This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3025541, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Autonomous Application of Netlist Transformations inside Lagrangian Relaxation-based Optimization

Apostolos Stefanidis, Dimitrios Mangiras, Chrysostomos Nicopoulos, David Chinnery, Giorgos Dimitrakopoulos

***Abstract*—Timing closure is a complex process that involves many iterative optimization steps applied in various phases of the physical design flow. Lagrangian Relaxation (LR)-based optimization has been established as a viable approach for this. We extend LR-based optimization by interleaving in each iteration various techniques, such as: gate and flip-flop sizing; buffering to fix late and early timing violations; pin swapping; gate merge/split transformations; and useful clock skew. In all cases, locally optimal decisions are made using LR-based cost functions. In each iteration of LR-based optimization, we leverage the Multi-Armed Bandit (MAB) model to automatically pick which optimization heuristic should be applied to the design. The goal is to improve the performance metrics based on the rewards learned from the previous applications of each heuristic and the runtime cost paid for the received reward. The fine-grained combination of an LR-based optimization flow with a statistical recommendation system allows for the autonomous execution of the optimization flow and results in significant quality-of-results improvement relative to the state-of-the-art. More specifically, our flow achieves 17% lower clock period, while also saving 15% power and 6% area, on average, on the TAU2019 benchmarks, as compared to the TAU2019 contest winner, and 25% better leakage power on the ISPD13 benchmarks, as compared to the best reported results.**

***Index Terms*—Power and Timing Optimization, Lagrangian Relaxation, Autonomous Optimization, Multi armed bandits**

I. Introduction

Multimode multi-corner timing-driven design optimization aims at satisfying timing constraints in all modes and corners, while improving the area and power performance of the design [1]. Fixing a violation in one timing scenario is likely to cause a new violation in another. This problematic behavior is accentuated when power also needs to be optimized, since the worst-case power corner could be different from the worst-case timing corner. Such a multi-objective problem is inherently complex and computationally challenging, when considering the highly increasing number of mode/corner combinations. Over the years, multiple optimization methods have been introduced to achieve significant Power-Performance-Area (PPA) improvement. Cell sizing, transistor voltage threshold selection (V_T-swap), netlist restructuring, timing-driven cell relocation, useful clock skew, and buffer insertion/deletion are just a few examples of optimization methods supported by modern physical design tools [2].

Even if each algorithm is effective in optimizing the design, the order of the application of the various algorithms is equally important to the final result. Currently, the order of applying the available design optimization methods is organized in reference flows based on human experience and on how well the flow has performed so far on other designs. If closure is not achieved by the reference flow, the flow is customized for the specific needs of the current design. Deciding on a new customization faces the exploitation versus exploration dilemma: exploit the same heuristics that have so far yielded high Quality-of-Results (QoR), or explore new heuristics with more uncertain QoR to gather more information and hope to reach a better overall solution in the end. Even if this dilemma can be answered optimally for a certain design and a fixed set of optimization heuristics, the answer cannot be directly transferred to a new design, nor can it be adapted to a different set of heuristics.

This work tackles this problem by leveraging a modified version of Multi-Armed Bandits (MABs) [3] to autonomously optimize unknown designs by choosing in each iteration of a Langrangian Relaxation (LR)-based optimizer which method to apply among a wide range of incremental optimization heuristics. Each heuristic is smoothly integrated without being disruptive to the overall optimization process, thus allowing the solution produced by one heuristic to gradually adapt to the solution produced by a previously applied heuristic. In all cases, the locally optimal decisions needed by each heuristic are taken using LR-based cost functions. This increases the modularity of the proposed approach, since any other local optimization heuristic with similar cost function could be included in the set of available optimizations.

The order of applying the optimization heuristics inside the LR-based optimizer is done autonomously by the proposed recommendation engine. For each one of the applied optimization algorithms, a reward is recorded based on a combination of PPA metrics and runtime. Improvements in PPA increase the reward given to each heuristic, while its increased execution time relative to a runtime target penalizes its future selection. The recommendation engine decides which optimization method to select next, following a balanced exploitation-exploration approach. Thus, the optimization of each design, even if enclosed in the same LR-based optimization loop, evolves autonomously by adapting both to the design's unique characteristics and to the phase of the overall optimization.

The contributions of this work are summarized as follows:

- A multi-corner LR-based global optimizer that applies a different local optimization heuristic in each iteration.

Apostolos Stefanidis, Dimitrios Mangiras and Giorgos Dimitrakopoulos are with the Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece. Dimitrios Mangiras is supported by the Onassis Foundation - Scholarship ID: G ZO 014-1/2018-2019.
(e-mail: apstefan@ee.duth.gr, dmangira@ee.duth.gr, dimitrak@ee.duth.gr).

Chrysostomos Nicopoulos is with the Department of Electrical and Computer Engineering at the University of Cyprus, Nicosia, Cyprus.
(e-mail: nicopoulos@ucy.ac.cy).

David Chinnery is with Mentor Graphics, a Siemens Business, Fremont, USA (e-mail: David_Chinnery@mentor.com).

- The elaboration of well-known netlist transformations, such as gate sizing, buffering, useful clock skew, gate merging and splitting, and pin swapping to operate using similar local-cost functions that include only Lagrangian Multipliers (LMs) as weights.
- A MAB-based selection policy that decides, based on a newly defined reward mechanism, which netlist transformation should be applied in each iteration of the LR-based optimization loop.

The proposed approach has been successfully applied to the TAU 2019 multi-corner design optimization contest benchmarks [4] and, for completeness, to the benchmarks of the ISPD13 gate sizing contest [5]. The introduced recommendation engine is demonstrated to be a very effective orchestrator of the overall optimization process. In all cases, the proposed approach successfully optimizes the designs and achieves significantly better results than state-of-the-art. Most importantly, the recommendation engine allows the addition of new heuristics that will be autonomously applied inside the LR-optimization loop, without requiring any manual tuning and without degrading the already achieved QoR.

The rest of the paper is organized as follows: Section II reviews state-of-the-art in design optimization. Section III presents the proposed flow that employs an autonomous recommendation engine inside an LR-based global optimizer. The reward functions that guide the recommendations are described in Section IV. The applied heuristics that use LR-based cost functions are presented in Section V. The experimental results are presented in Section VI, while conclusions are drawn in Section VII.

II. Related Work

Design optimization includes a multitude of different techniques. A standard industry approach is to iteratively select the targets on timing-critical paths and try different optimizations on each cell, picking the best alternative, before optimizing the next target. This can be quite runtime expensive, and such greedy approaches can be sub-optimal from a global perspective. Typical optimization techniques include gate sizing, buffering, useful skew, remapping and swapping connections to logically equivalent pins of a gate.

Early work on gate sizing proposed convex optimization with a continuous range of transistor sizes [6]. Subsequent work suggested use of more accurate polynomial models to provide a convex formulation to ensure optimality [7]. However, convex delay models are too inaccurate for modern technologies and do not adequately model standard-cell libraries with discrete cell sizes. Discrete gate sizing has historically been solved with greedy gate-sizing approaches, such as proposed by Coudert [8], focusing on timing-critical portions of the circuit and the best delay vs. power sensitivity trade-off, or vice versa, when trying to reduce power in portions of the circuit with timing slack. Greedy gate-sizing, which is still commonly used in commercial EDA tools, has been shown to be suboptimal in comparison to global optimization [9].

Several global optimization approaches for discrete gate sizing have been proposed, resizing gates to the discrete version that achieves the closest slack-slew values to those optimally calculated. Nguyen *et al.* [10] used inaccurate timing models for standard-cell delays, ignoring slew dependence and wire loads, and used Linear Programming (LP) to assign timing slack to resize gates for maximizing power savings subject to timing constraints. Chinnery *et al.* [9] improved this by accurately accounting for the delay changes on the neighboring gates due to a resize, but the runtime of this timing-accurate LP approach was prohibitive on larger designs, due to roughly quadratic runtime growth with design size. Held *et al.* [11] assigned slew targets, instead of delay targets. Fatemi *et al.* [12] presented sensitivities that can be used for timing, power, area, and slew optimization across multiple corners. The core algorithm in these papers can be sped up by relaxing the timing constraints with Lagrangian relaxation.

LR has been widely used for design optimization in recent years. It was first used by Chen *et al.* [13] for continuous wire and gate sizing. Hu *et al.* [14] used LR to define optimal continuous gate sizes that are then clustered to discrete sizes based on proximity to the optimal size. Ozdal *et al.* [15] formulated the LR sub-problem to trade off leakage power and fix timing violations, choosing gate sizes with Dynamic Programming (DP). Flach *et al.* [16] sized each gate to locally minimize the LR cost. This provided great leakage power savings, with faster runtime than other approaches, achieving the best results to date on the ISPD 2012 and 2013 benchmarks. Sharma *et al.* [17] extended this work with multi-threading and a new LM update method to reduce the number of iterations to converge, achieving a 15× speedup at the cost of 2.5% higher leakage power compared to [16]. We use essentially the same LR gate-sizing approach as in [16] and [17], extending it for register resizing and for handling sizing when facing conflicting late/early timing violations.

Roy *et al.* [18] extended the LR formulation to handle multiple modes and corners. Daboul *et al.* [19] used a resource sharing formulation, which is a specialization of LR. Shklover *et al.* [20] added clock skew to the LR formulation and resized both gates and clock buffers. Sharma *et al.* [21] combined LR gate sizing and slack-based clock skew assignment, while, more recently, Mangiras *et al.* [22] extended the LR formulation for timing-driven placement to include flip-flops, gates, and local clock buffers, leading to efficient placements. Our LR formulation in this paper includes multi-mode/multi-corner timing constraints and various optimizations.

Delay buffer insertion is a standard technique to fix early timing violations by increasing the path delay. Huang *et al.* [23] used an LP formulation to minimize the number of added hold buffers. Tu *et al.* [24] tackled hold violations across different power modes in ultra-low voltage designs. Wu *et al.* [25] presented an LP approach to model setup and hold constraints and assign delays that should be inserted on each node to solve hold violations. They then used DP to perform buffer insertion. Han *et al.* [26] proposed an integer LP hold-buffer insertion approach that achieves hold timing closure across multiple corners. Similarly, our framework has a delay buffer insertion transform that is used to fix hold violations.

Buffers can also be used to increase the drive strength and speed up timing on nets with high fanout and significant

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3025541, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

load capacitance. Van Ginneken [27] found the optimal buffer positions along the route of a net given a set of arrival timing constraints. Lillis *et al.* [28] extended this to account for multiple buffer types; while Wang *et al.* [29] presented a lower complexity buffering algorithm. Jiang *et al.* [30] formulated simultaneous transistor sizing and buffer insertion used for late critical path isolation. Liu *et al.* [31] used an LR-based cost function for buffer insertion on each net using only one buffer size from the library, while Ho *et al.* [32] allowed for multiple buffer options. Finally, Hu *et al.* [33] proposed an approximation method that reduces runtime with minimum impact on the result. For the TAU 2019 contest, the benchmarks of which we used in our evaluations, buffering is restricted to be either at the driver, or directly on the input pin of a cell. So, our buffering approach is simplistic, and, while permitting multiple buffers to be placed next to each other, we do not consider buffering at different locations along a net.

In our previous work [34], we extended the LR-based framework to include a set of various netlist transformations. In this work, we enhance and generalize our previous work by building an automatic recommendation system [3] to select which transformation would be applied in each iteration of the LR-optimization loop, thereby offering improved QoR. A preliminary approach of using traditional MAB algorithms [35] to coordinate optimization was presented in [36]. However, said work took a reverse approach: a MAB-guided global loop, with both LR and timing-based heuristics within each loop. In each round, one optimization algorithm is selected and applied to the design. When the algorithm finishes, a reward is recorded, in order to guide the selection of the following rounds using a traditional MAB algorithm that operates optimally in a stationary context, even though the problem at hand is dynamic, *i.e.*, the design changes as it gets optimized and the same optimization would yield a different reward when applied in a different round. Therefore, the overall optimization followed in [36] is purely statistical, without guarantees of convergence. On the contrary, this work employs a LR-based global optimizer that operates on a uniform cost function that is minimized locally by each selected optimization method. Also, the proposed rewarding mechanism and recommendation system are more elaborate and calibrated appropriately for the dynamic nature of the optimization process. Moreover, while the results of [36] were promising, they could not surpass the QoR obtained with deterministic approaches. In contrast, the proposed approach in this article substantially improves the QoR over existing state-of-the-art.

In a similar vein to our proposed work, but in a different context, adaptive optimization flows were explored in the context of compilers [37] [38], high-level synthesis for FPGAs [39], logic synthesis [40], and dynamic clock and voltage scaling for run-time power optimization [41].

III. Orchestrating LR Optimization with Autonomous Recommendations

Formulating a design optimization problem can consider timing, power, and area objectives, or constraints in various

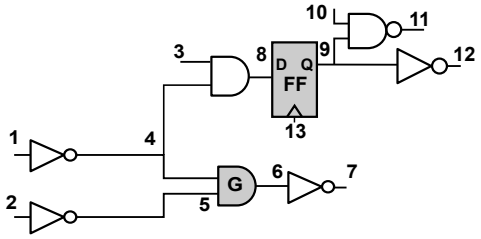


Fig. 1. Example design for demonstrating LM distribution and local cost calculation.

combinations. In this work, we aim at minimizing the sum of power, area, and Total Negative Slack (TNS) of the design (TNS is the sum of timing violations at the timing endpoints), such that the timing constraints are met.

$$\textbf{min}: \sum_{c\in\text{cells}} P(c) + A(c) - \sum_{j\in\text{POs}} slk_j^L - \sum_{j\in\text{POs}} slk_j^E \quad (1)$$
$$\textbf{s.t.:}\ slk_j^L \le 0 \text{ and } slk_j^E \le 0, \forall j \in \text{POs}$$
$$slk_j^L \le r_j^L - a_j^L \text{ and } slk_j^E \le a_j^E - r_j^E, \forall j \in \text{POs}$$
$$a_i^L + d_{i\to j}^L \le a_j^L \text{ and } a_i^E + d_{i\to j}^E \ge a_j^E, \forall \text{arcs } i \to j$$

$P(c)$, $A(c)$ are the leakage power and area of cell c; slk_j is endpoint js negative slack; a_j and r_j are pin js arrival and required times; and $d_{i\to j}$ is the arc delay from pin i to pin j. The arc delays include both cell input pin to output pin delays (cell arc delays), and the wire delay from a cell output pin to the input pin of another cell. The indices E and L represent early and late timing, respectively, and timing endpoint primary outputs (POs) include the input-D pins of all flip flops and the primary outputs of the design.

Lagrangian relaxation is applied on the formulated problem to incorporate the constraints into the cost function and simplify the cost function [13]. Each constraint is multiplied by λ weights called Lagrangian Multipliers (LMs), as shown in (2). The higher the LM value, the more critical the respective constraint is.

$$\begin{aligned}\textbf{min}: &\sum_{c\in\text{cells}} P(c) + A(c) - \sum_{j\in\text{POs}} slk_j^L - \sum_{j\in\text{POs}} slk_j^E + \\ &\sum_{j\in\text{POs}} \left(\lambda_{j0}^L slk_j^L + \lambda_{j0}^E slk_j^E\right) + \\ &\sum_{j\in\text{POs}} \left(\lambda_{j1}^L (slk_j^L - r_j^L + a_j^L) + \lambda_{j1}^E (slk_j^E - a_j^E + r_j^E)\right) + \\ &\sum_{i\to j\in\text{arcs}} \lambda_{i\to j}^L (a_i^L + d_{i\to j}^L - a_j^L) + \lambda_{i\to j}^E (a_j^E - a_i^E - d_{i\to j}^E) \quad (2)\end{aligned}$$

By differentiating (2) with respect to the arrival times, according to the Karush-Kuhn-Tucker optimality conditions, we end up with the following LM flow-conservation rules [13]:

$$\sum_{i\in\text{fanin}(j)} \lambda_{i\to j}^L = \sum_{k\in\text{fanout}(j)} \lambda_{j\to k}^L \text{ and } \sum_{i\in\text{fanin}(j)} \lambda_{i\to j}^E = \sum_{k\in\text{fanout}(j)} \lambda_{j\to k}^E \quad (3)$$

The LMs for the output pin of a cell are proportionally distributed to the LMs of the cells input-output arcs. For example for gate G in Fig. 1, net 6s outgoing LM is propagated to the LMs into net 6, preserving the equalities:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3025541, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

$\lambda^L_{4\to6} + \lambda^L_{5\to6} = \lambda^L_{6\to7}, \lambda^E_{4\to6} + \lambda^E_{5\to6} = \lambda^E_{6\to7}$. For flip-flops (FFs), following [22], the LMs added at their output Q pin are distributed to the clock-to-Q timing arc. For the example shown in Fig. 1, this translates to $\lambda^L_{13\to9} = \lambda^L_{9\to11} + \lambda^L_{9\to12}, \lambda^E_{13\to9} = \lambda^E_{9\to11} + \lambda^E_{9\to12}$.

Substituting (3) into (2) simplifies the objective function to (4). In the final simplified cost function (4), leakage power, area and delay are scaled by η, β and θ, respectively, to tackle the issue of power, area, and delay having different units. The scaling factors are derived according to the method proposed in [42].

$$\textbf{min} \sum_{c\in\text{cells}} \eta P(c) + \beta A(c) + \theta \sum_{i\to j} (\lambda^L_{i\to j} d^L_{i\to j} - \lambda^E_{i\to j} d^E_{i\to j}) \quad (4)$$

In this work, we follow an iterative optimization approach, where each iteration tries to optimize the global cost function (4) using one out of a set of eight netlist transformations, i.e., netlist change or restructuring optimization heuristics that try to minimize localized versions of the global cost function.

The overall LR-based design optimization loop is shown in Fig. 2. Initially, all cells are sized to the smallest size that does not violate any maximum load or slew constraints [43]. Then, the LMs for every arc are initialized to 1, and the LR-based optimization begins.

At the beginning of each iteration, and before applying the selected transformation, timing and LMs are updated using the process described in Sections III-A and III-B, respectively. The netlist transformation that is applied on the design uses the updated LM values for all local optimum decisions. Which netlist transformation is applied to the design is decided using an autonomous recommendation engine that operates according to the rules described in Section III-C.

The optimization loop stops when the solution quality does not improve for a number of consecutive iterations. If timing constraints are satisfied, the quality of the solution is equal to the area/power improvement. On the contrary, if timing violations are still present, termination is judged by TNS improvement. In the end, a set of brute-force recovery steps are executed.

To provide the recommendation engine the opportunity to run many different transformations before deeming that the design can no longer be optimized, the number of iterations that we wait before stopping the optimization is arbitrarily set to be 25% higher than the number of available transformations, i.e., 10 iterations for 8 netlist transformations.

A. Incremental Timing Updates Critical corners

At the start of each iteration, we should be aware of the timing of the design for all corners under consideration. Initially, an incremental timing update for all corners is performed and the two critical corners are selected: the most critical early and late corner. The critical late corner is the one with the worst late TNS, or the corner with the lowest late total slack, if no corner has late violations. The same applies for the most critical early corner.

To save runtime during optimization, incremental timing update on all corners is performed once every five iterations. In the other iterations, timing updates are performed only on the already identified critical corners. The critical corners are not statically determined throughout the optimization, but they are re-evaluated every five iterations of the LR optimization loop. Our experimental results show that updating the most critical corners more often introduces unnecessary runtime overhead without significant improvement of the overall QoR.

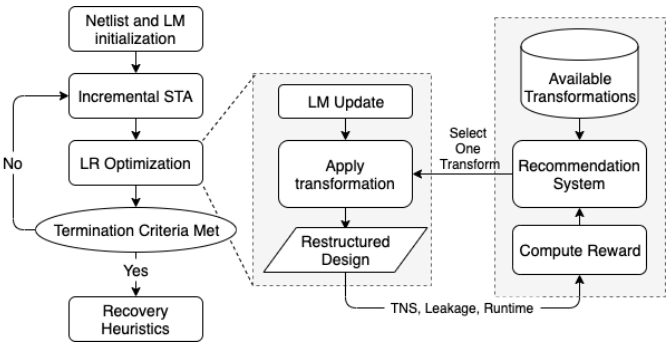


Fig. 2. The overall design optimization flow. Netlist transformations are applied sequentially on the design, as selected autonomously by the recommendation/rewarding system. Each transformation relies on LM weights, thus contributing to the convergence of the global LR-based optimization loop.

B. LM Updates

LMs are updated using the modified sub-gradient method proposed in [44]:

$$\lambda^L_{j0} = \lambda^L_{j0}\left(\frac{a^L_j}{r^L_j}\right), \lambda^E_{j0} = \lambda^E_{j0}\left(\frac{r^E_j}{a^E_j}\right) \forall i \in \text{POs}$$

$$\lambda^L_{i\to j} = \lambda^L_{i\to j}\left(\frac{a^L_i + d^L_{i\to j}}{a^L_j}\right), \lambda^E_{i\to j} = \lambda^E_{i\to j}\left(\frac{a^E_j}{a^E_i + d^E_{i\to j}}\right) \forall \text{arc}_{i\to j}$$

The delays, the arrival times, and the required arrival times for early-late timing modes are calculated from the corresponding critical corner. Gate sizing methods like [16] and [17] utilized exponential LM updates for faster convergence. Even if this is the proper choice for gate-sizing only transformations, it does not fit well in our flow that utilizes many different netlist transformations. In this case, the LMs should adapt smoothly to the changes in the designs timing, allowing for better co-operation across the various optimization heuristics.

The updated values of output LMs should be distributed to all nets satisfying the flow conservation conditions (3). The distribution is performed by traversing the circuit in reverse topological order. At each visited cell, the sum of LMs at the output pins is distributed to the LMs of the input pins. When an LM value needs to be distributed to multiple incoming arcs, this distribution is done based on the ratio of the LMs of the corresponding timing arcs. Such distribution increases the LMs on critical paths, and, therefore, the timing violations are expected to be minimized. Also, since LMs are accumulated at each branching point, the higher the number of violating endpoints affected by an arc, the higher the value of the corresponding LM.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3025541, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

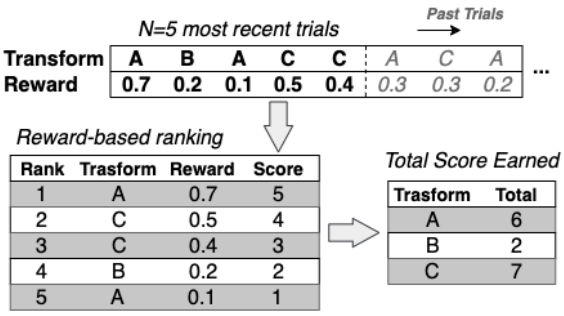


Fig. 3. Reward-based relative ranking of the transformations applied in the last N most recent trials.

C. Automatic Transformation Recommendation

The proposed recommendation engine takes the performance observed by each netlist transformation so far and tries to pick the next transformation. The goal is to exploit as much as possible the current best transformation, while maintaining some exploration of the other transformations, in case one of them becomes more efficient at a later stage of the optimization. Well known bandit algorithms, such as the Upper Confidence Bound (UCB) and its variants [35], have been proven to optimally solve the exploration vs. exploitation dilemma, albeit in a *stationary* context [3]. However, the design optimization problem that we are trying to solve is inherently *dynamic*. The design is altered in each iteration, and successive plays of the same optimization algorithm would yield different rewards [36]. To bypass this inherent inefficiency when using bandit algorithms in a dynamic environment, we make decisions based on differential criteria, following the approach used in OpenTuner [37].

Score Computation: When transformation k has been applied on the design in a certain iteration, we record its earned reward, R_k, using the reward function (7) described in Section IV. Then, we compute the average payoff for each one of the transformations, as follows:

1) Sort the transformations applied in the last N trials according to their received rewards. In the example shown in Fig. 3, the last five ($N = 5$) trials are ranked based on the reward earned when applied to the design. Transform A had the highest reward of 0.7 and it is ranked first. Transform C follows with a reward of 0.5.
2) Each transform receives a score according to its position in the rank. Position 1 receives a score of N, position 2 a score of $N - 1$, while the last position receives the lowest score of 1.
3) Compute the total score of each transform by adding the score of all of its appearances in the rank list. Thus, in the example of Fig. 3, C receives a total score of 7, since it occupied the second and the third positions in the rank with corresponding scores of 4 and 3. The total score of each transform actually reflects how good the transform was in the last N trials relative to the others.

The normalized payoff Q_k of each transform k is just the normalized version of its total score:

$$Q_k = \frac{\text{score}(k)}{1 + 2 + ... + N} \tag{5}$$

Selection policy: The recommendation engine would select transformation k to apply in this trial, which maximizes

$$(1 - c)Q_k + c\sqrt{\frac{2 \ln N}{N_k}}, \tag{6}$$

where N is the window size and N_k is the number of times that k has been played in the last N trials. c is a tuning variable between 0 and 1 that, when increased, favors the exploration of rarely selected transformations (low values of N_k) versus exploiting the ones with the best performance so far (high values of Q_k). For the example shown in Fig. 3, $(Q_A, N_A) = (6/15, 2)$, $(Q_B, N_B) = (2/15, 1)$, and $(Q_C, N_C) = (7/15, 2)$. Assuming a balanced exploit vs. explore strategy with $c = 0.5$, transformation C would be applied in the next trial.

The initial payoffs Q of each transform can be obtained either by applying each transformation once (no pretraining), or by using the average scores derived from the previous application of this algorithm on various benchmarks (pretraining).

Productivity Rule: Once a transformation is applied on the design, the changes that it caused *are not necessarily kept*. If the quality of the obtained result is worse by more than 10% in any of the targeted aspects, i.e., timing, leakage power, and area, then the changes are discarded and the design is recovered to its previous state. We allow this small degradation to occur, since it may help the subsequent transformations. For instance, when timing is closed, meaning that timing violations are below a certain threshold [45] (arbitrarily chosen to be 10% of the clock period in our experiments), and we are optimizing for leakage power and area, we allow for small power degradation, since this can lead to positive timing slack that would allow us to optimize leakage power later on.

IV. Guiding the Automatic Recommendation: The Reward Function

To sort out the relative performance of the most recently applied transformations, each transformation k applied to the design in the ith iteration receives a reward $R_k(i)$. The reward function demonstrates the effect of the chosen optimization method on the performance metrics that characterize the circuits behavior, as well as to the runtime cost paid in achieving such metrics. According to the productivity rule, a reward is recorded only if the restructured netlist is actually kept, i.e., it does not deteriorate any QoR metrics by more than 10%. Overall, the reward $R_k(i)$ is equal to:

$$R_k(i) = r_k \left(\delta R_{\text{timing}}(i) + (1 - \delta) R_{\text{power-area}}(i)\right) \tag{7}$$

$R_{\text{timing}}(i)$ and $R_{power-area}(i)$ represent how efficient the selected transformation was in improving TNS, and reducing leakage power or area, respectively. Factor δ defines the importance of timing optimization compared to power-area optimization and it is changed during the flow's execution. For instance, in the first iterations, we prioritize timing optimizations with $\delta = 0.8$. Once timing closure is achieved, power-area reduction is prioritized by changing δ to 0.2.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3025541, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Parameter r_k is a scaling factor that depends solely on the runtime of transformation k. We include this in the reward function, so that the recommendation algorithm is able to distinguish between two methods that have a similar QoR impact, but different runtime needs. We penalize the reward received by slow methods with low values for r_k. Initially, $r_k = 1$ for all transformations. On the following iterations,

$$r_k = 1 - \frac{\text{avg\_runtime}_k}{\text{runtime\_target}} \tag{8}$$

A high average runtime for method k will penalize that method when it is comparable to the runtime target. Setting a strict runtime target can reduce the total runtime of the optimization, since faster methods will be picked more often by receiving higher rewards, possibly at the cost of inferior QoR. When assigning a relaxed runtime target, r_k is used as a tie-breaker between equally efficient methods in terms of timing and power/area, but with different runtimes.

A. Timing Reward

The timing reward $R_{\text{timing}}(i)$ represents the tradeoff between timing improvement $\Delta T(i)$ and power/area overhead $\Delta PA(i)$ achieved in the ith iteration.

$$R_{\text{timing}}(i) = \begin{cases} \frac{\Delta T(i)}{\min(\Delta PA(i), 0.01)}, & \text{if timing is improved} \\ 0, & \text{if timing is not improved} \end{cases}$$

If timing closure has not been achieved, timing improvement ΔT refers to the average percentage of TNS improvement between two consecutive iterations i and $i-1$ for late and early mode:

$$\Delta T(i) = \frac{1}{2}\frac{\text{TNS}_i^L - \text{TNS}_{i-1}^L}{\text{TNS}_{i-1}^L} + \frac{1}{2}\frac{\text{TNS}_i^E - \text{TNS}_{i-1}^E}{\text{TNS}_{i-1}^E} \tag{9}$$

After timing closure, ΔT refers to the percentage of total slack increase.

Similarly, $\Delta PA(i)$ is the percentage increase of the total sum of scaled leakage power and area of all cells in one iteration relative to the previous iteration:

$$\Delta PA(i) = \frac{\sum(\eta P + \beta A)_i - \sum(\eta P + \beta A)_{i-1}}{\sum(\eta P + \beta A)_{i-1}} \tag{10}$$

The definition of the timing reward reflects our strategic decision *to prefer overall efficiency over exclusive TNS improvement*. For instance, a method that achieves TNS reduction of 10%, while degrading power and area by 5%, is preferred over a method that reduces TNS by 20%, but degrades power and area by 15%.

$\Delta PA(i)$ is lower-bounded to 0.01 in the denominator of R_{timing} to avoid cases where the power/area difference is too small and diminishes the impact of timing in the reward. A very low $\Delta PA(i)$, can cause the timing reward to overshoot regardless of the actual timing improvement. Without this lower bound, a method that marginally improves timing by 0.1% and degrades leakage power and area by 0.001% would receive a timing reward of 100, which is unrealistically high for the small impact the method had. By lower-bounding the power and area improvement, the reward will be equal to 0.1.

B. Power-Area Reward

The power/area reward has a different meaning depending on the phase of the optimization. If timing has not closed, power reward reflects the tradeoff between timing degradation and power/area improvement. If timing has closed, and the LR-optimization focuses on power/area reduction, the reward is simply the percentage of power/area reduction across iterations, irrespective of timing. This is safe to do, since, according to the productivity rule, if the applied transformation worsens any quality metric by more than 10%, its effect is not kept and the design is restored to its previous state. Therefore, once timing has closed in some iteration, any timing degradation that may appear as a result of the applied power-area optimizations will be very small due to the 10% per-iteration limit. Overall,

$$R_{\text{power-area}}(i) = \begin{cases} 0, & \text{if power-area increased} \\ \frac{\Delta PA(i)}{\min(\Delta T(i), 0.01)}, & \text{if timing is not closed} \\ \Delta PA(i), & \text{if timing is closed} \end{cases}$$

This time, we bound $\Delta T(i)$ to 0.01, in order to avoid methods that cause a minimal TNS change getting high rewards regardless of their power and area effect.

V. Netlist transformations

The automatic recommendation engine embedded inside the LR-based optimization loop utilizes a set of eight netlist transformations that collectively optimize the design.

Two versions of flip-flop and gate resizing are employed that choose an appropriate size and threshold voltage for all cells in the design. The utilized cell resizing algorithm is a generalized version of previous algorithms proposed in the context of LR-based sizing [16], [17], while also covering flip-flop sizing and conflicting cases that cannot be handled by previous approaches. The arsenal of transformations also includes two versions of buffering: (a) hold-buffer insertion for reducing early timing violations, and (b) local late buffering that conditionally inserts buffers at the output of cells with high-fanout loads. Pin swapping attempts to swap the sink pin of the most timing critical net of the gate with another equivalent pin, in order to help the timing-critical net with its violations. Simplified gate merging/splitting locally compose or decompose adjacent gates to trade off number of levels versus gate complexity per level. Finally, the eighth transformation involves applying useful clock skew on flip-flops.

All of the above methods are driven by the LM values of each iteration and try to optimize their LR-based cost functions using only slack information to ensure that they are not degrading the timing violations. Any other local optimization heuristics that operate in a similar manner can be added in the library of available transformations.

A. Cell Resizing

The cell resizing Algorithm 1 examines all different versions of each cell and selects the one that minimizes the local cost

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3025541, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Algorithm 1: Cell resizing algorithm

```
foreach cell c ∈ candidate_cells in topological order do
    best_cost ← localCost(c) ;
    best_version ← cellVersion(c) ;
    neg_slack ← localNegativeSlack(c) ;
    foreach version v ∈ equivalent_versions do
        resize cell c to v;
        if (violates_load_constraints(c)) then
            skip version;
        end
        update timing locally;
        new_neg_slack ← localNegativeSlack(c) ;
        if new_neg_slack < γ · neg_slack then
            skip version;
        end
        new_cost ← localCost(c);
        if new_cost < best_cost then
            best_cost ← new_cost ;
            best_version ← new_version ;
        end
    end
    resize cell c to best_version ;
    update timing locally ;
end
```

function (11), without introducing load violations and without degrading the slack of its nets over a threshold γ [16].

$$\eta P(v) + \beta A(v) + \theta \sum_{i \to j \in \text{local\_arcs}} \left(\lambda^L_{i\to j} d^L_{i\to j} - \lambda^E_{i\to j} d^E_{i\to j}\right) \quad (11)$$

$P(v)$ and $A(v)$ are the leakage power and area, respectively, of size v of the examined cell.

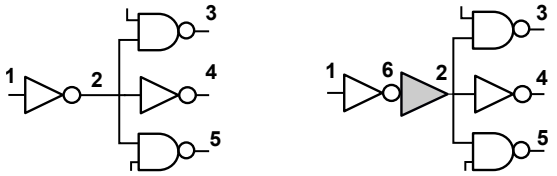
The local timing arcs for each cell include the arcs of the resized cell, its immediate fanin and fanouts, and the arcs that share a common fanin with the resized cell. For gate G in Fig. 1, the local arcs include: arcs of G $(4 \to 6, 5 \to 6)$, the fanin arcs $(1 \to 4, 2 \to 5)$, the fanout arcs $(6 \to 7)$, and the common fanin arcs $(4 \to 8)$. Flip-flops are handled similarly to gates. For flip-flop FF in Fig. 1, the local arcs include arcs $(13 \to 9)$, $(3 \to 8, 4 \to 8)$ and $(9 \to 11, 9 \to 12)$.

Handling of conflicting constraints: Relying on the local cost function (11) may lead to contradicting sizing decisions when a cells output pin has both early and late violations. No matter which decision the sizer takes, there is no choice that will reduce both timing violations. The traditional LR-based sizing algorithm [16], [17] would try to reduce the violations on the side with the higher LM. For example, if the early LMs are higher than the late ones, it would try to downsize the cell. However, this choice worsens late slack. So, eventually, the algorithm balances the slacks on both (early-late) sides, without truly solving any violations.

In such cases, we prioritize late timing violations over early ones, since early violations can possibly be fixed by other techniques such as hold-buffer insertion. To do this, we omit the terms $\lambda^E_{i\to j} d^E_{i\to j}$ from the cost function (11). Thus, only late-mode LMs and delays determine the cost function and guide the sizing decisions.

Full and fast gate sizing: To allow the recommendation engine to trade off higher sizing quality with lower runtime, we employ two sizing transformations: *full cell sizing* that examines every sequential and combinational cell; and *fast cell sizing* that operates on a selected subset of cells.

Specifically, when timing is not closed, fast cell sizing resizes only cells that either have negative slack on their pins, or are immediate fanouts of cells with negative slack (i.e., to downsize a load to speed up the path). When timing has closed, fast cell sizing picks the cells with the highest power/area potential that also have positive slack. Power potential refers to the difference between their current leakage power and the smallest leakage power they can have without causing maximum slew and load violations, which is the leakage they had after the initial downsizing. Area potential is defined similarly.



(a) Attempting buffering on pin 2 (b) After buffer insertion

Fig. 4. Example of late buffer insertion on pin 2. The local cost (11) without a buffer is computed over the arcs for pin 2: $1 \to 2, 2 \to 3, 2 \to 4, 2 \to 5$. The local cost with a buffer inserted is the cost of the arcs around the buffer: $1 \to 6, 6 \to 2, 2 \to 3, 2 \to 4, 2 \to 5$, including the buffer's power and area. If the new cost is lower than the original, the buffer is kept provided that no maximum slew/load violation is introduced.

B. Local Buffer Insertion

Buffering is a versatile design optimization that can be efficiently applied for various design targets. In this work, we employ two forms of local buffering that target early and late timing violations:

1) Late timing violations: Buffering can reduce the output load of cells with a large fanout, thus decreasing their delay and helping reduce late slack violations.

Initially, the gates with negative late slack are sorted by their output capacitance over input capacitance ratio. Then, buffer insertion is attempted on the source of the fanout net of the top 100 gates. All available buffer choices are examined and the one with the lowest cost is kept. If the lowest cost with the buffer added is smaller than the cost without the buffer and buffer insertion does not introduce slew/load violations, the selected buffer is actually inserted. The local cost around the buffer is calculated using equation (11) involving all arcs connected to the net where the buffer will be inserted. An example of the involved timing arcs is shown in Fig. 4.

2) Early timing violations: Solving early (hold) timing violations requires slowing down the signal propagation on violating paths. This can be achieved by appropriate delay buffer insertion. Hold buffers should be inserted in positions where they will affect many hold violating paths, thus minimizing the number of buffers that need to be added. We avoid adding buffers on paths with both late (setup) and early violations. Hold buffers are only inserted in pins where there is room to trade off positive late slack with negative early slack.

For every hold violating endpoint, we store the worst early path through it and keep in a list of candidates the paths pin

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3025541, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

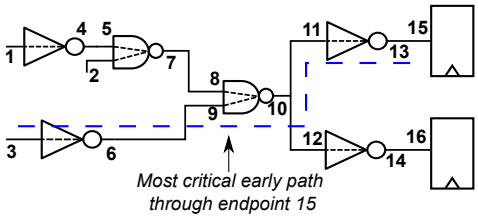


Fig. 5. Hold buffer insertion example. Pin 10, which is part of the early critical path, has the highest $\lambda^E - \lambda^L$ difference and is examined to receive extra buffers for alleviating hold-time violations.

with the highest difference $\lambda^E - \lambda^L$ (i.e., critical in early timing and not critical in late timing) and with positive late slack. Then, on those candidate pins, we add delay by using a chain of identical inverters, gradually consuming the available positive late slack.

Let us assume that, in Fig. 5, the candidate pin for the worst early path through endpoint 15 could be pin 10. To add a chain of buffers at pin 10, without the need to resize that driving cell, we want the cell driving pin 10 to see a similar output load. If adding a buffer at the output reduces the load, it would speed up the driver, contrary to our purpose of adding delay on the path with the early timing violation. In Fig. 5, the appropriate buffer for adding delay on pin 10 should have an input capacitance close to $C_{\text{in}}^{11} + C_{\text{in}}^{12}$, plus any effective wire capacitance of the net connecting pins 10, 11, and 12.

Since all the buffers of the chain would see the same load (similar to their input capacitance), we can assume that they will all have the same delay d^E in early mode timing. Therefore, for a negative early slack of slk^E, we need at least $N_{\text{min}} = -slk^E/d^E$ buffers to remove early slack violations.

At the same time, assuming that the available positive late slack is slk^L, we cannot add more than $N_{\text{max}} = slk^L/d^L$ buffers, where d^L is the delay of the same buffer in late mode timing. Since we decided not to consume more than 50% of the available positive late slack, we limit the maximum number of buffers allowed to $N_{max}/2$. Overall, the number of buffers added is equal to $\min(N_{\text{min}}, N_{\text{max}}/2)$ provided that buffer addition does not introduce any maximum slew/load violation.

C. Cell merging and splitting

While gate sizing and buffering transformations are very effective in improving timing and reducing the area and power of the design, merge-split transformations can broaden the solution space by trading off the number of logic levels with the simplicity of the gates per level. In this work, we explore simple merge/split options across independent AND/OR/NAND/NOR gates, while complex cases of merging/splitting combined AND-OR Boolean functions are not supported. Both transformations are applied on every cell eligible for merging or splitting in topological order.

1) Cell merging: Cell merging replaces two serially connected logic gates with a multi-input gate of the same functionality. For example, in Fig. 6, two-input AND gates G1 and G2 are replaced by a three-input AND gate G3. The new gate is always assumed to replace the end gate, e.g., G2 in Fig. 6, and spatially placed in its position. The wire resistance and capacitance (RC) parasitics of the net connecting the two gates (net 7 in Fig. 6) are added to the input net parasitics of the fanin gate, i.e., nets 4 and 5 in Fig. 6. To judge if this replacement is beneficial, we compute the local LR cost (11) before and after the merge. If the LR cost after the merge is lower, the local slack is not degraded, similar to the cell sizing local slack check in Algorithm 1, and there are no load violations introduced, the merge is accepted.

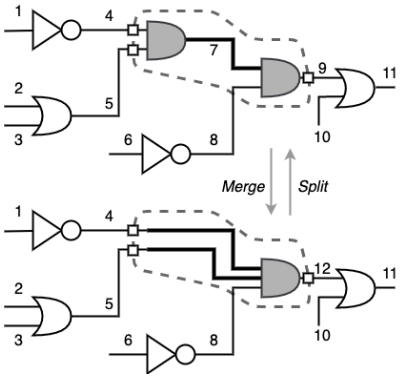


Fig. 6. Merge/Split example highlighting the placement of the replaced gate and the timing arcs used for local cost calculations.

For computing the local cost before the merge in the example shown in Fig. 6, we use the timing arcs of the two gates that will be merged, the arcs of their fanins and their immediate fanouts: $\{1 \rightarrow 4, 2 \rightarrow 5, 3 \rightarrow 5, 4 \rightarrow 7, 5 \rightarrow 7, 6 \rightarrow 8, 7 \rightarrow 9, 8 \rightarrow 9, 9 \rightarrow 11\}$.

After the merge, some timing arcs inevitably vanish. Therefore, for the example shown in Fig. 6, the local cost *after the merge* is calculated by using arcs $4 \rightarrow 12, 5 \rightarrow 12$, and $8 \rightarrow 12$, in the place of arcs $4 \rightarrow 7, 5 \rightarrow 7, 7 \rightarrow 9$, and $8 \rightarrow 9$. The LM used for these new timing arcs is derived after redistributing locally the LM values that were already present on the fanout of the cell to be merged. To compute the local cost for the merged gate, we need to know its exact size. Instead of trying various sizes (this is a task of cell sizing), the new merged cell is assigned the size of the first gate, or the size that exhibits the closest input capacitance to that gate.

2) Cell splitting: The cell splitting transformation separates a multi-input gate to two gates with fewer input pins. The gate splitting algorithm is applied on every multi-input AND/OR/NAND/NOR cell that can be split in cells with a smaller number of inputs.

To limit the possible splitting options, in this work, a cell is always split to a cell with two inputs, and a second cell for the rest of the inputs. The sizes of the resulting gates will be the same as the size of the original gate, or the one that matches more closely its input capacitance. The two new cells are connected serially (connected with a net of minimum length), with the multi-input one driving the two-input cell. The most critical late net is connected to the two-input cell and the most critical early net is connected to the multi-input cell. If the same net is the most critical in both early and late timing, it will be assigned to the position satisfying the most critical mode, thus improving the timing of the local

most critical path. A split actually occurs only when the local cost after the split (calculated similarly to cell merging) is less than the cost before the split, without introducing maximum load/slew violations.

D. Pin Swapping

Cell merging/splitting can be combined with pin swapping to further improve timing, by taking advantage of cells where the input-to-output timing differs across pins. When logic functionality allows, we examine connecting the most timing-critical input to the fastest input-to-output path of the gate. The pin swapping algorithm visits every combinational gate with negative slack, or on the immediate fanout of a cell with negative slack in topological order. For each gate, the input pin with the most negative early or late slack is swapped with the other equivalent pins. After each swap, the timing is updated locally and the local cost function (11) of the gate is recalculated. After every swap is done, the swap with the best local cost which doesn't violate slew/load constraints is kept and the next gate is processed.

E. Useful Clock Skew Assignment

To maximize the overall optimization efficiency, we combine the aforementioned datapath-driven optimizations with clock skew optimizations, which allow for wider-range timing tuning. Instead of setting up a new linear program to determine optimal clock skew offsets [46], we decide locally and incrementally on the clock arrival time of each flip-flop, thus allowing useful clock skew to adjust smoothly to the optimizations of the other netlist transformations.

Based on the formulation presented in [22], LMs can be propagated to the clock pins of the flip-flops as follows: $\lambda^L_{CLK} = \lambda^L_Q + \lambda^E_D$, $\lambda^E_{CLK} = \lambda^E_Q + \lambda^L_D$. Thus, the sign of $\lambda^L_{CLK} - \lambda^E_{CLK}$ determines if we should delay or speed up clock arrival. If it is positive, a constant delay d_{clk} is subtracted from clock arrival, favoring slk^L_Q and slk^E_D; instead, if it is negative, the same delay is added to clock arrival in favor of slk^L_D and slk^E_Q. In both cases, the update of the useful clock skew is kept as long as it does not degrade the late/early worst slack on all data pins of the flip-flop. d_{clk} equals the average delay of all available clock buffers driving a minimum sized flip-flop of the library.

Since we operate in a multi-corner environment, the delay added should be scaled across corners. Since the clock arrival offsets would be implemented using the available clock buffers of the library, we compute a scaling factor for each corner by approximating the ratio of the clock buffer delay of that corner compared to the buffer delay of the typical corner, for multiple input slew and output load values. So, when a delay d_{clk} is added on a clock pin for the typical corner, a delay $d_{clk} \cdot ratio_{cr}$ will be added for corner cr, where $ratio_{cr}$ is the average buffer delay ratio for corner cr and the typical corner.

F. Recovery heuristics after the LR optimization loop

After the LR-based optimization is complete, some recovery steps are executed as a way to perform minor enhancements to the QoR and eradicate any small leftover timing violations. In contrast to the eight previous transformations, these recovery steps cannot be selected by the recommendation engine.

The power/area reduction step attempts to downsize every gate in forward topological order [16]. If the downsize degrades the local slack, the move is reverted; else it is kept. After every forward topological pass, an incremental timing update is performed. The process stops when the TNS is degraded, or when no more downsizes are possible.

The timing recovery step resizes the driver cell affecting the most violating endpoints [16]. The sizes tested are the immediately smaller and bigger size. After either move, we perform local timing update on the critical corner and calculate the new local negative slack on the cells output. If it is improved compared to the initial slack, we perform an incremental timing update and ensure that the TNS has improved too. If it has, this cell version is kept, and the process is repeated. If the timing has not improved, we revert the change and move on to the next most critical cell. Timing recovery stops if all timing violations are solved, if the TNS stops improving, or if a certain number of incremental timing updates is reached. Timing recovery is performed twice: once for the remaining late timing violations, and once for the early timing violations.

For the last remaining early timing violations, a buffer chain is inserted in front of every hold-violating endpoint, until the violations are fixed. This is the last optimization step performed, ensuring that the final design is hold-violation-free.

VI. Experimental results

The proposed method was implemented in C++ inside RSyn [47] after extending it for multi-corner timing analysis. The presented results were evaluated using the benchmarks of TAU 2019 [4] and ISPD 2013 contests [5]. All experiments were performed on a Linux workstation using a 3.6 GHz Intel Core i7-4790 with 4 cores and 32 GB of RAM. The results are validated using OpenTimer [48], which is the reference timer in the TAU 2019 contest.

A. Setup of the recommendation engine

The recommendation engine operates on a sliding window of $N = 32$ rewards (four times the number of available transformations). A very small window is prone to being overflowed by one method, which is then repeatedly ranked highly, not because of its quality but because of its high presence in the window. On the other hand, a very large window can give a method a high ranking because of its performance many iterations ago, even if the timing profile of the design has completely changed.

To avoid blindly running every heuristic at the start of the optimization, the recommendation engine was pre-trained using ten average-sized benchmarks from the TAU 2019 and ISPD13 benchmark suites. Each one of these benchmarks was optimized by running every method serially (for example, full gate sizing on iteration 1, pin swapping on iteration 2, late buffering on iteration 3, etc.) and storing the rewards of the 32 first iterations. Thus, the initial window included four times

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3025541, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE I
THE INITIAL AVERAGE REWARDS FOR EACH TRANSFORMATION. WHEN TIMING IS NOT CLOSED, THE TIMING REDUCTION REWARDS ARE USED. WHEN TIMING IS CLOSED, REWARDS ARE RE-INITIALIZED TO THE AVERAGE REWARD OF THE POWER REDUCTION PHASE.

Method	Full sizing	Pin Swap	Late Buff.	Hold Buff.	Clock Skew	Gate Merge	Gate Split	Fast sizing
Timing reduction	0.133	0.043	0.129	0.012	0.247	0.078	0.176	0.180
Power reduction	0.262	0.027	0.020	0.020	0.128	0.182	0.081	0.282

each all eight transformations. The duration of the pre-training runs was 10 minutes.

During pre-training, we recorded for each transformation two sets of average rewards. The one refers to the timing reduction phase (i.e., when timing is not closed) and the other to the power reduction phase (i.e., when timing is closed and power-area reduction is targeted). The initial rewards of each transformation derived in the training phase are shown in Table I. At the start of the optimization, the initial sliding window is populated by the average rewards of the transformations during the pre-timing closure phase. Once timing closure is achieved, the sliding window is re-initialized to the average rewards of each transformation in the power reduction phase.

The behavior of the recommendation engine also depends on the explore coefficient c in (6). Setting c high will treat all methods equally, instead of focusing on the most efficient ones. On the contrary, setting c at a low value will limit the optimization to only a few methods, failing to reap the benefits of the combined application of all heuristics. We observed the best results when $c = 0.35$ to 0.5. The results presented were run for c being equal to 0.4.

Finally, the runtime target of the runtime scaling factor (8) that determines how much each transformation is penalized due to runtime was set to 40 mins for all designs. This target actually has no effect in small designs and only constrains the runtime of large designs. Larger runtime targets improved only slightly the overall QoR. Note that the runtime target is not a hard constraint, but just a penalty factor that helps increasing the rewards of fast transformations relative to equally-good but slower transformations.

B. Results on the benchmarks of the TAU 2019 contest

The TAU results were compared against the TAU contest winners results [49], which were extracted by running the executable that they submitted to the contest (and also provided to us). Each benchmark includes SPEF and SDC files for each netlist. The TAU benchmarks provide five different standard-cell library files, one for each corner. Each library provides a range of cells of varying complexity. It includes positive, negative and non unate cells, registers with Set/Reset pins, and complex combinational cells, such as half and full adders. Each library cell has a variety of different versions, ranging from 1 to 6 separate sizes and 1 threshold voltage. The designs only have timing constraints on register-to-register paths, leaving primary inputs and outputs unconstrained. There is also a different transition time constraint for each corner.

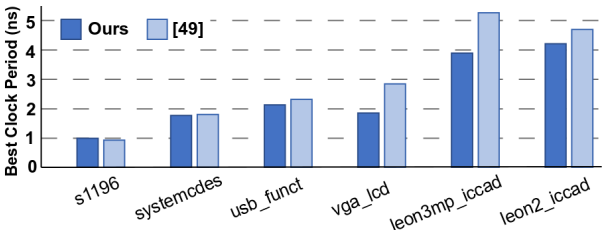


Fig. 7. The best clock period achieved for each benchmark of the TAU 2019 contest covering all corners, by the proposed method and the winner of the contest [49].

Also, all benchmarks include clock trees that are unrealistic, since they do not have RC parasitics, and the clock arrival times are highly unbalanced. For this reason, the TAU 2019 contest winner removed the clock buffers from the clock tree and rebuilt it without including any RC parasitics of the nets connecting the clock-tree buffers. Note that assuming ideal wires on the clock tree was permitted by the contest, but not realistic. To allow for a more realistic comparison (assuming that designs are compared in a pre-CTS stage before clock tree synthesis), we removed all clock buffers from both the initial set of benchmarks fed to our algorithm, and from the final netlist of the TAU 2019 winner. Nevertheless, for the latter, we kept the assigned clock pin arrival times (acting as useful clock skew values), so that the timing performance of the TAU 2019 winner remained unaffected.

Fig. 7 presents the best clock period for which each flow achieves closure for all corners. Our flow achieves 17% lower period, on average. We achieve a better clock period for every design, with the exception of s1196.

Table II depicts the leakage and area achieved when both flows are constrained by the worst clock period of Fig 7. Our flow achieves a lower leakage and area for every benchmark, resulting in 6% area and 15% power improvements, on average. Our QoR on the larger benchmarks is much superior to the result of the contest winner, in exchange for a longer runtime. The runtime overhead mainly stems from methods that operate on the entire design, such as full gate sizing and cell merging/splitting. Even if gate-sizing is multithreaded using eight threads each time is applied, all the rest transformations are single-threaded.

The proposed recommendation engine can operate equally well (albeit with an increased runtime), even if it is not pre-trained and the average payoff of each transformation is initialized to 0. Table III depicts the leakage power and area results, when the designs are optimized for the clock period of Fig. 7 (the best our flow can achieve), with and without pretraining (with pr. and w/o pr., respectively). In all cases, it is evident that the proposed flow, *even when applied blindly*, achieves equally good, or better QoR. Pre-training just improved runtime by favoring faster convergence.

Therefore, the proposed optimization flow not only achieves timing closure and competitive power/area reductions, but it also does so autonomously, without relying on any previous knowledge of the circuits, and without requiring any manual tuning/intervention.

In order to demonstrate the importance of the proposed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3025541, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE II
COMPARISON OF THE LEAKAGE AND AREA AT THE CLOCK PERIOD WHERE BOTH FLOWS ACHIEVE CLOSURE FOR ALL CORNERS

Design	#cells	Period (ps)	Leakage (μW)			Area (μm^2)			Runtime (min)		
			Ours	[49]	Save(%)	Ours	[49]	Save(%)	Ours	# Iters	[49]
s1196	584	985	12	13	7.69	552	569	2.99	0.03	24	0.03
systemcdes	2825	1788	74	96	22.92	3368	3975	15.27	0.19	46	0.35
usb_funct	10535	2306	370	402	7.96	17599	17812	1.20	0.77	45	0.87
vga_lcd	87958	2826	2780	3106	10.50	142153	146257	2.81	6.82	20	0.40
leon3mp_iccad	649191	5246	19351	23816	18.75	957947	1030860	7.07	57.25	22	6.03
leon2_iccad	793286	4677	23989	30354	20.97	1203920	1312960	8.30	46.72	19	7.53
average saves	-	-	-	-	14.80	-	-	6.27	-	-	-

TABLE III
COMPARISON OF THE LEAKAGE AND AREA AT THE BEST CLOCK PERIOD ACHIEVED BY OUR FLOW, WITH (WITH PR.) AND WITHOUT (W/O PR.) USING PRETRAINED REWARD INITIALIZATION.

Design	Period (ps)	Leakage (μW)		Area (μm^2)		Runtime (min)	
		with pr.	w/o pr.	with pr.	w/o pr.	with pr.	w/o pr.
s1196	985	12	12	552	551	0.03	0.03
systemcdes	1754	80	81	3510	3533	0.16	0.17
usb_funct	2116	404	412	18412	18654	1.51	2.89
vga_lcd	1840	2796	2781	140813	141477	9.42	10.33
leon3mp_iccad	3872	19686	19613	963614	962976	92.84	93.68
leon2_iccad	4190	24170	24003	1208100	1206040	91.81	110.35

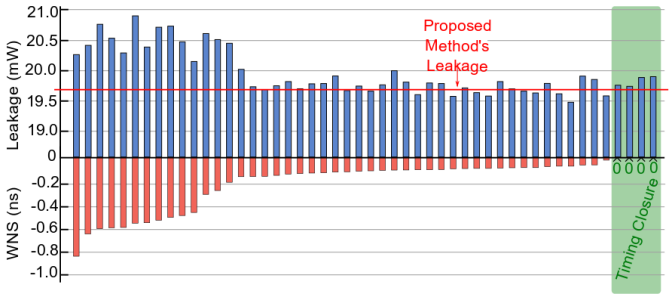


Fig. 8. WNS and leakage power for 50 random experiments on the leon3mp_iccad, without using the proposed recommendation engine and targeting the best clock period achieved by the proposed flow.

recommendation engine to the final QoR, we ran additional experiments, whereby the recommendation engine is disabled and the transformation executed in each iteration is picked randomly. The final WNS and the leakage power for 50 random experiments on the *leon3pm_iccad* design are presented in Figure 8, when the designs are optimized for the best clock period of Fig. 7. The results are sorted by the achieved WNS.

The 4 trials shown on the right-hand side of Fig. 8 managed to close timing, but with worse leakage power than the proposed approach, while the other 46 trials achieve a wide range of worse timing and leakage performance. It should be noted that, even if the applied transformations are selected randomly in each iteration, their result is kept only if it improves the design according to the proposed productivity rule. Therefore, in the end, only the results of useful transformations are kept. This indicates that, even if the proposed netlist transformations provided to the LR-based optimizer are effective, they cannot reach the QoR achieved when these methods are appropriately rewarded and orchestrated, as done by the proposed recommendation system.

C. The interaction of netlist transformation with the recommendation engine

In this sub-section, our goal is to demonstrate the behavior of the recommendation engine, and which transformations it decided to prioritize in order to achieve the overall results of Table II. Fig. 9 depicts the percentage that each transformation is utilized per benchmark during the LR-optimization loop, as decided independently, by the MAB-based recommendation engine. In all cases, the lion's share belongs to useful clock skew and the two versions of cell sizing, since they are the strongest methods for both TNS and power reduction. Because of the runtime scaling factor, the optimization method picks fast gate sizing as often as full gate sizing.

Hold buffering usually has the lowest pick rate, since the designs do not originally have early violations, so it can only be useful around the end of the flow. Even then, since we do not allow for early TNS to degrade too much, early buffering cannot receive a high enough reward.

Late buffering is picked infrequently for the opposite reason. It is very useful in the first iterations to buffer large fanout nets, but its benefit diminishes later on. This behavior is highlighted in Fig. 10, which depicts the normalized payoff, calculated on equation (5) of early and late buffering over time for the design usb_funct. The payoff of late buffering is high at the start of the flow and gradually decreases, while the payoff of early buffering increases over time, but still stays low. Recall that the payoff demonstrates how high or low a method is ranked compared to the other methods.

Utilizing more elaborate methods for buffering nets with

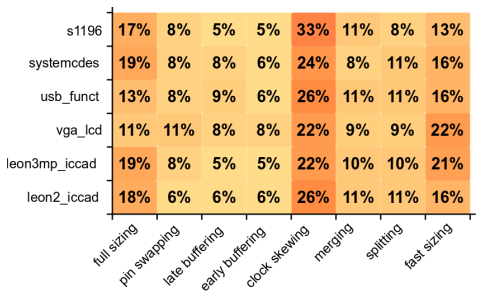


Fig. 9. How often each transformation is selected per benchmark.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3025541, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE IV
THE IMPACT OF OMITTING A TRANSFORMATION ON THE RESULTS FOR USB_FUNCT. IN ALL CASES LISTED BELOW THE "FULL FLOW" ROW IN THE TABLE, HOLD TIMING WAS CLOSED, AS ENSURED BY THE POST-PASS OF INSERTING BUFFERS TO FIX ANY REMAINING HOLD VIOLATIONS.

Netlist		TNS (ns)	WNS (ns)	Leakage (μW)	Area (μm^2)	Runtime (min)
Initial		-392.697	-1.104	392	17561	-
Initial downsizing		-736.635	-1.678	394	17597	-
Full flow		0	0	404	18412	1.51
Opt removed	Full sizing	-0.010	-0.010	408	18549	1.67
	Pin Swap	-0.052	-0.052	413	18352	1.36
	Late Buff.	-0.036	-0.036	399	18466	2.24
	Hold Buff.	-0.027	-0.027	398	18322	1.54
	Clock Skew	-1.375	-0.075	426	18908	9.76
	Gate merge	0	0	413	18671	1.59
	Gate split	-0.138	-0.072	395	18234	1.38
	Fast sizing	-0.027	-0.027	407	18507	1.28
	Sizing	-0.030	-0.030	455	19490	4.19

timing violations than the local buffering approaches used in this work could possibly increase the pick rate of buffering transformations. It is part of our future plans to adapt efficient dynamic buffering techniques to use LM-based cost functions and include them in our set of available transformations.

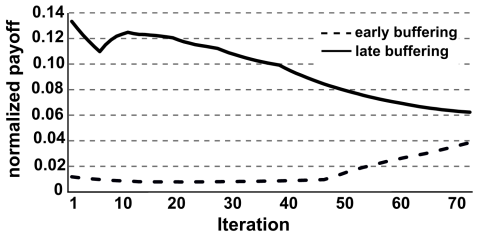


Fig. 10. The normalized payoff of early and late buffering over time for usb_funct.

Gate merging receives higher rewards in the power reduction phase, since it reduces the total number of cells, while gate splitting is more effective during the timing reduction phase. Pin swapping has a low pick rate, since its immediate effect on the timing is usually small.

In order to better quantify the contribution of each method to the final QoR, we re-ran our flow by excluding an optimization method on each run. Table IV presents the results for the indicative design usb_funct for the best achieved period. It can be observed that clock skew scheduling is the most efficient method for reducing timing violations, and cell sizing for reducing leakage power and area. Nevertheless, removing any method will worsen the QoR or the runtime, as the optimization framework will need to achieve its effect by running other methods for potentially more iterations, which is an overhead that can become prohibitively expensive for larger designs. Therefore, all methods are required to achieve the best QoR in the fastest runtime.

The number of netlist transformations employed does not alter the tuning of the proposed recommendation system. In fact, appropriate netlist transformations can be added, or removed, in a plug-and-play manner. To demonstrate this, we ran experiments with a reduced number of available netlist transformations. Also, to examine if the recommendation system can still adapt and produce reasonable results, despite the varying number of available optimization methods, *we did not retune any parameter for those runs.*

TABLE V
AUTONOMOUS OPTIMIZATION USING IN EACH CASE A REDUCED NUMBER OF HEURISTICS ON THE USB_FUNCT DESIGN.

# Opt. used	Opt. available	TNS (ns)	WNS (ns)	Leakage (μW)	Area (μm^2)	Runtime (min)
2	Clock Skew Fast sizing	-0.601	-0.222	390	18118	1.09
4	Previous + Full sizing Gate merge	-0.554	-0.153	393	18187	1.75
6	Previous + Gate split Pin Swap	-0.042	-0.037	402	18378	1.11
8	Previous + Late Buff. Early Buff.	0	0	404	18412	1.51

The results obtained for the same indicative *usb_funct* design are presented in Table V. In the first scenario, we use fast cell sizing and clock skew assignment as the only available transformations. Both represent the two most utilized methods according to Fig. 9. The following three scenarios each add two more transformations, following the order of their average utilization. In every case, the obtained timing, power, area, and runtime results are reasonable. The quality improves as more netlist transformations are added, since the proposed recommendation system can autonomously adapt and combine the benefits of a wider range of methods, without requiring any manual calibration.

D. Results on the benchmarks of the ISPD 2013 contest

The presented flow was also applied on the ISPD 2013 gate sizing contest benchmarks. In this case, each cell in the library has ten sizes available at three threshold voltages, with a total of 30 sizes per cell. There is only one setup timing corner, and registers are actually non-sizeable, since there is only one version of flip-flop cells available. For each benchmark, two clock period targets are given: a fast and a slow target. Each design has a SPEF file describing a set of RC parasitics for each net. The ISPD library file does not include buffers, so they are replaced by inverters. Every buffering heuristic ensures that the number of added inverters is even.

The results obtained after running the proposed optimization on the benchmarks of the ISPD 2013 contest are depicted in Table VI. Since timing constraints were met in all cases, only leakage power is reported together with the measured runtime. Table VI also includes the results achieved by two state-of-the-art gate sizing methods [16], [17], which yield the best results in this benchmark set.

The proposed optimization that includes all eight available netlist transformations achieves superior results in most cases, achieving 25% better leakage power, on average, than the most efficient previous work [16].

The runtime evaluation results in Table VI indicate that the proposed approach is comparable to the state-of-the-

TABLE VI
RESULTS FOR THE ISPD 2013 CONTEST BENCHMARKS.

Design	#cells	Leakage (mW)			Runtime (min)		
		Ours	[16]	[17]	Ours	[16]	[17]
usb_phy_slow	623	1	1	1	0.03	0.49	0.22
usb_phy_fast		1	2	2	0.06	0.42	0.23
pci_bridge32_slow	30763	53	57	58	1.93	10.53	0.97
pci_bridge32_fast		60	85	90	1.70	22.62	1.54
fft_slow	33792	84	87	88	1.61	25.71	1.37
fft_fast		131	194	213	3.30	40.43	1.64
cordic_slow	42937	220	267	293	4.24	69.04	2.29
cordic_fast		785	980	1080	4.38	117.08	5.66
des_perf_slow	113346	340	327	332	14.38	132.27	7.27
des_perf_fast		663	644	639	21.19	347.87	26.16
edit_dist_slow	129227	428	416	440	5.74	123.90	4.92
edit_dist_fast		532	535	549	10.44	352.96	6.66
matrix_mult_slow	159642	451	443	448	7.62	226.13	8.80
matrix_mult_fast		721	1541	1633	21.56	395.96	13.94
netcard_slow	984094	3663	5155	5170	58.36	483.55	24.67
netcard_fast		3772	5182	5205	64.52	400.89	30.60

art. Note that the reported runtimes for the gate-sizing-only techniques are taken verbatim from their respective papers. Consequently, those runtimes correspond to other machines with different specifications than the one we used. Therefore, the comparisons can only be broadly and generally indicative. Regardless, we include those runtime numbers here in a big-picture context, to demonstrate that the runtimes of the proposed approach are reasonable, as compared to the others.

Finally, it should be noted that, since our cell sizing algorithm is effectively an enhanced version of the LR-based sizers used in [16] and [17], our optimization can approach the results reported in said papers using gate-sizing-only transformations. The extra savings reported, which are significant for the larger benchmarks, are the outcome of the autonomous orchestration of all available netlist transformations. More specifically, like in the TAU 2019 benchmarks, cell sizing and clock skew scheduling are the most efficient methods for reducing leakage power and timing violations respectively, with the rest of the methods having smaller but considerably positive impact on the final result.

VII. CONCLUSIONS

This work investigates adaptive design optimization techniques, where a modified version of Multi-Armed Bandits is employed to autonomously optimize unknown designs. In each iteration of an LR-based global optimizer, the MAB-based orchestrator chooses a particular method to apply among several netlist transformations. The introduced recommendation engine can start either blindly, or with an initial estimate of rewards from profiling the optimization transforms across a subset of smaller, representative benchmarks. It adapts on-the-fly across iterations, based on analysis of which optimization approaches have been most successful thus far on this design. This dynamic adaptation is performed while balancing the exploration of alternative optimizations versus the reward, and by avoiding the need for any manual tuning. The proposed approach achieves superior results on the TAU 2019 and ISPD 2013 benchmarks, as compared to current state-of-the-art.

We have provided a new generalized paradigm for a digital circuit optimization engine that combines the following: (a) state-of-the-art Lagrangian relaxation that identifies how best to weigh (with Lagrange multipliers) the delay versus power tradeoff at each cell to achieve lower power and timing closure; (b) plug-and-play capability for different optimization transforms that are typically used in an industrial EDA flow; and (c) a system to recommend which type of optimization may give more benefit at this point during optimization. The heuristics optimize the design based on the local LR cost function, harmoniously operating under the guidance of the proposed recommendation engine to achieve timing convergence and lower power. Any optimization method that can be adapted to use the local LR cost function can be added.

REFERENCES

[1] N. D. MacDonald, "Timing Closure in Deep Submicron Designs," in *Design Automation Conference (DAC)*, 2010.

[2] L. Lavagno, I. Markov, G. Martin, and L. Scheffer, *Electronic Design Automation for IC Implementation, Circuit Design, and Process Technology*. Taylor and Francis group, 2016.

[3] T. Lattimore and C. Szepesvari, *Bandit Algorithms*, 2018. [Online]. Available: http://banditalgs.com

[4] G. Chen, T.-W. Huang, and J. Shah, "Design Optimization Contest," in *ACM Intern. Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, 2019. [Online]. Available: https://sites.google.com/view/tau-contest-2019/home

[5] M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke, and C. Zhuo, "An Improved Benchmark Suite for the ISPD-2013 Discrete Cell Sizing Contest," in *Int. Symp. on Physical Design*, 2013, pp. 168–170.

[6] J. P. Fishburn and A. E. Dunlop, "TILOS: A Posynomial Programming Approach to Transistor Sizing," in *Int. Conf. CAD*, 1985, pp. 326–328.

[7] M. Ketkar, K. Kasamsetty, and S. Sapatnekar, "Convex Delay Models for Transistor Sizing," in *Design Automation Conf.*, 2000, pp. 655–660.

[8] O. Coudert, "Gate Sizing for Constrained Delay/Power/Area Optimization," *IEEE Trans. on VLSI Systems*, vol. 5, no. 4, pp. 465–472, 1997.

[9] D. G. Chinnery and K. Keutzer, "Linear Programming for Sizing, Vth and Vdd Assignment," in *Int. Symp. Low Power Electronics and Design*, 2005, pp. 149–154.

[10] D. Nguyen, A. Davare, M. Orshansky, D. Chinnery, B. Thompson, and K. Keutzer, "Minimization of Dynamic and Static Power Through Joint Assignment of Threshold Voltages and Sizing Optimization," in *Int. Symp. Low Power Electronics and Design*, 2003, pp. 158–163.

[11] S. Held, "Gate Sizing for Large Cell-based Designs," in *Design, Automation and Test in Europe*, 2009, pp. 827–832.

[12] H. Fatemi, A. Kahng, H. Lee, J. Li, and J. P. de Gyvez, "Enhancing Sensitivity-based Power Reduction for an Industry IC Design Context," *Integration*, vol. 66, pp. 96–111, 2019.

[13] C.-P. Chen, C. C. N. Chu, and D. F. Wong, "Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation," *IEEE Trans. on CAD*, vol. 18, no. 7, pp. 1014–1025, 1999.

[14] S. Hu, M. Ketkar, and J. Hu, "Gate Sizing for Cell Library-Based Designs," in *Design Automation Conference (DAC)*, 2007, pp. 847–852.

[15] M. Ozdal, S. Burns, and J. Hu, "Algorithms for Gate Sizing and Device Parameter Selection for High-Performance Designs," *IEEE Trans. on CAD*, vol. 31, no. 10, pp. 1558–1571, 2012.

[16] G. Flach, T. Reimann, G. Posser, M. Johann, and R. Reis, "Effective Method for Simultaneous Gate Sizing and *V*th Assignment Using Lagrangian Relaxation," *IEEE Trans. on CAD*, vol. 33, no. 4, pp. 546–557, 2014.

[17] A. Sharma, D. Chinnery, T. Reimann, S. Bhardwaj, and C. Chu, "Fast Lagrangian Relaxation Based Multi-Threaded Gate Sizing Using Simple Timing Calibrations," *IEEE Trans. on CAD*, vol. 39, no. 7, pp. 1456–1469, 2019.

[18] S. Roy, D. Liu, J. Singh, J. Um, and D. Z. Pan, "OSFA: A New Paradigm of Aging Aware Gate-Sizing for Power/Performance Optimizations Under Multiple Operating Conditions," *IEEE Trans. on CAD*, vol. 35, pp. 1618–1629, 2016.

[19] S. Daboul, N. Hhnle, S. Held, and U. Schorr, "Provably Fast and Near-Optimum Gate Sizing," *IEEE Trans. on CAD*, vol. 37, no. 12, pp. 3163–3176, 2018.

[20] G. Shklover and B. Emanuel, "Simultaneous Clock and Data Gate Sizing Algorithm with Common Global Objective," in *Int. Symp. Physical Design*, 2012, pp. 145–152.

[21] A. Sharma, D. Chinnery, and C. Chu, "Lagrangian Relaxation Based Gate Sizing With Clock Skew Scheduling - A Fast and Effective Approach," in *Int. Symp. on Physical Design*, 2019, pp. 129–137.

[22] D. Mangiras, A. Stefanidis, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos, "Timing-Driven Placement Optimization Facilitated by Timing-Compatibility Flip-Flop Clustering," *to appear in IEEE Trans. on CAD*, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8840912

[23] S. Huang, G. Jhuo, and W. Huang, "Minimum Buffer Insertions for Clock Period Minimization," in *Int. Symp. Computer, Communication, Control and Automation*, 2010, pp. 426–429.

[24] W. Tu, C. Chou, S. Huang, S. Chang, Y. Nieh, and C. Chou, "Low-power Timing Closure Methodology for Ultra-low Voltage Designs," in *Int. Conf. CAD*, 2013, pp. 697–704.

[25] P. Wu, M. D. F. Wong, I. Nedelchev, S. Bhardwaj, and V. Parkhe, "On Timing Closure: Buffer Insertion for Hold-violation Removal," in *Design Automation Conference (DAC)*, 2014, pp. 1–6.

[26] I. Han, D. Hyun, and Y. Shin, "Buffer Insertion to Remove Hold Violations at Multiple Process Corners," in *Asia and South Pacific Design Automation Conf.*, 2016, pp. 232–237.

[27] L. P. P. P. van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay," in *Int. Symp. Circuits and Systems*, vol. 2, 1990, pp. 865–868.

[28] J. Lillis, C.-K. Cheng, and T. Y. Lin, "Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model," *IEEE J. of Solid-State Circuits*, vol. 31, no. 3, pp. 437–447, 1996.

[29] X. Wang, W. Liu, and M. Yu, "A Distinctive o(mn) Time Algorithm for Optimal Buffer Insertions," in *Int. Symp. Quality Electronic Design*, 2015, pp. 293–297.

[30] Y. Jiang, S. S. Sapatnekar, C. Bamji, and J. Kim, "Interleaving Buffer Insertion and Transistor Sizing into a Single Optimization," *IEEE Trans. VLSI Systems*, vol. 6, no. 4, pp. 625–633, 1998.

[31] I.-M. Liu, A. Aziz, D. F. Wong, and H. Zhou, "An Efficient Buffer Insertion Algorithm for Large Networks Based on Lagrangian Relaxation," in *Int. Conf. Computer Design*, 1999, pp. 210–215.

[32] Y.-J. Ho and W.-K. Mak, "Power and Density-Aware Buffer Insertion," in *Int. Symp. On VLSI Design, Automation and Test*, 2008, pp. 287–290.

[33] S. Hu, Z. Li, and C. J. Alpert, "A Fully Polynomial Time Approximation Scheme for Timing Driven Minimum Cost Buffer Insertion," in *Design Automation Conf.*, 2009, pp. 424–429.

[34] A. Stefanidis, D. Mangiras, C. Nicopoulos, D. Chinnery, and G. Dimitrakopoulos, "Design Optimization by Fine-Grained Interleaving of Local Netlist Transformations in Lagrangian Relaxation," in *Proceedings of the International Symposium on Physical Design*, 2020, pp. 87–94.

[35] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Mach. Learn.*, vol. 47, no. 2-3, pp. 235–256, 2002.

[36] A. Stefanidis, D. Mangiras, C. Nicopoulos, and G. Dimitrakopoulos, "Multi-Armed Bandits for Autonomous Timing-driven Design Optimization," in *International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2019, pp. 17–22.

[37] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe, "Opentuner: An extensible framework for program autotuning," in *Intern. Conf. on Parallel Architectures and Compilation*, 2014, pp. 303–316.

[38] Z. Wang and M. OBoyle, "Machine Learning in Compiler Optimization," *Proceedings of the IEEE*, vol. 106, no. 11, pp. 1879–1901, 2018.

[39] C. Xu, G. Liu, R. Zhao, S. Yang, G. Luo, and Z. Zhang, "A Parallel Bandit-Based Approach for Autotuning FPGA Compilation," in *Int. Symp. on Field-Programmable Gate Arrays*, 2017, pp. 157–166.

[40] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, "DRiLLS: Deep Reinforcement Learning for Logic Synthesis," 2019. [Online]. Available: https://arxiv.org/pdf/1911.04021.pdf

[41] S. Sadasivam, Z. Chen, J. Lee, and R. Jain, "Efficient Reinforcement Learning for Automating Human Decision-making in SoC Design," in *Design Automation Conference (DAC)*, 2018.

[42] T. J. Reimann, C. N. C. Sze, and R. Reis, "Cell Selection for High-Performance Designs in an Industrial Design Flow," in *International Symposium on Physical Design*, 2016, pp. 65–72.

[43] L. Li, P. Kang, Y. Lu, and H. Zhou, "An Efficient Algorithm for Library-based Cell-type Selection in High-performance," in *Int. Conf. CAD*, 2012, pp. 226–232.

[44] H. Tennakoon and C. Sechen, "Nonconvex Gate Delay Modeling and Delay Optimization," *IEEE Trans. on CAD*, vol. 27, pp. 1583–1594, 2008.

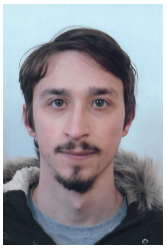
[45] A. Sharma, D. Chinnery, S. Dhamdhere, and C. Chu, "Rapid Gate Sizing with Fewer Iterations of Lagrangian Relaxation," in *Int. Conf. CAD*, 2017, pp. 337–343.

[46] J. P. Fishburn, "Clock Skew Optimization," *IEEE Trans. on Computers*, vol. 39, no. 7, pp. 945–951, 1990.

[47] G. Flach, M. Fogaça, J. Monteiro, M. Johann, and R. Reis, "Rsyn: An Extensible Physical Synthesis Framework," in *Int. Symp. On Physical Design*, 2017, pp. 33–40.

[48] T. W. Huang and M. D. F. Wong, "OpenTimer: A High-performance Timing Analysis Tool," in *Int. Conf. CAD*, 2015, pp. 895–902.

[49] H.-H. Cheng, T.-W. Lin, Y.-C. Lin, I. H.-R. Jiang, and P.-Y. Lee, "Team iTimer, TAU workshop 2019," 2019. [Online]. Available: https://sites.google.com/view/tau-contest-2019/home



Apostolos Stefanidis received the Diploma in electrical and computer engineering from the Democritus University of Thrace, Xanthi, Greece, in 2017, where he is currently pursuing his Ph.D. degree.

His current research interests include electronic design automation, with emphasis in machine learning applications on autonomous timing and power optimization.

Dimitrios Mangiras received the Diploma in electrical and computer engineering from the Democritus University of Thrace, Xanthi, Greece, in 2017, where he is currently pursuing the Ph.D. degree.

His research interests include electronic design automation for physical design, clock tree synthesis and machine-learning based optimization as well as, design of energy-efficient integrated circuits and automated verification methodologies.

Chrysostomos Nicopoulos received the B.S. and Ph.D. degrees in electrical engineering with a specialization in computer engineering from Pennsylvania State University, State College, PA, USA, in 2003 and 2007, respectively.

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus. His current research interests include networks-on-chip, computer architecture, multi/many-core microprocessor and computer system design.

David Chinnery received a Ph.D. in Electrical Engineering from the University of California at Berkeley in 2006. He is the author of two books on Closing the Gap Between ASIC and Custom with tools and techniques for high-performance and low-power design. He is an author of two chapters in the EDA for Integrated Circuits Handbook, and various conference papers.

David has worked for the past 8 years at Mentor Graphics, where he is the R&D manager for the Optimization group of the Nitro place-and-route tool. Previously, David worked for 5 years in the CAD group at Advanced Micro Devices supporting both custom and synthesized microprocessor designs.

Giorgos Dimitrakopoulos received the B.S, MSc and Ph.D. degrees in Computer Engineering from University of Patras, Patras, Greece, in 2001, 2003 and 2007, respectively.

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece. He is interested in the design of digital integrated circuits, electronic design automation, and computer architecture, with emphasis in low-power systems design.