Soft-Clustering Driven Flip-flop Placement Targeting Clock-induced OCV

Dimitrios Mangiras[†], Pavlos Mattheakis[‡], Pierre-Olivier Ribet[‡], Giorgos Dimitrakopoulos[†]
[†]Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece
[‡]Mentor, a Siemens Business, Grenoble, France

ABSTRACT

On-Chip Variation (OCV) in advanced technology nodes introduces delay uncertainties that may cause timing violations. This problem drastically affects the clock tree that, besides the growing design complexity, needs to be appropriately synthesized to tackle the increased variability effects. To reduce the magnitude of the clock-induced OCV, we incrementally relocate the flip-flops and the clock gaters in a bottom-up manner to implicitly guide the clock tree synthesis engine to produce clock trees with increased common clock tree paths. The relocation of the clock elements is performed using a soft clustering approach that is orthogonal to the clock tree synthesis method used. The clock elements are repeatedly relocated and incrementally re-clustered, thus gradually forming better clusters and settling to more appropriate positions to increase the common paths of the clock tree. This behavior is verified by applying the proposed method in industrial designs, resulting in clock trees which are more resilient to process variations, while exhibiting improved overall timing.

CCS CONCEPTS

• **Hardware** → **Electronic design automation**; **Physical design (EDA)**; **Placement**; **Physical synthesis**; **Clock-network synthesis**.

KEYWORDS

on-chip variations; soft clustering; flip-flop placement; clock tree synthesis

ACM Reference Format:
Dimitrios Mangiras, Pavlos Mattheakis, Pierre-Olivier Ribet, Giorgos Dimitrakopoulos. 2020. Soft-Clustering Driven Flip-flop Placement Targeting Clock-induced OCV. In *Proceedings of the 2020 International Symposium on Physical Design (ISPD '20), March 29-April 1, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3372780.3375564

1 INTRODUCTION

The On-Chip Variation (OCV) effect refers to the intrinsic variability involved in semiconductor manufacturing processes and the fluctuation of operating conditions, such as voltage and temperature, and how they impact a circuit's timing [7].

Due to OCV, some cells may be faster or slower than expected, thus introducing delay uncertainties in data and clock path delays, resulting in more stringent timing constraints. In order to model them, timing derates are introduced that are multiplied with the net and cell delays [2]. For example, in the case of the clock tree, a launch clock path can be slower than expected, while a capture clock path can be faster than expected. In this case, if there is no sufficient positive slack, the increase of clock skew uncertainties may cause a violation of the late (setup) timing constraints. Nevertheless, this opposite derating of the launch and the capture clock paths cannot occur on the part of the clock tree that is common to both paths. The common path should not be derated since the clock can be either slower or faster for both the launch and capture paths. Common path pessimism removal discards this artificial pessimism during timing analysis [2].

Previous research tries to alleviate the impact of OCV either during Clock Tree Synthesis (CTS), or by optimizing already synthesized clock trees. Optimizing the quality of the top-level clock tree by reducing clock divergence and optimizing placement of clock logic and buffers was the goal in [3] and [21]. In [18], a statistical centering based clock routing method is proposed that makes the clock skew more tolerant to interconnect variations. The work of [26] reconstructs the topology of a synthesized clock tree by reconnecting buffers for removing OCV timing violations, while improving the lower bounds on the Worst Negative Slack (WNS) and the Total Negative Slack (TNS).

The work in [23] improves timing in multiple modes and multiple corners by applying useful clock skew on an already constructed clock tree. A formulation based on linear programming similar to [17, 22] computes optimal positive or negative clock skew offsets that are applied on the clock tree using buffer insertion, removal, or relocation. Similarly, the work of [12] minimizes the sum of skew variations over all adjacent sink pairs using both global and local optimization that includes solving a linear program and utilizing machine learning to predict the impact of local moves on clock latency. Improving the correlation between the predicted clock skew offsets for tackling OCV and the achieved timing quality after CTS was the focus of [8].

Non-tree clock structures have also been tested as a means for reducing clock-induced OCV. Clock meshes [10] and clock trees with cross links [9, 20] represent the most relevant approaches. Non-tree structures reduce clock skew and improve the robustness of clock networks compared to tree-shaped clock networks, but incur an additional non-trivial cost.

In this work, we try to bring appropriately selected flip-flop and clock gaters closer, while respecting both their initial spatial locality and the functional clock-gating hierarchy, in order to create a better seed for CTS to produce clock trees with less path divergence that are inherently less sensitive to clock-induced OCV.

ISPD '20, March 29-April 1, 2020, Taipei, Taiwan
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7091-2/20/03...$15.00
https://doi.org/10.1145/3372780.3375564

Once global placement and in-place datapath optimization have finished, flip-flops and clock gaters are first clustered in a bottom-up fashion, using soft clustering [11], and then moved iteratively closer to the weighted mean location of the center of all neighbor clusters. *Flip-flop-to-cluster membership is not a hard decision and it is not fixed at any stage of the algorithm.* The membership of flip-flops to clusters is quantified by weights that model the physical proximity and timing adjacency of the examined clock cells as well as their timing criticality. Membership weights are not constant but they are dynamically updated as flip-flop/clock-gater relocation evolves. It should be noted that, in this work, clustering is only used for guiding clock cell relocation, and it does not lead to circuit restructuring, as done in clustering-driven CTS engines [6, 25], or in other approaches that use clustering for register clumping [5, 13, 19, 27] or multibit register composition [15, 16, 24].

The presented approach has been integrated into Mentor's Nitro SoC^{TM} place-and-route flow and tested using six industrial designs implemented at various technologies between 14 and 28nm. The results after post-CTS optimizations, show that the proposed pre-CTS clock cell relocation effectively guides CTS to produce clock trees with increased common paths. As a result, timing metrics are improved in all cases, without degrading overall clock tree complexity. In overall, the main contributions of this work are summarized as follows:

- The utilization of soft clustering, for the first time, for guiding clock cell relocation across clock hierarchy.
- The novel definition of membership grades that quantifies effectively physical and timing adjacency.
- The evaluation of the proposed approach under a realistic industrial setup.

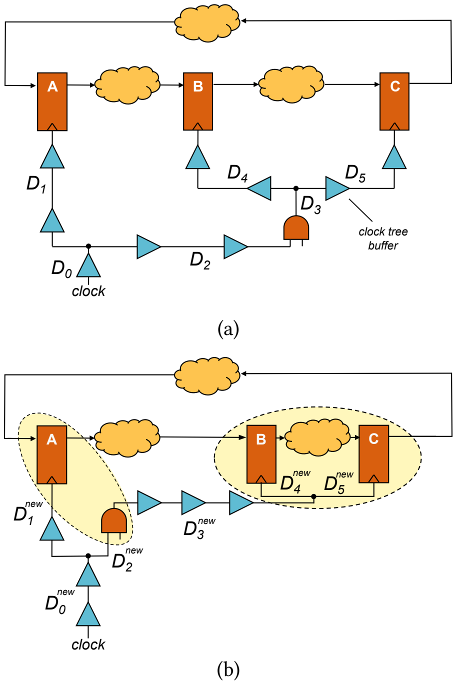
The proposed method *is applied before CTS* and *is inherently orthogonal* to any previous work that optimizes directly the clock tree topology for reducing clock-induced OCV. Even if clock cell relocation is allowed in modern CTS or post-CTS optimization flows, still the distances that the cells can travel are significantly restricted compared to the pre-CTS stage of the implementation flow that is the focus of this work.

The rest of the paper is organized as follows: Section 2 presents the motivation behind the hierarchical clustering-based flip-flop relocation. Section 3 describes the overall flip-flop relocation algorithm, while Section 4 presents the experimental results. Conclusions are drawn in Section 5.

2 MOTIVATION–PROBLEM FORMULATION

In this paper, we focus on changing the clock cell placement to allow the CTS engine to produce clock trees with as many common paths as possible thus reducing the effect of clock-induced OCV timing degradation. One generic approach to guide CTS into placing selected clock cells on the same clock branch is to bring those cells closer in the physical layout. To take advantage of this property, we need to identify (a) which clock cells, when put closer, would alleviate OCV derates, and (b) how to relocate the selected cells. The following example will shed more light on to how we attack the problem of OCV-aware relocation of clock cells.

Let us consider the example shown in Fig. 1(a) that involves three flip-flops A, B, C, and a clock gater that drives flip-flops B and C. The clock skew for each one of the three register-to-register timing paths AB, BC, and CA is the delay difference between the clock launch and the capture path. The delay of any common path between the launch and capture paths is omitted when computing their delay difference. Due to OCV, when considering the worst-case scenario in terms of late constraints, the delay of the launch clock path is increased by a derating factor γ. For the sake of simplicity, we also assume that the delay of the capture clock path is decreased by the same factor. Therefore, the late clock skew of all paths, after the clock divergence point for each path, is the difference between the maximum clock path delay of the launch path and the minimum clock path delay of the capture path.



(a)

(b)

Figure 1: Physical proximity of selected clock cells driven by the same clock nets reduces clock divergence and improves common clock path delay. (a) The original clock tree, and its (b) optimized version to tackle clock-induced OCV.

$$\begin{aligned}
\text{skew}_{AB}^{\text{late}} &= D_1^{\max} - (D_2^{\min} + D_3^{\min} + D_4^{\min}) \\
&= (\overline{D}_1 - \overline{D}_2 - \overline{D}_3 - \overline{D}_4) + \gamma(\overline{D}_1 + \overline{D}_2 + \overline{D}_3 + \overline{D}_4) \\
\text{skew}_{BC}^{\text{late}} &= D_4^{\max} - D_5^{\min} \\
&= (\overline{D}_4 - \overline{D}_5) + \gamma(\overline{D}_4 + \overline{D}_5) \\
\text{skew}_{CA}^{\text{late}} &= (D_2^{\max} + D_3^{\max} + D_5^{\max}) - D_1^{\min} \\
&= (\overline{D}_2 + \overline{D}_3 + \overline{D}_5 - \overline{D}_1) + \gamma(\overline{D}_2 + \overline{D}_3 + \overline{D}_5 + \overline{D}_1)
\end{aligned}$$

The equations are derived assuming that $D_i^{\max} = \overline{D}_i(1 + \gamma)$ and $D_i^{\min} = \overline{D}_i(1 - \gamma)$, where $\overline{D}_i$ represents the mean delay (without OCV) and includes both wire and logic delay.

To reduce the impact of OCV, we need to minimize the sum of clock path delays multiplied by γ. To do so for this example, we start from the lower levels of the clock hierarchy and try to minimize delays $\overline{D}_4$ and $\overline{D}_5$. As shown in Fig. 1(b), this is achieved by moving flip-flops B and C closer relative to Fig. 1(a). This relocation of flip-flops B and C guides the CTS to place the common branch point of B and C in close proximity, effectively diminishing the derating effect on path BC. This movement is justified, since B and C are the endpoints of the same clock net of the same timing path. Of course, moving B and C closer, for improving OCV, should not degrade data-path timing.

Next, we move up the clock hierarchy and focus on minimizing the OCV derates on paths AB and CA. In this case, our goal is to transfer part of the delays $\overline{D}_1$ and $\overline{D}_2$ to the common path D_0. To achieve this, we need to bring the clock gater and flip-flop A closer, as shown in Fig. 1(b). Flip-flop A and the clock gater are driven by the same clock net and participate in the same timing paths. Therefore, bringing them closer increases the probability that CTS will drive them with a common clock path.

On the contrary, moving A closer to B, or C, would not improve clock-induced OCV and their relative placement should be decided by other criteria. These flip-flops belong to different clock sub-nets and their last common clock point is above the clock gater. The gater inevitably acts as a divergence point for the clock net, separating the clock branch that drives B and C from the clock branch that drives flip-flop A, irrespective of the physical proximity of A to B and C. The same arguments hold for all flip-flops that belong to different clock nets. Such flip-flops can be considered irrelevant for the interaction of their placement with OCV, since their clock divergence point belongs to an upper level of the clock tree.

In any case, moving clock cells closer should not tradeoff an increase in clock latency. Latency increase is avoided if cells don't move far away from the cells they drive in the clock tree hierarchy. For instance, in Fig. 1(b) the clock gater should approach A but at the same time, it should not move too far away from flops B and C. This the reason for not drawing the gater next to A in this example.

This example highlights that, by creating physical clusters of selected clock cells (*timing neighbors*), after examining the cells' launch-capture connectivity and their position on the clock tree hierarchy, we increase the probability that those cells are put on the same clock branch during CTS.

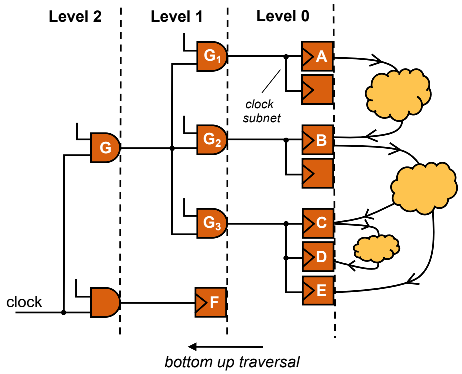


Figure 2: An example to explain the definition of timing neighbor clock cells, depending on their connectivity and positions in the functional clock tree hierarchy.

Two clock cells are timing neighbors if they have clock pins on the same net that belong to the launch and capture clock parts of a constrained timing path. For example, in Fig. 2, flip-flop C is a timing neighbor of flip-flop D and not of flip-flop B. Even if flip-flop C is connected to both flip-flops B and D, it neighbors only with flip-flop D since flip-flop B has its clock pin on different net from C. Also, clock gater G_1 is a timing neighbor cell for gater G_2, since they have their clock pins on the same net (the net driven by gater G) and there is a constrained timing path that connects them through their children ($G_1 \longrightarrow A \longrightarrow B \longrightarrow G_2$). However, the same is not true for G_1 and G_3. Although they are endpoints of the same clock net, there is no timing path that connects their leaves.

Algorithm 1: OCV-aware clock Cell Relocation

```
1  foreach level k of the clock tree - bottom up do
2      foreach clock net n of level k do
3          Cells[n] ← all cells at the endpoints of n;
4          Clusters[n] ← InitClusters(Cells[n]);
5          repeat // Cluster and Relocate
6              repeat // Soft Clustering
7                  Compute $m(i, j)$ $\forall i \in$ Cells[n] and
                    $j \in$ Clusters[n] using eq. (2);
8                  Update the centers of all Clusters[n] using
                    eq. (4);
9              until convergence;
               // Cell relocation
10             foreach cell $i \in$ Cells[n] do
11                 if i not timing critical && not reached
                    displacement limit then
12                     Move i closer to the weighted mean
                        location of the centers of Clusters[n];
13                 end
14             end
15             UpdateTiming();
16         until no cell moved;
17     end
18 end
```

3 SOFT CLUSTERING-BASED PLACEMENT

The proposed approach is based on a repetitive process of incremental clustering and clock cell relocation with the goal to bring timing neighbors closer and increase the probability that they are driven by the same clock branch after CTS.

Algorithm 1 depicts the overall structure of the proposed method. The clock cells are examined hierarchically beginning from the leaves of each clock net. The clock cells at the sinks of each clock net are clustered using the k-Harmonic Means (kHM) soft clustering algorithm [28]. kHM begins with an initial guess of the solution (line 4 of Alg. 1), and then refines the position of the centers until it reaches convergence, i.e., the positions of the cluster centers change by less than 1% per iteration (lines 6–9 of Alg. 1).

In contrast to hard clustering algorithms [14, 27], kHM is a soft clustering algorithm and allows the cells to belong to more than one cluster [11]. Function $m(s_i, c_j)$, with $0 \leq m(s_i, c_j) \leq 1$ and $\sum_j^k m(s_i, c_j) = 1$, defines the grade of membership of clock cell s_i to the jth cluster with center c_j. This membership function effectively combines the physical location of the cells with the location of their timing neighbors.

Once the cell-to-cluster memberships have been computed, each examined cell tries to approach the center of the clusters according to the computed membership grades (lines 10–14 of Alg. 1). The soft membership nature of the proposed clustering algorithm allows *all nearby clusters* to contribute to the movement of each clock cell. This feature would have been impossible with clustering algorithms that employ hard membership functions.

Once all candidate cells have tried one new position, routing and timing are incrementally updated to reflect the available slacks at the inputs and the outputs of the affected cells (line 15 of Alg. 1).

3.1 Initialize cluster centers

The kHM soft clustering algorithm is executed independently on each clock net n, assuming a predetermined number of clusters K. The number of clusters is computed based on a maximum-allowed cluster size that tries to mimic the maximum-fanout constraint imposed on the clock tree.

During initialization, we partition the cells driven by each clock net in equally-sized groups of clock elements (flip-flops and clock gaters), and select randomly the position of one cell from each group as the initial cluster center. As it will be shown in Algorithm 2, the initial cluster centers are derived after taking into account the cells of each net as well as the cluster centers already defined for the hierarchically lower clock nets. This addition is needed to avoid clock cells being placed far away from the cells they drive, which could increase clock wirelength and latency. The upper levels of the clock-tree hierarchy are sparse and involve in most cases a few clock gaters placed far apart from each other. Therefore, bringing those cells closer, as dictated by the proposed method, would risk to separate them from the cells they drive. This risk does not appear on the lower levels of the clock tree where bringing cells closer involves only local moves.

The recursive partitioning of the cells driven by each clock net is highlighted in Algorithm 2. In particular, we first define the bounding box that encloses all cells of clock net n, i.e., Cells$[n]$. Then, we add to Cells$[n]$ all cluster centers of the lower level of the clock tree that are connected to each cell of Cells$[n]$ and placed inside the bounding box of n (Line 4 of Algorithm 2). In this way, the cluster initialization of clock net n is done on *AllCells* that includes both the cells connected to n and the pre-defined cluster centers of the hierarchically lower clock sub-nets. Then, the set of points that determine the initialization of the clusters of net n are first sorted geographically, and they are recursively partitioned to equally-sized sets using *RecPartition* function described in Algorithm 2.

3.2 Compute membership function

In kHM clustering algorithm [28] the probability of a cell s_i being a member of the jth cluster is determined by the harmonic average of the distances of each cell to the centers of all K clusters and is given by:

$$d(s_i, c_j) = \frac{\left\|s_i - c_j\right\|^{-p-2}}{\sum_{k=1}^{K} \left\|s_i - c_k\right\|^{-p-2}} \tag{1}$$

Parameter p is set to 4, to distinguish more clearly the cells that are located far from the center of the cluster relative to those that are placed in a nearby position. For traditional 2D clustering, the scaled physical distance of a cell from the centers of all clusters given by $d(s_i, c_j)$ would have been a sufficient clustering quality metric [27, 28]. However, *for the OCV-aware placement of sequential cells, this is not enough.* A cluster is a good candidate for cell s_i, if the timing neighbors of s_i, denoted as $\mathcal{N}(s_i)$, also belong to the same cluster, especially the most timing critical ones. In this way, CTS is guided to put them on the same clock branch and effectively reducing clock-induced OCV.

Algorithm 2: Initialize Clusters

```
1  function InitClusters (Cells[n])
       // Set #clusters using cells of subnet n
2      $K \leftarrow$ Cells[n] / MAX_CLUSTER_SIZE;
3      BB← BoundingBox(Cells[n]);
       // Include cluster centers of next level that are inside BB
4      AllCells← Cells[n] ∪ Valid_Centers;
5      step← AllCells / $K$;
6      SortedCells ← Sort AllCells according to their $(x, y)$
         co-ordinates ($x$ first).;
7      return RecPartition(SortedCells, step);
8  endfunction
   // Recursive partitioning to equal size groups
9  function RecPartition (SortedCells, step)
10     if sizeof(SortedCells) ≤ step then
11         return a random point $j \in$ SortedCells;
12     end
       // Split SortedCells in two sets
13     $C_1$ ←SortedCells[1 : step];
14     $C_2$ ←SortedCells[(step+1) : sizeof(SortedCells)];
15     RecPartition($C_1$, step);
16     RecPartition($C_2$, step);
17 endfunction
```

3.2.1 The proposed membership grade. Membership grade $m(s_i, c_j)$ should reflect both the spatial proximity of s_i to the center of the jth cluster c_j, as expressed by $d(s_i, c_j)$, as well as the physical proximity of the neighbors of s_i to the same cluster. Effectively, the closer a timing neighbor of s_i is to cluster j the larger the "pressure" towards cell s_i to group to the same cluster as well.

To express these dependencies, we define the membership grade of cell s_i to the jth cluster as follows:

$$m(s_i, c_j) = a \cdot d(s_i, c_j) + (1 - a) \frac{\sum\limits_{k=1}^{|\mathcal{N}(s_i)|} t(s_k, s_i) d(s_k, c_j)}{\sum\limits_{k=1}^{|\mathcal{N}(s_i)|} t(s_k, s_i)} \tag{2}$$

The distance $d(s_k, c_j)$ of each timing neighbor s_k of s_i contributes relative to its timing criticality $t(s_k, s_i)$ with respect to s_i (see Section 3.2.2). The more critical the timing path that connects s_k and s_i, the stronger the need to bring them closer, expecting that CTS will drive them with a common clock tree path.

For $a = 1$, membership is determined only by the physical distance of each cell to the center of each cluster. This corresponds to traditional flip-flop clumping [27], where flip-flops are clustered together based only on their (x, y) coordinates. On the contrary, $a = 0$ would try to bring closer all timing neighbors ignoring the original locations of the cells. Empirically, picking an intermediate value for $a = 0.35$ offers a balanced clustering that could realistically increase the common clock paths and decrease OCV derating.

3.2.2 Timing criticality of timing neighbors. The timing criticality $t(s_k, s_i)$ expresses how critical s_k is, in terms of timing, with respect to the timing paths launching at s_i. It is computed by mapping the effective slack $eslk(s_k, s_i)$ of all neighbors $s_k \in \mathcal{N}(s_i)$ of s_i to a value in the range [0,1], using the function (3) suggested in [1]. If s_k is the most critical neighbor of $\mathcal{N}(s_i)$, then $t(s_k, s_i) = 1$, while $t(s_k, s_i) \rightarrow 0$ if s_k has much greater slack than the average slack of the rest neighbors.

$$t(s_k, s_i) = e^{b\left(\frac{minES - eslk(s_k, s_i)}{avgES - minES}\right)} \tag{3}$$

Effective slack $eslk(s_k, s_i)$, is the total negative slack at s_k due to paths launching at s_i, or the worst positive slack when no negative timing path exists between s_i and s_k. Terms $minES$ and $avgES$ are the minimum and the average effective slack of all neighbors $\mathcal{N}(s_i)$, and b is a tuning parameter; $b = 2$ was used since it consistently gave better results.

To compute the effective slack $eslk(s_k, s_i)$, we need to consider the following cases:

- If s_k and s_i are both flip-flops, then we consider the timing paths that connect them directly.
  - If s_k is a launch flip-flop for cell s_i, effective slack corresponds to its output-Q pin slack.
  - If s_k is a capture flip-flop for s_i, effective slack is the slack at its input-D pin.
- If s_k and s_i are clock gaters we consider the D/Q pin slacks of the flip-flops placed at the endpoints of their transitive fanout and not just the slack of their enable pins. This is done, because we want s_i to approach the clock gater (s_k) that drives the more critical flip-flops.

For instance, following the clock tree hierarchy shown in Fig. 2, $eslk(G_1, G_2)$ involves the timing path $A \longrightarrow B$ and is equal to the slack of the Q pin of flip-flop A. Similarly, for $eslk(G_2, G_1)$ we should examine again the path $A \longrightarrow B$, but in this case we consider the slack at the input-D pin of flip-flop B. For $eslk(G_3, G_2)$, we examine the paths $B \longrightarrow C$ and $B \longrightarrow E$ and consider the slack on the input-D pin of flip-flops C and E. Path $C \longrightarrow D$ – that is internal to the subnetwork rooted by G_3 – does not contribute to the effective slack of any of its timing neighbors.

3.3 Update cluster center

Once the membership grade of each cell s_i placed at (x_{s_i}, y_{s_i}) to all clusters has been computed, the location of the center of each cluster (x_{c_j}, y_{c_j}) needs to be updated using (4).

$$\begin{aligned} x_{c_j} &= \frac{\sum_{i=1}^{\#cells} m(s_i, c_j) w(s_i) x_{s_i}}{\sum_{i=1}^{\#cells} m(s_i, c_j) w(s_i)} \\ y_{c_j} &= \frac{\sum_{i=1}^{\#cells} m(s_i, c_j) w(s_i) y_{s_i}}{\sum_{i=1}^{\#cells} m(s_i, c_j) w(s_i)} \end{aligned} \tag{4}$$

Computing the position of the cluster center also takes into account the grade of influence w and the grade of membership m of each cell. This is a unique feature of the kHM soft-clustering algorithm, and makes it less sensitive to the initialization of the cluster centers. The grade of influence $w(s_i)$ of cell s_i to the positions of all clusters is given by [28]:

$$w(s_i) = \frac{\sum_{j=1}^{K} \left\|s_i - c_j\right\|^{-p-2}}{\left(\sum_{j=1}^{K} \left\|s_i - c_j\right\|^{-p}\right)^2} \tag{5}$$

By definition, the impact of cells that are not close to any center is increased, while the impact of cells that are close to one or more center is decreased. This principle helps in spreading the centers to cover the positions of all cells.

3.4 Relocate Cells

After soft clustering has converged, each cell moves closer to the centers of the more preferable clusters. With this move it attracts its timing neighbors to prefer the same cluster, and increases the probability of sharing a common clock branch with them after CTS.

Clock cell s_i placed at (x_{s_i}, y_{s_i}) in the current iteration is relocated to $(x_{s_i}^{new}, y_{s_i}^{new})$. The new location should be closer to the clusters preferred by s_i, i.e., its membership grade for them is high. For this reason, s_i is relocated to the weighted mean of the location of the centers of all nearby clusters using (6). The contribution of each cluster center to the new location is proportional to the membership grade of s_i to each cluster.

$$x_{s_i}^{new} = \frac{\sum_{j=1}^{K} m(s_i, c_j) x_{c_j}}{\sum_{j=1}^{K} m(s_i, c_j)}, \quad y_{s_i}^{new} = \frac{\sum_{j=1}^{K} m(s_i, c_j) y_{c_j}}{\sum_{j=1}^{K} m(s_i, c_j)} \tag{6}$$

As long as we completely avoid any hard cell-to-cluster assignment, we allow cell relocation to evolve more smoothly across iterations. If instead we employed hard assignments, the cell would move closer only to the center of its assigned cluster, thus possibly leading to ping-pong movements when cell-to-cluster assignments changed across iterations. Also, since the relocation of each cell is biased by the relocation of its timing neighbors, it means that, after several iterations, a better global solution is reached.

A cell is allowed to move and approach its new location when it has positive slack to spend and has not reached its displacement limit yet. Our goal is to utilize some of the positive slack of certain clock cells to form large common branches in the clock tree in other parts of the design that are more timing critical. In any case, we avoid creating a tradeoff between improving clock-induced OCV and degrading data-path timing. A flip-flop is considered safe to move to improve the common clock branches of its timing neighborhood, as long as the input-D and output-Q pin slacks are both positive. Similarly, a clock gater is safe to move when its input-enable pin slack is positive and there is no critical flip-flop at the subtree rooted on this clock gater.

Fig. 3 illustrates graphically the overall cell relocation process. The new location is the weighted mean of the centers of all nearby clusters and cell movement is bounded by its timing feasible region [4], i.e., the common region formed by the transformation to equivalent distance of the positive slacks of the fanin and fanout nets. Each flip-flop's input and output positive slack define a diamond centered by the fanin and fanout gates of the clock cell, while its half diagonal corresponds to the equivalent distance computed using Elmore delay. In the case of nets with multiple endpoints, each endpoint is considered individually and the intersection of the diamonds of all endpoints is kept.

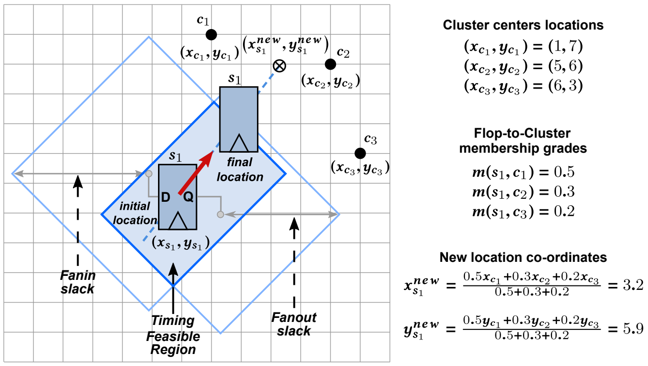

c_1
(x_{c_1}, y_{c_1})
$(x_{s_1}^{new}, y_{s_1}^{new})$
c_2
(x_{c_2}, y_{c_2})
s_1
c_3
(x_{c_3}, y_{c_3})
s_1
final location
D
Q
initial location
(x_{s_1}, y_{s_1})
Fanin slack
Timing Feasible Region
Fanout slack
Cluster centers locations
$(x_{c_1}, y_{c_1}) = (1, 7)$
$(x_{c_2}, y_{c_2}) = (5, 6)$
$(x_{c_3}, y_{c_3}) = (6, 3)$
Flop-to-Cluster membership grades
$m(s_1, c_1) = 0.5$
$m(s_1, c_2) = 0.3$
$m(s_1, c_3) = 0.2$
New location co-ordinates
$x_{s_1}^{new} = \frac{0.5x_{c_1}+0.3x_{c_2}+0.2x_{c_3}}{0.5+0.3+0.2} = 3.2$
$y_{s_1}^{new} = \frac{0.5y_{c_1}+0.3y_{c_2}+0.2y_{c_3}}{0.5+0.3+0.2} = 5.9$

Figure 3: A flip-flop approaching its new location computed as the weighted mean of the three cluster centers. Its relocation is limited by the current timing feasible region.

The cell is legalized instantly to the new suggested position, or to a close-by available position chosen by the legalizer (within 5 rows). Also, when we relocate a cell, the routing congestion and/or row utilization may be degraded due to this movement. To avoid such deteriorations, we do not perform relocations to areas where the routing congestion and/or row utilization is already high, since such movements would affect these metrics even more negatively.

When a cell is relocated it alters its own timing profile as well as the timing profile of all connected cells. Thus, timing and routing needs to be updated to have an accurate view of the timing slacks of each cell. However, performing such incremental updates per cell movement is prohibitive in terms of runtime. As depicted in Algorithm 1, timing and routing are updated once every iteration, after moving many cells. In the meantime, the slacks per cell remain inaccurate. However, respecting the maximum displacement limit and the fact that no cell moves beyond its timing feasible region partially alleviates the problem.

Finally, after each cell relocation two metrics should be updated: its own membership and influence grades relative to every cluster, and the membership grade of every timing neighbor, even if neighbors were not actually moved. After that, the position of the centers of all clusters should be updated, too. Therefore, the re-execution of soft clustering after partial cell relocation needs fewer iterations to converge. At some point, cells and their clusters have been stabilized with no cell being able to move. Please note that once a cell moves closer to the centers of its preferred clusters, it directly impacts the grade of membership of all other connected cells, and all together affect the new position of the centers of all clusters. This orchestrated movement gradually makes the clusters more distinct.

3.5 Algorithm complexity

The proposed cell clustering and relocation is executed independently per clock net in a bottom up manner. Let's assume that each clock net consists of N endpoints (clock cells). In the worst case, the N cells can be split to N/K clusters while each cell can have N timing neighbors. Therefore, letting each one of the N cells of the clock net examine all possible cluster centers and all possible timing neighbors leads to a complexity of $O(N^3)$ per clock net. For C clock nets in the functional clock tree hierarchy, the overall complexity is $O(CN^3)$. In practice, the runtime complexity is lower since the designs consist of many small clock nets (i.e, small N per net, large number of nets C) due to the deep functional clock-gating hierarchy seen in modern designs.

In all cases we consider all clock nets. Thus, we cannot reduce the contribution of C in the runtime complexity. To limit the runtime complexity of the proposed method, we restrict the computation involved per clock net. In our experiments, each cell of a clock net cannot examine more than 50 timing neighbors of the same clock net (the ones placed far away are avoided), while it considers at most 20 nearby cluster centers.

4 EXPERIMENTAL RESULTS

The proposed flip-flop and clock gater placement methodology has been implemented in C++ and integrated in the Nitro-SoC place-and-route tool. It is executed after global placement and data-path optimization. The former provides valid cell locations, whereas the latter fuels the cells with positive slack allowing them to cover bigger distances. Once the proposed algorithm has concluded, cell group placement constraints are generated for all clustered cells. The cell groups act as fences prohibiting the clustered cells to move away from their initial position. The rest of the implementation flow remains unchanged and runs to completion.

To judge the overall effectiveness of the proposed method, in terms of timing, OCV robustness, and clock tree complexity, we compare it with two versions of the reference flow: The first one "Base" represents the industrial quality flow which was originally used to implement the designs. The second one, denoted as "Cluster", activates physical register clustering at the same point in the flow as the proposed method, and implements the algorithm presented in [27]. In particular, this physical clustering [27] utilizes a modified version of k-means algorithm, driven by placement and physical distance criteria, to create physical groups of flip-flops with the goal to simplify the clock tree and reduce clock power. However, we included this technique in our comparisons to highlight that flop grouping techniques that rely on hard flop-to-cluster assignments and use only physical distances for clustering, while ignoring timing criticalities and flip-flop communication, cannot reduce succesfuly the impact of clock-induced OCV. Our implementation of "Cluster" creates groups of tightly placed flip-flops without necessarily forming banks of regularly-placed flip-flops as done in [27]. "Cluster" and the proposed approach cannot move clock cells more than the maximum allowed displacement of 20 rows.

The effectiveness of the proposed method is evaluated on real industrial designs that cover different complexities spanning from 82K up to 1.54M cells and implemented in different technologies between 28 and 14nm. All designs but D2 are constrained with Advanced OCV (AOCV) derates [7]. D2 has simple OCV derates.

4.1 Timing comparisons

The results obtained by the three methods under comparison with respect to timing for setup and hold constraints are shown in Table 1. The first noticeable result is that in all cases, when applying the proposed flip-flop relocation, the worst-negative slack (WNS) and total negative slack (TNS), for both setup and hold analysis, is reduced; an indication that the criticality of certain paths due to OCV is reduced.

Table 1: The timing and row utilization of all designs for the reference implementation flow (Base), the modified flow including the physical register clustering (Cluster) of [27] and the proposed OCV-aware clock cell relocation (New).

Design		Setup		Hold		Util
		WNS (ps)	TNS (ns)	WHS (ps)	THS (ns)	(%)
D1 - 14nm	Base	-337.2	-29.7	0.0	0.0	70.9
82K cells	Cluster	-320.0	-28.8	0.0	0.0	70.5
4.5K regs	New	-297.0	-25.2	0.0	0.0	70.9
D2 - 28nm	Base	-396.0	-885.0	-134.0	-0.6	65.1
199K cells	Cluster	-409.0	-1148.1	-104.0	-7.5	63.6
16K regs	New	-368.0	-768.2	-1.0	-0.1	62.9
D3 - 16nm	Base	-43.0	-0.6	-15.0	-0.6	55.5
542K cells	Cluster	-137.0	-0.9	-17.0	-0.1	55.8
35K regs	New	-24.0	-0.3	-14.0	-0.1	55.7
D4 - 22nm	Base	-232.0	-564.2	0.0	0.0	80.3
557K cells	Cluster	-288.0	-677.0	0.0	0.0	80.8
47K regs	New	-223.0	-392.5	0.0	0.0	80.6
D5 - 16nm	Base	-802.0	-442.9	-35.0	-1.4	56.5
611K cells	Cluster	-668.0	-487.0	-49.0	-0.9	55.6
45K regs	New	-379.0	-100.6	-30.0	-0.6	56.7
D6 - 14nm	Base	-103.0	-41.1	-93.0	-6.0	64.7
1545K cells	Cluster	-68.0	-20.6	-170.0	-20.2	63.8
71K regs	New	-59.0	-16.4	-68.0	-1.9	65.1

Contribution (%) of "New" to the runtime of the full flow

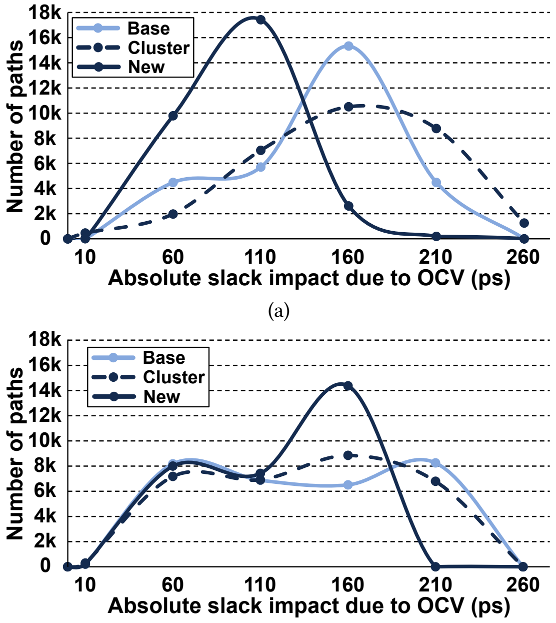
D1	D2	D3	D4	D5	D6
0.83%	0.38%	8.18%	1.23%	6.20%	1.69%

Flip-Flops and clock-gaters have exchanged some of their positive slack to help critical cells reduce the OCV effect on their timing paths. Setup TNS has reduced by 42% on average, while setup WNS is better by 28% on average. Worst Hold Slack (WHS) and Total Hold Slack (THS) have been improved by 45% and 73% respectively. The average savings include the savings of the proposed design relative to both "Base" and "Cluster" method used for comparison. TNS and THS reductions are more distinct since the criticality of the timing neighbors of each cell (see eq. (3)) takes into account the sum of the negative slacks of the related timing paths.

At the rightmost column of Table 1 we report the utilization for each design. The maximum local utilization does not increase since we employ safeguards to avoid relocating cells to areas with already high utilization or routing congestion. The average utilization change is negligible although D2 exhibited a significant utilization improvement with the proposed methodology.

It should be noted that the reported timing violations were collected after the post-CTS optimizations of an industrial quality flow. It is very common in physical design that improvements obtained at specific points in the flow to be partially lost by the optimizations performed later on. However, the proposed cell relocation integrated smoothly with the rest of the flow providing by far the best overall timing Quality-of-Results (QoR).

At the bottom of Table 1, the runtime of the proposed work is reported as the contribution (%) of "New" to the runtime of the full flow. The percentages reported correspond to the *single-threaded* execution of the proposed cell relocation algorithm on a machine with four Intel Xeon CPUs at 2.60 GHz and with 250GB memory. The runtime complexity is heavily dependent of the functional clock tree hierarchy, i.e., on the number of clock nets and the endpoints per clock net. Since the proposed method is executed independently on the cells of each clock net, in our future work, we plan to assign the soft-clustering and cell relocation on each clock net to different threads.



(a)

(b)

Figure 4: The histogram of the impact of clock-induced OCV on late slack on designs (a) D3 and (b) D6.

4.2 Clock-induced OCV redistribution

To observe more clearly how the proposed method guided the CTS engine to produce clock trees with increased common clock tree paths for the communicating sequential elements, we computed the histogram of the impact of clock-induced OCV on late slack on a large set of timing paths.

For the 30K most critical paths of each design, we measured the difference of the path's late slack with and without OCV derates. Then, to produce the required histograms, we split the paths to bins according to the derived slack value. For instance, a bin of 60ps containing a certain number of paths means that those paths are derated in overall by 60ps due to clock-induced OCV relative to the case that OCV is neglected. The histograms of the impact of OCV on late slack for two representative designs and for the three methods under comparison are shown in Fig. 4. Similar results are obtained for other designs.

Both diagrams of Fig. 4 reveal that the proposed method ("New") correctly identified the paths most heavily affected by OCV and restructured them to increase their common clock path. For example, for "New" the majority of the paths in the case of D3 in Fig. 4(a), are affected by 60–110 ps due to OCV. On the contrary, for the baseline design, the impact of OCV is more pronounced since the majority of the paths in this case experience a slack impact of 160ps due to OCV. Similarly, for design D6 in Fig. 4(b), the proposed method achieved to shift the OCV impact of 210 ps and above to 160 ps. On the other hand, the restructuring of the clock tree triggered by

Table 2: Clock tree characteristics for all three methods under comparison.

Design		Clock tree				
		Buffers	WL (mm)	Cap (pF)	Lat (ps)	Skew (ps)
D1	Base	64	18.7	8.4	352	88
	Cluster	65	19.0	8.5	320	88
	New	64	18.1	8.2	341	108
D2	Base	342	103.6	33.6	635	162
	Cluster	300	102.6	33.3	604	134
	New	303	99.5	32.4	535	124
D3	Base	1285	211.9	85.5	690	164
	Cluster	1201	210.8	84.7	740	140
	New	1216	211.2	84.2	677	166
D4	Base	6650	326.1	150.0	661	98
	Cluster	6688	338.2	151.5	599	110
	New	6637	327.2	149.8	679	123
D5	Base	5719	250.4	238.3	1642	143
	Cluster	6009	267.1	250.9	1749	142
	New	5611	253.5	239.9	1646	112
D6	Base	9463	569.6	774.0	1911	197
	Cluster	9540	580.7	778.5	1808	236
	New	9650	571.1	776.0	1540	173

"Cluster" [27] distributes the slack across critical paths in a non-favorable manner. This behavior highlights that simple physical clustering of clock cells is not enough for tackling the effect of clock-induced OCV.

4.3 Clock tree complexity

The reduction of slack degradation due to OCV and the overall improvement of timing achieved by the proposed method, as observed at the end of the flow, does not incur any significant overhead to the complexity of the clock tree. This conclusion is supported by the results in Table 2 which shows the number of clock buffers, the clock wirelength, and the total clock capacitance including both wire and pin capacitances, as well as the average clock latency and the clock skew. In most of the cases, the clock tree QoR metrics exhibited insignificant differences. However, there were cases where noticeable changes were observed. For instance, the clock latency of "New" for D6 improved compared to the "Base" at the cost of more clock repeaters. Our work did not intend to exercise this trade-off. This was a decision made by the CTS implementation engine.

5 CONCLUSIONS

Applying iteratively soft clustering and clock-cell relocation improves the physical proximity of timing-neighbor and reduces the clock-induced OCV by increasing the common clock tree paths. To achieve this result at the end of the flow requires a balanced approach that would take into account the spatial proximity and the timing criticality of the cells and their timing neighbors in the functional clock-tree hierarchy. Such metrics have been gracefully combined in the soft clustering algorithm that drives clock cell relocation. The results across six industrial benchmarks demonstrate the effectiveness of the proposed approach in producing robust clock trees with significantly improved timing.

ACKNOWLEDGMENTS

We would like to thank J. Shinnerl, D. Chinnery and J. de San Pedro, Mentor, a Siemens Business for their valuable comments. Dimitrios Mangiras is supported by the Onassis Foundation - Scholarship ID: G ZO 014-1/2018-2019. This research has been supported by a research grant from Mentor, a Siemens Business to DUTH.

REFERENCES

[1] C. J. Alpert, Miloš Hrkić, J. Hu, A. B. Kahng, J. Lillis, B. Liu, S. T. Quay, S. S. Sapatnekar, A. J. Sullivan, and P. Villarrubia. 2001. Buffered Steiner Trees for Difficult Instances. In *Proc. of the Intern. Symp. on Physical Design (ISPD)*. 4–9.

[2] J. Bhasker and Rakesh Chadha. 2009. *Static Timing Analysis for Nanometer Designs: A Practical Approach.* Springer.

[3] Tuck-Boon Chan, Kwangsoo Han, Andrew B. Kahng, Jae-Gon Lee, and Siddhartha Nath. 2014. OCV-aware Top-level Clock Tree Optimization. In *Proc. of the Great Lakes Symposium on VLSI (GLSVLSI)*. 33–38.

[4] Zhi-Wei Chen and Jin-Tai Yan. 2013. Routability-constrained Multi-bit Flip-flop Construction for Clock Power Reduction. *Integration VLSI J.* 46, 3 (June 2013).

[5] Yongseok Cheon, Pei-Hsin Ho, A. B. Kahng, S. Reda, and Qinke Wang. 2005. Power-aware placement. In *in Design Automation Conference (DAC)*. 795–800.

[6] Chao Deng, Yi-Ci Cai, and Qiang Zhou. 2015. Register Clustering Methodology for Low Power Clock Tree Synthesis. *Journal of Computer Science and Technology* 30, 2 (Mar 2015), 391–403.

[7] Ahran Dunsmoor and Dr. João Geada. May 21, 2012. Applications and Use of Stage-based OCV. *in EDA Designline* (May 21, 2012). https://www.edn.com/design/eda-design/4373361/Applications-and-Use-of-Stage-based-OCV

[8] Rickard Ewetz. 2017. A Clock Tree Optimization Framework with Predictable Timing Quality. In *Proc. of the Design Automation Conference (DAC)*. 72:1–72:6.

[9] R. Ewetz and C. Koh. 2015. Cost-Effective Robustness in Clock Networks Using Near-Tree Structures. *IEEE Trans. on CAD* 34, 4 (April 2015), 515–528.

[10] Matthew R. Guthaus, Gustavo Wilke, and Ricardo Reis. 2013. Revisiting Automated Physical Synthesis of High-performance Clock Networks. *ACM Trans. Des. Autom. Electron. Syst.* 18, 2 (April 2013), 31:1–31:27.

[11] Greg Hamerly and Charles Elkan. 2002. Alternatives to the K-means Algorithm That Find Better Clusterings. In *Proc. of the Intern. Conference on Information and Knowledge Management (CIKM)*. 600–607.

[12] K. Han, J. Li, A. B. Kahng, S. Nath, and J. Lee. 2015. A Global-local Optimization Framework for Simultaneous Multi-mode Multi-corner Clock Skew Variation Reduction. In *Proc. of the Design Automation Conference (DAC)*. 26:1–26:6.

[13] W. Hou, D. Liu, and P-H. Ho. 2009. Automatic Register Banking for Low-power Clock Trees. In *Proc. of the Int. Symp. on Quality of Electronic Design (ISQED)*. 647–652.

[14] Anil K. Jain. 2010. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters* 31, 8 (2010), 651 – 666.

[15] Andrew B. Kahng, Jiajia Li, and Lutong Wang. 2016. Improved Flop Tray-based Design Implementation for Power Reduction. In *Proc. International Conference on Computer-Aided Design (ICCAD)*.

[16] T. Lee, D. Z. Pan, and J. Yang. 2018. Clock Network Optimization With Multibit Flip-Flop Generation Considering Multicorner Multimode Timing Constraint. *IEEE Trans. on CAD* 37, 1 (Jan 2018), 245–256.

[17] J. Lu and B. Taskin. 2009. Post-CTS clock skew scheduling with limited delay buffering. In *IEEE Intern. Midwest Symp. on Circ. and Syst.* 224–227.

[18] U. Padmanabhan, J. M. Wang, and J. Hu. 2008. Robust Clock Tree Routing in the Presence of Process Variations. *IEEE Trans. on CAD* 27, 8 (Aug 2008), 1385–1397.

[19] D. Papa, C. Alpert, C. Sze, Z. Li, N. Viswanathan, G-J. Nam, and I. Markov. 2011. Physical Synthesis with Clock-Network Optimization for Large Systems on Chips. *IEEE Micro* 31, 4 (July 2011), 51–62.

[20] A. Rajaram and D. Z. Pan. 2006. Variation Tolerant Buffered Clock Network Synthesis with Cross Links. In *Proc. of the Intern. Symp. on Physical Design (ISPD)*. 157–164.

[21] Anand Rajaram and David Z. Pan. 2008. Robust Chip-level Clock Tree Synthesis for SOC Designs. In *Proc. of the Design Automation Conference (DAC)*. 720–723.

[22] V. Ramachandran. 2012. Construction of minimal functional skew clock trees. In *Proc. of the Intern. Symp. on Physical Design (ISPD)*. 119–120.

[23] S. Roy, P. Mattheakis, L. Masse-Navette, and D. Z. Pan. 2015. Clock Tree Resynthesis for Multi-Corner Multi-Mode Timing Closure. *IEEE Trans. on CAD* 34, 4 (2015), 589–602.

[24] I. Seitanidis, G. Dimitrakopoulos, P. Mattheakis, L. Masse-Navette, and D. Chinnery. 2019. Timing-Driven and Placement-Aware Multi-Bit Register Composition. *IEEE Trans. on CAD* 38, 8 (Aug 2019), 1501–1514.

[25] Rupesh S. Shelar. 2007. An Efficent Clustering Algorithm for Low Power Clock Tree Synthesis. In *Proc. of the Intern. Symp. on Physical Design (ISPD)*. 181–188.

[26] Necati Uysal and Rickard Ewetz. 2018. OCV Guided Clock Tree Topology Reconstruction. In *Proc. of the ASP-Design Automation Conference (ASPDAC)*. 494–499.

[27] Gang Wu, Yue Xu, Dean Wu, Manoj Ragupathy, Yu-yen Mo, and Chris Chu. 2016. Flip-flop Clustering by Weighted K-means Algorithm. In *Proc. of the Design Automation Conference (DAC)*. 82:1–82:6.

[28] B. Zhang. 2000. *Generalized k-harmonic means – boosting in unsupervised learning.* Technical Report. Technical Report HPL-2000-137, Hewlett-Packard Labs.