Optimization of Dual-Threshold Circuits

Konrad Engel Thomas Kalinowski Roger Labahn
Institut für Mathematik, Universität Rostock, 18051 Rostock, Germany
{konrad.engel, thomas.kalinowski, roger.labahn}@uni-rostock.de

Frank Sill Dirk Timmermann
Institut für Angewandte Mikroelektronik und Datentechnik, Universität Rostock,
Richard-Wagner Straße 31, 18119 Rostock-Warnemünde, Germany
{frank.sill, dirk.timmermann}@uni-rostock.de

May 04, 2005

Abstract

In this paper, we consider an optimization problem on directed acyclic graphs which is motivated by a standard task in low power VLSI design. With each vertex v of a directed acyclic graph, we associate two delay values $d_0(v) \leq d_1(v)$ and two leakage values $c_0(v) \geq c_1(v)$. The objective is to choose one of the indices 0 or 1 for each vertex, such that in first instance the corresponding total delay along any directed path is minimal, and in second instance the total leakage is minimized. We prove that a very restricted special case of this problem already is NP-hard, and we present four different heuristic approaches to the problem. Further, we test our algorithms on ISCAS85 benchmark circuits.

MSC: 90C35, 06A07, 94C15

Keywords: directed acyclic graphs, Sperner family, cutset, low-power design, leakage power, threshold voltage, timing constraints, VLSI CAD

1 Introduction

Small mobile applications are the current driving power for the chip industry. But the freedom from cables requires integrated circuits with small running times and low power dissipation. Furthermore, high integration density and high performance are desired. This resulted in an aggressive downscaling per each technology generation, and lowering of the supply voltage. To meet the performance requirements, the transistor threshold voltage has to be scaled down. Unfortunately, such scaling increases the sub-threshold leakage current, thereby increasing leakage power. Thus, in current technologies up to 50% of the power dissipation is due to leakage currents [9].

There are several ways to handle the leakage power problem in high performance applications. Kao et al. [6] propose to use additional sleep transistors, which can disconnect the supply from the circuit in idle modes. Several authors [1, 3, 15, 19, 20] considered the usage of multiple supply voltages. The dynamic scaling of supply voltage adjusted to required performance was proposed Maken et al. [12]. Another approach is the application of two device types, which vary in their threshold voltages. This Dual Threshold CMOS (DTCMOS) design technique uses fast low threshold voltage (LTV) and slow high threshold voltage (HTV) devices. The advantage of HTV devices is their small leakage current. Thus, the aim of DTCMOS is to maximize the gain in leakage at the HTV devices without worsening the performance of the circuit. The problem of choosing adequate values for the two threshold voltages is adressed in [5, 11]. We assume these values are fixed, so our task is to assign one of two given threshold voltages to each gate. Li et al. [10] have shown that already very special problems of assigning dual threshold voltages to devices are NP-complete. Several heuristics for this problem have been proposed in the literature. Li et al. [10] replace step by step LTV devices by HTV devices using different score functions for choosing the gate that is replaced. Sundararajan and Parhi [18] consider the set of gates that can potentially be implemented as HTV devices and determine an adequate subset by solving an integer linear programming problem (ILP). In the heuristic variant of their algorithm solving of the ILP is replaced by a sequence of solvings of an LP-relaxation. The algorithm of Wei et al. [21, 22] starts with all gates at low threshold voltage, visits the gates in a breadth first order, and switches a gate to high threshold voltage if this does not violate the timing constraint. Sill et al. [16] present a simple algorithm, which sequentially scans the circuit and transfers every device in critical paths into an LTV device. Several authors [7, 8, 13, 17, 23] propose combinations of threshold voltage assignment and device sizing.

As usual, also in this paper the circuit is modeled as a directed acyclic graph (dag), where for each vertex we have to decide wether it operates on high or on low threshold voltage. The objective is to make this decision in such a way that the performance of the circuit is optimal and the leakage is minimized. The paper is organized as follows. In Section 2, we formally state the problem as an optimization problem on a dag, and we give a proof of the NP-completeness of a rather restricted version of it. In Sections 3-5, we propose four heuristic algorithms for the selection of the HTV devices, and finally in Section 6, we present some test results on benchmark circuits.

2 Mathematical formulation of the problem

Let $G = (V, E)$ be a dag and let $d_i, c_i : V \to \mathbb{R}_+$ $(i = 0, 1)$ be weight functions on the vertex set such that for all $v \in V$, $d_0(v) \leq d_1(v)$ and $c_0(v) \geq c_1(v)$. $d_0(v)$ and $d_1(v)$ can be interpreted as the delays of vertex v, where v is of LTV and HTV type, respectively. Similarly, $c_0(v)$ and $c_1(v)$ are the corresponding leakage values of vertex v. In the following we use graph theoretic terminology and speak of length and cost instead of delay and leakage. Throughout we assume that G has a unique source q and a unique sink s with $d_0(v) = d_1(v) = c_0(v) = c_1(v) = 0$ for $v \in \{q, s\}$. This can always be done without loss of generality by adding vertices q and s, connecting q to all sources of G (primary inputs) and connecting all sinks of G (primary outputs) to s. Let $x : V \to \{0, 1\}$ be a decision function. The pair (G, x) is called a *realization.* The *cost* of the realization is given by

$$c(G, x) := \sum_{v \in V} c_{x(v)}(v).$$

For a path $P = (v_0, \ldots, v_k)$ in G and a realization (G, x), the *length* of P is defined by

$$d(P, x) := \sum_{v \in P} d_{x(v)}(v).$$

The *length* of the realization (G, x) is given by

$$d(G, x) := \max\{d(P, x) \ : \ P \text{ is a } q\text{-}s\text{-path}\}.$$

Let x_0 be the zero function, i.e. $x_0(v) = 0$ for all $v \in V$. Clearly, (G, x_0) realizes the minimum length $d_{min}(G) := d(G, x_0)$. The *dual-threshold complementary metaloxid semiconductor problem* is the following:

DTCMOS: Find a decision function $x : V \to \{0, 1\}$ such that $d(G, x) = d_{min}(G)$ and $c(G, x)$ is minimum.

Let $c^*(G) = \sum_{v\in V} c_1(v)$. If we replace the cost functions c_0, c_1 by new functions c'_0, c'_1 defined by $c'_0(v) := c_0(v) - c_1(v)$ and $c'_1(v) := c_1(v) - c_1(v) = 0$, then the new cost is given by $c'(G,x) = c(G,x) - c^*(G)$, and $c'(G,x)$ is minimum iff $c(G,x)$ is minimum. Hence we may assume w.l.o.g. that $c_1(v) = 0$ for all $v \in V$. For brevity we set $c(v) := c_0(v)$ and obtain

$$c(G,x) = \sum_{v\in V : x(v)=0} c(v).$$

Theorem 1. DTCMOS *is NP-hard.*

Proof. We prove the theorem by reduction from 0-1-knapsack. Consider an arbitrary instance $((a_1,\dots,a_n),\ (c_1,\dots,c_n),\ B)$ of 0-1-knapsack, i.e. a_i, c_i $(i = 1,\dots,n)$ and B are positive integers, and the task is to determine an optimal solution $\boldsymbol{x} = (x_1,\dots,x_n) \in \{0,1\}^n$ of the problem

$$\sum_{i=1}^{n} a_i x_i \le B, \qquad \sum_{i=1}^{n} c_i x_i \to \max\,. \tag{1}$$

Now we define a dag $G = (V,E)$ with (see Figure 1)

$$\begin{aligned} V &= \{v_0 = q, v_1, \dots, v_n, v_{n+1} = s, v_{n+2}\},\\ E &= \{v_0v_1, v_1v_2, \dots, v_nv_{n+1}, v_0v_{n+2}, v_{n+2}v_{n+1}\}. \end{aligned}$$

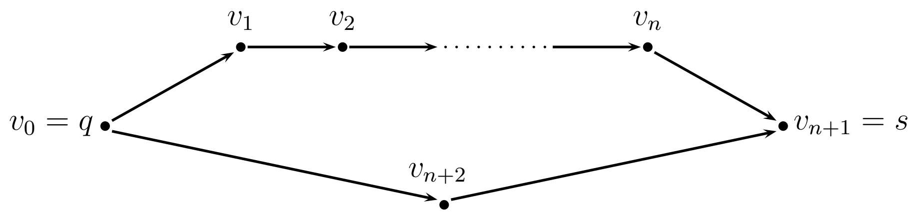

Figure 1: The dag $G = (V,E)$ for the reduction of 0-1-knapsack.

Let

$$d_0(v_i) = \begin{cases} 0 & \text{if } 0 \le i \le n+1,\\ B & \text{if } i = n+2, \end{cases} \qquad d_1(v_i) = \begin{cases} a_i & \text{if } 1 \le i \le n,\\ 0 & \text{if } i \in \{0, n+1\},\\ B+1 & \text{if } i = n+2, \end{cases}$$

$$c(v_i) = \begin{cases} c_i & \text{if } 1 \le i \le n,\\ 0 & \text{otherwise.} \end{cases}$$

Obviously, $d_{min}(G) = B$ and $x : V \to \{0,1\}$ is a solution of **DTCMOS** iff $x_i = x(v_i)$ $(i = 1,\ldots,n)$ gives an optimal 0-1-solution of the problem

$$\sum_{i=1}^{n} a_i x_i \le B, \qquad \sum_{i=1}^{n} c_i(1 - x_i) \to \min,$$

which is equivalent to (1). □

If we replace the condition $d(G,x) = d_{min}(G)$ by the weaker condition that $d(G,x)$ is bounded from above by some input constant, we come (with some change of notation) to the basic circuit implementation problem which was shown to be NP-hard by Li et al. [10]. Theorem 1 gives reason to look for well-founded heuristics for **DTCMOS**. Though the problem can be formulated as a 0-1-linear programming problem ILP-solvers are not the adequate tool since the graph can be very large, more than 50,000 vertices in some cases. We present four such heuristics and compare their performance on practical examples. But first we need some more notation. For $x : V \to \{0,1\}$ and $v \in V$, we put

$$\begin{aligned} e(v,x) &:= \max\{d(P,x) \ : \ P \text{ is a } q\text{-}v\text{-path}\} - d_{x(v)}(v), \\ h(v,x) &:= \max\{d(P,x) \ : \ P \text{ is a } v\text{-}s\text{-path}\}. \end{aligned}$$

The values $e(v,x)$ and $h(v,x)$ can be easily obtained by a recursive version of depth first search (DFS) in time $O(|E|)$ (cf. Sedgewick [14], 105 ff.). Clearly,

$$d(G,x) = \max\{e(v,x) + h(v,x) \ : \ v \in V\} = e(s,x).$$

Interpreting the vertices as jobs, where an edge vw indicates that job v has to be completed before job w can be started, the value $e(v,x)$ can be considered as the earliest starting time for job v, while $d(G,x) - h(v,x)$ is the latest starting time under the timing constraint. The *slack* at vertex v is defined by $slack(v,x) := d(G,x) - (e(v,x) + h(v,x))$. For the sake of brevity we put $dif(v) := d_1(v) - d_0(v)$.

3 The single and the multiple switch algorithm

Let v be a vertex with $slack(v,x) \ge dif(v)$ and $x(v) = 0$. Our first lemma states that one can switch $x(v)$ to 1 without changing the length of the realization. This lemma is implicitly contained in several papers, e.g. [2, 3, 10].

Lemma 1. *Let v be a vertex with $slack(v,x) \geq dif(v)$, $x(v) = 0$ and let $x' : V \to \{0,1\}$ be given by $x'(v) := 1$ and $x'(w) := x(w)$ for $w \in V \setminus \{v\}$. Then*

$$d(G,x') = d(G,x) \quad \textit{and} \quad c(G,x') \leq c(G,x).$$

Proof. Since $d_1(v) \geq d_0(v)$ and $c(v) \geq 0$, obviously

$$d(G,x') \geq d(G,x) \quad \text{and} \quad c(G,x') \leq c(G,x).$$

Let P be an arbitrary q-s-path. It remains to show that $d(P,x') \leq d(G,x)$. If P does not contain v, then clearly

$$d(P,x') = d(P,x) \leq d(G,x).$$

So assume $v \in P$. Then

$$\begin{aligned} d(P,x') &= d(P,x) + dif(v) \\ &\leq e(v,x) + h(v,x) + dif(v) \\ &= d(G,x) - slack(v,x) + dif(v) \leq d(G,x). \end{aligned}$$

□

A vertex is called a *candidate vertex* if its decision can be switched from 0 to 1 according to Lemma 1. The *candidate set* C is the set of all candidate vertices, i.e.

$$C := \{v \in V \ : \ x(v) = 0 \text{ and } slack(v,x) \geq dif(v)\}.$$

Lemma 1 can be generalized as follows.

Lemma 2. *Let $S \subseteq V$ be a subset of vertices such that $x(v) = 0$ and $slack(v,x) \geq \sum_{w \in S} dif(w)$ for all $v \in S$. Moreover, let*

$$x'(w) := \begin{cases} 1 & \textit{if } w \in S, \\ x(w) & \textit{otherwise.} \end{cases}$$

Then

$$d(G,x') = d(G,x) \quad \textit{and} \quad c(G,x') \leq c(G,x).$$

Proof. As in Lemma 1 we only have to prove that for any q-s-path P, $d(P,x') \leq d(G,x)$. Again the remaining interesting case is that P contains vertices from S. For a fixed vertex v from $P \cap S$, we have

$$\begin{aligned} d(P, x') &= d(P, x) + \sum_{w \in P \cap S} dif(w) \\ &\leq d(P, x) + \sum_{w \in S} dif(w) \\ &\leq e(v, x) + h(v, x) + \sum_{w \in S} dif(w) \\ &= d(G, x) - slack(v, x) + \sum_{w \in S} dif(v) \leq d(G, x). \end{aligned}$$

$\square$

Lemmas 1 and 2 are fundamental for the two algorithms in this section. In both algorithms, one starts with the zero function x_0 and then successively switches one or more vertices $v \in V$ from $x(v) = 0$ to $x(v) = 1$ without changing the length but decreasing the cost. The choice of such vertices can be done in the following way: from a heuristic point of view it is wise to take vertices with large slack (they do not have much influence on other vertices) and with large cost (they contribute more to decrease the cost). Let $f : \mathbb{R}_+^2 \to \mathbb{R}_+$ be a function that is increasing in both variables, e.g. $f(x, y) = x + \lambda y$ ($\lambda \in \mathbb{R}_+$), $f(x, y) = xy$ or $f(x, y) = x\sqrt{y}$. It is used to evaluate the vertices of the candidate set C by

$$\begin{aligned} weight(v) &:= f(\, slack(v, x),\, c(v)\,) \qquad \text{or} \\ weight(v) &:= f(\, slack(v, x) - dif(v),\, c(v)\,). \end{aligned}$$

In our implementation we worked with

$$weight(v) = slack(v, x) - dif(v) + \lambda c(v). \tag{2}$$

Other score functions for ranking the candidate vertices have been considered e.g. by Li et al. [10]. In the first algorithm we proceed by switching one single vertex of maximal weight in each step.

Single Switch Algorithm (SSA)
INPUT: $G = (V, E)$, $c(v)$, $d_i(v)$ ($i = 1, 2$, $v \in V$)
OUTPUT: decision function $x : V \to \{0, 1\}$
Initialization
$x := x_0$
Iteration
do

determine $slack(v, x)$ for all $v \in V$
determine the candidate set C
if ($C \neq \emptyset$)
compute $weight(v)$ for all $v \in C$
choose a vertex $v \in C$ with maximal weight
$x(v) := 1$
while $C \neq \emptyset$
return x

Let $n := |V|$. Under the realistic assumption that the vertex degrees are bounded by a constant each iteration step has time complexity $O(n)$. So the total time complexity is $O(n^2)$. In practical examples, we observed that the number of switched vertices grows linearly with n. So for large n, it is better to switch more than one vertex in each iteration step. For this, we order the vertices of the candidate set C by decreasing weight and take a large initial segment of this ordering for the set S used in Lemma 2. This initial segment can be determined step-by-step as follows. Let $C = \{v_1, \ldots, v_k\}$ with $weight(v_1) \geq \cdots \geq weight(v_k)$. At stage $S = \{v_1, \ldots, v_l\}$ $(l < k)$, we can insert v_{l+1} into S if $slack(v_i, x) \geq \sum_{j=1}^{l+1} dif(v_j)$ is true for all $i = 1, \ldots, l+1$. With

$$sum_l := \sum_{j=1}^{l} dif(v_j),$$

$$gap_l := \min\{slack(v_i, x) - sum_l \; : \; i = 1, \ldots, l\}$$

for $(l = 1, \ldots, k)$, this is equivalent to $gap_{l+1} \geq 0$. Obviously $sum_{l+1} = sum_l + dif(v_{l+1})$ and $gap_{l+1} = \min\{gap_l - dif(v_{l+1}),\ slack(v_{l+1}, x) - sum_{l+1}\}$. This leads to the following procedure.

Initial Segment Procedure (ISP)
INPUT: $C = \{v_1, \ldots, v_k\}$ with $weight(v_1) \geq \cdots \geq weight(v_k)$
OUTPUT: set S of vertices that can be switched
Initialization
$S := \{v_1\}$, $gap := slack(v_1, x) - dif(v_1)$, $sum := dif(v_1)$, $l := 1$
Iteration
do
$l := l + 1$
$sum := sum + dif(v_l)$
$gap := \min\{gap - dif(v_l),\ slack(v_l, x) - sum\}$
if $gap \geq 0$
$S := S \cup \{v_l\}$

while ($gap \geq 0$ and $l < k$)
return S

To order the set C one needs time $O(k \log k)$. From a practical point of view, $\log k$ can be considered constant, but moreover, one can keep C "small", e.g. by working only with the smaller set

$$C' = \{v \in C \ : \ weight(v) \geq \overline{weight}\},$$

where $\overline{weight}$ is the average weight of a vertex of C,

$$\overline{weight} := \frac{1}{|C|} \sum_{v \in C} weight(v).$$

This reduces the required time and appeared to be practically sufficient. The whole algorithm reads:

Multiple Switch Algorithm (MSA)
INPUT: $G = (V, E)$, $c(v)$, $d_i(v)$ $(i = 1, 2,\ v \in V)$
OUTPUT: decision function $x : V \to \{0, 1\}$
Initialization
$x := x_0$
Iteration
do
  determine $slack(v, x)$ for all $v \in V$
  determine the candidate set C
  if ($C \neq \emptyset$)
    compute $weight(v)$ for all $v \in C$
    $\overline{weight} := \frac{1}{|C|} \sum_{v \in C} weight(v)$
    $C' := \{v \in C \ : \ weight(v) \geq \overline{weight}\}$
    order C' by decreasing weight
    determine S using **ISP**
    put $x(v) := 1$ for all $v \in S$
while $C \neq \emptyset$
return x

We cannot prove a time complexity better than $O(n^2)$, but in practical examples we observed an essential acceleration compared to SSA.

4 The k-family algorithm

As in the previous section we start with the zero function x_0 and switch a set of vertices from the candidate set C to $x(v) = 1$ keeping the length $d(G, x)$

invariant. Let
$$maxdif(G) := \max\{dif(v) \ : \ v \in V\}.$$
We may assume $maxdif(G) > 0$, because otherwise an optimal solution is given by $x(v) = 0$ for all $v \in V$. For any real number $\kappa \geq 0$, let
$$k := \max\{\lfloor\kappa\rfloor, 1\}$$
and define a partially ordered set (poset) $(Q_x^\kappa, \leq)$ by
$$Q_x^\kappa := \begin{cases} \{v \in V \ : \ x(v) = 0 \text{ and } slack(v,x) \geq \kappa \cdot maxdif(G)\} & \text{if } \kappa > 1, \\ C & \text{if } \kappa \leq 1, \end{cases} \tag{3}$$
$$\text{and } v \leq w \ :\Leftrightarrow \ \text{There is a } v\text{-}w\text{-path in } G. \tag{4}$$
A subset $S \subseteq Q_x^\kappa$ is called a *k-family* in $(Q_x^\kappa, \leq)$ if there is no chain in $(Q_x^\kappa, \leq)$ containing more than k elements from S.

Lemma 3. *Let S be a k-family in $(Q_x^\kappa, \leq)$ and let*
$$x'(w) := \begin{cases} 1 & \textit{if } w \in S, \\ x(w) & \textit{otherwise.} \end{cases}$$
Then
$$d(G, x') = d(G, x) \quad \textit{and} \quad c(G, x') \leq c(G, x).$$

Proof. The case $k = 1$ can be proved as Lemma 1. So let $k > 1$. As in the proofs of Lemmas 1 and 2 we only have to prove that for any q-s-path P, we have $d(P, x') \leq d(G, x)$, and it is sufficient to consider a path P containing at least one vertex $v \in S$. By definition of a k-family, P contains at most k elements of S. We proceed as in the proof of Lemma 2 and obtain
$$\begin{aligned} d(P, x') &\leq d(G, x) - slack(v, x) + \sum_{w \in S \cap P} dif(w) \\ &\leq d(G, x) - slack(v, x) + k \cdot maxdif(G) \\ &\leq d(G, x) - slack(v, x) + \kappa \cdot maxdif(G) \leq d(G, x). \end{aligned}$$
□

Switching the elements of a k-family S reduces the cost by $\sum_{v \in S} c(s)$, hence it is wise to choose a k-family of maximum cost. For the same reason as in Section 3 it is helpful to replace $c(v)$ by a weight $weight(v)$, where e.g.
$$weight(v) = 1 + maxdif(G) - dif(v) + \lambda c(v). \tag{5}$$

So we have reduced the original problem to the task of finding a k-family of maximum weight. A min-cost-flow algorithm for this problem is described in Chapter 4 of [4]. If $|Q_x^\kappa| \leq k$, clearly the whole set Q_x^κ can be chosen. It remains to describe the choice of κ. In order to keep the computing time small it is desirable to work with a reasonably small poset Q_x^κ. This can be achieved by prescribing some bound B, e.g. $B = 100$, and requiring that the cardinality of Q_x^κ is not much greater than B. Let $C = \{v_1, \ldots, v_l\}$ with $slack(v_1, x) \geq \cdots \geq slack(v_l, x)$ and let $j = \min\{B, l\}$. Now for

$$\kappa := \frac{slack(v_j, x)}{maxdif(G)}$$

by (3) it follows that $\{v_1, \ldots, v_j\} \subseteq Q_x^\kappa$ and for $k > 1$ only those vertices v_i with $i > j$ belong to Q_x^κ for which $slack(v_i, x) = slack(v_j, x)$. This leads to the following algorithm.

k-Family Algorithm (k-FA)
INPUT: $G = (V, E)$, $c(v)$, $d_i(v)$ ($i = 1, 2$, $v \in V$), bound B
OUTPUT: decision function $x : V \rightarrow \{0, 1\}$
Initialization
$x := x_0$
Iteration
do
  determine $slack(v, x)$ for all $v \in V$
  determine the candidate set C
  if $(C \neq \emptyset)$
    order C by decreasing values of $slack(v, x)$
    pick w as the element at position $\min\{B, |C|\}$ in C
    $\kappa := \frac{slack(w,x)}{maxdif(G)}$
    $k := \max\{\lfloor\kappa\rfloor, 1\}$
    determine Q_x^κ according to (3)
    compute $weight(v)$ for all $v \in Q_x^\kappa$
    determine a k-family S of maximum weight in $(Q_x^\kappa, \leq)$
    put $x(v) := 1$ for all $v \in S$
while $C \neq \emptyset$
return x

By Lemma 3 the length $d(G, x)$ is always equal to $d_{min}(G)$ and as long as $C \neq \emptyset$, in each loop at least one more vertex switches to $x(v) = 1$. So the algorithm terminates after finitely many steps. As in MSA algorithm one could reduce the ordering of the candidate vertices to a certain subset $C' \subseteq C$.

5 The k-cutset algorithm

Again, we start with the zero function x_0. Assume that after some steps, we have a realization (G, x) with $d(G, x) = d_{min}(G)$. As in the previous sections we work with the set C of candidate vertices. We switch them all simultaneously, i.e. we put

$$x'(v) := \begin{cases} 1 & \text{if } v \in C, \\ x(v) & \text{otherwise.} \end{cases}$$

Clearly this improves the cost, i.e. $c(G, x') \leq c(G, x)$, but in general, we also have $d(G, x') > d(G, x)$. Hence we have to switch back some vertices from $x'(v) = 1$ to $x'(v) = 0$ in order to obtain the minimum length $d_{min}(G)$. Now we change notation, and assume that (G, x) is a realization with $d(G, x) > d_{min}(G)$. We describe a step by which $d(G, x)$ is decreased and the cost increase is minimum. It is presented in two variants.

Here is the first variant. Let $G_x = (V_x, E_x)$ be the *critical graph*, i.e. the graph defined by

$$\begin{aligned} V_x &:= \{v \in V \ : \ slack(v, x) = 0\}, \\ E_x &:= \{vw \in E \ : \ v, w \in V_x \text{ and } e(v, x) + d_{x(v)}(v) = e(w, x)\}. \end{aligned}$$

Note that $q, s \in V_x$. Using the edge identity $e(v, x) + d_{x(v)}(v) = e(w, x)$, one can easily derive that in G_x, every q-s-path has length $e(s, x) = d(G, x)$. Vice versa, every q-s-path in G having length $d(G, x)$ is obviously a path in G_x. Let

$$k := \left\lceil \frac{d(G, x) - d_{min}(G)}{maxdif(G)} \right\rceil . \tag{6}$$

We say that a set $S \subseteq V_x$ is a *k-cutset* in G_x iff every q-s-path in G_x has at least k vertices in S.

Lemma 4. *Let $d(G, x) > d_{min}(G)$ and assume that $S \subseteq V_x$ is a set of vertices with $x(v) = 1$ for all $v \in S$ and such that*

$$x'(v) := \begin{cases} 0 & \textit{if } v \in S, \\ x(v) & \textit{otherwise,} \end{cases}$$

defines a realization (G, x') with $d(G, x') = d_{min}(G)$. Then S is a k-cutset in G_x.

Proof. Let P be any q-s-path in G_x. We have $d(P, x) = d(G, x)$. Let $T = P \cap S$ be the set of vertices on P that have been switched. We have to show that $|T| \geq k$. Clearly,

$$d(P, x') = d(P, x) - \sum_{v \in T} dif(v) \geq d(G, x) - |T| maxdif(G). \tag{7}$$

Assume that $|T| < k$. Then by (6), $|T| maxdif(G) < d(G,x) - d_{min}(G)$, and from (7) we obtain $d(P, x') > d_{min}(G)$, a contradiction to our assumption. $\square$

Our approach is to switch the vertices of some k-cutset in G_x to zero, and iterate this until $d_{min}(G)$ is reached. A first try could be to choose a k-cutset S with minimum weight, where the weight is defined by $weight(S) = \sum_{v \in S} c(v)$. In practical examples many vertices have the same value of $c(v)$, hence it is helpful to take also other objectives into account, in particular the amount of length decrease. In our implementation we use the weight function

$$weight(v) = c(v) - \mu \, dif(v), \quad weight(S) = \sum_{v \in S} weight(v),$$

where μ is some constant bounded by

$$0 \leq \mu \leq \min_{v:dif(v)>0} \frac{c(v)}{dif(v)}.$$

We have chosen μ very small, e.g. $\mu = 0.0001$. The determination of a minimum k-cutset can be accomplished by a min-cost-flow algorithm. This is completely described in Chapter 4 of [4] in terms of posets. (We obtain a poset from G_x by taking the reflexive and transitive hull). Now the length decreasing part of our algorithm can be described as follows.

Length Decreasing Procedure (LDP)
INPUT: $G = (V, E)$, $c(v)$, $d_i(v)$ $(i = 1, 2,\ v \in V)$, old $x : V \to \{0, 1\}$
OUTPUT: new $x : V \to \{0, 1\}$
while $d(G, x) > d_{min}(G)$
  determine $slack(v, x)$ for all $v \in V$
  determine the critical graph $G_x = (V_x, E_x)$
  $k := \left\lceil \frac{d(G,x) - d_{min}(G)}{maxdif(G)} \right\rceil$
  determine a k-cutset S in G_x with minimum weight
  put $x(v) := 0$ for all $v \in S$
return x

Since in each q-s-path of length $d(G, x)$, at least $k \geq 1$ vertices are switched to zero, the length indeed decreases and after finitely many steps we obtain a realization (G, x) with $d(G, x) = d_{min}(G)$.

Now we present the second length decreasing variant. Instead of working only with vertices of zero slack we also allow vertices with "small" slack. The idea behind this approach is to "destroy" in one step not only the paths with maximum length, but also paths with "large" length. For a nonnegative real λ, let $G_x^\lambda = (V_x^\lambda, E_x^\lambda)$ be the λ-*critical graph*, defined by

$$\begin{aligned} V_x^\lambda &:= \{v \in V \ : \ slack(v,x) \leq \lambda(d(G,x) - d_{min}(G))\}, \\ E_x^\lambda &:= \{vw \in E \ : \ v, w \in V_x^\lambda, \ e(v,x) + d_{x(v)}(v) = e(w,x) \\ &\qquad\qquad \text{or } h(x,v) - d_{x(v)}(v) = h(w,x)\}. \end{aligned}$$

Of course, G_x^λ contains also q-s-paths P with $d(P,x) < d(G,x)$, but one can expect that these paths still have large length. Let $d_\lambda(G,x)$ be the minimum length of a q-s-path in G_x^λ. This value can be computed efficiently, e.g. with a slight modification of the DFS based algorithm for the determination of the maximum length in G. Note that $\lim_{\lambda \to +0} d_\lambda(G,x) = d(G,x)$. Let

$$k := \left\lceil \frac{d_\lambda(G,x) - d_{min}(G)}{maxdif(G)} \right\rceil.$$

As for Lemma 4, one can prove that, in order to achieve $d_{min}(G)$, one has to switch vertices of a k-cutset to zero, and again, it is a good idea to take a k-cutset of minimum weight. The algorithm presented in [4] for Hasse diagrams of posets directly applies to general dags. In contrast to the first variant, also for $d(G,x) > d_{min}(G)$ it can happen that $k \leq 0$ because, in general, $d_\lambda(G,x) < d(G,x)$, i.e. the case $d_\lambda(G,x) < d_{min}(G)$ is possible. If this is the case, we decrease λ, e.g. by $\lambda := \lambda/2$, and iterate until $d_\lambda(G,x) > d_{min}(G)$, hence $k \geq 1$. Altogether, we have the following extended version:

Extended Length Decreasing Procedure (ELDP)
INPUT: $G = (V,E)$, $c(v)$, $d_i(v)$ $(i = 1,2,\ v \in V)$, old $x : V \to \{0,1\}$, λ
OUTPUT: new $x : V \to \{0,1\}$
while $d(G,x) > d_{min}(G)$
  determine $slack(v,x)$ for all $v \in V$
  do
    determine the λ-critical graph $G_x^\lambda = (V_x^\lambda, E_x^\lambda)$
    determine $d_\lambda(G,x)$
    $k := \left\lceil \frac{d_\lambda(G,x) - d_{min}(G)}{maxdif(G)} \right\rceil$
    if $(k \leq 0)$ $\lambda := \lambda/2$
  while $k \leq 0$
  determine a k-cutset S in G_x^λ with minimum weight
  $x(v) := 0$ for all $v \in S$
return x

As in the basic version it is easy to see that after finitely many steps a realization (G,x) with $d(G,x) = d_{min}(G)$ will be obtained.

At the beginning of this section we already mentioned that we start with a realization (G, x) with $d(G, x) = d_{min}(G, x)$, switch all vertices of the candidate set C to 1 and then switch back some vertices to 0 using (E)LDP. By allowing only vertices from C for the back-switching we assure that in the end the number of vertices v with $x(v) = 1$ has increased by at least 1 compared with the realization we started with, because otherwise the last k-cutset in (E)LDP would not have had minimum weight. Of course, G_x respective G_x^λ can also contain vertices that do not belong to C. By putting $weight(v) = \infty$ for $v \in V_x \setminus C$ resp. $v \in V_x^\lambda \setminus C$ we make sure that these vertices cannot be switched back to 0. Now the whole algorithm reads as follows:

k-Cutset Algorithm (k-CA)
INPUT: $G = (V, E)$, $c(v)$, $d_i(v)$ $(i = 1, 2,\ v \in V)$, λ_0
OUTPUT: decision function $x : V \to \{0, 1\}$
Initialization
$x := x_0$
Iteration
do
  determine $slack(v, x)$ for all $v \in V$
  determine the candidate set C
  if $(C \neq \emptyset)$
    $x(v) := 1$ for all $v \in C$
    $weight(v) := c(v) - \mu dif(v)$ for all $v \in C$
    $weight(v) := \infty$ for all $v \in V \setminus C$
    $\lambda := \lambda_0$
    call **(E)LPD**
while $C \neq \emptyset$
return x

Since in each loop the number of vertices v with $x(v) = 1$ increases by at least 1, the algorithm terminates after finitely many steps.

6 Experimental results

We implemented the four algorithms in C++. The ten ISCAS85 netlists form the test data. Clearly, the results strongly depend on the functions $d_i, c_i : V \to \mathbb{R}_+$ $(i = 0, 1)$. We worked with *unit-leakage*, i.e. $c_1(v) = 0$, $c_0(v) = 1$ for all $v \in V$, and with *average-leakage*. The delay and average-leakage values are contained in Table 1. Without knowing these values it is not possible to compare the performance of the algorithms with other

	Delay $[ns]$		average-leakage $[\mu W]$	
Gate type	d_0	d_1	c_0	c_1
INV	37	46	92.8	12.6
NAND	43	58	135.0	20.3
AND	59	81	253.9	37.5
NOR	66	90	86.0	10.6
OR	71	98	151.9	20.9

Table 1: The delay and average-leakage values

algorithms from the literature. Unfortunately, the numbers of vertices of a given circuit differ from paper to paper. The following tables contain the percentage of improvement of the cost compared with the zero function x_0. For instance, if we have a circuit with n vertices and the algorithm returns a decision with n_0 LTV-vertices (i.e. $|\{v \in V \ : \ x(v) = 0\}| = n_0$) and n_1 HTV-vertices, (i.e. $|\{v \in V \ : \ x(v) = 1\}| = n_1$) then the cost improvement in the case of unit-leakage is $(n_1/n) \cdot 100\%$. The bold numbers indicate the best result for the respective circuit. One can see that for unit-leakage k-

Circuit	$\|V\|$	SSA	MSA	k-FA	k-CA
c432	155	52.26	51.61	**54.19**	**54.19**
c499	474	67.72	67.51	68.78	**70.04**
c880	393	**84.22**	83.97	**84.22**	**84.22**
c1355	738	70.87	71.00	**75.07**	74.80
c1908	453	**72.19**	71.74	**72.19**	**72.19**
c2670	663	**91.10**	**91.10**	**91.10**	**91.10**
c3540	1093	87.56	87.65	**87.74**	**87.74**
c5315	1781	92.70	**92.76**	**92.76**	92.64
c6288	2701	64.68	64.61	64.39	**65.27**
c7552	1874	**95.84**	95.79	95.41	95.68

Table 2: Leakage improvement for unit-leakage

CA and the k-FA have in general best performance while for average leakage MSA and k-CA are mostly the best ones. The parameter λ in (2) and (5), and the initial value for λ in the k-CA have some but no great influence on the results. Good choices for λ are given in Table 4. Note that λ does not have an influence in SSA and MSA in case of unit-leakage. The algorithms are fast enough to allow several runs with different values of λ resp. λ_0. Here we present the running times (in seconds) of the algorithms on a 1.8GHz PC for the largest ISCAS85 circuits c7552 and c6288 and for a special 64-bit-

Circuit	$\lvert V\rvert$	SSA	MSA	k-FA	k-CA
c432	155	**44.76**	43.94	43.87	44.16
c499	474	56.10	56.43	57.08	**58.28**
c880	393	**72.27**	**72.27**	**72.27**	**72.27**
c1355	738	57.43	58.79	**62.39**	62.37
c1908	453	59.45	60.61	60.47	**61.07**
c2670	663	77.85	77.89	77.79	**77.90**
c3540	1093	**75.28**	**75.28**	75.05	75.06
c5315	1781	79.03	**79.12**	78.87	78.59
c6288	2701	53.13	52.59	53.08	**53.60**
c7552	1874	81.49	**81.54**	81.00	80.75

Table 3: Leakage improvement for average-leakage

	SSA	MSA	k-FA	k-CA
unit-leakage	–	–	50	0.1
average-leakage	0.01	0.01	0.2	0.1

Table 4: Choice for λ resp. λ_0.

multiplier with 56353 vertices where the λ's were chosen according to Table 4.

	Circuit	$\lvert V\rvert$	SSA	MSA	k-FA	k-CA
unit-leakage	c7552	1874	6.44	0.55	4.70	4.22
	c6288	2701	10.53	1.66	12.20	4.95
	64-bit-mult	56353	7301.14	323.94	3375.14	648.78
average-leakage	c7552	1874	6.64	0.45	1.32	0.39
	c6288	2701	10.27	1.97	7.00	7.78
	64-bit-mult	56353	7087.64	329.34	2850.59	766.23

Table 5: Running times of the different algorithms in seconds.

The cost improvements for the 64-bit-multiplier are shown in Table 6. These results suggest that for very large circuits (more than 50,000 vertices) MSA seems to be the best choice: although SSA yields a slightly better leakage reduction this is compensated by the significantly smaller running time of MSA.

	SSA	MSA	k-FA	k-CA
unit-leakage	**71.02**	70.73	67.54	69.41
average-leakage	**62.21**	61.73	58.43	59.96

Table 6: Leakage improvements for the 64-bit-multiplier.

References

[1] J. Chang and M. Pedram. Energy minimization using multiple supply voltages. *IEEE Transactions on Very Large Scale Integration Systems*, 4:436–443, 1997.

[2] C. Chen, E. Bozorgzadeh, A. Srivastava, and M. Sarrafzadeh. Budget management with applications. *Algorithmica*, 34:261–275, 2002.

[3] C. Chen and M. Sarrafzadeh. An effective algorithm for gate-level power-delay tradeoff using two voltages. In *Proceedings of the International Conference on Computer Design*, pages 222–227, 1999.

[4] K. Engel. *Sperner Theory.* Cambridge University Press, 1997.

[5] M. Hamada, Y. Ootaguro, and T. Kuroda. Utilizing surplus timing for power reduction. In *Proc. IEEE Custom Integrated Circuits Conference*, pages 89–92, 2001.

[6] J. Kao, A. Chandrakasan, and D. Antoniadis. Transistor sizing issues and tool for multi-threshold CMOS technology. In *Proceedings of the 34th ACM/IEEE conference on Design Automation (DAC)*, pages 409–414, 1997.

[7] T. Karnik, Y. Ye, J. Tschanz, L. Wei, S. Burns, V. Govindarajulu, V. De, and S. Borkar. Total power optimization by simultaneous dual-Vt allocation and device sizing in high performance microprocessors. In *Proceedings of the 39th ACM/IEEE conference on Design Automation (DAC)*, pages 486–491, 2002.

[8] M. Ketkar and S.S. Sapatnekar. Standby power optimization via transistor sizing and dual threshold voltage assignment. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design (ICCAD)*, pages 375–378, 2002.

[9] N.S. Kim, T. Austin, T. Blaauw, T. Mudge, K. Flautner, H.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *IEEE Computer*, 36(12):68–75, 2003.

[10] W.-N. Li, A. Lim, P. Agrawal, and S. Sahni. On the circuit implementation problem. In *Proceedings of the 29th ACM/IEEE conference on Design Automation (DAC)*, pages 478–483, 1993.

[11] M. Liu, W.-S. Wang, and M. Orshansky. Leakage power reduction by dual-Vth designs under probabilistic analysis of Vth variation. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design (ISLPED'04)*, pages 2–7, 2004.

[12] P. Maken, M. Degrauwe, M. Van Paemel, and H. Oguey. A voltage reduction technique for digital systems. In *1990 IEEE International Solid-State Circuits Conference*, pages 238–239, 1990.

[13] D Nguyen, A. Davare, M. Orshansky, D. Chinnery, B. Thomson, and K. Keutzer. Minimization of dynamic and static power through joint assignment of threshold voltages and sizing optimization. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 158–163, 2003.

[14] R. Sedgewick. *Algorithms in C++ Part 5: Graph Algorithms*. Addison-Wesley, 3rd edition, 2001.

[15] W.-T. Shiue and C. Chakrabarti. Low-power scheduling with resorces operating at multiple voltages. *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, 47(6):536–543, 2000.

[16] F. Sill, F. Grassert, and D. Timmermann. Low power gate-level design with mixed-Vth (MVT) techniques. In *17th Symposium on Integrated Circuits and Systems (SBCCI)*, pages 278–282, 2004.

[17] S. Sirichotiyakul, T. Edwards, C. Oh, R. Panda, and D. Blaauw. Duet: An accurate leakage estimation and optimization tool for dual-V_t circuits. *IEEE Transactions on VLSI Systems*, 10(2):79–90, 2002.

[18] V. Sundararajan and K. Parhi. Low power synthesis of dual threshold voltage CMOS VLSI circuits. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 139–144, 1999.

[19] K. Usami and M. Horowitz. Clustered voltage scaling for low-power design. In *International Symposium on Low Power Design*, pages 3–8, 1995.

[20] K. Usami and M. Igarashi. Low-power design methodology and applications utilizing dual supply voltages. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 123–128, 2000.

[21] L. Wei, Z. Chen, K. Roy, M.C. Johnson, Y. Ye, and V.K. De. Design and optimization of dual-threshold circuits for low-voltage low-power applications. *IEEE Transactions on VLSI Systems*, 7(1):16–24, 1999.

[22] L. Wei, Z. Chen, K. Roy, Y. Ye, and V. De. Mixed-V_{th} (MVT) CMOS circuit design methodology for low power applications. In *Proceedings of the 36th ACM/IEEE conference on Design Automation (DAC)*, pages 430–435, 1999.

[23] L. Wei, K. Roy, and C.-K. Koh. Power minimization by simultaneous dual-Vth assignment and gate-sizing. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 413–416, 2000.