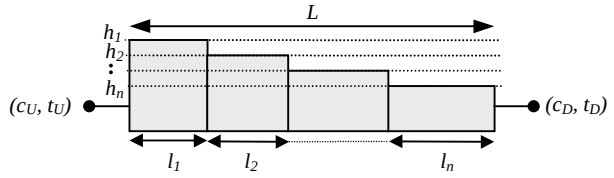

Figure 4. *The new wire sizing problem (WS') for a single wire branch in an interconnect tree.*

Consider a particular (c_D, t_D) and c_U combination. Let $c_i = c(h_i)$ for $1 \le i \le n$. The delay t_U for this wire branch is

$$t_U = \frac{r_0 l_1}{h_1}\left(\frac{c_1 l_1}{2} + c_2 l_2 + \cdots + c_n l_n + c_D\right)$$
$$+ \frac{r_0 l_2}{h_2}\left(\frac{c_2 l_2}{2} + c_3 l_3 + \cdots + c_n l_n + c_D\right)$$
$$\vdots$$
$$+ \frac{r_0 l_n}{h_n}\left(\frac{c_n l_n}{2} + c_D\right) + t_D$$
$$= \frac{1}{2}\boldsymbol{l}^T \Phi \boldsymbol{l} + \rho^T \boldsymbol{l} + t_D$$

where

$$\Phi = \begin{pmatrix} c_1 r_0/h_1 & c_2 r_0/h_1 & c_3 r_0/h_1 & \cdots & c_n r_0/h_1 \\ c_2 r_0/h_1 & c_2 r_0/h_2 & c_3 r_0/h_2 & \cdots & c_n r_0/h_2 \\ c_3 r_0/h_1 & c_3 r_0/h_2 & c_3 r_0/h_3 & \cdots & c_n r_0/h_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_n r_0/h_1 & c_n r_0/h_2 & c_n r_0/h_3 & \cdots & c_n r_0/h_n \end{pmatrix},$$
$$\rho = \begin{pmatrix} c_D r_0/h_1 \\ c_D r_0/h_2 \\ c_D r_0/h_3 \\ \vdots \\ c_D r_0/h_n \end{pmatrix}, \text{ and } \boldsymbol{l} = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \\ \vdots \\ l_{n1} \end{pmatrix}$$

Therefore, *WS'* can be formulated as the following modified convex quadratic program.

$$\begin{array}{lll} \textbf{MCQP}: & \text{Minimize} & t_U = 1/2\boldsymbol{l}^T\boldsymbol{\Phi}\boldsymbol{l} + \boldsymbol{\rho}^T\boldsymbol{l} + t_D \\ & \text{Subject to} & l_1 + \ldots + l_n = L \\ & & c_1 l_1 + \ldots + c_n l_n + c_D = c_U \\ & & l_i \ge 0 \text{ for } 1 \le i \le n \end{array} \quad (2)$$

Notice that the matrix $\boldsymbol{\Phi}$ here is the same as the one in ***CQP***. Hence it is positive definite and has a tri-diagonal inverse.

As shown in the original ***QP*** approach in [1], the above formulation can be extended easily to handle simultaneous buffer insertion and wire sizing. For fixed (c_D, t_D) and c_U values, each combination of number of buffers and buffer sizes corresponds to one instance of ***MCQP***. However, if buffers of different sizes are considered, many instances of ***MCQP*** need to be solved. Suppose there are q different choices of buffer size in the buffer library and m buffers are inserted. Then there are q^m choices of buffer sizes, and hence q^m instances of ***MCQP*** to solve. The algorithm will be slow if m is large.

In order to guarantee the number of buffers inserted in each wire to be small, we divide each long wire into several wires shorter than a critical length. The critical length is defined as the maximum length such that at most one buffer is needed in the optimal solution [10]. This length depends on the technology parameters and the bounds on wire width and buffer size and can be determined experimentally. Using this idea, for fixed (c_D, t_D) and c_U values, only $1+q$ instances (one instance for no buffer and one instance for each of the q buffer sizes) need to be considered. The simultaneous buffer insertion and wire sizing problem for one buffer of size B is shown in Figure 5.

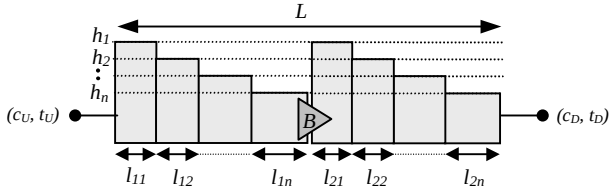

Figure 5. *The simultaneous buffer insertion and wire sizing problem.*

4.2 Extended Active Set Method (EASM)

In this subsection, we derive a very efficient algorithm to solve ***MCQP*** based on the idea of active set method.

Active set method is a popular technique for solving quadratic programming problems. It has been proved to be efficient in practice. The idea underlying the active set method for solving a general convex quadratic program is to partition the inequality constraints into two groups: active and inactive. In each iteration, the active inequality constraints are treated as equality constraints and the inactive constraints are essentially ignored. Then, the resulting equality constrained program is solved. If the solution is infeasible with respect to the original program, some inactive constraints will be added to the set of constraints. If the solution is feasible but not optimal (i.e., some Lagrangian multipliers are negative for the minimization problem), some constraints will be removed from the current active set. The process is repeated until the optimal solution is found. We give a brief outline on how to solve ***QP*** by active set method in the following. Readers are encouraged to read [9, Chapter 11] for more details of active set method.

If a convex quadratic program consists of equality constraints only, it is particularly easy to solve. Consider the following program:

$$\begin{aligned} &\text{Minimize} && 1/2\boldsymbol{l}^T\boldsymbol{\Phi}\boldsymbol{l} + \boldsymbol{\rho}^T\boldsymbol{l} \\ &\text{Subject to} && \boldsymbol{\Gamma}\boldsymbol{l} = \boldsymbol{b} \end{aligned} \quad (3)$$

Where $\boldsymbol{\Phi}$ is positive definite and $\boldsymbol{\Gamma}$ is of full rank. Consider the associated Lagrangian:

$$\mathcal{L}(\boldsymbol{l}, \boldsymbol{\lambda}) = 1/2\boldsymbol{l}^T\boldsymbol{\Phi}\boldsymbol{l} + \boldsymbol{\rho}^T\boldsymbol{l} + \boldsymbol{\lambda}^T(\boldsymbol{\Gamma}\boldsymbol{l} - \boldsymbol{b}) \quad (4)$$

The Lagrange necessary conditions of optimality are $\partial l(\boldsymbol{l}, \boldsymbol{\lambda})/\partial l_i = 0$ and $\partial l(\boldsymbol{l}, \boldsymbol{\lambda})/\partial \lambda_i = 0$ for all i. The conditions can be written in matrix form as follows:

$$\begin{aligned} \boldsymbol{\Phi}\boldsymbol{l} + \boldsymbol{\Gamma}^T\boldsymbol{\lambda} + \boldsymbol{\rho} &= 0 \\ \boldsymbol{\Gamma}\boldsymbol{l} - \boldsymbol{b} &= 0 \end{aligned} \quad (5)$$

Since $\boldsymbol{\Phi}$ is positive definite and $\boldsymbol{\Gamma}$ is of full rank, it can be shown that the conditions can be uniquely solved:

$$\begin{aligned} \boldsymbol{\lambda} &= -(\boldsymbol{\Gamma}\boldsymbol{\Phi}^{-1}\boldsymbol{\Gamma}^T)^{-1}(\boldsymbol{\Gamma}\boldsymbol{\Phi}^{-1}\boldsymbol{\rho} + \boldsymbol{b}) \\ \boldsymbol{l} &= -\boldsymbol{\Phi}^{-1}\boldsymbol{\Gamma}^T\boldsymbol{\lambda} - \boldsymbol{\Phi}^{-1}\boldsymbol{\rho} \end{aligned} \quad (6)$$

CQP and ***MCQP*** consist of inequality constraints as well. It has been shown in [1] that inequality constraints in ***CQP*** can be handled efficiently by active set method. The same idea can be applied to ***MCQP***. The ***MCQP*** problem also has an extra equality constraint on c_U. The major difference between solving ***MCQP*** and ***CQP*** is at least two wire segments need to be inactive at any time in the active set method. This is because we have two equality constraints (i.e., the total wire length constraint and the upstream capacitance constraint) that need to be satisfied. To ensure the feasibility of the solution, we need to start the active set method with a feasible initial solution for vector $\boldsymbol{l}$. Then, we iteratively calculate a new solution $\boldsymbol{l}'$ which will be the new direction to move $\boldsymbol{l}$. The idea is to move $\boldsymbol{l}$ stepwise toward the optimal solution and make sure that each step stays within the feasible region. The algorithm ***EASM*** which we used to solve ***MCQP*** is summarized in Figure 6.

Algorithm *EASM* (Extended Active Set Method)
1. Find an initial feasible solution for $\boldsymbol{l}$.
2. Set the active set $\boldsymbol{A} = \varnothing$.
3. Solve for $\boldsymbol{l}'$ with respect to $\boldsymbol{A}$ as in ***CQP***.
4. Calculate $\boldsymbol{d} = \boldsymbol{l}' - \boldsymbol{l}$.
5. If ($\boldsymbol{l}'$ is not feasible)
6. Calculate α_k according to $\boldsymbol{d}$ and $\boldsymbol{l}$ (where α_k is the step size which is selected to be as large as possible to maintain feasibility).
7. Calculate new $\boldsymbol{l} = \boldsymbol{l} + \boldsymbol{d}\,\alpha_k$.
8. Add the l element which gives α_k to $\boldsymbol{A}$. Go back to step 3.
9. Else if ($\boldsymbol{l}'$ is feasible)
10. If ($\boldsymbol{d} \neq 0$)
11. Update $\boldsymbol{l} = \boldsymbol{l}'$ and go back to step 3.
12. Else if ($\boldsymbol{d} = 0$) /* local minimal reached */
13. Solve for $\boldsymbol{\lambda}$.
14. If ($\boldsymbol{\lambda} < 0$) /* check for optimality */
15. Drop all the l which correspond to negative $\boldsymbol{\lambda}$ from $\boldsymbol{A}$.
16. Go back to step 3.
17. Else if ($\boldsymbol{\lambda} \geq 0$)
18. Optimal $\boldsymbol{l}$ obtained.

Figure 6. *The Algorithm **EASM** (Extended Active Set Method).*

4.3 The DQP Algorithm

The ***DP*** approach discussed in Section 2 is the basic framework of our ***DQP*** algorithm. The ***DP*** idea is used to handle the tree structure of interconnects (i.e. , to combine the set of solutions of adjacent subtrees together). We do not divide the wires into segments and handle the segments by dynamic programming. Each wire branch is handled as a whole by the ***QP*** approach.

However, since we do not know the upstream resistance at a node during the bottom-up traversal of the dynamic programming, we need to consider many different c_U values and calculate the optimal delay t_U corresponding to each c_U value. In general, except for the leaf nodes (the nodes which connect to sinks), each wire branch can have more than one (c_D, t_D) pair at the downstream node and a set of c_U at the upstream node. Each combination of c_U and (c_D, t_D) forms a ***MCQP*** problem instance.

Let N_c be a user-defined parameter specifying the number of different c_U values used at each node. Let q be the number of choices of buffer size. For each wire branch, there are $1+q$ cases to consider (one case for no buffer and one case for each of the q buffer sizes). For each case, the set of c_U is chosen by first determining an upper bound and a lower bound on the upstream capacitance. Then $N_c/(q+1)$ discrete values are selected evenly from the range.

After the root of the interconnect tree is reached, the interconnect optimization solution can be constructed by a top-down traversal. With the knowledge of the driver resistance, we can select the best (c, t) pair at the root which results in minimum delay time. The rest of the solution can be obtained by recursively traversing the tree in a top-down manner and selecting the best (c, t) pair at each node.

The ***DQP*** algorithm is summarized in Figure 7. In Section 4.4, we introduce a constant reusing technique to speed up the solving of ***MCQP*** instances in Step 3.

4.4 Constant Reusing Technique

In this subsection, we present a constant reusing technique to speed up the ***DQP*** algorithm.

EASM can be applied directly in ***DQP*** to solve the ***MCQP*** instances for all wires. Each ***MCQP*** instance can be solved very

Algorithm *DQP* (Hybrid Dynamic/Quadratic Programming)

1. Start from a leaf node. /* the node which connect to a sink */
2. Append a branch of wire to the upstream of this node.
3. Apply ***EASM*** to calculate an upstream delay time t_U for each selected upstream capacitance c_U according to all available downstream delay time t_D, and capacitance c_D.
4. If (current node is the root)
5. Search for the smallest delay according to the driver resistance and (c, t) pair set at the root.
6. Construct the solution top-down and then exit the program.
7. Else if (number of children of a node = 2)
8. Merge the two (c, t) lists and prune redundant (c, t) pairs.
9. Go back to step 2.
10. Else if (number of children of a node = 1)
11. (c_D, t_D) set of the current node = (c_U, t_U) set of the child of the current node.
12. Go back to step 2.

Figure 7. *The Hybrid Dynamic/Quadratic Programming Algorithm.*

efficiently by ***EASM***. However, for each wire branch, there are many (c_D, t_D) pairs at the downstream node, many c_U values at the upstream node, and several choices of buffer size. Therefore, there can be a lot of ***MCQP*** instances. If those ***MCQP*** instances are solved independently by ***EASM***, experimental results show that the ***DQP*** approach only has very limited improvement over the pure ***DP*** approach. However, we recognize that all ***MCQP*** instances are the same except that the parameters L, c_U, c_D and t_D are different. Many constant values computed in one ***MCQP*** instance can be stored and reused by other ***MCQP*** instances.

By observing the equations (2) and (6), we recognize that $\boldsymbol{l}$, $\boldsymbol{\lambda}$ and t_U can be expressed by three separate linear functions in terms of downstream capacitance c_D, total length of the wire branch L, upstream capacitance c_U, and downstream delay time t_D as shown in (7).

$$\boldsymbol{\lambda} = \boldsymbol{v}_{\lambda 1} c_D + \boldsymbol{v}_{\lambda 2} L + \boldsymbol{v}_{\lambda 3} c_U, \qquad \boldsymbol{l} = \boldsymbol{v}_{l1} c_D + \boldsymbol{v}_{l2} L + \boldsymbol{v}_{l3} c_U, \qquad (7)$$

$$\text{and} \quad t_U = v_{t1} c_U^2 + v_{t2} L^2 + v_{t3} c_D^2 + v_{t4} c_U L + v_{t5} L c_D + v_{t6} c_D c_U + t_D$$

All v_{ti}'s, $\boldsymbol{v}_{\lambda i}$'s, and $\boldsymbol{v}_{ti}$'s in (7) are independent of L, c_U, c_D and t_D. Their values only depend on the electrical parameters (e.g., r_0, c_0, and c_f) and the set of wire widths being used. In the active set method, some segments are set to inactive at each iteration. Hence the set of wire widths being used may only be a subset of H. So when a particular set of wire widths is first used in some iteration of the active set method, the constants v_{ti}'s, $\boldsymbol{v}_{\lambda i}$'s, and $\boldsymbol{v}_{ti}$'s can be computed and stored. When the same set of wire widths is used again in another iteration, the constant stored can be reused. This constant reusing technique only increases the memory usage moderately but reduces the run time dramatically. Please refer to Appendix for the equations to calculate constants v_{ti}'s, $\boldsymbol{v}_{\lambda i}$'s, and $\boldsymbol{v}_{ti}$'s.

5. EXPERIMENTAL RESULTS

The ***DQP*** algorithm is implemented as a C program. We test the program on a PC with a 500 MHz Pentium III processor and 256 MB of memory. We use the parameters for the 0.18 µm technology listed in [11]. The results are compared with the pure ***DP*** approach. We use 6 trees with 2 to 100 sinks. The length of the tree wires range from 1000 µm to 15000 µm.

Figure 8 shows the delay time of the solutions obtained by ***DQP*** verses the number of upstream capacitance choices N_c on the input tree with 10 sinks. The results in Figure 8 show that N_c does not need to be selected as a large value to obtain a good solution quality. Figure 9 shows a similar chart for the pure ***DP*** with the same input tree. The l_s is the segment length in µm that we used in ***DP*** to divide a wire branch into equal-length segments.

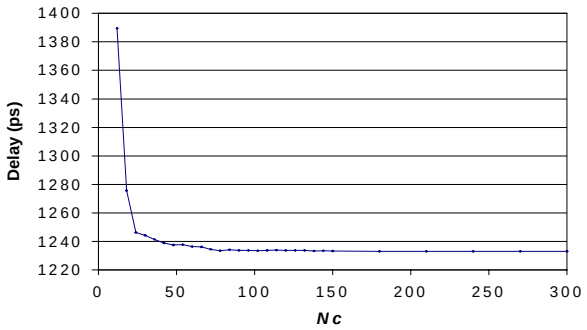

Figure 8. *Delay vs. N_c for* ***DQP***

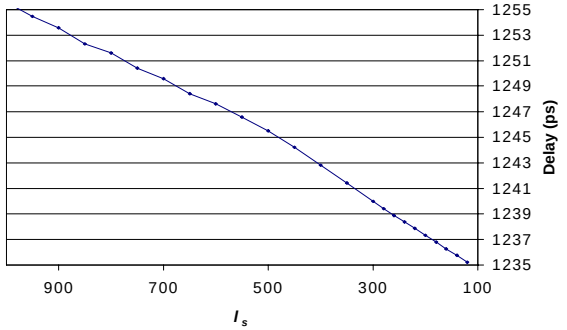

Figure 9. *Delay vs. l_s for* ***DP***

For the purpose of comparison, we set the parameters l_s and N_c such that the quality of solution for ***DP*** and ***DQP*** are similar. We set l_s = 200 and N_c = 90 in our experiment. The experimental results are shown in Table 1.

# of Sinks	Optimal Delay (ps)	***DP*** (l_s = 200)			***DQP*** (N_c = 90)		
		Run Time (sec.)	Memory Used (Mb)	Delay (ps)	Run Time (sec.)	Memory Used (Mb)	Delay (ps)
2	293.72	1.77	6.17	294.70	0.27	0.00+0.9	295.59
5	666.60	6.44	22.12	668.83	0.76	0.01+0.9	668.53
10	1232.95	12.66	42.93	1237.34	1.41	0.02+0.9	1233.80
15	1430.64	20.82	70.35	1435.33	1.95	0.03+0.9	1431.50
30	1937.31	38.20	129.95	1939.95	4.09	0.05+0.9	1937.86
100	2256.82	N/A	N/A	N/A	13.28	0.17+0.9	2265.29

Table 1. *The comparison between* ***DP*** *and* ***DQP***.

The estimated optimal delay value is obtained by setting N_c to 1800 in ***DQP***. In our experiment, ***DP*** with l_s = 200 cannot be run successfully for the input file containing 100 sink nodes on the PC we used. We estimate the reason is because of memory limitations of the PC. The memory data column of ***DQP*** contains two parts. The first part is the memory dynamically allocated for (c, t) pairs. The second part is the memory used to store the constants being reused.

Our experimental results suggest that ***DQP*** is better than ***DP*** in run time, memory, and accuracy (except for the delay of the smallest input tree). For small problems and for the same quality of solution, the run time advantage of ***DQP*** over ***DP*** is relatively

less. This is because of the overhead and the lower constant reusing rate of the ***EASM*** algorithm. However, for bigger problems, the run time advantage of ***DQP*** over ***DP*** is more significant.

We also observe that the constant reusing technique can enhance the efficiency of our algorithm significantly. By applying the constant reusing technique, the run time of our program can be reduced by around 75% compared to the implementation without constant reusing technique.

6. DISCUSSION

In this paper, only the timing optimization of an interconnect tree is presented. However, interconnect power and area optimization can be easily applied to the same algorithm framework.

In general, a better circuit can be obtained by applying interconnect optimization in a larger scale at the circuit level. This research can only be applied to an interconnect tree. For future research, it will be interesting to extend the work to consider simultaneous wire sizing, buffer insertion, buffer sizing, and gate sizing to a whole circuit.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for helpful comments on the manuscript.

REFERENCES

[1] C. C. N. Chu and D. F. Wong. "A Quadratic Programming Approach to Simultaneous Buffer Insertion/Sizing and Wire Sizing." *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 6, pages 787-798, June 1999.

[2] J. Lillis, C.-K. Cheng, and T.-T. Lin. "Optimal and Efficient Buffer Insertion and Wire Sizing." *Proc. Of Custom Integrated Circuit Conf.*, pages 259-262, 1995.

[3] Semiconductor Industry Association. *The International Technology Roadmap for Semiconductors*. 1999.

[4] L. P. P. P. van Ginneken. "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay." *Proc. Intl. Symp. on Circuits and Systems*, pages 865-868, 1990.

[5] J. Lillis, C.-K. Cheng, and T.-T. Lin. "Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model." *IEEE J. Solid-State Circuits*, 31(3):437-447, March, 1996.

[6] C. Alpert and A. Devgan. "Wire Segmenting for Improved Buffer Insertion." *Proc. ACM/IEEE Design Automation Conf.*, pages 588-593, 1997.

[7] M. Lai and D. F. Wong, "A Memory-Efficient Implementation of Dynamic Programming for Interconnect Optimization." manuscript.

[8] W. C. Elmore, "The Transient Response of Damped Linear Network with Particular Regard to Wideband Amplifiers." *J. Applied Physics*, 19:55-63, 1948.

[9] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison Wesley, second edition. 1984.

[10] J. Cong and Z. Pan, "Interconnect Delay Estimation Models for Synthesis and Design Planning." In *Proc. Asia and South Pacific Design Automation Conf.*, pages 97-100, Jan. 1999.

[11] J. Cong, L. He, K.-Y Khoo, C.-K. Koh, and Z. Pan. "Interconnect Design for Deep Submicron ICs." *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pages 478-485, 1997.

APPENDIX

Equations for Constant Reusing Technique

The wire sizing problem is used below to illustrate the idea. Extension to include buffer insertion is easy. Consider the following convex quadratic program formulation corresponding to an active set $\boldsymbol{A}$ for the active set method.

$$\begin{aligned} &\text{Minimize} && 1/2\,\boldsymbol{l}_A{}^T \boldsymbol{\Phi}_A \boldsymbol{l}_A + \boldsymbol{\rho}_A{}^T \boldsymbol{l}_A \\ &\text{Subject to} && \boldsymbol{\Gamma}_A \boldsymbol{l}_A = \boldsymbol{b} \end{aligned} \qquad \text{(a1)}$$

where $\Gamma_A = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ c_1 & c_2 & \cdots & c_n \end{pmatrix}$, $\boldsymbol{l}_A = \begin{pmatrix} l_1 \\ l_2 \\ \vdots \\ l_n \end{pmatrix}$, and $\boldsymbol{b} = \begin{pmatrix} L \\ c_U - c_D \end{pmatrix}$

From (6) and (a1), we can derive the following equations.

$$\lambda_A = \begin{pmatrix} D-A \\ F-B \end{pmatrix} c_D - \begin{pmatrix} C \\ E \end{pmatrix} L - \begin{pmatrix} D \\ F \end{pmatrix} c_U \qquad \text{(a2)}$$

where $\begin{pmatrix} C & D \\ E & F \end{pmatrix} = \left(\Gamma_A \Phi_A{}^{-1} \Gamma_A\right)^{-1}$,

and $\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} C\sum_{j=1}^{n}\sum_{i=j-1}^{j+1}\left(\theta_{ij} r_0/h_j\right) + D\sum_{j=1}^{n}\sum_{i=j-1}^{j+1}\left(c_i \theta_{ij} r_0/h_j\right) \\ E\sum_{j=1}^{n}\sum_{i=j-1}^{j+1}\left(\theta_{ij} r_0/h_j\right) + F\sum_{j=1}^{n}\sum_{i=j-1}^{j+1}\left(c_i \theta_{ij} r_0/h_j\right) \end{pmatrix}$

θ_{ij} is the element in $\Phi_A{}^{-1}$.

$$\boldsymbol{l}_A = \boldsymbol{\alpha}\, c_D + \boldsymbol{\beta}\, L + \boldsymbol{\gamma}\, c_U \qquad \text{(a3)}$$

where

$$\alpha = \begin{pmatrix} -(D-A)\sum_{j=1}^{2}\left(\theta_{1j}\right) - (F-B)\sum_{j=1}^{2}\left(\theta_{1j} c_j\right) - \sum_{j=1}^{2}\left(\theta_{1j}\, r_0/h_j\right) \\ -(D-A)\sum_{j=1}^{3}\left(\theta_{2j}\right) - (F-B)\sum_{j=1}^{3}\left(\theta_{2j} c_j\right) - \sum_{j=1}^{3}\left(\theta_{2j}\, r_0/h_j\right) \\ -(D-A)\sum_{j=2}^{4}\left(\theta_{3j}\right) - (F-B)\sum_{j=2}^{4}\left(\theta_{3j} c_j\right) - \sum_{j=2}^{4}\left(\theta_{3j}\, r_0/h_j\right) \\ \vdots \\ -(D-A)\sum_{j=n-1}^{n}\left(\theta_{nj}\right) - (F-B)\sum_{j=n-1}^{n}\left(n_{nj} c_j\right) - \sum_{j=n-1}^{n}\left(\theta_{nj}\, r_0/h_j\right) \end{pmatrix}$$

$$\beta = \begin{pmatrix} C\sum_{j=1}^{2}\left(\theta_{1j}\right) + E\sum_{j=1}^{2}\left(\theta_{1j} c_j\right) \\ C\sum_{j=1}^{3}\left(\theta_{2j}\right) + E\sum_{j=1}^{3}\left(\theta_{2j} c_j\right) \\ C\sum_{j=2}^{4}\left(\theta_{3j}\right) + E\sum_{j=2}^{4}\left(\theta_{3j} c_j\right) \\ \vdots \\ C\sum_{j=n-1}^{n}\left(\theta_{nj}\right) + E\sum_{j=n-1}^{n}\left(\theta_{nj} c_j\right) \end{pmatrix}, \text{ and } \gamma = \begin{pmatrix} D\sum_{j=1}^{2}\left(\theta_{1j}\right) + F\sum_{j=1}^{2}\left(\theta_{1j} c_j\right) \\ D\sum_{j=1}^{3}\left(\theta_{2j}\right) + F\sum_{j=1}^{3}\left(\theta_{2j} c_j\right) \\ D\sum_{j=2}^{4}\left(\theta_{3j}\right) + F\sum_{j=2}^{4}\left(\theta_{3j} c_j\right) \\ \vdots \\ D\sum_{j=n-1}^{n}\left(\theta_{nj}\right) + F\sum_{j=n-1}^{n}\left(\theta_{nj} c_j\right) \end{pmatrix}$$

$$\begin{aligned} t_U &= 1/2\, \boldsymbol{l}_A{}^T \boldsymbol{\Phi}_A \boldsymbol{l}_A + c_D\, \boldsymbol{\sigma}^T \boldsymbol{l}_A + t_D \\ &= (1/2\, \boldsymbol{\alpha}^T \boldsymbol{\Phi}_A \boldsymbol{\alpha} + \boldsymbol{\sigma}^T \boldsymbol{\alpha})\, c_D{}^2 + (1/2\, \boldsymbol{\beta}^T \boldsymbol{\Phi}_A \boldsymbol{\beta})\, L^2 + (1/2\, \boldsymbol{\gamma}^T \boldsymbol{\Phi}_A \boldsymbol{\gamma})\, c_U{}^2 \\ &\quad + (\boldsymbol{\alpha}^T \boldsymbol{\Phi}_A \boldsymbol{\gamma} + \boldsymbol{\sigma}^T \boldsymbol{\gamma})\, c_U\, c_D + (\boldsymbol{\alpha}^T \boldsymbol{\Phi}_A \boldsymbol{\beta} + \boldsymbol{\sigma}^T \boldsymbol{\beta})\, c_D\, L \\ &\quad + (\boldsymbol{\beta}^T \boldsymbol{\Phi}_A \boldsymbol{\gamma})\, c_U\, L + t_D \end{aligned} \qquad \text{(a4)}$$

where

$$\sigma = \begin{pmatrix} r_0/h_1 \\ r_0/h_2 \\ \vdots \\ r_0/h_n \end{pmatrix}$$