An approximation algorithm for threshold voltage optimization

SIAD DABOUL, STEPHAN HELD, JENS VYGEN, AND SONJA WITTKE,
University of Bonn, Germany

We present a primal-dual approximation algorithm for minimizing the leakage power of an integrated circuit by assigning gate threshold voltages. While most existing techniques do not provide a performance guarantee, we prove an upper bound on the power consumption.

The algorithm is practical and works with an industrial sign-off timer. It can be used for post-routing power reduction or for optimizing leakage power throughout the design flow.

We demonstrate the practical performance on recent microprocessor units. Our implementation obtains significant leakage power reductions of up to 8% on top of one of the most successful algorithms for gate sizing and threshold voltage optimization. After timing-aware global routing we achieve leakage power reductions of up to 34%.

CCS Concepts: • **Hardware** → *Physical synthesis*; *Circuits power issues*;

Additional Key Words and Phrases: V_t optimization; leakage power; time-cost tradeoff; set cover

ACM Reference Format:
Siad Daboul, Stephan Held, Jens Vygen, and Sonja Wittke. 2018. An approximation algorithm for threshold voltage optimization. 1, 1 (June 2018), 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Threshold voltage (V_t) optimization is a crucial step in VLSI design. It is often combined with simultaneous sizing as in recent sensitivity-based [8, 9] or Lagrangian relaxation approaches [5, 15], among which [5] reported the best results on the ISPD 2012 and 2013 gate sizing contest benchmarks. Its integration into an industrial design environment [14, 15] also achieved substantial power reductions on industrial designs. However, threshold voltage optimization finds its individual application in post-routing power reduction [1, 13]. Most modern cell libraries offer different V_t choices with the same footprint. Thus, the voltage threshold of a gate can be changed without requiring routing changes.

Overuse of low V_t gates leads to a large leakage power consumption, which can be prohibitive. In contrast to the size of a gate which has an approximately linear influence on the power, the static power consumption usually depends exponentially on the used V_t level of a gate. The static leakage of an inverter in the library of the ISPD 2013 gate sizing contest with respect to the chosen V_t level and size is illustrated in Figure 1.

Typically there are 10–20 different gate sizes available. In contrast to this the V_t optimization problem is highly discrete, providing only 2–4 alternative V_t levels. The large differences in the

Author's address: Siad Daboul, Stephan Held, Jens Vygen, and Sonja Wittke,
University of Bonn, Research Institute for Discrete Mathematics, Lennéstr. 2, Bonn, Germany, 53113, {daboul,held,vygen}@dm.uni-bonn.de.

XXXX-XXXX/2018/6-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

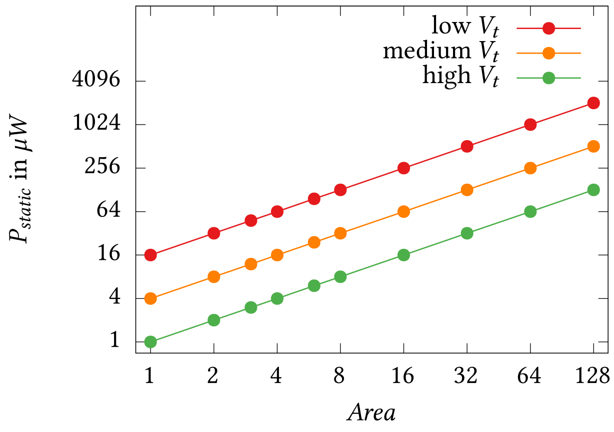

Fig. 1. The cell library for an inverter from the ISPD 2013 contest (logarithmic scales). The specific leakage numbers are artificial, but the picture is structurally consistent with industrial gate libraries.

leakage power consumption of the individual threshold implementations make the problem difficult in practice.

Shah et al. [16] propose a continuous formulation for simultaneous gate sizing and V_t assignment, in which the V_t levels are always snapping to integral values. However, the relaxation is not convex and is not known to be efficiently solvable with any useful approximation guarantee.

Liu and Hu [11] combine Lagrangian relaxation with dynamic programming for rounding to a discrete solution, resolving inconsistencies due to reconvergent paths heuristically. This approach was later refined in by Ozdal, Burns, and Hu[12].

Algorithms for pure threshold voltage optimization include the conjugate gradient method applied to a certain continuous problem relaxation [1] or greedy algorithms [13].

The problem of choosing a V_t level for each gate while satisfying the timing constraints is similar to the discrete time-cost tradeoff (TCT) problem in directed graphs. Here we are given an acyclic digraph where every edge has a set of possible execution times with associated costs. The task is to choose a realization for every edge such that the maximum execution time of a path respects some delay bound and the total cost is minimized.

Grigoriev and Woeginger showed that unless P=NP the discrete time-cost tradeoff problem cannot be approximated by a smaller factor than the vertex cover problem [7], which cannot be approximated by a factor 1.36 unless P=NP [3]. Svensson [18] showed that under the unique games conjecture there is not even a constant-factor approximation algorithm.

Skutella found a bicriteria (α, β)-approximation algorithm which, given a fixed parameter $0 < \mu < 1$, computes a solution such that the optimum cost is exceeded at most by a factor of $\alpha = \frac{1}{1-\mu}$ and the deadline D is exceeded by a factor of at most $\beta = \frac{1}{\mu}$ in polynomial time [17]. This gives a deterministic $(2, 2)$-approximation. He also showed how to obtain a randomized solution with an expected guarantee of (γ, γ), where $\gamma = \frac{e}{e-1} \approx 1.58$. Exceeding the deadline so much is of course out of the question in chip design.

2 OUR CONTRIBUTION

We present a new global algorithm for the problem of assigning a V_t level to each gate. Iteratively trying to improve the most critical gates is a common approach that is used systematically or as

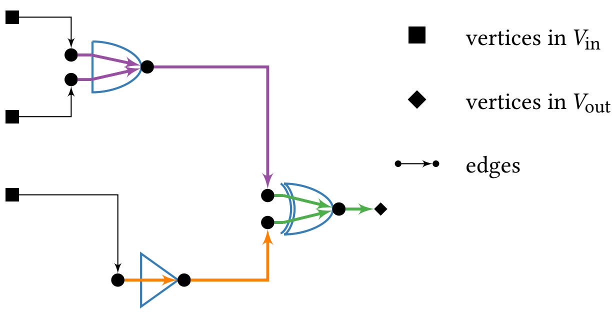

Fig. 2. An example of a timing graph. The blue gates indicate how the graph arises from the underlying chip. Edges of each E_g have the same color.

a refinement in many gate sizing & V_t assignment algorithms [4, 5, 8, 9]. Our algorithm uses the same idea but guides the iterative refinement with a global primal-dual cost function. This allows us to prove an a-priori bound on the leakage power consumed by our solution.

For the first time we obtain an a-priori approximation guarantee for the problem of minimizing the power consumption while maximizing the total negative slack (TNS) or the true total negative slack (TTNS) [14] (see also Section 5.1) of a design. Our result also translates to the TCT problem, where it provides the best known approximation guarantee for bounded path lengths.

Our algorithm has the following characteristics:

- We formulate the V_t optimization problem as a set cover problem with an exponentially large universe.
- An approximate solution to this set cover problem can be computed efficiently by a primal-dual algorithm based on Bar-Yehuda and Even [2].
- We prove that the acceleration cost of our solution never exceeds the optimum by a factor more than k, where k is the maximum number of gates on a signal path.
- The algorithm computes a lower bound on the best possible solution which we use to evaluate our results.
- Our algorithm respects the discrete nature of the problem. We do not make any convexity assumptions and do not solve a continuous relaxation.
- We do not relax timing constraints but achieve best possible solutions with respect to timing.
- We demonstrate the practical performance of our algorithm. On top of a highly optimized gate sizing and V_t assignment computed by a state-of-the-art algorithm [5, 15], we obtain additional leakage power reductions of up to 8%.
- Our algorithm is footprint-preserving and can be applied after detailed routing late in the design flow.
- After running a timing-aware global routing algorithm, we can reduce the leakage power consumption even by up to 34%.

The remainder of this paper is organized as follows. In Section 3, we give a formal problem definition. Then in Section 4, we present the new approximation algorithm together with an example and a theoretical quality analysis. In Section 5, we shortly present further variants of the problem and algorithm, and a four-step flow for V_t assignment. Finally, we present experimental results in Section 6 and conclusions in Section 7.

3 PRELIMINARIES

In the following we will assume that an instance can be modeled by a directed acyclic graph $G = (V, E)$ (the timing graph), a set of gates $\mathcal{G}$, and a set of V_t levels $\{1, \ldots, z\}$. We call vertices $v \in V$ without entering edges input vertices $v \in V_{\text{in}}$; analogously we define output vertices V_{out}. Each gate $g \in \mathcal{G}$ is represented in G by a subset E_g of edges of G where $E_g \cap E_{g'} = \emptyset$ for any pair of different gates $g, g' \in \mathcal{G}$. A sample timing graph is shown in Figure 2. For easier notation we assume that every gate has some implementation for each V_t level in $\{1, \ldots, z\}$. Here, 1 is the fastest V_t level (lowest V_t) and z the slowest one (highest V_t).

A V_t assignment is a map $\varphi : \mathcal{G} \to \{1, \ldots, z\}$. We denote the assignment that maps every gate to the fastest implementation by $\mathbb{1}$. The delay of a timing edge $e \in E$ depends on the assignment φ and is denoted by $d_\varphi : E \longrightarrow \mathbb{R}_{\geq 0}$. For a path P in G we define $d_\varphi(P) = \sum_{e \in E(P)} d_\varphi(e)$. The power of a gate is given by a function power(g, i), where $i \in \{1, \ldots, z\}$ specifies the chosen V_t level.

For a path P, its delay bound T, and assignment φ, the slack is defined by slack$(P, \varphi) = T - d_\varphi(P)$. For pins $v \in V$ and edges $e \in E$ we denote by slack(v, φ) and slack(e, φ) the minimum slack of a path that contains v and e, respectively. The total negative slack (TNS) is the sum of all negative endpoint slacks

$$\text{TNS}(\varphi) = \sum_{v \in V_{\text{out}}} \min\{0, \text{slack}(v, \varphi)\}.$$

We consider the following V_t optimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{g \in \mathcal{G}} \text{power}(g, \varphi(g)) \\ \text{subject to} \quad & \varphi : \mathcal{G} \longrightarrow \{1, \ldots, z\} \\ & \text{TNS}(\varphi) = \text{TNS}(\mathbb{1}). \end{aligned} \tag{1}$$

We will also discuss some variants of this problem in Section 5.

4 V_t OPTIMIZATION ALGORITHM

We will now describe our proposed V_t optimization algorithm (Algorithm 1). We start by assigning every gate $g \in \mathcal{G}$ to the highest available V_t level $\varphi(g) = z$. Over the course of the algorithm we will maintain a reduced cost function c_g which is guiding the optimization globally. Initially, c_g is given by the additional cost needed to accelerate gate g to the next lower V_t level

$$c_g \;=\; \text{power}(g, z-1) - \text{power}(g, z).$$

Then we proceed to iteratively accelerate a path P that violates the timing constraints. We do this by accelerating the cheapest gate g^* on P with respect to the reduced cost function c_g. Note that this accelerates all paths through g, not only P. We then reduce the values c_g for every gate on P by exactly c_{g^*}. This process is iterated until the timing constraints are met.

Note that our particular reduced cost update is important for achieving globally good solutions. If a gate occurs frequently on some violated path P, it becomes more attractive to be accelerated due to its lowered cost. The cost update is a core ingredient for proving the approximation guarantee of our algorithm in Section 4.2, and also needed for good practical results as the following example demonstrates.

4.1 Example

Consider the instance in Figure 3, where an inverter drives $K \gg 1$ other inverters (the instance can be easily adjusted to avoid high fanouts using a higher depth). If the deadline is $T = 3$ and

```
for $g \in \mathcal{G}$ do
    $\varphi(g) \leftarrow z$                                              ▷ initially use slowest $V_t$
    $c_g \leftarrow \text{power}(g, z-1) - \text{power}(g, z)$              ▷ reduced costs
while $\exists v \in V_{\text{out}} : \text{slack}(v, \varphi) < \min\{0, \text{slack}(v, \mathbb{1})\}$ do
    $P \leftarrow$ most critical path ending in $v$ w.r.t. $\varphi$
    $\mathcal{G}(P) \leftarrow$ gates on timing path $P$
    $g^* \leftarrow \text{argmin}_{g \in G_P} c_g$                           ▷ cheapest $g$ w.r.t. $c_g$
    $\gamma \leftarrow c_{g^*}$
    for $g \in \mathcal{G}(P)$ do
        $c_g \leftarrow c_g - \gamma$                                       ▷ reduce $c_g$ for gates on $P$
    $\varphi(g^*) \leftarrow \varphi(g^*) - 1$                              ▷ accelerate $g^*$
    if $\varphi(g^*) > 1$ then
        $c_{g^*} \leftarrow \text{power}(g^*, \varphi(g^*) - 1) - \text{power}(g^*, \varphi(g^*))$
                                                                           ▷ re-initialize $c_{g^*}$
    else
        $c_{g^*} \leftarrow \infty$
return $\varphi$
```

Algorithm 1: V_t optimization algorithm

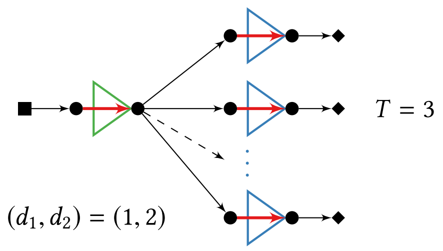

Fig. 3. An example where a greedy algorithm might speed up all K gates on the right, while our primal-dual algorithm chooses the left gate and at most one gate from the right side. (see Section 4.1)

the inverters have either delay 1 or 2 with an acceleration cost of 1, the optimum is given by only accelerating the driving inverter.

Already for $k = 2$ a simple greedy approach, that sorts all acceleration possibilities of the gates by the gain $\frac{\Delta\text{delay}}{\Delta\text{power}}$ and iteratively accelerates a gate that minimizes this (negative) ratio, e.g. a discrete variant of the TILOS algorithm [4], does not achieve a constant approximation guarantee. It may instead choose to accelerate all K inverters instead (in fact it will always do so if we reduce their acceleration cost by some small constant $\epsilon > 0$). Similarly, always choosing the cheapest gate on a critical path (without our reduced cost update) will lead to the same behaviour.

Our primal-dual algorithm might also start accelerating one of the inverters on the right. However, this reduces the cost of the driving inverter on the common path to 0 (or ϵ). This one will, thus, be the cheapest choice in the next iteration, and our algorithm will accelerate at most two inverters in total.

4.2 Algorithm Analysis

Before we analyse the algorithm, we point out the theoretic connection to the set cover problem. In the set cover problem we are given a set system $\mathcal{S} = \{S_1, \ldots, S_r\}$ where $\cup_{i=1}^{r} S_i =: \mathcal{U}$ and a cost function $\text{cost} : \mathcal{S} \to \mathbb{R}_{\geq 0}$. We are then looking for a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $\cup_{S \in \mathcal{S}'} S = \mathcal{U}$ and $\sum_{S \in \mathcal{S}'} \text{cost}(S)$ is minimum.

If we assume that our instance has only two V_t levels, that is $z = 2$, we can formulate it as a set cover problem in the following way. We call a set of gates $G \subseteq \mathcal{G}$ critical if in every timing feasible solution at least one gate in G has the lower V_t level. Let $\mathcal{P}$ denote the set of all paths from an input vertex to an output vertex. For a path $P \in \mathcal{P}$ we denote by $\mathcal{G}(P)$ the gates that have some edge on P. Our universe will then be $\mathcal{U} = \{G \subseteq \mathcal{G}(P) : P \in \mathcal{P}, G \text{ is critical}\}$. For every gate $g \in \mathcal{G}$ we define $S_g = \{G \subseteq \mathcal{G} : g \in G, G \in \mathcal{U}\}$.

It is easy to see that for a set $X \subseteq \mathcal{G}$ we have $\cup_{g \in X} S_g = \mathcal{U}$ if and only if accelerating the gates in X yields a timing feasible solution. In the special case $z = 2$, Algorithm 1 is an adaptation of the primal-dual algorithm of Bar-Yehuda and Even [2] for the set cover problem. The algorithm has an approximation guarantee of $\max_{u \in \mathcal{U}} |\{S \in \mathcal{S} : u \in S\} \leq \max_{P \in \mathcal{P}} |\mathcal{G}(P)|$. We will now give an elementary proof of this bound for arbitrary z.

To prove quality guarantees of our new algorithm, we make two mild assumptions:

A1 Lowering the voltage threshold of a gate does not increase the delay of any edge in E.
A2 The delay $d_\varphi(P)$ of a path P can only be reduced by lowering $\varphi(g)$ for a gate $g \in \mathcal{G}(P)$.

The first assumption is usually fulfilled if the input pin capacitances of a gate g do not depend on its voltage threshold $\varphi(g)$. The second assumption would follow from the first assumption in a path-based timing analysis. In any case, deviations from these assumptions in practice are usually small. We can prove the following worst-case guarantee.

Theorem 1. *Assume that A1 and A2 hold. Algorithm 1 returns a feasible solution $\bar{\varphi}$ to Problem (1). The power increase over the cheapest possible solution, choosing z everywhere, is at most k times greater than the power increase of an optimum solution φ^*:*

$$\sum_{g \in \mathcal{G}} \big(\text{power}(g, \bar{\varphi}(g)) - \text{power}(g, z)\big)$$
$$\leq k \sum_{g \in \mathcal{G}} \big(\text{power}(g, \varphi^*(g)) - \text{power}(g, z)\big),$$

where k is the maximum number of gates on any path in G.

The algorithm can be implemented to run in time $O(z|\mathcal{G}|\theta)$, where θ is the running time for identifying and traversing a critical path.

Proof. Obviously, the algorithm stops only when φ is a feasible solution. It stops after at most $(z-1)|\mathcal{G}|$ iterations of the while loop, proving the total running time bound.

Let $\mathcal{U} := \{(P, \varphi) : P \text{ is a path from } v \in V_{\text{in}} \text{ to } w \in V_{\text{out}} \text{ with } \text{slack}(P, \varphi) < \min\{0, \text{slack}(w, \mathbb{1})\}\}$ be the set of pairs with path P and V_t assignment φ for which P is too slow. Suppose we add $y(P, \varphi) := 0$ for all $(P, \varphi) \in \mathcal{U}$ in the initialization (before line 4), and $y(P, \varphi) := \gamma$ before line 11 of the algorithm (for the current values of P, φ, and γ). These numbers are needed only for the following analysis.

For any gate $g \in \mathcal{G}$ and any $i \in \{2, \ldots, z\}$ we have, while $\varphi(g) = i$, the invariant

$$c_g + \sum_{(P, \widehat{\varphi}) \in \mathcal{U} : g \in \mathcal{G}(P), \widehat{\varphi}(g) = i} y(P, \widehat{\varphi}) = \text{power}(g, i-1) - \text{power}(g, i). \tag{2}$$

Moreover, we have $c_g \geq 0$ at any stage, and $c_g = 0$ when $g = g^*$ is accelerated in Line 11. Let $\bar{\varphi}$ denote the output of the algorithm. At termination, we have for $g \in \mathcal{G}$ with $\bar{\varphi}(g) < i$ the property

$$\sum_{(P,\widehat{\varphi})\in\mathcal{U}: g\in\mathcal{G}(P), \widehat{\varphi}(g)=i} y(P,\widehat{\varphi}) \;=\; \text{power}(g, i-1) - \text{power}(g,i). \tag{3}$$

Let φ^* be an optimal solution. By definition, and by assumptions A1 and A2, for every $(P,\widehat{\varphi}) \in \mathcal{U}$ there exists a $g \in \mathcal{G}(P)$ with

$$\varphi^*(g) \;<\; \widehat{\varphi}(g). \tag{4}$$

Using equations (2), (3) and (4) we conclude:

$$\begin{aligned}
&\sum_{g\in\mathcal{G}} \big(\text{power}(g,\bar{\varphi}(g)) - \text{power}(g,z)\big)\\
&= \sum_{g\in\mathcal{G}} \sum_{i=\bar{\varphi}(g)+1}^{z} \big(\text{power}(g,i-1) - \text{power}(g,i)\big)\\
&\overset{(3)}{=} \sum_{g\in\mathcal{G}} \sum_{i=\bar{\varphi}(g)+1}^{z} \sum_{(P,\widehat{\varphi})\in\mathcal{U}: g\in\mathcal{G}(P), \widehat{\varphi}(g)=i} y(P,\widehat{\varphi})\\
&= \sum_{g\in\mathcal{G}} \sum_{(P,\widehat{\varphi})\in\mathcal{U}: g\in\mathcal{G}(P), \widehat{\varphi}(g)>\bar{\varphi}(g)} y(P,\widehat{\varphi})\\
&\overset{(5)}{\leq} \sum_{(P,\widehat{\varphi})\in\mathcal{U}} y(P,\widehat{\varphi})\,|\mathcal{G}(P)|\\
&\overset{(6)}{\leq} k \sum_{(P,\widehat{\varphi})\in\mathcal{U}} y(P,\widehat{\varphi})\\
&\overset{(4)}{\leq} k \sum_{(P,\widehat{\varphi})\in\mathcal{U}} y(P,\widehat{\varphi})\,|\{g\in\mathcal{G}(P) : \varphi^*(g) < \widehat{\varphi}(g)\}|\\
&= k \sum_{g\in\mathcal{G}} \sum_{(P,\widehat{\varphi})\in\mathcal{U}: g\in\mathcal{G}(P), \varphi^*(g)<\widehat{\varphi}(g)} y(P,\widehat{\varphi})\\
&= k \sum_{g\in\mathcal{G}} \sum_{i=\varphi^*(g)+1}^{z} \sum_{(P,\widehat{\varphi})\in\mathcal{U}: g\in\mathcal{G}(P), \widehat{\varphi}(g)=i} y(P,\widehat{\varphi})\\
&\overset{(2)}{\leq} k \sum_{g\in\mathcal{G}} \sum_{i=\varphi^*(g)+1}^{z} \big(\text{power}(g,i-1) - \text{power}(g,i)\big)\\
&= k \sum_{g\in\mathcal{G}} \big(\text{power}(g,\varphi^*(g)) - \text{power}(g,z)\big).
\end{aligned}$$

In (5) we use that the contribution to y of an element $(P,\widehat{\varphi}) \in \mathcal{U}$ can be counted at most once for every gate $g \in \mathcal{G}(P)$, which are $|\mathcal{G}(P)|$ many. For (6) note that every path contains at most k gates. □

The theorem obviously implies the following bound on the total power consumption.

Corollary 2. *Assume that A1 and A2 hold. The solution $\bar{\varphi}$ of Algorithm 1 exceeds the power of an optimum solution φ^* by at most a factor k:*

$$\sum_{g\in\mathcal{G}} \mathrm{power}(g,\bar{\varphi}(g)) \le k \sum_{g\in\mathcal{G}} \mathrm{power}(g,\varphi^*(g)),$$

where k is the maximum number of gates on any path in G.

Note that in our setting the number of gates on a longest path is a small constant compared to the total number of gates. The running time θ is usually linear in $|E|$, but Algorithm 1 can as well be used with a path-based timing analysis. See also Sections 5.6 and 6.1 for practical running time reductions.

Algorithm 1 also provides a lower bound on the optimum power consumption. As we will see this bound is much tighter than k in practice (cf. Section 6).

Corollary 3. *Assume that A1 and A2 hold. Let φ^* be an optimal solution. Let $y(P,\varphi)$ be defined as in the proof of Theorem 1. At any point of Algorithm 1 we have*

$$\sum_{g\in\mathcal{G}} \mathrm{power}(g,\varphi^*(g)) \ge \sum_{(P,\varphi)\in\mathcal{U}} y(P,\varphi) + \sum_{g\in\mathcal{G}} \mathrm{power}(g,z).$$

Proof. This is part of the inequality chain (in particular the last two inequalities) in the proof of Theorem 1. □

The lower bound can be computed easily by summing up the reduced costs of all accelerated gates and adding the high VT power. As the y values are non-decreasing over the course of the algorithm, they determine a lower bound at every intermediate step.

Note that our algorithm guarantees a close to optimum solution for low power designs with small depth.

Consider an instance with small k, where the optimum solution φ^* uses only slightly more power than the solution φ^z, which assigns every gate to the slowest available realization. More precisely, assume $\sum_{g\in\mathcal{G}} \mathrm{power}(g,\varphi^*(g)) \le (1+\epsilon)\sum_{g\in\mathcal{G}} \mathrm{power}(g,\varphi^z(g))$ for some $\epsilon > 0$. By considering a modified instance where φ^z has cost 0 one can easily see that our algorithm will return a solution $\bar{\varphi}$ such that $\sum_{g\in\mathcal{G}} \mathrm{power}(g,\bar{\varphi}(g)) \le (1+k\epsilon)\sum_{g\in\mathcal{G}} \mathrm{power}(g,\varphi^z(g))$. The approximation ratio that we obtain is therefore given by $\rho \le \frac{1+k\epsilon}{1+\epsilon}$. If we have $k = 10$ and there is an optimum solution that uses 1% more power than φ^z, i.e. $\epsilon = 0.01$ we are guaranteed to obtain a $\frac{1.1}{1.01} \approx 1.089$ approximation.

We point out that the primal-dual cost update is essential to obtain a good approximation guarantee. Other algorithms that greedily accelerate the critical path do usually not give any guarantee. In the example in Section 4.1, our algorithm accelerates at most $k = 2$ inverters, as Theorem 1 guarantees.

4.3 Sharpness of the analysis

It can be seen that our analysis in Theorem 1 is sharp. Indeed, suppose we have an inverter chain of $k+1$ gates with cycle time $T = 1$, where all gates have delay 0 for low V_t and power 0 for high V_t, the first gate has power $1+\epsilon$ for low V_t, and delay 1 for high V_t, and the other gates have power 1 for low V_t and delay $\frac{1}{k}$ for high V_t. The algorithm will put all but the first gate on low V_t and spend power k, while the optimum with power $1+\epsilon$ is exactly the opposite. However, the cell library assumed in this example has unrealistically varying delay power tradeoffs for the different gates.

5 VARIANTS AND IMPLEMENTATION

In the following, we discuss several enhancements to improve the applicability on industrial designs.

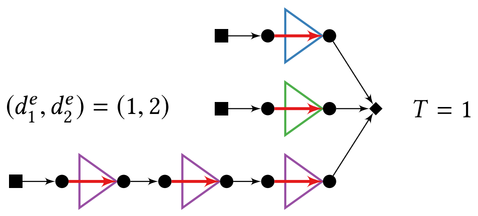

Fig. 4. Situation in which some paths are not optimized due to a hopeless path. All gates have a fast delay of $d_1^e = 1$ and a slow delay of $d_2^e = 2$.

5.1 Handling critical subpaths

Algorithm 1 finds a solution that maximizes the TNS while approximating the minimum power consumption. Assume we are given an instance with two inverter chains of length 1 and one inverter chain of length 3 that all share a common output vertex $t \in V_{\text{out}}$. If we have a cycle time of $T = 1$ and all inverters have fast delay 1 and slow delay 2 the optimum solution will completely accelerate the long path with three inverters as the optimum attainable TNS is -2. This instance is depicted in Figure 4. Note that the critical inverters on the short paths are not accelerated, even though doing so would remove the timing violation on the corresponding paths.

This shows that one should consider a more accurate timing metric than TNS. Reimann et al. suggested to use the so-called TTNS (true TNS) to evaluate the timing on less critical subpaths [14]. The TTNS is maximized if every path with negative slack is as fast as possible.

Our algorithm can be extended to find a solution with $\text{TTNS}(\varphi) = \text{TTNS}(\mathbb{1})$ by changing the condition of the while loop in line 4 to look for an edge $e \in E$ for which $\text{slack}(e, \varphi) < \min\{0, \text{slack}(e, \mathbb{1})\}$ and selecting a most critical path through that edge in line 5. It is straightforward to prove that this is also a k-approximation in terms of a cheapest solution in which every negative path is as fast as possible. We can also stop once $\text{TTNS}(\varphi) \geq \Theta$ for a given threshold $\Theta \leq \text{TTNS}(\mathbb{1})$. In our experiments, we chose Θ as the TTNS of the initial solution, for which we want to improve the leakage power.

5.2 Power recovery

Once Algorithm 1 terminates some gates can usually be decelerated again without introducing timing violations. For example, a fanout inverter on the right side in Figure 3 can be decelerated after the driving inverter on the left has been accelerated. For the Bar-Yehuda and Even algorithm the so-called reverse delete step [6] serves this purpose. In this post-processing routine the gates are considered in the reverse order in which they were accelerated by the algorithm and decelerated if this does not introduce any timing violation.

Alternatively, it is also tempting to decelerate gates in non-increasing order of their static leakage. As this order experimentally led to better leakage reductions, we incorporated it to post-process the result of Algorithm 1.

In contrast to the greedy approaches described in Section 4.1, Algorithm 1 together with the recovery step solves the instance in Figure 3 optimally.

5.3 Breaking ties

By A1 we assume that lowering the voltage threshold does not increase the delay of any edge. We verify this by measuring the slack gain after every acceleration and rejecting the change if the path slack degraded. This hardly ever occurs but we also use the slack change to break ties if there are multiple gates with the same reduced cost. To avoid excessive runtime increases we configure the signoff timer to only update delays and slacks locally for these slack change evaluations.

5.4 Slew violation removal

Extensive V_t optimization may lead to violations of slew limits at some sink pins. We neglect these violations during the course of Algorithm 1 and the power recovery step in the previous section.

However, at the very end we eliminate slew limit violations by lowering V_t levels or, only if the footprint need not be preserved, by changing gate sizes. In our experiments, this affects a negligible fraction of the gates.

5.5 Preprocessing

In practice, instances often contain hopeless paths, where all gates have to be assigned to the lowest V_t level. We can significantly speed up our algorithm by identifying and accelerating these paths in a preprocessing step.

For the TTNS optimization we set all gates to the fastest alternative and fix this lowest V_t level for all gates whose slack is still negative. It is easy to see that such gates need to be set to the fastest alternative in every feasible solution.

For the case of maximizing the TNS this approach can pre-assign more gates to low V_t than necessary. For instance in Figure 4, the upper two gates do not influence the TNS. In this case we can only fix gates whose deceleration would decrease the TNS. We can identify these gates by propagating slack deltas in reverse topological order.

5.6 Disjoint paths for running time reduction

We can obtain a significant practical speedup by considering not only a single most critical path but a set of gate-disjoint critical paths independently. In every iteration we compute a set of disjoint violated V_{in}-V_{out}-paths and for each of these paths we accelerate exactly one gate with the minimum reduced cost. This reduces the number of iterations and the number of global timing updates before collecting the path(s), leading to a significant running time reduction.

We collect these paths by traversing the timing graph in reverse topological order, while blocking gates that already occur in some path. Note that the bound in Theorem 1 still holds, because in the proof we do not use that Algorithm 1 selects a most critical path. It is sufficient to pick any violated path.

However as we will see later (Section 6), in practice the leakage might degrade a little when using too many paths simultaneously. Thus, we select a certain fraction of the most critical paths. To this end we use a sliding slack window that selects a subset of the timing endpoints. Initially, we select all timing endpoints which are within $r := 1\,\text{ps}$ from the global worst slack.

To always select a good portion of failing paths, we adjust the window as follows. If we selected less than $\frac{\alpha\cdot|\mathcal{G}'|}{1000}$ paths in an iteration, where $\mathcal{G}' \subseteq \mathcal{G}$ is the set of gates with negative slack at the start of the algorithm and $\alpha > 0$ is a parameter, we increase $r \leftarrow 1.15r$. Otherwise, we set $r \leftarrow 1.15^{-1}r$. Due to the multiplicative update of r the slack window will quickly be large enough to select a sufficient amount of paths. Therefore the exact choice of r or the search factor of 1.15 do not play a large role with respect to the measured runtime.

For our experiments we used $\alpha = 1$ unless stated otherwise. In Section 6.1 we analyse the dependency on α experimentally.

5.7 Overall V_t assignment flow

Sometimes we do not want to obtain the best possible TTNS, instead the goal is to achieve the quality of the initial solution φ^I using less power. In any case we always maximize the worst slack and the TNS. This can be achieved by multiple calls to slight variants of Algorithm 1, which are described in our overall optimization flow in Algorithm 2.

```
1 Run Algorithm 1 until the TNS is maximized.
2 Run the variant of Algorithm 1 described in Section 5.1, but without initialization (lines 1–3 of
    Algorithm 1), and stop as soon as TTNS($\varphi$) ≥ TTNS($\varphi^I$). If after accelerating a gate the initial
    leakage power $\sum_{g \in \mathcal{G}}$ power($g, \varphi^I(g)$) is exceeded we stop the algorithm.
3 Power recovery (Section 5.2).
4 Slew violations removal (Section 5.4), and (when sizing is allowed) placement legalization.
```

Algorithm 2: Optimization flow

6 EXPERIMENTAL RESULTS

We evaluated our implementation of Algorithms 1 and 2 on the industrial 22nm microprocessor instances that were also used in [15]. For these instances, we can use one of the most successful algorithms by Reimann et al. [15] for initial gate sizing and V_t assignment, and measure the additional leakage power reduction achieved by our algorithm.

Our implementation is integrated into the IBM microprocessor design flow. For every instance $z = 3$ V_t levels were available. We used the sign-off timing engine EinsTimer for all timing calculations inside our algorithm and for the numbers in the tables. Wire delays were estimated using the MAISE delay model [10], which is the default in the design environment. Here we apply our algorithm on a highly optimized netlist. First, we use the gate sizing algorithm by Reimann et al. [15] to reduce the static power consumption by up to 10% and the total power by up to 8.3%. Then, we are then able to obtain additional static power reductions of up to 8% by using our algorithm. The experiments were performed on a cluster of Linux servers with Intel Xeon CPUs with clock frequencies between 2.6 and 3.4 GHz.

Unfortunately, only the instances from [15] at the beginning of the physical design flow were available to us, but not the final gate sizing instances used in [15]. Thus, we reran the physical design flow, which is the reason why our numbers slightly deviate from [15]. Note that Reiman et al. [15] also reran the whole flow compared to their previous work [14].

We also report the lower bound $P^{\text{lb}}_{\text{static}}$ on the minimum leakage for TNS maximization, which is computed according to Corollary 3. Algorithm 2 stops once the initial leakage power or TTNS are reached, way before the TTNS is maximized. At this point the bound induced by the y-variables is not valid for the current TTNS, but only for the maximum TTNS. Thus, we always report the valid bound for TNS maximization.

We ran three variants of our algorithm. We disabled the preprocessing step as we stop the optimization as soon as the initial power is exceeded, which is not possible when a preprocessing is used. The results are given in Table 2. The rows of Table 2 refer to the following experiments/flows:

- Initial: An unrouted industrial instance after placement, and full timing optimization including a net-based layer, wiring width, and spacing assignment. Wires are estimated as short Steiner trees on the respective layers assuming the assigned width and spacing. The snapshot is the result of an industrial design flow.

- Lagrange [15]: We use the implementation of the gate sizing and V_t assignment algorithm by Reimann et al. [15], which is integrated into the industrial design environment. Note that [15] is an industrial adaption of [5], the winner of the ISPD'13 contest. We observe similar power improvements and running times as the original paper [15]. While they report a running time of about 13 hours for the largest instance uP_10 with 126k gates we measured around 11.5 hours [15].
- Alg. 2 TNS-opt: We apply our optimization flow (Algorithm 2) on the result of the Lagrange flow but omit the second step which optimizes the TTNS. The purpose of this step is primarily to serve for comparison with the lower leakage bound $P^{\text{lb}}_{\text{static}}$.
- Alg. 2: We apply our optimization flow (Algorithm 2) on the result of the Lagrange flow.
- Alg. 2 post-GR: We apply our optimization flow (Algorithm 2) on the result of the Lagrange flow followed by an industrial timing-driven global routing. Here, gate sizing is forbidden for slew recovery to preserve gate footprints.

The columns show the instance names, the particular flow, the number of gates $|\mathcal{G}|$, the maximum number k of gates on a signal path, the worst slack WS, the total negative endpoint slack TNS, the true total negative slack TTNS [14], the leakage power consumption $P^{\text{apx}}_{\text{static}}$ before power recovery, the leakage power $P^{\text{recov}}_{\text{static}}$ after power recovery and the leakage power $P^{\text{fixup}}_{\text{static}}$ after violation fixup, its relative difference to the Lagrange flow in percent $\Delta P^{\text{fixup}}_{\text{static}}$, the lower bound $P^{\text{lb}}_{\text{static}}$ for TNS optimization according to Corollary 3, the ratio between the lower bound and the given solution, the total power consumption P_{total}, its relative difference to the Lagrange flow in percent ΔP_{total} and the running time of the V_t assignment or gate sizing algorithm respectively. Slew limit violations were negligible at the end of each flow.

Algorithm 2 without global routing shows significant reductions of the leakage power compared to the result of "Lagrange". On uP_14 the reduction is 8.7%. Here we are provably less than 4% away from the optimum solution. Algorithm 2 maximizes the TNS, which, thus, is never worse than the TNS of "Lagrange" and in most cases it yields a better TNS. For the three instances uP_02, uP_05, and uP_12, the power limit was reached in Step 2. The subsequent power recovery pushed the leakage power below the limit.

The effect of the power recovery is usually small, but on some instances as uP_01 it can save up to 2% of power in the post-GR mode. Similarly the increase of power by the violation fix up is not significant as only few violations are introduced to begin with. For cap violations there were at most 2 additional cap violations on a single instance and in total there is 1 cap violation less after violation fixup.

We point out that in every run the TNS what maximized by our algorithm. To verify this we analyzed the change of endpoint slacks on the 75787 endpoints of all instances uP_01-uP_14 between the end of Algorithm 2 and the reference algorithm [15] for which either of the algorithms did not meet the slack target. Due to slight timing degradations by the power recovery, the worst degradation at a single pin which we measured was -0.79ps. The best improvement of a pin was 14.3ps. The average improvement is 0.12ps, and the total improvement across all instances is 9.2ns.

By degrading the TTNS a major leakage reduction is possible as e.g. instance uP_13 shows where the TNS-opt flow reduces the leakage by 28%. The worst approximation guarantee we obtain is 3.54 on instance uP_11, in any case the computed guarantee is significantly better than k which ranges from 13 to 50.

When used after timing-aware global routing, the leakage reduction by Algorithm 2 is even more significant. Algorithm 2, which does not require any re-routes, reduces the leakage power by up to 34%. The reason is that the timing-aware global wires mostly result in faster signal delays and the design flow uses slightly pessimistic delay estimates before global routing. Thus, the TNS and

	Instance with 110k gates						Instance with 1500k gates					
α	0.1	0.25	0.5	0.75	1.0	2.0	0.1	0.25	0.5	0.75	1.0	2.0
Iterations	15929	6226	3180	2174	1669	952	14974	5948	3119	2187	1637	1112
Runtime $[h{:}m]$	2:13	1:09	0:47	0:41	0:37	0:30	16:15	11:31	6:34	6:15	5:27	4:53
Leakage $[\mu W]$	40.73	40.72	40.68	40.68	40.63	40.77	361.40	361.47	361.75	361.74	362.17	362.49

Table 1. Running times and total leakages on two designs depending on α, i.e. the number of selected paths.

TTNS after global routing improve compared to the Steiner estimates with layer assignment. This becomes also noticeable in the mostly reduced lower bounds on the leakage power. However, some wire delays and sometimes even the WS degrade after global routing, e.g. on uP_01 and uP_11.

6.1 Running time evaluation

In addition to the results in Table 2 we also tried to analyse the scalability and running time efficiency of our algorithm. To do this we measured the running time on two larger designs, a large one with 1.5 million gates and a moderate instance with 110k gates. To investigate the scaling of the algorithm we disabled any preprocessing. As indicated in Section 5.6 the number of paths that is selected in every iteration has a big impact on the total running time. Therefore we tried various different values for the parameter α introduced in Section 5.6.

For the larger instance approximately 1 million acceleration operations were performed in order to reach a timing feasible solution. A naive implementation of our algorithm would thus perform 1 million iterations, each of which requires a full timing analysis of the instance.

In practice the situation looks much better, indeed we can accelerate about 1000 paths independently on this instance as can be seen in Table 1. Note that even if the number of iterations is about inversely proportional to α the running time doesn't fully scale as we evaluate slack changes as described in Section 5.3.

As bigger instances usually allow more paths to be collected, the number of iterations is almost constant, implying an almost linear practical running time of our algorithm. If α is too large, we obtain slightly worse results, thus our default choice of $\alpha = 1$.

With preprocessing and $\alpha = 1$ these two instances run in 0:20 and 2:31 hours, respectively.

7 CONCLUSIONS

We presented a primal-dual V_t optimization algorithm with a provable performance guarantee. In practice, it achieved leakage reductions of up to 8% on netlists that where pre-optimized by one of the most successful algorithms for gate sizing and V_t assignment [15].

Our approach also yields lower bounds which show that we solve some of the instances almost optimally.

After global routing the reduction grows up to 34% without changing any footprints. This indicates that the final V_t optimization should be done after routing, when most accurate delay estimates can be used.

REFERENCES

[1] Abrishami, H., Lou, J., Qin, J., Froessl, J., and Pedram, M., *Post sign-off leakage power optimization* (2011), *Proc. DAC*, 453–458.

[2] Bar-Yehuda, R. and Even, S., *A linear-time approximation algorithm for the weighted vertex cover problem* (1981), *Journal of Algorithms 2(2)*, 198–203.

[3] Dinur, I. and Safra, S., *On the hardness of approximating minimum vertex cover* (2005), *Annals of Mathematics 162(1)*, 439–485.

[4] Fishburn, J.P. and Dunlop, A.E., *TILOS: A posynomial programming approach to transistor sizing* (1985), *Proc. ICCAD*, 326–328.

[5] Flach, G., Reimann, T., Posser, G., Johann, M. de O., and Reis, R., *Effective method for simultaneous gate sizing and V th assignment using Lagrangian relaxation* (2014), *IEEE TCAD 33(4)*, 546–557.

[6] Goemans, M.X. and Williamson, D.P., *The primal-dual method for approximation algorithms and its application to network design problems* (1997), *Approximation Algorithms for NP-hard problems*, Hochbaum, D. (ed.), PWS Publishing, 144–191.

[7] Grigoriev, A. and Woeginger, G.J., *Project scheduling with irregular costs: complexity, approximability, and algorithms* (2004), *Acta Informatica 41(2)*, 83–97.

[8] Hu, J., Kahng, A.B. , Kang, S., Kim, M.-C., and Markov, I.L., *Sensitivity-guided metaheuristics for accurate discrete gate sizing* (2012), *Proc. ICCAD*, 233–239.

[9] Kahng, A.B. , Kang, S., Lee, H., Markov, I.L., and Thapar, P., *High-performance gate Sizing with a signoff timer* (2013), *Proc. ICCAD*, 450–457.

[10] Liu, F. and Feldmann, P., *MAISE: An Interconnect Simulation Engine for Timing and Noise Analysis* (2008), *Proc. ISQED*, 621–626.

[11] Liu, Y. and Hu, J. *A new algorithm for simultaneous gate sizing and threshold voltage assignment* (2010), *IEEE TCAD 29(2), 223–234.*

[12] Ozdal, M., Burns, S., and Hu, J., Algorithms for Gate Sizing and Device Parameter Selection for High-Performance Designs (2012), IEEE TCAD 31(10), 1558–1571.

[13] Rahman, M. and Sechen, C., Post-Synthesis Leakage Power Minimization (2012), Proc. DATE, 99–104.

[14] Reimann, T., Sze, C.C.N., and Reis, R., Challenges of cell selection algorithms in industrial high performance microprocessor designs (2015), Integration, the VLSI Journal, (52), 347–354.

[15] Reimann, T., Sze, C.C.N., and Reis, R., Cell selection for high-performance designs in an industrial design flow (2016), Proc. ISPD, 65–72.

[16] Shah, S., Srivastava, A., Sharma, D., Sylvester, D., Blaauw, D., and Zolotov, V., Discrete vt assignment and gate sizing using a self-snapping continuous formulation (2005), Proc. ICCAD, 705–712.

[17] Skutella, M., Approximation algorithms for the discrete time-cost tradeoff problem (1998), Mathematics of Operations Research 23(4), 909–929.

[18] Svensson, O., Hardness of Vertex Deletion and Project Scheduling (2012), Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, Springer, 301–312.

Instance	Flow	$\|\mathcal{G}\|$	k	WS [ps]	TNS [ns]	TTNS [ns]	P^{apx}_{static} [μW]	P^{recov}_{static} [μW]	P^{fixup}_{static} [μW]	ΔP^2_{static}	P^{lb}_{static} [μW]	$\frac{P_{static}}{P^{lb}_{static}}$	P_{total} [μW]	ΔP_{total}	Time [h:m:s]
uP_01	Initial	99k	28	-69.5	-101.4	-602.6			81.7	+13.2%			95.1	+11.2%	
	Lagrange [15]			-69.5	-96.4	-590.5			72.2		21.2	3.40	85.5		6:27:19
	Alg. 2 TNS-opt			-69.5	-94.8	-894.5	56.3	56.3	56.4	-21.8%	21.2	2.66	69.8	-18.4%	49:05
	Alg. 2			-69.5	-95.1	-590.5	71.2	70.3	70.5	-2.4%	21.2	3.32	83.8	-2.0%	1:00:46
	Alg. 2 post-GR			-69.9	-78.2	-448.6	66.0	64.7	64.9	-10.1%	20.7	3.06	78.2	-8.6%	51:14
uP_02	Initial	10k	50	-156.9	-1.9	-10.0			1.2	+6.8%			2.4	+4.4%	
	Lagrange [15]			-157.0	-1.9	-10.8			1.1		0.9	1.25	2.3		45:16
	Alg. 2 TNS-opt			-157.0	-1.8	-18.4	1.1	1.1	1.1	-6.1%	0.9	1.18	2.3	-3.0%	0:49
	Alg. 2			-157.0	-1.8	-11.6	1.1	1.1	1.1	-1.0%	0.9	1.24	2.3	-0.5%	1:15
	Alg. 2 post-GR			-154.6	-1.9	-9.8	1.1	1.1	1.1	-0.9%	0.9	1.24	2.3	-1.0%	0:45
uP_03	Initial	9k	22	7.0	-0.0	-0.0			2.7	+2.0%			52.7	+0.7%	
	Lagrange [15]			7.0	-0.0	-0.0			2.7		1.7	1.54	52.4		57:52
	Alg. 2 TNS-opt			7.0	-0.0	-0.0	2.5	2.5	2.5	-5.5%	1.7	1.46	52.3	-0.2%	4:32
	Alg. 2			7.0	-0.0	-0.0	2.5	2.5	2.5	-5.1%	1.7	1.46	52.3	-0.2%	4:23
	Alg. 2 post-GR			7.0	-0.0	-0.0	2.3	2.2	2.3	-15.4%	1.7	1.30	51.4	-1.9%	4:02
uP_04	Initial	7k	25	-11.2	-0.7	-0.7			1.6	+0.3%			2.9	+0.3%	
	Lagrange [15]			-11.2	-0.7	-0.7			1.6		1.6	1.01	2.9		31:18
	Alg. 2 TNS-opt			-11.2	-0.7	-0.7	1.6	1.6	1.6	-0.1%	1.6	1.01	2.9	-0.0%	0:08
	Alg. 2			-11.2	-0.7	-0.7	1.6	1.6	1.6	-0.1%	1.6	1.01	2.9	-0.0%	0:08
	Alg. 2 post-GR			-5.5	-0.4	-0.4	1.6	1.6	1.6	-0.2%	1.6	1.01	2.9	-0.6%	0:06
uP_05	Initial	16k	22	-76.6	-36.6	-64.0			20.3	+2.8%			67.3	+1.0%	
	Lagrange [15]			-76.6	-36.8	-64.2			19.7		9.3	2.12	66.6		31:20
	Alg. 2 TNS-opt			-76.6	-36.7	-72.3	18.9	18.9	18.9	-4.0%	9.3	2.03	65.8	-1.2%	10:03
	Alg. 2			-76.6	-36.7	-64.5	19.7	19.7	19.7	-0.2%	9.3	2.11	66.6	-0.1%	10:18
	Alg. 2 post-GR			-69.5	-27.3	-46.9	18.0	18.0	18.0	-8.8%	9.0	1.93	63.4	-4.8%	7:09
uP_06	Initial	77k	29	-108.9	-15.9	-25.6			35.7	+5.4%			147.6	+1.3%	
	Lagrange [15]			-108.9	-14.5	-24.0			33.9		23.5	1.44	145.8		3:20:22
	Alg. 2 TNS-opt			-108.9	-13.3	-26.5	31.9	31.9	32.2	-5.0%	23.5	1.37	144.1	-1.2%	25:04
	Alg. 2			-108.9	-13.3	-23.9	32.2	32.1	32.5	-4.2%	23.5	1.38	144.3	-1.0%	24:52
	Alg. 2 post-GR			-107.7	-9.8	-15.2	28.2	28.1	28.3	-16.4%	22.3	1.21	140.2	-3.9%	17:10
uP_07	Initial	72k	25	-33.9	-38.6	-231.6			60.8	+9.1%			73.2	+7.5%	
	Lagrange [15]			-33.9	-39.2	-229.2			55.7		19.0	2.93	68.1		4:34:21
	Alg. 2 TNS-opt			-33.9	-36.6	-343.3	47.8	47.7	47.9	-14.1%	19.0	2.52	60.2	-11.5%	46:30
	Alg. 2			-33.9	-36.7	-228.6	54.5	53.9	54.1	-3.0%	19.0	2.85	66.4	-2.4%	51:00
	Alg. 2 post-GR			-32.4	-28.6	-153.0	49.7	49.1	49.2	-11.6%	18.4	2.59	61.6	-9.5%	44:02
uP_08	Initial	18k	28	-72.6	-35.1	-176.4			16.8	+8.1%			85.9	+2.7%	
	Lagrange [15]			-72.6	-34.5	-176.8			15.5		6.1	2.53	83.7		1:13:35
	Alg. 2 TNS-opt			-72.6	-34.5	-248.8	11.8	11.8	11.8	-23.8%	6.1	1.93	79.9	-4.5%	11:04
	Alg. 2			-72.6	-34.5	-176.4	14.7	14.6	14.7	-5.6%	6.1	2.39	82.7	-1.1%	13:50
	Alg. 2 post-GR			-66.6	-26.7	-124.0	13.8	13.7	13.7	-11.7%	6.1	2.24	80.7	-3.5%	12:01
uP_09	Initial	18k	22	-23.2	-8.8	-36.2			14.5	+10.3%			47.8	+3.2%	
	Lagrange [15]			-22.7	-8.6	-37.1			13.1		5.8	2.26	46.3		1:15:09
	Alg. 2 TNS-opt			-22.7	-8.2	-54.1	11.4	11.4	11.4	-13.0%	5.8	1.97	44.6	-3.7%	10:02
	Alg. 2			-22.7	-8.2	-36.8	12.7	12.6	12.7	-3.6%	5.8	2.18	45.9	-1.0%	10:54
	Alg. 2 post-GR			-22.4	-6.6	-24.2	11.6	11.5	11.6	-11.8%	5.6	1.99	44.5	-3.9%	9:33
uP_10	Initial	126k	23	-43.8	-76.0	-342.6			91.6	+17.0%			395.2	+5.3%	
	Lagrange [15]			-39.9	-80.5	-395.3			78.3		25.8	3.04	375.2		9:14:53
	Alg. 2 TNS-opt			-40.0	-73.9	-531.1	67.0	67.0	67.2	-14.2%	25.8	2.61	364.0	-3.0%	1:45:27
	Alg. 2			-39.9	-74.0	-392.8	73.5	73.2	73.4	-6.3%	25.8	2.85	370.3	-1.3%	1:55:15
	Alg. 2 post-GR			-31.4	-30.8	-119.0	52.1	51.5	51.7	-34.0%	25.9	2.01	343.1	-8.6%	1:47:15
uP_11	Initial	25k	38	-140.7	-167.2	-886.7			39.7	+6.4%			61.6	+4.0%	
	Lagrange [15]			-140.3	-165.2	-878.4			37.3		10.1	3.67	59.2		1:36:54
	Alg. 2 TNS-opt			-140.3	-165.4	-990.7	27.0	27.0	27.0	-27.5%	10.1	2.66	48.9	-17.4%	12:07
	Alg. 2			-140.3	-165.1	-877.8	35.9	35.9	36.0	-3.6%	10.1	3.54	57.8	-2.2%	15:42
	Alg. 2 post-GR			-142.6	-157.8	-826.6	35.5	35.5	35.6	-4.6%	9.9	3.50	57.4	-2.9%	13:18
uP_12	Initial	18k	38	-417.8	-342.0	-696.1			5.1	+5.7%			25.5	+1.9%	
	Lagrange [15]			-417.8	-342.5	-699.4			4.8		3.7	1.30	25.0		1:10:42
	Alg. 2 TNS-opt			-417.8	-340.5	-753.7	4.5	4.5	4.6	-4.9%	3.7	1.23	24.8	-1.0%	3:43
	Alg. 2			-417.8	-340.5	-712.3	4.8	4.8	4.8	-0.2%	3.7	1.29	25.0	-0.0%	4:19
	Alg. 2 post-GR			-416.2	-332.9	-682.8	4.8	4.8	4.8	-0.3%	3.7	1.29	24.9	-0.3%	2:35
uP_13	Initial	20k	17	-47.6	-20.8	-103.4			19.6	+6.8%			80.2	+1.9%	
	Lagrange [15]			-47.6	-20.6	-103.6			18.4		6.5	2.85	78.7		1:08:54
	Alg. 2 TNS-opt			-47.6	-20.4	-152.7	13.2	13.2	13.2	-28.1%	6.5	2.05	73.5	-6.6%	8:09
	Alg. 2			-47.6	-20.5	-103.3	18.3	18.1	18.1	-1.5%	6.5	2.81	78.4	-0.4%	10:28
	Alg. 2 post-GR			-42.9	-17.9	-88.8	17.2	17.1	17.1	-6.9%	6.3	2.65	77.2	-1.9%	8:42
uP_14	Initial	13k	13	-54.8	-5.1	-9.2			8.2	+0.1%			17.9	+0.0%	
	Lagrange [15]			-54.8	-5.1	-9.2			8.2		7.3	1.13	17.9		23:16
	Alg. 2 TNS-opt			-54.8	-5.1	-9.2	7.5	7.5	7.5	-8.7%	7.3	1.03	17.2	-4.0%	0:32
	Alg. 2			-54.8	-5.1	-9.2	7.5	7.5	7.5	-8.7%	7.3	1.03	17.2	-4.0%	0:32
	Alg. 2 post-GR			-54.7	-5.0	-9.0	7.4	7.4	7.5	-8.9%	7.2	1.03	17.2	-4.1%	0:33

Table 2. Results on 22nm microprocessor instances. Alg. 2 runs as a postprocessing of the Lagrange [15] algorithm.