Performance Optimization Under Rise and Fall Parameters

Rajeev Murgai
Fujitsu Laboratories of America, Inc., Sunnyvale, CA
murgai@fla.fujitsu.com

Abstract

Typically, cell parameters such as the pin-to-pin intrinsic delays, load-dependent coefficients, and input pin capacitances have different values for rising and falling signals. The performance optimization algorithms, however, assume a single value for each parameter. No work has been done to study the impact of separate rise and fall values on the complexity of optimization. In this paper, we take the first step towards understanding this impact. We pick two problems that have polynomial-time complexities if a single value for each cell parameter is assumed. The first problem is that of buffer insertion on a fixed topology net to maximize the required time at the source of the net. The second is the gate resizing problem (and the more general technology mapping problem) for minimizing the circuit delay under the simplest, load-independent delay model. We show that under separate rise and fall parameters, both these problems become NP-complete. To the best of our knowledge, this is the first such result showing the effect of rise and fall parameters on the complexity of performance optimization problems. We then address the important question of devising a good practical algorithm for local fanout optimization.

1 Gate Delay Models

The model used to calculate the delay through a gate is of central importance in timing analysis and optimization.

Given a single-output gate (or cell) g, let $\delta(i,g)$ denote the delay from an input pin i of the gate g to the output of g. We will use g to denote the output of g as well. Two delay models are popular: load-independent and load-dependent. The **load** c_g refers to the cumulative capacitance seen at the output of g. It is the sum of the input pin capacitances $\gamma(p)$ of all the fanout pins p of g.

In the **load-independent delay model**, the delay from an input pin i of a gate g to the output of g, $\delta(i,g)$ is the intrinsic delay

$$\delta(i,g) = \alpha(i,g) \qquad (1)$$

In the **load-dependent delay model**,

$$\delta(i,g) = \alpha(i,g) + \beta(i,g)c_g \qquad (2)$$

Here,
c_g = load capacitance at the output of the gate g,
$\alpha(i,g)$ = intrinsic delay from i to g,
$\beta(i,g)$ = drive capability or load coefficient of the path from i to g.

The gate library specifies α and β parameters for all input-pin to output-pin paths within each gate and γ values for all the input pins. In general, $\alpha(i,g)$ and $\beta(i,g)$ are different for different input pins i. If g has a single input pin (e.g., buffers, inverters) or if α and β values are identical for all input pins, we will drop the argument i.

The above description assumes a single value for each parameter α, β, and γ. However, it is well-known that delays for the rising and the falling transitions can be quite different. In fact, every cell in the industrial cell libraries we have access to has different rise and fall delay parameter values. Quite often, these values are far off from each other. For instance, in one of our sub-micron technologies, the rise and fall α values for a path in a simple cell differed by 45% and the β values, by 100%! To handle this scenario, we use the subscripts r and f to denote rise and fall. For instance, $\alpha_{rr}(i,g)$ denotes the intrinsic delay from input pin i to the output g when i switches from 0 to 1 and as a result g also switches from 0 to 1. Similarly, $\alpha_{rf}(i,g)$ is the intrinsic delay when g falls due to a rising transition on i. Thus, for each input to output path (i,g), there are four α and four β values corresponding to rising and falling transitions at i and at g. Although the results that we prove in this work hold for this general delay model, for simplicity we assume that the circuit consists only of simple gates such as buffers, inverters, multi-input AND, NAND, OR, and NOR gates. For these gates the output is a unate function of each of the inputs. Then out of rr and rf, only one case is possible for an (i,g) pair. Only two out of four α (and also β) values are possible: α_r, α_f, β_r, and β_f; the subscript denotes the transition at the gate output. We write these values as pairs: (α_r, α_f) and (β_r, β_f).

The gate delay parameters (α_r, α_f) and (β_r, β_f) are used to compute the **arrival times** at various gates and the delay through the circuit as follows. At each cell, both rise and fall arrival times are stored. The rise (fall) arrival time at a gate g denotes the maximum possible time it takes for a transition to travel from a primary input to the output of g and g makes a rising (falling) transition as a result. A topological traversal of the circuit from primary inputs to outputs is used to compute the rise and fall arrival times at each gate g using the rise and fall arrival times already computed at the fanin gates and the gate delays δ through g. An inversion through the cell should be considered appropriately while computing the times. For instance, since a falling transition at the input of an inverter generates a rising transition at its output, the fall arrival time at the inverter's input should be used to compute the rise arrival time at its output. The arrival time at a primary output is the maximum of the rise and fall arrival times at that output. The **delay of the circuit** is the maximum arrival time at a primary output. A similar but reverse topological traversal of the circuit starting from the primary outputs computes **required times** at the cells and primary inputs.

2 Performance Optimization

Most of the research in performance optimization, including almost the entire body of theoretical work, considers only a single value for each cell parameter α, β, and γ [14, 12, 4, 10, 7, 1, 3, 5, 15]. In reality, each cell in the library has different values for rising and falling transitions. To bridge this gap between research and reality, most optimization tools approximate each cell parameter by taking either an average or the maximum of rise and fall values. Clearly, both these strategies for computing the circuit delay are approximations: the first is optimistic, and the second, pessimistic.

In this work, we show that it is non-trivial to extend certain algorithms that consider a single value for each cell parameter to those that incorporate both rise and fall values, while guaranteeing optimality. We identify two problems in performance optimization that can be solved in polynomial-time under the assumption of a single value for each cell parameter:

1. local fanout optimization (LFO) problem with the net topology fixed,
2. gate resizing/technology mapping problem for minimizing maximum delay under the load-independent delay model.

We prove that both problems become NP-complete with different rise and fall values. This, we believe, is the first work that highlights the complexity arising out of separate rise and fall values.

The paper is organized as follows. Section 3 addresses the complexity of the local fanout optimization problem under rise and fall parameters. The problems of gate resizing and technology mapping are described in Section 4. Section 5 addresses the question of devising a good practical algorithm for local fanout optimization under separate rise and fall parameters.

3 Local Fanout Optimization (LFO)

The fanout optimization problem for a single gate/net is called the local fanout optimization problem, and can be stated as follows:

- *Given a library $\mathcal{L}$ of buffers and inverters, and for each $b \in \mathcal{L}$ its input load $\gamma(b)$, its load coefficient $\beta(b)$, and its intrinsic delay $\alpha(b)$;*
- *Given the source gate s of a signal/net N, with intrinsic delay $\alpha(j,s)$ and load coefficient $\beta(j,s)$ for each input pin j of s;*
- *Given n destinations or sinks, with required time $q(i)$, load $\gamma(i)$, and polarity $p(i)$ for each sink i;*
- *Find a tree of buffers and inverters that distributes the signal N to all the sinks and maximizes the minimum required time at input pins of the source s.*

Note that fanout optimization makes sense only under the load-dependent delay model.

We will distinguish between two cases of fanout optimization. In the first case, we have the freedom to determine the topology or structure of each net tree and then insert buffers on various edges of the tree. We will call it **LFO-NTU** (for **net topology unknown**). In the second case, each net's tree topology is already determined (e.g., the parent nodes of sinks are already known) – either by a fanout tree generator or a global router; only the buffer types and insertion points need to be determined. We will call this **LFO-NTF** (for **net topology fixed**).

Note that LFO-NTF is a special case of LFO-NTU. It is well-known that the LFO-NTU problem is NP-complete [1, 14]. However, LFO-NTF can be solved in polynomial time by a dynamic programming algorithm [10, 14] if the delay parameters (i.e., α, β, γ) have identical values for rising and falling transitions (see Section 5). It turns out that for different rise and fall values, the problem becomes intractable. We show this next.

The previous formulation of LFO assumed single values for all the parameters. Now consider LFO-NTF with separate rise and fall values for α, β, and γ. The required times are also different for rising and falling transitions.

- *Given a library $\mathcal{L}$ of buffers and inverters, and for each $b \in \mathcal{L}$ its input load $(\gamma_r(b), \gamma_f(b))$, its intrinsic delay $(\alpha_r(b), \alpha_f(b))$, and its load coefficient $(\beta_r(b), \beta_f(b))$;*
- *Given the source gate s of a fixed-topology tree net N, with intrinsic delay $(\alpha_r(j,s), \alpha_f(j,s))$ and load coefficient $(\beta_r(j,s), \beta_f(j,s))$ for each input pin j of s;*
- *Given n destinations or sinks of N, with required time $(q_r(i), q_f(i))$, load $(\gamma_r(i), \gamma_f(i))$, and polarity $p(i)$ for each sink i;*
- *Determine the types and locations of buffers and inverters that should be placed on the edges (net segments) of the net N to maximize the minimum of rise and fall required times at the input pins of the source s.*

To prove our result, we impose the following restrictions on LFO-NTF:

1. At most one buffer can be inserted on one edge. This is to limit the buffering choices at each net. This assumption is made by several LFO algorithms [10, 14, 13].
2. A buffer can only be inserted on an internal edge. In other words, a buffer cannot be inserted on an edge that is directly incident upon a sink. Since in practice buffers are inserted at branching points (Steiner nodes) of the net [10, 14]), this is not really a restricting assumption.

Let us call the version of LFO-NTF with different rise and fall parameters and with restrictions (1) and (2) LFO-NTF-DRF (DRF stands for different rise and fall). We prove that LFO-NTF-DRF is NP-complete.

Theorem 3.1 *LFO-NTF-DRF is NP-complete.*

Proof It is easy to see that LFO-NTF-DRF is in NP. Given a buffering arrangement on N, the rise and fall required times at the input pins of the source gate s can be computed in linear time.

To prove NP-hardness, we transform the NP-complete problem PARTITION [2] to LFO-NTF-DRF. PARTITION, stated as a decision problem, is as follows:
INSTANCE: A finite set A and a weight $w(a) \in Z^+$ for each $a \in A$.
QUESTION: Is there a subset $A' \subseteq A$ such that

$$\sum_{a \in A'} w(a) = \sum_{a \in A - A'} w(a)? \tag{3}$$

From a general instance of PARTITION, we build a specific instance of LFO-NTF by constructing a net N with chain-topology, as shown in Figure 1 (a). N has a source gate s (with a single input T) and $|A|$ sinks. All sinks have positive polarities. For each $a_i \in A$, there is a sink a_i, with input capacitance $\gamma_r(a_i) = \gamma_f(a_i) = w(a_i)$. The library has two non-inverting buffers: *the rise-buffer R* and *the fall-buffer F*, which have the following parameter values:

$\alpha_r(R) = \alpha_f(R) = 0$	$\alpha_r(F) = \alpha_f(F) = 0$
$\beta_r(R) = 1, \beta_f(R) = 0$	$\beta_r(F) = 0, \beta_f(F) = 1$
$\gamma_r(R) = \gamma_f(R) = 0$	$\gamma_r(F) = \gamma_f(F) = 0$

The intrinsic rise and fall delays (α_r and α_f) of both buffers are zero. Only the appropriate load coefficients β are non-zero: for the rise-buffer R, $\beta_r = 1$ and for the fall-buffer F, $\beta_f = 1$.

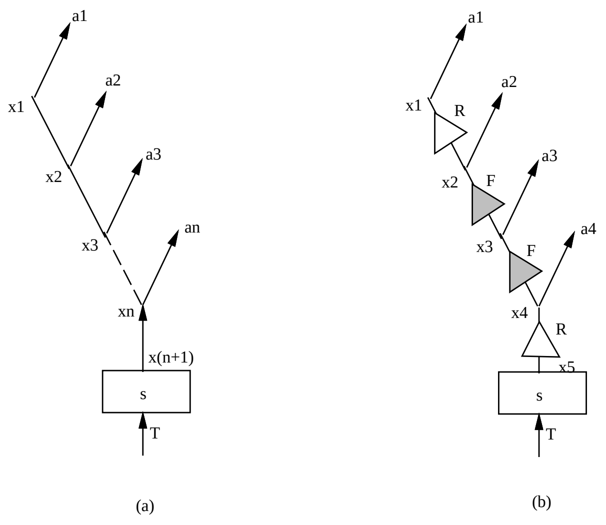

Figure 1: Constructing net N from PARTITION instance

The input-pin capacitances are also zero. Then, $\delta_r(R)$, the delay through R for the rising signal, is

$$\delta_r(R) = \alpha_r(R) + \beta_r(R)\ c_R = c_R. \tag{4}$$

Similarly,

$$\delta_f(R) = \alpha_f(R) + \beta_f(R)\ c_R = 0. \tag{5}$$

For the fall-buffer F,

$$\delta_r(F) = \alpha_r(F) + \beta_r(F)\ c_F = 0. \tag{6}$$

$$\delta_f(F) = \alpha_f(F) + \beta_f(F)\ c_F = c_F. \tag{7}$$

Here c_R and c_F denote load capacitances at the output of R and F respectively.

Let

$$W(A) = \sum_{a \in A} w(a) \tag{8}$$

The source gate s has the following parameter values:

$$\alpha_r(s) = \alpha_f(s) = 0, \beta_r(s) = \beta_f(s) = W(A), \gamma_r(s) = \gamma_f(s) = 0.$$

Finally, let the required times $q_r(a_i) = q_f(a_i) = 0$ for all sinks a_i.

We prove that *there exists a buffering of the net N such that both $q_r(T)$ and $q_f(T)$ are at least $-W(A)/2$ if and only if there exists a subset A' of A such that (3) holds.* Since the required times of all the sinks are identically zero, this is equivalent to proving that the rise and fall delays through N (i.e., the maximum delay from T to some sink) are at most $W(A)/2$ if and only if there exists a subset A' of A such that (3) holds.

First note that in accordance with the restriction (2) on LFO-NTF-DRF, no buffer can be placed on an edge of type (x_i, a_i); it can only be placed on an edge of type (x_{j+1}, x_j). Then, the delay through the net N is the delay from T to the sink a_1, i.e., the sum of the delays through s and through all the buffers on the path from x_{n+1} to x_1.

If: Assume A' such that (3) holds. We derive a buffering of N as follows. We insert a buffer on each edge (x_{i+1}, x_i). For each item a_j in A', we insert the rise-buffer R on (x_{j+1}, x_j). On all other edges, we insert fall-buffers F. Of course, edges (x_i, a_i) are left unbuffered. This defines the buffering arrangement of N.

For instance, if $|A| = 4$ and $A' = \{a_1, a_4\}$, the resulting buffering arrangement is shown in Figure 1 (b). Rise-buffers R are placed on edges (x_2, x_1) and (x_5, x_4), and fall-buffers F on (x_3, x_2) and (x_4, x_3).

Let us compute the rise delay through the net N. First, note that both rise and fall delays through s are 0. Since s sees a capacitive load c_s of 0 (there is either R or F on the edge (x_{n+1}, x_n) and the input pin capacitances of R and F are zero), the delay through s

$$\delta(s) = \alpha(s) + \beta(s)c_s = 0 + \beta(s)0 = 0$$

As shown in (6), the fall-buffers on the net do not contribute to the rise delay. As for the rise-buffers, consider such a buffer, say on (x_{i+1}, x_i). It sees a capacitive load of $w(a_i)$ corresponding to the sink a_i, since the buffer on (x_i, x_{i-1}) has zero input capacitance. So, from (4), its rise delay contribution is $w(a_i)$. Therefore, the rise delay through the net N is simply given by the total capacitance driven by the rise-buffers present on N. Since rise-buffers on N correspond to items in A', the rise-delay through N is equal to the sum of the weights of the items in the set A'. In our example of Figure 1 (b), this is $w(a_1) + w(a_4)$. Similarly, the fall delay through the net is given by the total capacitance driven by all the fall-buffers present on N, which is the sum of the weights of the items in the set $A - A'$. From (3) and (8), it follows that both rise and fall delays are $W(A)/2$.

Only If: Assume there exists a buffering of N such that both rise and fall delays through N are at most $W(A)/2$.

Observe that given any sink a_i, there must be at least one buffer between a_i and the source s. Otherwise, s would be driving a load of at least $w(a_i)$, whose net delay contribution would be $\beta(s)\ w(a_i) = W(A)\ w(a_i) > W(A)/2$, a contradiction! Let B_i be the first such buffer closest to a_i. If B_i is a rise-buffer, a_i contributes $w(a_i)$ towards the rise delay through B_i and through N, otherwise it contributes $w(a_i)$ towards the fall delay. Thus, each sink a_i contributes $w(a_i)$ either to the rise delay or to the fall delay through N (but not both). This implies that the sum of the rise and the fall delays through N is $W(A)$ (the delay through s is zero), which implies that both rise and fall delays through N are exactly $W(A)/2$. Let U' be the set of sinks contributing towards the rise delay. Then, it follows that U' is the desired set A' of PARTITION satisfying (3). ■

4 Gate Resizing/Technology Mapping

Consider the following scenario. We are given a circuit composed of cells from a cell-library. For each cell C_i, many different sizes are available in the library, each size having possibly different area, input pin capacitances γ, intrinsic delays α, and load coefficients β. *The gate resizing problem is to select the size of each cell such that the circuit delay is minimized. We assume the load-independent pin-to-pin delay model*, in which the delay through a path within a cell is simply the intrinsic delay α.

If only one value were to be used for each cell parameter (i.e., α), the problem can be solved optimally by a dynamic programming algorithm [4] as follows. Traverse the network gates in a topological order from primary inputs towards primary outputs. When a cell C is reached, the arrival times at all its input pins are known. For each available size of the cell C, compute the arrival time at the output of C using the arrival times at its input pins and the pin-to-pin δ delays for the cell size. Pick the

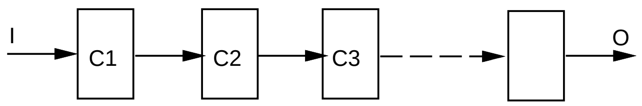

Figure 2: Chain-circuit for proving NP-completeness of gate resizing

size that minimizes the arrival time at the output of C. Continue the traversal and size selection until the primary outputs are reached.

If both α_r and α_f are specified, the circuit delay is max {circuit rise delay, circuit fall delay}, which is what we wish to minimize. One natural strategy for using the dynamic programming paradigm is the following.

For each cell, select the size that minimizes the maximum of rise and fall arrival times at that cell.

However, this is a non-optimal strategy, as the following example illustrates.

Example 4.1 *Consider a circuit with only two AND cells C and O, with C feeding O. The inputs of C are the primary inputs, arriving at times zero. The output of O is the only primary output of the circuit. Let there be two sizes of the cell C: the first one has $\alpha_r = 6, \alpha_f = 8$ and the second has $\alpha_r = 9, \alpha_f = 6$. This strategy would select the first size for C, since it has smaller maximum delay. Let O have only one size: $\alpha_r(O) = 2$ and $\alpha_f(O) = 10$. Then the size selected at C results in a circuit delay of* $\max\{6+2, 8+10\} = 18$. *Had we selected the second size for C, it would have resulted in a smaller delay of 16 (*$= \max\{9+2, 6+10\}$*).*

As this example shows, using this strategy we cannot decide locally at a gate the best size for it. We need to examine the fanouts as well. However, that may generate an exponential number of solutions by essentially enumerating all possible size selection choices in the circuit.

It turns out that deriving an efficient optimum solution is difficult because the problem itself is intractable. We prove that the problem of gate resizing with different rise and fall parameter values is NP-complete even under the load-independent delay model, which is the simplest possible model.

Theorem 4.1 *Given different rise and fall parameter values for the gates, the gate resizing problem is NP-complete under the load-independent delay model.*

Proof That the problem is in NP is easy to see. To prove NP-hardness, the transformation is once again from PARTITION. Given an instance of PARTITION, we construct a circuit with $|A|$ single-input, single-output non-inverting cells $C_1, C_2, \ldots C_{|A|}$, which are connected in a chain, with the output of C_i connected to the input of C_{i+1} (Figure 2). The circuit has a single input and a single output.[1] The cell C_i corresponds to the item a_i of A. Each cell C_i comes in two sizes: C_i^R and C_i^F, and they have the following delay parameters:

$$\alpha_r(C_i^R) = w(a_i), \qquad \alpha_f(C_i^R) = 0. \tag{9}$$

$$\alpha_r(C_i^F) = 0, \quad \alpha_f(C_i^F) = w(a_i). \tag{10}$$

The size R contributes only to the rise delay at its output, and the size F only to the fall delay.

[1]The argument in the proof remains the same if each gate has other fanins that are primary inputs.

We show that there exists $A' \subseteq A$ such that (3) is satisfied if and only if the rise and fall delays through the circuit are each at most $W(A)/2$.

Only If: Given A' such that (3) is satisfied. If $a_i \in A'$, select the size C_i^R for the cell C_i; otherwise, select C_i^F. Since each cell is non-inverting, rise delay of the circuit is the sum of the rise delays of all the cells. Since the rise delay through a cell that corresponds to an item not in A' is zero, and through a cell that corresponds to an item in A' is the weight of the corresponding item, the rise delay of the circuit is precisely $\sum_{a \in A'} w(a) = W(A)/2$ (from (3)). Similarly, the fall delay is $\sum_{a \in A - A'} w(a) = W(A)/2$.

If: Assume there exists a size for each cell such that both rise and fall delays are at most $W(A)/2$. In fact, both must be exactly equal to $W(A)/2$, since each cell C_i contributes $w(a_i)$ either to the rise delay or the fall delay through the circuit and hence the total contribution of all the cells to rise or fall delay through the circuit is $W(A)$. Create the set A' as follows. If the size C_i^R is selected for the cell C_i, place the corresponding item a_i in A'. It is easy to see that (3) is satisfied. ■

Notes:

- In [6], Li *et al.* proved that the problem of gate resizing for minimizing the circuit delay under area constraints is NP-complete. Our proof (of Theorem 4.1) can be obtained by replacing the cell delay and area parameters in the proof of [6] with the rise and fall delay parameters.
- Although some of the α values in the proof are zero, an alternate proof that uses strictly positive α values can also be constructed by adding a constant ν to all the cell delays.[2]
- We used the simplest load-independent delay model to prove the complexity result. Clearly, the gate resizing problem remains NP-complete for the more realistic load-dependent delay model as well.
- Since gate resizing is a special case of technology mapping, the previous theorem also establishes that the problem of technology mapping for minimum circuit delay given separate rise and fall delay parameters under the load-independent delay model is NP-complete.
- The proof builds a circuit that is a single chain of cells. Hence, the minimum-delay resizing and mapping problems are NP-complete even for circuits with such a simple chain topology.

5 Practical Considerations

Since cells in the technology libraries have separate rise and fall parameters, we need to solve LFO-NTF and gate resizing problems with these parameters. In this section, we address LFO-NTF.

If each delay parameter has a single value, LFO-NTF can be solved in polynomial time by Ginneken's algorithm [10], which we briefly describe next. Given a net N with fixed topology (as shown in Figure 3) and required times at sinks, Ginneken's algorithm determines an optimum choice of buffers and their locations on the net to maximize the required time at the net source. The algorithm traverses nodes of the net bottom-up: starting from the net sinks and proceeding towards the root s. At an intermediate node (Steiner node) v, there is a choice: *should a buffer be inserted at v or not.* In fact if the buffer library has B buffers, there are $(B+1)$ possibilities. The algorithm constructs a set of solutions $S(v)$ at v to capture all

[2]We thank Dave Wallace for pointing this out.

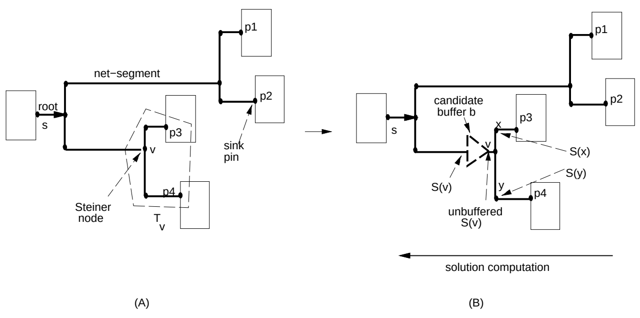

Figure 3: A single net (A); solution computation (B)

these possibilities. A solution is a pair (c, q), where c is the capacitance of the tree T_v rooted at v and q is the required time at v. If x and y are v's children (Figure 3), $S(v)$ is constructed by first *combining* $S(x)$ and $S(y)$ and then considering all buffering possibilities at v. Thus $S(v)$ captures all buffering possibilities at all the net nodes in T_v. At the net root s, the solution of $S(s)$ that maximizes the required time at the input pins of s is the optimum solution for the net N. Stated as above, the algorithm takes exponential time: it enumerates all possible buffering choices for N. Ginneken made a key observation, which reduces the complexity of the algorithm to polynomial: *For* $(c, q), (c', q') \in S(v)$, *if* $(c' \geq c)$ *and* $(q' < q)$, *then* (c', q') *is sub-optimal.* This is so because a larger load with smaller required time can only make the delay worse and is thus sub-optimal. It turns out that most of the solutions are sub-optimal. Sub-optimal solutions should either not be generated or, if generated, be thrown away (pruned) immediately. In Ginneken's algorithm, when $S(x)$ and $S(y)$ are combined, sub-optimal solutions are not generated. During buffered solution construction, sub-optimal solutions are generated but immediately detected and thrown away. This results in an efficient polynomial-time algorithm for a single net.

One way to solve LFO-NTF-DRF is by setting for each buffer and inverter $\alpha = \max\{\alpha_r, \alpha_f\}$, $\beta = \max\{\beta_r, \beta_f\}$, the input pin capacitance $\gamma = \max\{\gamma_r, \gamma_f\}$, and then using Ginneken's algorithm [10]. We call this approximation **same-rise-fall**. This approximation is employed by many academic and commercial tools for various performance optimization sub-problems. For instance, the timing-driven technology mapper [14] of the Berkeley logic synthesis system sis [11] uses this approximation.

Another way to solve LFO-NTF-DRF is by modifying Ginneken's algorithm to accommodate rise and fall values. Instead of (c, q), a solution is now (c^r, q^r, c^f, q^f). We call this option **diff-rise-fall**. c^r & q^r denote the load capacitance and required time for the rising signal and c^f & q^f the load capacitance and required time for the falling signal. As mentioned earlier, a fast identification and pruning of sub-optimal solutions is key to an efficient and effective buffer optimization algorithm. Under separate rise and fall values, the sub-optimality of a solution can be checked as follows. Given solutions $\sigma_1 = (c_1^r, q_1^r, c_1^f, q_1^f)$ and $\sigma_2 = (c_2^r, q_2^r, c_2^f, q_2^f)$ at a node, σ_1 can be thrown away if all of the following conditions hold:

$$c_1^r \geq c_2^r, \; c_1^f \geq c_2^f, \; q_1^r \leq q_2^r, \; q_1^f \leq q_2^f \qquad (11)$$

Although it is possible to come up with straight-forward algorithms for combining and pruning the solution sets, they are not time- and space-efficient. We could neither devise a data-structure that effectively handles four components in a solution nor come up with any insight to efficiently remove sub-optimal solutions. In the worst case, combining $S(x)$ and $S(y)$ can take time and space equal to $|S(x)|\ |S(y)|$, and pruning a set can take time quadratic in the size of the set. This gives rise to the worst-case exponential time complexity. These observations are consistent with the NP-completeness result. Note that the *diff-rise-fall* algorithm, although exponential in the worst case, is optimum for a single net.

Next, we compare the performance of *same-rise-fall* and *diff-rise-fall* on real designs.

5.1 Experimental Results

We performed two experiments. In the first experiment, our intent was to compare the performance of *diff-rise-fall* and *same-rise-fall* on individual nets. We took all the critical nets of an industrial design ex6 (Table 1). There were 225 of them, with the number of sinks ranging from 1 to 17. On each net, we applied *diff-rise-fall* and *same-rise-fall* and compared the resulting delay improvements at the source of the net. On average, *diff-rise-fall* resulted in 0.029ns higher required time at the source than *same-rise-fall*. The maximum difference in the improvements was, however, large: 0.43ns.

In the second experiment, we compared these two methods in the context of entire circuits. So we embedded the two methods in a global buffering scheme. This scheme selects nets for buffering in an iterative manner. In each iteration, it evaluates nets for delay improvement and buffers a subset of nets that yield a positive improvement. A delay trace is performed on the circuit. If the circuit delay improves, the iteration is repeated with the updated delay values. Otherwise, we stop. We used real industrial designs as our benchmarks. Table 1 shows relevant design statistics such as the technology used, numbers of cells and nets in the design, and the total cell area. ex1 to ex3 are very small fragments of real designs; the rest are real industrial designs. Of them, the largest, ex8, is a *hi-vision TV encoder/decoder* design. It has about 172K cells and 211K nets. We applied both buffering methods on these designs. The results are reported in Table 2. We report the original design delay, the final delay after buffering, area penalty and CPU time taken by each method. *diff-rise-fall* yields only 0.71% better final delays as compared to *same-rise-fall* on average. And its area penalty is actually worse than *same-rise-fall* by 50%. Most importantly, on average, its CPU time is 8.8 times that of *same-rise-fall*. In fact, in ex5, there were 5 net nodes for which more than one million solutions were generated by *diff-rise-fall*!

Note that for ex3, *diff-rise-fall* produces slightly worse delay than *same-rise-fall*. This is because although *diff-rise-fall* is an optimum algorithm for a single net, the net selection strategy used to apply it to the entire circuit is heuristic.

We can conclude that at least for LFO-NTF, the *same-rise-fall* approximation generates results that are very close to those generated by an exact, worst-case exponential algorithm *diff-rise-fall* in the context of entire circuits and runs much faster.

6 Conclusions

In this paper, we showed that certain problems in performance optimization that can be solved in polynomial time under the single value assumption for each delay parameter become NP-complete under separate rise and fall values. To the best of our knowledge, this is the first theoretical result that highlights the complexity resulting from considering separate rise and fall parameters.

Although the NP-completeness result implies that we can bid farewell to finding the optimum solution for these problems ef-

ex	tech.	#cells	#nets	cell area (in BC)	r-time (sec)
ex1	0.35 μ	32	48	105	13
ex2	0.35 μ	356	409	1567	15
ex3	0.25 μ	268	355	1952	16
ex4	0.35 μ	$\sim$ 17.1K	$\sim$ 26.0K	$\sim$ 122.6K	110
ex5	0.5 μ	$\sim$ 35.3K	$\sim$ 36.9K	$\sim$ 93.0K	213
ex6	0.35 μ	$\sim$ 40.0K	$\sim$ 48.1K	$\sim$ 200.2K	328
ex7	0.35 μ	$\sim$ 86.7K	$\sim$ 108.1K	$\sim$ 381.6K	574
ex8	0.35 μ	$\sim$ 172.2K	$\sim$ 210.9K	$\sim$ 718.6K	2384

1K = 1000, r-time = time to read cell library & design data
1 BC = area of the smallest inverter in the library

Table 1: Benchmark statistics

ex	*o.d.*	diff-rise-fall			same-rise-fall		
		n.d.	ΔA	cpu	*n.d.*	ΔA	cpu
	(ns)	(ns)	(BC)	(sec)	(ns)	(BC)	(sec)
ex1	6.16	3.60	6	0	3.62	6	0
ex2	7.84	4.81	81	0	4.81	62	0
ex3	4.69	4.42	49	2	4.39	76	0
ex4	7.44	7.32	78	21	7.32	33	25
ex5	14.93	13.99	609	4711	14.33	161	86
ex6	11.38	8.97	619	1033	9.09	569	207
ex7	18.41	10.33	210	2150	10.51	257	854
ex8	56.36	42.85	1622	2831	43.04	1626	530
avg		0.9929	1.5	8.8	1.0	1.0	1.0

o.d. = original circuit delay, *n.d.* = new delay,
ΔA = area penalty; a 200 MHz Ultrasparc
with 1GB RAM was used for all experiments.

Table 2: *diff-rise-fall* vs. *same-rise-fall*

ficiently, we did present a simple and effective algorithm for the local fanout optimization (net topology fixed) problem with different rise and fall parameters that gives almost the same quality solutions as a more CPU-intensive exact algorithm. This, in a sense, lends credence to simple heuristic approaches that handle separate rise and fall values by approximating them by the worse of the two.

Note that certain performance optimization problems are known to be hard even in the absence of rise and fall parameters. For instance, the gate resizing/technology mapping problem for minimizing the maximum circuit delay under the load-dependent delay model is NP-complete [8]. So are the problems of global fanout optimization [9] (both with net topology fixed and with net topology unknown) and local fanout optimization with net topology unknown [1, 14]. Clearly, with separate rise and fall parameters, such problems will remain NP-complete.

Finally, we note that the NP-completeness proofs we presented used transformations from PARTITION. Although PARTITION is NP-complete, it is not NP-complete *in the strong sense* [2]. It can be solved optimally in pseudo-polynomial time by dynamic programming. This leaves open the possibility of pseudo-polynomial optimum algorithms for LFO-NTF-DRF and gate resizing. We will pursue this matter in near future.

Acknowledgements

We thank Mukul Ranjan Prasad for implementing an initial version of the *diff-rise-fall* algorithm for a single net, and Robert Carragher for helping with the framework issues.

References

[1] C. L. Berman, J. L. Carter, and K. F. Day. The Fanout Problem: From Theory to Practice. In C. L. Seitz, editor, *Advanced Research in VLSI: Proceedings of the 1989 Decennial Caltech Conference*, pages 69–99. MIT Press, March 1989.

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Co., NY, 1979.

[3] K. Kodandapani, J. Grodstein, A. Domic, and H. Touati. A Simple Algorithm for Fanout Optimization Using High-Performance Buffer Libraries. In *Proceedings of the International Conference on Computer-Aided Design*, pages 466–471, 1993.

[4] Y. Kukimoto, R. K. Brayton, and P. Sawkar. Delay-Optimal Technology Mapping by DAG Covering. In *Proceedings of the Design Automation Conference*, pages 348–351, 1998.

[5] D. Kung. A Fast Fanout Optimization Algorithm for Near-Continuous Buffer Libraries. In *Proceedings of the Design Automation Conference*, pages 352–355, 1998.

[6] W. N. Li, A. Lim, P. Agarwal, and S. Sahni. On the Circuit Implementation Problem. In *Proceedings of the Design Automation Conference*, pages 478–483, 1992.

[7] J. Lillis, C. K. Cheng, and T. T. Y. Lin. Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model. In *Proceedings of the International Conference on Computer-Aided Design*, pages 138–143, 1995.

[8] R. Murgai. On The Complexity of Minimum-delay Gate Resizing/Technology Mapping Under Load-Dependent Delay Model. In *IWLS*, 1999.

[9] R. Murgai. On The Global Fanout Optimization Problem. In *Proceedings of the International Conference on Computer-Aided Design*, 1999.

[10] Lukas P. P. P. van Ginneken. Buffer Placement in Distributed RC-tree Networks for Minimum Elmore Delay. In *Proceedings of the International Symposium on Circuits and Systems*, pages 865–868, 1990.

[11] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. *SIS: A System for Sequential Circuit Synthesis.* Memorandum No. UCB/ERL M92/41, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, May 1992.

[12] K. J. Singh. *Performance Optimization of Digital Circuits.* PhD thesis, UC Berkeley, December 1992.

[13] K. J. Singh and A Sangiovanni-Vincentelli. A Heuristic Algorithm for the Fanout Problem. In *Proceedings of the Design Automation Conference*, pages 357–360, June 1990.

[14] H. Touati. *Performance-oriented Technology Mapping.* PhD thesis, UC Berkeley, November 1990. UCB/ERL M90/109.

[15] H. Vaishnav and M. Pedram. Routability-Driven Fanout Optimization. In *Proceedings of the Design Automation Conference*, pages 230–235, 1993.