A Combinatorial Approximation Algorithm for Selecting the Gate Sizes from Finite Sets in VLSI Circuits

Nodari Vakhania[*1] and Frank Werner[2]

[1]Science Faculty, State University of Morelos
Av. Universidad 1001, Cuernavaca 62210, Morelos, Mexico
Inst. of Computational Math., Akuri 8, Tbilisi 93, Georgia
tel.: +52 777 329 70 20
[2]Fakultat fur Mathematik, Otto-von-Guericke-Universitat,
PSF 4120, 39106 Magdeburg, Germany
Email address: frank.werner@ovgu.de
tel.: +49 391 67 12025
fax: +49 391 67 11171
e-mail: nodari@uaem.mx ; frank.werner@ovgu.de

Abstract

In this paper, we consider a problem of VLSI design occurring in the routing phase. The problem is to determine the optimal size selection for the gates in a combinatorial circuit which uses the problem of finding a shortest path in an oriented acyclic graph for making certain updates between any two successive iterations. For this NP-hard problem, we give an approximation algorithm.

Key words: VLSI design, combinatorial circuits, NP-hard problems, approximation algorithm, shortest paths

1. Introduction

The design of VLSI (very large scale integrated) circuits belongs to the hardest problems in combinatorial optimization. The design of such circuits is a multi-stage process which can be roughly divided into three types of sub-problems: *partitioning, placement, and routing*.

In the partitioning phase, the chip is split into smaller pieces which can be easier treated. Here it is typically assumed that these pieces can be designed independently of the other ones.

In the placement stage, the locations of all circuit gates (also called blocks) within the chip are fixed and a list of the gates which need to be connected with wires is generated. Typically, a cost is assigned to each placement and then this cost function is minimized.

[*] Corresponding author

We only note that another type of a placement problem is floor planning which arises if a circuit is decomposed into a number of gates which are to be routed separately.

In the routing phase within the physical design of VLSI circuits, one wishes to find a realization of the connections determined in the placement phase. More precisely, routing deals with finding the layouts for the wires connecting the terminals on the gates.

Since the number of possible routes is huge, the routing problem is computationally very hard. Even the determination of an optimal layout (e.g., one with a minimum wire length) for a single net is NP-hard so that the existence of a polynomial algorithm is very unlikely. For this reason, most routing algorithms are of heuristic nature.

Due to the complexity of the problem, routing is often divided further into global routing and detailed routing. While in global routing, the exact geometric details are still ignored, and in some sense, only `loose' routes for the wires are determined, the detailed routing phase completes this point-to-point wiring by specifying such geometric information as the location and width of the wires and their layer assignments. In the detailed routing phase, the output from the global routing is used and then the exact geometric layout of the wires for connecting the gates is determined. From a graph-theoretical point, the detailed routing problems include the determination of vertex-disjoint Steiner trees in a grid. In particular, one of the main problems in detailed routing consists in channel routing. However, even by using this splitting into global and detailed routing, each of these phases remains NP-hard. For instance, the global routing problem is NP-hard since it is at least as hard as the minimum Steiner problem in graphs which is contained as a special case.

In the routing phase, the primary goal is to determine feasible routes, and if so, a particular objective function is considered. Often it is distinguished between two-terminal and multi-terminal nets. Many of the algorithms for such problems are variants of shortest path algorithms. For global routing, two specific graph models are typically used, namely a grid graph model and a channel-intersection graph model. In two-terminal algorithms, often Dijkstra's algorithm is employed on intersection graphs or Maze routing and Hadlock's algorithm is used on grid graphs. The objective function is typically described as a function minimizing the cost of the connections such as the wire length or edge congestions, or a linear combination of several terms (see e.g. [20]).

As already mentioned, typically heuristic or approximation algorithms are used for the particular sub-problems arising in the VLSI design. Since there exists a huge number of publications dealing with the particular sub-problems resulting in the different stages, we can mention here only a few works. An early introduction into the VLSI design for

analog and digital circuits can be found e.g. in [11]. Sherwani [17] presented the basic concepts and algorithms in the area of VLSI design.

For the circuit partitioning and placement phases, e_ective memetic algorithms have been presented and discussed e.g. in [1]. Particularly for the routing phase, a recent overview on global routing in VLSI dealing with algorithms, theory and computation has been given in [9]. Several integer linear programming models for global routing have been presented in [4]. Global routing was also considered in [8], where also a polynomial time approximation algorithm has been given for the global routing problem which is based on an integer programming formulation. This algorithm ensures that all routing demands are satisfied concurrently such that the overall cost is approximately minimized. It turned out that their new algorithms performed well compared with other algorithms based on integer programming models. Combinatorial algorithms particularly for the detailed routing phase have been presented in [19]. There have been recently developed some refinement techniques to the global routing problem (see e.g. [21]).

Genetic algorithms for the problems of channel routing can be found e.g. in [15]. More general, the discussion of genetic algorithms for VLSI design, layout and test automation was discussed in detail in [16]. Recent particle swarm algorithms for routing in VLSI circuits have been given e.g. in [2]. A recent optimization algorithm for the VLSI design which is based on grid graphs was suggested in [13]. This algorithm constructs a maze routing path such that the interconnect delay from the source to the sink is minimized. The authors introduce a novel look-ahead scheme which is applied to speed up the running time of the algorithm and which provided a significant improvement in the performance over some existing routing algorithms.

In this paper, we consider a problem occurring in the routing phase of VLSI design. We consider a combinatorial circuit with a number of gates with assigned available integer size and integer delay values resulting from the sizes of the gates. The goal is to determine a feasible size selection for the gates such that in the corresponding oriented acyclic graph with the gates as nodes, the resulting critical path has a minimal length. For this NP-hard problem, we present an approximation algorithm. In contrast to most other algorithms in this area which are either based on continuous optimization or on a simply greedy approach, our approximation algorithm has a maximum absolute error equal to the maximum difference between the largest and smallest overall delays taken over all gates.

The greedy algorithm is the most widely used approach for gate sizing (see e.g. [14, 10, 18, 5]. This method iteratively resizes the nodes on (or near) the critical path by means of particular heuristics. We note that there exist several papers in the literature dealing with gate-size selection problems using a di_erent setting than ours. Among them, we mention here e.g. the papers [6, 3, 12]. In [6], the problem of choosing optimal gate sizes from a

library to minimize the total circuit area subject to timing constraints was considered. Beeftink et al. [3] presented an algorithm for selecting a good set of gate sizes to minimize of a prescribed measurement. The error function quanti_es the discrepancy in the measurement when a required gate size is replaced by an available gate size. The algorithm searches for a set of gates minimizing the delay error or the size error. Joshi and Boyd [12] considered the problem of choosing the gate sizes in a circuit to minimize the total area subject to timing constraints. In contrast to the problem considered in this paper, a continuous problem in the form of a geometric program (i.e., the objective and constraint functions have a special form) is considered, where upper bounds on the gate delays are used. For this problem, large circuits have been considered such that the associated geometric program has about three million variables and more than six million monomial terms in its constraints. A comparative study of some algorithms for gate sizing can be found e.g. in [7]. In particular, this paper compares _ve di_erent algorithms on constraint free delay optimization and delay constraint power optimization. It turned out that one of the approaches was superior to the widely used greedy approach. We also note that in contrast to other papers on gate size selection dealing with continuous functions, we select the gate sizes from _nite sets and present a combinatorial approach in this paper.

The remainder of this paper is organized as follows. In Section 2, we formulate the problem considered in more detail. The main concepts of the algorithms are presented in Section 3. Finally, in Section 4, we give some concluding remarks.

2. The Problem

The problem investigated in this paper can be formally described as follows. We are given an oriented acyclic graph $G = (V, E)$. Each node of G represents one of n + 2 objects (we call them further gates). Gate $i \in V$ is characterized by the set of available integer sizes and the set of corresponding integer time delays. Namely, to the gate $i \in V$, one of the following integer sizes $\alpha_{i1} \leq \alpha_{i2} \leq \cdots \leq \alpha_{im}$ can be assigned. The delay time of gate i depends on the size of this object as well as on the sizes of its immediate successors: from one side, with increasing the size of a gate its delay is decreasing; but from the other side, with increasing the size of its successors the delay of a gate is increasing. So, for each gate $i \in V$, we are given two sets of integer delay values

$$\acute{\delta}_{i1} \leq \acute{\delta}_{i2} \leq \cdots \leq \acute{\delta}_{im}$$

And

$$\acute{\acute{\delta}}_{i1} \leq \acute{\acute{\delta}}_{i2} \leq \cdots \leq \acute{\acute{\delta}}_{im}$$

(in general, we will have $\acute{\delta}_{ij} > \acute{\acute{\delta}}_{ij}, i\epsilon V, j = 1,2,\dots,m$). The delay $\acute{\delta}_{ij}$ is the delay of the gate i imposed by the size α_{ij} . Moreover, is the additional delay which will imply gate i of size $\acute{\acute{\delta}}_{ij}$ for its direct predecessor gates (their delays will increase if the size of gate i is increased).

We consider the overall delays

$\delta_{ij} = \acute{\delta}_{ij} + \acute{\acute{\delta}}_{ij}, i\epsilon V, j = 1,2,\dots,m.$

The gates 0 and $n + 1$ are fictitious gates. We represent them in G as source and sink nodes, respectively. We set $\alpha_{0j} = 0$; $\acute{\delta}_{0j} = 0$; the values $\acute{\acute{\delta}}_{0j}$ are not defined $(\delta_{0j} = \acute{\delta}_{0j})$, $\acute{\delta}_{n+1j} = 0, j = 1,2,\dots,m$ and the values $\acute{\acute{\delta}}_{n+1j}$ are given numbers. If the gate l has the size α_{lj}, then to any arc $(k,l) \in E$, the delay of gate $l\delta_{lj}$ is assigned. A solution of the problem is defined by the assignment function

$\sigma: i \to \alpha_{ij(i)}, i \in V, 1 \le j(i) \le m.$

A solution $\sigma = \{\sigma_{ij(i)}\}$ is feasible if

$\sum_{i=0}^{n+1} \sigma_{ij(i)} \le B_{max},$

where B_{max} is a given integer bound. A feasible solution σ, which yields the minimal length of a critical path in its corresponding graph G_σ, is an optimal solution of the problem.

3. Main Concepts and the Algorithm

First, we give a brief overview of the algorithm we subsequently present. We obtain an initial feasible solution $\sigma_0$by assigning to each gate its smallest available size. Obviously, if this solution is not feasible, then there exists no feasible solution. Then we iteratively build a new feasible solution by selecting a particular gate on the currently determined critical path, and we assign to it a new size. The gate and size selection is accomplished by means of the introduced rate functions. The rate function measures the maximum delay fall on one unit of increased size. We also note that some of the assignments might be later revised.

Next, we present the main concepts of the algorithm in more detail. As noted above, the algorithm presented here obtains an initial feasible solution by assigning to each gate $i \in V$ its smallest available size α_{i0}. If this solution is not feasible, i.e., if the sum of assigned sizes of all gates is greater than the given bound, then there does not exist a

feasible solution and the algorithm stops. Otherwise, it determines a critical path P_0 in the obtained graph G_0. Then one particular gate is determined on P_0 and a new, greater than the current, size is assigned to it. This decreases the delay of this gate and hence, the length of P0 is decreased. Then the algorithm proceeds with the next iteration determining again a critical path in the new graph and the node on it whose size is increased at that iteration. The algorithm continues in an analogous way until if finds out that it cannot increase further the size of the selected gate. The distance between the obtained feasible solution and an optimal one is no more than some constant Δ derived from the given problem instance.

We denote by P_h the particular critical path determined at iteration h. The gate whose size assignment is increased at this iteration, is called a substitution gate. We denote the graph corresponding to iteration h by G_h. The graph G_h is obtained from G_{h-1} by the correction of the time delays with respect to the substitution at iteration $h-1$. We denote by $\tau(P)^h$ the length of the path P in the graph G_h. We use the short form $\tau(P_h)$ for the length of the critical path P_h at iteration h.

We call the sum of the sizes of all gates at iteration h the overall size at that iteration. Moreover, we denote by $h(i)$ the iteration of the last substitution of gate i by iteration h.

Further, we denote by $\gamma(i,h)$; $0 \le \gamma(i,h) \le m$, the size selection of gate i at the beginning of iteration h. $\gamma(i,h) = 0, i \in V$; and $\gamma(i,0), i \in V$, are not defined since at the beginning of iteration 0, we have no size selections. So, the size of gate $i \in V$ at iteration h is $\alpha_{i\gamma(i,h)}$, and the corresponding delay is$\delta_{i\gamma(i,h)}$. We call $\gamma(i,h)$ the regular size selection of gate i at iteration h. The estimating size selection of gate i at iteration h, $\gamma^*(i,h)$ is defined by

$$\frac{\delta_{i\gamma(i,h)}-\delta_{i\gamma^*(i,h)}}{\alpha_{i\gamma^*(i,h)}-\alpha_{i\gamma(i,h)}} = max\left\{\frac{\delta_{i\gamma(i,h)}-\delta_{ik}}{\alpha_{ik}-\alpha_{i\gamma(i,h)}} | k > \gamma(i,h)\right\}$$

Later, we use the notations

$$\Delta\delta_{ih} = \delta_{i\gamma(i,h)} - \delta_{i\gamma^*(i,h)}$$

and

$$\Delta\alpha_{ih} = \alpha_{i\gamma^*(i,h)} - \alpha_{i\gamma(i,h)}.$$

Notice that all the above magnitudes are positive.

The estimating size selection is evaluated iteratively for each gate on the currently determined critical path for determining the substitution gate. Only to the selected

substitution gate, there will be assigned the (new) estimating size selection while the size selection of all other gates will remain the regular ones.

Let h be the iteration of the latest substitution of the gate i. We say that gate i is revised at iteration $\acute{h} > h$ if $\gamma(i, \acute{h}) := \gamma(i, h)$, i.e., to gate i, there is reassigned its regular size selection corresponding to the beginning of the iteration of its last substitution or, equivalently, the size of gate i is decreased by $\Delta\alpha_{ih}$ and its delay is increased correspondingly by $\Delta\delta_{ih}$. Accordingly, we say that gate i is restored at iteration h^*, $h^* > \acute{h}$ if $\gamma(i, h^*) := \gamma^*(i, h)$, i.e., if the estimating size selection of gate i at iteration h is reassigned to gate i at iteration h^*.

If we substitute gate i at iteration h, we may revise some gates, substituted earlier and related to gate i in a certain way. The revision of these gates may cause the restoration of some other related gates, and so on. We discuss this in detail later, now we give a simple example for the illustration.

We briey sketch the fragment of a graph G_h. Gate i is the substitution gate at iteration h, and the gates j and k are the former substitution gates at the iterations h_1 and h_2, $h_1 < h_2 < h$, respectively (here we just assume that the three gates are substitution gates, later we discuss how we determine the substitution gate). The gate i is such that $i \in P_{h1}, i \in P_{h2}, i \in P_h$. We just indicated that i is a substitution gate at iteration h. This means that its current (regular) size will be replaced by the greater (estimating) size which will cause the decrease of the length of P_h. However, gate i belongs to two other (former) critical paths at the iterations h_1 and h_2 (we now assume that the three paths are related in a certain way, we specify what we mean by this later). Therefore, if we revise the size assignment of the gates j and k, we will still decrease the paths p_1 and p_2. As a result, we increase the size of one gate instead of increasing the size of three gates, while the length of (at least) three related critical paths are decreased, although we are forced to revise two former substitution gates (we could substitute only gate i from the beginning if we would know that this gate will later belong to several related critical paths).

Now, if the gates j and k are such that some other gates have been revised at their stage of substitution, then the revision of the gates j and k might cause the restoration of those other gates, as we will see later.

While we substitute the gate i, we might or might not declare it as active for the specific set of gates. If we declare the gate i as active, we will later consider the possibility of its revision. If we do not declare i as active, then its size selection will certainly not be revised further since we do not doubt about its new estimating size selection.

The set of active gates of a gate i at iteration h is denoted by a_{ih}. Below we specify the set of gates for which we will declare a substitution gate as active considering three separate cases.

Case (a). At the initial stage 0, there is no substitution gate and therefore, there is no active gate, i.e., $a_{j0} = \emptyset, j \in V$. Suppose that $i \in P_0$ is a substitution gate at iteration 0. Assume that

$\tau(P_0) \geq \tau(P_1) \geq \tau(P_0)^1$

(the length of the critical path P_0 at iteration 1 is less than the length of the critical path at iteration 1 and therefore, the critical path at iteration 1 cannot be again P_0). In this case, we declare i as active for all $p \in P_0, p \neq i$ at iteration 1,

$a_{p1} := a_{p0} \cup \{i\}$.

We apply this rule at any further iteration h unless the corresponding substitution gate i is such that $a_{ih} \neq \emptyset$ (we consider this case below).

Case (b). Assume that i is the substitution gate at iteration h such that $a_{ih} \neq \emptyset$ but $a_{jh} = \emptyset$ for any other $j \in a_{ih}$.

Similarly to the case (a), we declare gate i as active for all gates $p \in P_h$, $(p \neq i)$ if the following additional condition holds:

$\sum_{k \in a_{ih}} \Delta\alpha_{kh(k)} < \Delta\alpha_{ih}$ (1)

While substituting gate i, we revise all gates $j \in a_{ih}$ (decreasing in this way the overall size) and all gates j become non-active at iteration $h + 1$, that is, we set

$a_{lh+1} := a_{lh} \backslash \{j\}$

for any l such that $j \in a_{lh}$. Now we declare gate i as active for $j \in a_{ih}$ if

$\sum_{k \in a_{ih} \backslash \{j\}} \Delta\alpha_{kh(k)} < \Delta\alpha_{ih}$ (2)

Finally, let $l \neq i$ be any gate such that $j \in a_{lh}$ for at least one $j \in a_{ih}$ and let $Q(i,l) = a_{ih} \cap a_{lh}$ (notice that $Q(i,l) \neq \emptyset$). We declare gate i as active for gate l if

$\sum_{k \in a_{ih} \backslash Q(i,l)} \Delta\alpha_{kh(k)} < \Delta\alpha_{ih}$ (3)

We declare a substitution gate i as active for gate p only if in the case of a later substitution of p, we will reduce the overall size if we revise gate i. When we revise gate

i, we restore all gates from a_{ih} which are not 'related' to the (new) substitution gate p (later we explain what is meant with 'related'). These sets of gates are specified in the left-hand sides of formulas (1)-(3).

Case (c). Assume that $a_{ih} \neq \emptyset$ and $a_{jh(j)} \neq \emptyset$ for at least one $j \in a_{ih}$.

First, we introduce the notion of the related gate set for i. The related gate set of i in relevance with $j \in a_{ih}$, $R_h(i,j)$ is the (possibly empty) set of gates that have been active for both gates i and j at the stage of the substitution of gate j:

$R_h(i,j) = a_{ih(j)} \cap a_{jh(j)}$.

The overall related gate set for gate i at iteration h is given by

$R_h(i) = \cup \{R_h(i,j) | j \in a_{ih}\}$.

The unrelated gate set for gate i in relevance with gate $j \in a_{ih}$ at iteration h is
$R_h^{-}(i,j) = \cup a_{jh(j)} \backslash R_h(i,j)$

and the overall unrelated gate set is

$R_h^{-}(i) = \cup \{R_h^{-}(i,j) | j \in a_{ih}\}$.

Here we give some intuitive explanation in connection with the introduced notions. Suppose that we substitute gate i revising (the former substitution) gate $j \in a_{ih}$ such that $a_{jh(j)} \neq \emptyset$. Let gate l be such that $l \in a_{jh(j)}$ but $l \notin a_{ih(j)}$, i.e., let $l \in R_h^{-}(i)$. Notice that $i \notin P_{h(l)}$ (otherwise gate l would be active for gate i at iteration $h(j)$). Then the revision of gate j will cause the increase of the critical path $P_{h(l)}$ (remind that gate l is also a former substitution gate and that it is revised at iteration h). So, the (new) substitution gate i is not related to the former substitution gate l and therefore, we restore its (last) substitution.

By our local balancing rule, whenever we substitute gate i, we revise gate $j \in aih(i)$ and restore all gates $l \in R_h^{-}(i,j)$.

Similar to the case (b), while substituting gate i, we revise all gates $j \in a_{ih}$ but in addition, we now apply the local balancing procedure and restore all gates $l \in R_h^{-}(i,j)$. Quite analogously to the case (b), we declare gate i as active for the gates of the type p, j and l (see the description above) if the following modifications of the formulas (1)-(3), respectively, hold:

$$\sum_{k \in a_{ih}} \left(\Delta\alpha_{kh(k)} - \sum_{q \in R^{-}{}_h(i,k)} \Delta\alpha_{qh(q)} \right) < \Delta\alpha_{ih} \qquad (1')$$

$$\sum_{k\in a_{ih\{j\}}}\left(\Delta\alpha_{kh(k)} - \sum_{p\in R^-{}_h(i,k)}\Delta\alpha_{ph(p)}\right) < \Delta\alpha_{ih} \quad (2')$$

$$\sum_{k\in a_{ih\setminus Q(i,l)}}\left(\Delta\alpha_{kh(k)} - \sum_{p\in R^-{}_h(i,k)}\Delta\alpha_{ph(p)}\right) < \Delta\alpha_{ih} \quad (3')$$

Now we want to extend our local balancing rule so that we can apply it for the case when, in general, $a_{lh(l)} \neq \emptyset$ $(l \in R^-{}_h(i,j))$, $a_{qh(q)} \neq \emptyset$ for $q \in R^-{}_{h(j)}(j,l)$, and so on.

Suppose that we apply the local balancing rule while substituting gate i at stage $h = h(i)$, revising gate j and restoring gate l correspondingly. Then we apply recursively the local balancing rule to gate l (whose size assignment, similarly to gate i, is increased) with a slight modification. Let us go back to the stage $h(j)$. At this stage, by the local balancing rule, the gate $q \in a_{lh(l)}$ would be restored or remained unchanged depending on whether $q \in R^-{}_{h(j)}(j,l)$ or $q \in R_{h(j)}(j,l)$, respectively. Therefore, while we restore gate l at stage h, we need to revise only those gates from $a_{lh(l)}$which were restored earlier, i.e., the gates from $q \in R^-{}_{h(j)}(j,l)$ (the restoration of gate l will cause the same effect as that at the stage of substitution of gate l except that when we restore gate l, we do not revise all gates $q \in a_{lh(l)}$ but only those gates q from the set $R^-{}_{h(j)}(j,l)$. We recursively continue an analogous procedure for the earlier stage (former) substitution gates until we reach the gates l^* with $a_{l^*h(l^*)} = \emptyset$.

The above extension of the local balancing rule is called the global balancing procedure according to which, whenever we substitute a gate, we rebalance, in the way explained above, the size assignments of the gates `involved' in this substitution.

The set of all gates involved in the global balancing procedure, corresponding to the substitution of gate i, can be represented in a useful way by a rooted tree $T(i,h)$, where the root represents the gate i. The set aih is represented by the successors of the node i, the set $a_{jh(j)}$ for a gate $j \in a_{ih}$ is represented by the successors of node j, and so on. The leaves of the tree $T(i,j)$ represent the gates k with $a_{kh(k)} = \emptyset$. We consider $T(i,h)$ as a symbolic representation of the fragment of the graph G_h. We do not put arcs in $T(i,h)$ since this could obviously violate the structure of G (not all successors of a node in $T(i,h)$ are real successors of this node in G). However, we can still consider the paths in $T(i,h)$ passing through the nodes related to the set of active gates and thus involved in the global balancing procedure. For example, we have $l \in R_h(i,j)$ or $l \in R^-{}_h(i,j)$ depending on whether $i \in P_{h(l)}$ or $i \notin P_{h(l)}$, respectively (notice that $j \in P_{h(l)}$).

To reflect the overall size variations caused by the global balancing procedure, we introduce the following notions.

Suppose that gate i has been a substitution gate. Then the perturbed balance coefficient for a gate $j \in a_{ih(i)}$ in relevance with gate i is recursively defined as follows:

$$\tilde{b}_{jh(i)}(i) = \begin{cases} \Delta\alpha_{jh(j)} & If\ a_{jh(j)} = \emptyset \\ \Delta\alpha_{jh(i)} - \sum_{k \in a_{jh(j)} \setminus R_{h(i)}(i,j)} \Delta\alpha_{kh(k)} & If\ a_{jh(j)} \neq \emptyset\ \&R^{-}{}_{h(j)}(j) = \emptyset \\ \Delta\alpha_{jh(j)} - \sum_{k \in a_{jh(j)} \setminus R_{h(i)}(i,j)} \Delta\alpha_{kh(k)} + \sum_{l \in R^{-}{}_{h(j)}(j)} \tilde{b}_{lh(l)}(k) & Otherwise \end{cases}$$

The balance coefficient b_{ih} reects the overall size shift which will be caused by the substitution of gate i:

$$b_{ih} = \begin{cases} \Delta\alpha_{ih} & If\ a_{ih} = \emptyset \\ \Delta\alpha_{ih} - \sum_{k \in a_{ih}} \Delta\alpha_{kh(k)} & If\ a_{ih} \neq \emptyset \& R^{-}{}_{h}(i) = \emptyset \\ \Delta\alpha_{ih} - \sum_{k \in a_{ih}} \Delta\alpha_{kh(k)} + \sum_{l \in R^{-}{}_{h}(i,k)} \tilde{b}_{lh(l)}(j) & Otherwise \end{cases}$$

Using the introduced notations, we can rewrite formulas (1')-(3') for the general case:

$$\sum_{k \in a_{ih}} \tilde{b}_{kh}(i) < \Delta\alpha_{ih} \quad (1^*)$$

$$\sum_{k \in a_{ih} \setminus \{j\}} \tilde{b}_{kh}(i) < \Delta\alpha_{ih} \quad (2^*)$$

$$\sum_{k \in a_{ih} \setminus Q(i,l)} \tilde{b}_{kh}(i) < \Delta\alpha_{ih} \quad (3^*)$$

The sets of active gates, generated in accordance with the points (a)-(c) might be corrected during the execution of the algorithm. We specify this in points (i)-(iii) below.

Let i be a substitution gate at iteration h. Then:

(i). The set of active gates of i is released by the stage $h + 1$, i.e.,

$a_{ih+1} := \emptyset$.

(ii). If, in addition, $\acute{h} > h$ is the earliest iteration such that, first, $i \in P_{\acute{h}}$ and second, i is not revised at any of the iterations $h + 1, h + 2, \ldots, \acute{h}$, then the last size assignment of gate i, $\alpha_{i,\gamma^*(i,h)}$ will not be further revised. So, we set

$$a_{k\acute{h}+1} := a_{k\acute{h}\setminus\{j\}}$$

for any k such that $i \in a_{k\acute{h}}$.

(iii). For any gate j to which, at iteration h, the two above points are not applicable, we set

$a_{jh+1} := a_{jh}$.

The following lemma shows that any restoration in the global balancing procedure increases the overall size.

Lemma 1. For any gates k and i, we have $\acute{b}_{kh(i)} \geq 1$.

Proof. The case $a_{kh(k)} = \emptyset$ is obvious. Suppose that $a_{kh(k)} \neq \emptyset$ and $R^-_{h(k)}(k) = \emptyset$. Observe that gate k has been declared as active for gate i at the iteration of its substitution. So, we could have three possibilities at iteration $h(k)$: Gate i could be of the type p, j or l, and one of the formulas (1), (2) or (3), respectively, had to be satisfied (see the case (b) above). Therefore, when we restore gate j in the global balancing procedure, the gates, which will be revised, are those specified in the left-hand sides of the formulas (1)-(3), respectively, and our claim is obvious.

The case $R^-_{h(k)}(k) \neq \emptyset$ now follows recursively from the de_nition of $\tilde{b}_{jh}(i)$ and the formulas (1*)-(3*).

From *Lemma 1* and the definition of the global balancing procedure, we get the following. *Corollary 1. Restoration (revision, respectively) of any gate in the global balancing procedure increases (decreases, respectively) the overall size.*

Lemma 2. Inequality

$\left|R^-_{h(i)}(i,k)\right| \leq max\{\alpha_{ih} - \alpha_{ip} | i \in V, p, l = 0,1, \dots, m, p < l\} + 1$ holds for any gates i, k such that $R^-_{h(i)}(i,k)$ is defined.

Proof. Since $R^-_{h(i)}(i,k)$ is defined, by definition, the gate k has to be declared as active for gate i at iteration $h(k)$. At that iteration, gate i could be a gate of one of the types p, j or l (see the proof of Lemma 1) and one of the formulas (1*), (2*) or (3*), respectively, had to be satisfied. Now, notice that the elements in the sum of the left-hand sides of the formulas (1*)-(3*) constitute exactly the set $R^-_{h(i)}(i,k)$, and our claim now follows from Lemma 1.

Consider the node $k \in P_h$. In general, we have
$k \in P_{h_1}, k \in P_{h_2}, \dots, k \in P_{h_p}, h_1, h_2, \dots, h_p \leq h$.
Suppose that the condition of proximity of these critical paths holds (see the definition of a active node). Then, by substituting the node k and revising all $l \in a_{kh}$, we are able to

decrease simultaneously all critical paths $P_{h_1}, P_{h_2}, P_{h_p}$ while increasing the overall size ones. In contrast to this fact, we could substitute for each critical path different nodes increasing the overall size several times (although decreasing the lengths of the paths further). We control this balance by means of the rate functions.

We define the rate function $r_{ih}, i \in P_h$ as follows:

$$r_{ih} = \begin{cases} \frac{\Delta\delta_{ih}}{(b_{ih}+1)} & If\ b_{ih} > 0 \\ \Delta\delta_{ih} & If\ b_{ih} = 0 \\ \Delta\delta_{ih}(b_{ih}+1) & If < 0. \end{cases}$$

It should be clear that the `profitability' of a particular node depends not only on its own characteristics but also on the structure of the delay graph and the distribution of the critical paths on it.

On the way of improving the initial solution σ_0, we iteratively increase the sizes of particular nodes decreasing in this way their delays. As a result, one or more critical paths are becoming non-critical, but we obtain new critical paths. Since we have an overall size bound, we are interested in the maximum `relative decrease' of the delay on one unit of increased size. This relationship is measured by the above rate function.

Thus, the main strategy of the algorithm is roughly the following: find not the substitution which maximally decreases the length of P_h but to substitute the node which will decrease the delay possibly less but the `overall decrease' of the delay on the unit of increased size will be the best. Thus, we prefer to make `smaller' improvements which are correct in the above sense. Below we give the general description of the algorithm.

Algorithm

Step 0

Construct the initial solution σ_h;

$h := 1$.

Step 1

Determine the critical path P_h and $i \in P_h$ with

$r_{ih} = max\{r_{kh} | k \in P_h\}$;

Substitute gate i;

Perform chain-revised and reverse-revised operations for gate i;

Determine the set $A_h(i)$.

Step 2

If $B_h < B_{max}$, then $h := h + 1$ and go to Step 1

else stop $\{\sigma_h \; is \; a \; \Delta - approximate \; solution\}$.

4. Concluding Remarks

In this paper, we have considered a problem arising in the design of VLSI circuits. In particular, we presented an approximate algorithm for determining the size of the gates which has the following properties:

1. Let σ_{opt} be an optimal solution, σ the solution determined by the above algorithm, and $\Delta = \max\{\delta_{ik} - \delta_{il} | i \in V, k, l \in \{1, \dots, m\} k \neq l\}$.

For the algorithm presented above, we obtained the estimation

$$\tau(\sigma) \leq \tau(\sigma_{opt}) + \Delta,$$

where $\tau(\sigma)$ denotes the length of a critical path in the graph G_σ.

2. Let

$$c = \left\lceil \frac{\max\{\delta_{ik} - \delta_{il} | i \in V, k, l \in \{1, \dots, m\}, k \neq l\}}{\min\{\alpha_{ik} - \alpha_{il} | i \in V, k, l \in \{1, \dots, m\}, k \neq l\}} \Big/ \frac{\min\{\delta_{ik} - \delta_{il} | i \in V, k, l \in \{1, \dots, m\}, k \neq l\}}{\max\{\alpha_{ik} - \alpha_{il} | i \in V, k, l \in \{1, \dots, m\}, k \neq l\}} \right\rceil + 1.$$

Then the running time of the algorithm is bounded by

$$c^2 B_{max} n^3 log n.$$

We also note that for taking into account the particular sizes of the predecessor nodes in determining the additional time delays caused by the gate i, we can extend the model considered by allowing for each gate $i \in V$ the time delays $\acute{\delta}_{i j_1 j_2}, j_1, j_2 \in [1, m]$, where j_1 and j_2 are the size selections of the gates i and $pred(i)$, respectively.

Acknowledgements

This work has been supported by Deutscher Akademischer Austauschdienst (DAAD) and by CONACyT grant 160162. The first author is grateful to Prof. Dr. Gunter Hotz, who suggested the problem, for many stimulating discussions.

References

[1] Areibi, S. and Yang, Z., 2004 Effective memetic algorithms for VLSI design = Genetic algorithms + Local search + Multi-level clustering, *Evolutionary Computation*, **12**, 327-353.

[2] Ayob, M.N., Yusof, Z.M., Adam, A., Abidim, A.F.Z. and Ibrahim, I., 2010, A particle swarm optimization approach for routing in VLSI, *2nd International Conference on Computational Intelligence Communication*, 49 - 53.

[3] Beeftink, F., Kudva, P., Kung, D. and Stok, L., 1998, Gate-size selection for standard cell libraries, *Proceedings* ICCAD, 545-550.

[4] Behjat, L., Vanelli, A. and Rosehart, W., 2005, Integer linear programming models for global Routing, *INFORMS Journal on Computing*, **18**, 137 -150.

[5] Borah, M., Owens, R.M. and Irwin, M.J., 1995, Transistor sizing for minimizing power consumption of CMOS circuits under delay constraint, *International Symposium on Low Power Design*, 167 -172.

[6] Chuang, W., Sapatneka, S. , Hajj, I.N., 1995, Timing and area optimization for standard-cell VLSI circuit design, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **14**, 308 - 320.

[7] Coudert, O. , Haddad, R. and Manne, S., 1996, New algorithms for gate sizing. A comparative study, *Proceedings of 33rd IEEE/ACM Design Automation Conference*, 734 - 739.

[8] Deza, A., Dickson, C., Terlaky, T., Vanelli, A. and Zhang, H., 2010, Global routing in VLSI design: Algorithm, theory and computational practice, *Optimization Online*, Manuscript, 24 pages.

[9] Dickson, C., 2007, Global routing in VLSI: Algorithms, theory, and computation, *Master Thesis*, McMaster University, 64 p.

[10] Fishburn. J.P., 1992, LATTIS: an iterative speedup heuristic for mapped logic, *Proceedings* of 29th ACM/IEEE Design Automation Conference, 488 - 491.

[11] Geiger, R. L., Allen, P.E. and Strader, N.R., 1984, VLSI design techniques for analog and digital circuits, MacGraw-Hill.

[12] Joshi, S. and Boyd, S., 2008, An efficient method for large-scale gate sizing, *IEEE Transactions on Circuits and Systems*, **55**, 2760 - 2773.

[13] Khalil-Hanui, M. and Shaik-Husin, N., 2009, An optimization algorithm based on grid-graphs for minimizing interconnect delay in VLSI layout design, *Malaysian Journal of Computer Science*, **22**, 19 - 33.

[14] Lee, C.M. and Soukop, H., 1984, An algorithm for CMOS timing and area optimization, *IEEE Journal of Solid-State Circuits*, **19, 5**, 781 - 787.

[15] Lienig, J. and Thulasariman, K., 1993, A genetic algorithm for channel routing in VLSI circuits, *Evolutionary Computation*, **1**, 203 - 311.

[16] Mazumder, P. and Rudnick, E., 1998, Genetic algorithms for VLSI design, layout and test automation, Prentice Hall.

[17] Sherwani, N., 1999, Algorithms for VLSI physical design automation, *Kluwer Academic Publishers*, Dordrecht.

[18] Shyu, J.M., Sangiovanni-Vincentelli, A., Fishburn, J.P.and Dunlop, A.E., 1988, Optimization based transistor sizing, *IEEE Journal of Solid-State Circuits*, **23, 2**, 400 - 409.

[19] Szeszler, D., 2005, Combinatorial algorithms in VLSI routing, *Ph. D. Thesis*.

[20] Terlaki, T., Vanelli, A. and Zhang, H., 2008, On routing in VLSI design and communication networks, *Discrete Applied Mathematics*, **156**, 2178 -2194.

[21] Yang, Z., Areibi, S. and Vanelli, A., 2007, An ILP based hierarchical global routing approach for VLSI ASIC design, *Optimization Letters*, **1**, 281 - 197.