A Machine Learning Approach for Power Gating the FPGA Routing Network

Zeinab Seifoori
Dept. of Computer Engineering
Sharif University of Technology
Tehran, Iran
seifoori@ce.sharif.edu

Hossein Asadi
Dept. of Computer Engineering
Sharif University of Technology
Tehran, Iran
asadi@sharif.edu

Mirjana Stojilović
School of Computer and Communication Sciences
École Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
mirjana.stojilovic@epfl.ch

***Abstract*—Power gating is a common approach for reducing circuit static power consumption. In FPGAs, resources that dominate static power consumption lie in the routing network. Researchers have proposed several heuristics for clustering multiplexers in the routing network into power-gating regions. In this paper, we propose a fundamentally different approach based on *K*-means clustering, an algorithm commonly used in machine learning. Experimental results on Titan benchmarks and Stratix-IV FPGA architecture show that our proposed clustering algorithms outperform the state of the art. For example, for 32 power-gating regions in FPGA routing switch matrices, we achieve (on average) almost 1.4× higher savings (37.48% vs. 26.94%) in the static power consumption of the FPGA routing resources at lower area overhead than the most efficient heuristic published so far.**

I. Introduction

Field-programmable gate arrays (FPGAs) offer fast time-to-market, reduced nonrecurring engineering costs, inexpensive design updates, and almost unlimited design flexibility. All these features make FPGAs an extremely attractive alternative to application-specific integrated circuits (ASICs). Yet, these benefits come at a cost of higher power consumption with respect to ASICs [1]. As a consequence, if FPGAs are to compete with ASICs for the applications that require low power consumption, we need new techniques and improved FPGA architectures.

A common approach for reducing FPGA static power consumption is through power gating the FPGA logic or the routing resources [2]–[7]. Controlling the power gating regions can be done either during FPGA configuration time (statically) or while an application is running (for example, via a control circuit that decides when and which regions to enable). In this paper, our focus is on statically-controlled power gating of FPGA routing resources, as the routing network contributes to more than 70% of total static power consumption [8], [9]. Our contributions in this paper can be summarized as follows:

- This paper is the first to use a machine learning technique to design power gating regions in the FPGA routing network with the goal to reduce FPGA static power consumption.
- We define *similarity metric*, *cluster pattern*, and *power gating efficiency*, and use them to design three clustering algorithms: *SiM*, *SiM-PR*, and *SiM-IPR*. They are all derived from *K*-means clustering—a common machine learning algorithm—and yet adapted to the problem at hand.
- We compare the efficiency in switching off unused routing multiplexers among our new algorithms, the standard *K*-means clustering algorithm, and the power gating strategies proposed by other researchers [5]–[7]. To choose the power-gating clusters and to test their efficiency, we use the Titan benchmarks [10] (industrial-size FPGA circuits covering a wide range of application domains) and the Intel Stratix-IV FPGA architecture, available in the latest Verilog-to-Routing (VTR) package [11]. This FPGA architecture faithfully illustrates heterogeneous FPGA logic and routing architectures. We use COFFE [12] and HSPICE to estimate the area overhead of the power-gating approaches and the static power consumption of the routing resources.

The results show that our clustering algorithms outperform the state of the art. For example, for 32 power-gating regions in the FPGA routing switch matrices, we achieve (on average) 1.39× higher savings (37.48% vs. 26.94%) in the static power consumption of the FPGA routing resources at lower area overhead than the most efficient heuristic available [7].

II. Related Work

Previous research on reducing FPGA power consumption are mostly studies that target the static power consumption through power gating of various FPGA resource types. Since our focus is on power gating the FPGA routing resources, we review here only the most closely related publications.

Bsoul et al. propose a power-gating scheme to reduce the static power consumption of both routing resources and logic blocks [5]. They control the power consumption of FPGA resources through dynamic coarse-grained power gating. In their work, the switch matrices (SMs) can operate in different power modes: “always off”, “power-controlled”, and “always on”. The “always off” SMs are permanently unused. The power consumption of “power-controlled” SMs is controlled dynamically by an on-chip controller. The power controller signals are routed through the “always on” SMs. Since Bsoul et al. do not focus on the power consumption reduction of partially unused SMs, the power efficiency of this method is

limited to the number of entirely unused SMs, which is not very high. They report 70% to 84% reduction in the static power consumption.

Li et al. propose a coarse-grained power-gating scheme that dynamically controls the power consumption of the power-gating regions [13], which include logic blocks and their corresponding connection blocks, through a power control hard macro (PCHM). Since the clock signals of the power gating regions come from the PHCM, this scheme can also decrease the dynamic power consumption of idle regions through clock gating. In addition, this study modifies the cost function of the placement algorithm to increase the power gating opportunities. Li et al. report up to 51% reduction in the power consumption.

Hoo et al. decrease the static power consumption of uni-directional switch matrices through a coarse-grained power gating technique [14]. In their approach, the buffers in each side of a switch matrix are grouped together. In addition, they modify the VTR routing algorithm to increase the power gating opportunities. Hoo et al. also dynamically turn off the FPGA modules during their idle periods, to increase the power savings. They reported that $\approx 40\%$ of the power gating regions could be turned off in their experiments.

Yazdanshenas et al. propose a fine-grained power-gating scheme to reduce the static power consumption of both unused SRAM cells in the logic blocks and the routing resources [15]. They divide each look-up table (LUT) into smaller ones to increase the controllability of the power consumption of the LUTs. They also control the power consumption of each switch block through one configuration cell. The experimental results show that the suggested architecture results in 4%, 27%, and 75% power saving in the switch blocks, the configuration blocks, and the LUTs, respectively.

Seifoori et al. show that the utilization rate of the routing resources is related to the FPGA routing architecture [7]. They examine power gating granularities and select the most efficient for various routing architectures. They report up to 57% reduction in the static power consumption, but the routing network in the target FPGA architecture in their work is outdated: it is composed of uniform wirelengths and simple routing switch patterns. In this work, we compare their most performing strategy with our approaches, and we do that using newer FPGA routing architecture.

Static power reduction obtained thanks to power gating depends on the number and composition of the power-gating regions. Providing an optimal or near optimal clustering of routing multiplexers in power-gating regions can significantly improve the static power consumption and mitigate the costs of area and power overheads caused by the added power-gating circuitry. All of the previous works that design statically-controlled power gating regions do it using heuristic approaches listed above. Here, we take a different approach: we first analyze the utilization of the routing resources and then decide how to cluster them into power-gating regions.

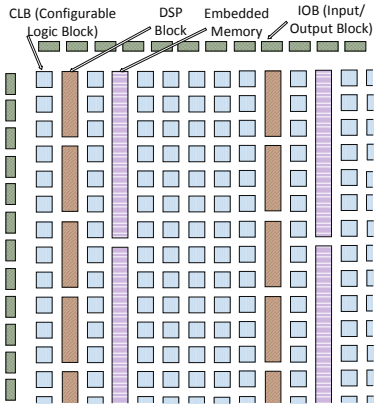

Fig. 1. FPGA architecture: besides logic blocks (in blue), FPGAs contain hard IP blocks (for example, DSPs), embedded memory blocks, external memory interfaces, transceivers, phased-locked loops, etc. Connectivity between all these elements is established using a configurable routing network, composed of horizontal and vertical routing wires and configurable switches.

III. Background and Motivation

In this section, we explain the basics of the FPGA architecture, motivate our work, and introduce *K*-means clustering.

A. FPGA Architecture

Traditionally, FPGAs are seen as arrays of logic blocks, distributed and block RAM memories, phased-locked loops, and I/O pins. Today, FPGAs are often equipped with specialized hardware, such as processor cores, hard external memory interfaces, on-chip hard IP blocks (for digital signal processing, for example), and transceivers for various signalling standards [16], [17] (Fig. 1).

Logic blocks, commonly referred to as the configurable logic blocks (CLBs) or the logic array blocks (LABs), are the most numerous elements of the FPGA. They implement the general purpose logic. The LABs are composed of

- **logic elements** (LEs), comprising look-up tables (LUTs), programmable registers, and a number of multiplexers that allow the use of either look-up tables or registers, and
- intra-LAB routing resources that connect the local feedback LE outputs or the inter-LAB wires to the LE inputs.

The connectivity between the FPGA blocks (logic elements, memory, hard IPs, etc.) is achieved using the FPGA routing network composed of **wires**, organized in horizontal and vertical routing channels, and routing **switch matrices**, which serve to enable the wires to connect to each other. Fig. 2 illustrates a part of a switch matrix found in FPGA routing networks, such as the Intel Stratix IV. Between two consecutive columns of FPGA resources, there are two types of vertical interconnects: C4 (wires that span four rows of the FPGA array) and C12 (long interconnects, wires that span 12 rows). Similarly, between two consecutive rows of FPGA elements, there are two types of horizontal interconnects: R4 and R20 (wires that span 20 columns). The nonuniform wirelengths are used to balance the flexibility, the delay, and the area of the routing network. The short wires (R4 and C4)

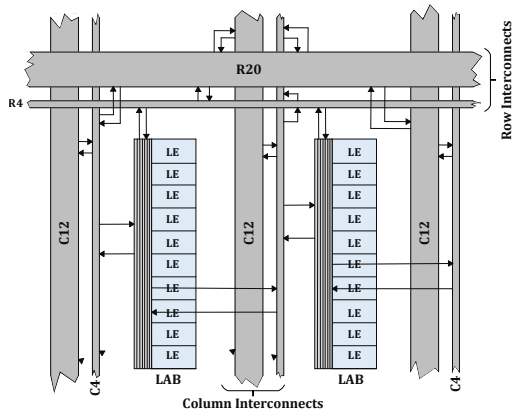

Fig. 2. Connectivity between the FPGA elements is achieved using the routing resources: wires organized in horizontal and vertical routing channels and routing switch matrices, which enable the wires to connect to each other. This figure illustrates a modern FPGA routing network based on the Intel Stratix IV FPGA architecture description available in VTR tool [11].

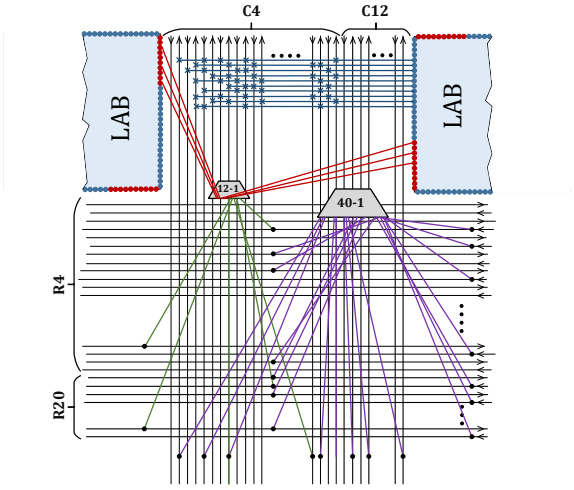

Fig. 3. An illustration of a part of a switch matrix inside the Stratix IV FPGA. It shows two multiplexer types (12:1 and 40:1) as well as two types of connections: (1) between the column wires (C4, C12) and the LAB inputs (in blue) and (2) among the LAB outputs (in red), the column interconnects, and the row interconnects.

are accessible by FPGA elements (can drive the LE inputs or be driven by the LE outputs). The wires in horizontal (row) and vertical (column) channels can be connected to form long routes. These connections are possible thanks to the multiplexers inside **switch matrices**. The multiplexers often have nonuniform sizes: large multiplexers (40:1) drive long wires, while small multiplexers (12:1) drive short wires, as illustrated in Fig. 3. Four different types of switch matrices can be identified in Stratix IV FPGA description [11]. They are listed in Table I and shown spatially distributed in Fig. 4.

B. Motivation

Routing resources can be clustered into power-gating regions to reduce static power consumption. Clustering can be coarse grained or fine grained [2]–[7], [13], [15]. However, both approaches have their advantages and drawbacks: the

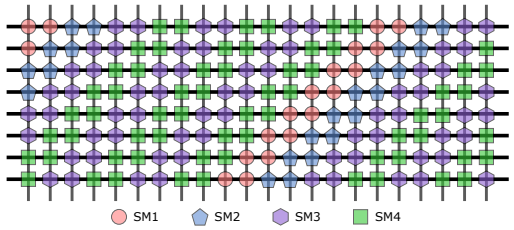

Fig. 4. Spatial distribution of switch matrices in Stratix IV, shown on a small part of the FPGA. Horizontal and vertical lines represent routing channels, composed of unidirectional wires of nonuniform lengths (Fig. 2). Four types of switch matrices can be identified (SM1, SM2, SM3, and SM4, listed in Table I). They differ in the number of 12:1 and 40:1 multiplexers. A periodic pattern in the switch matrix distribution can be observed, which is in fact expected given that the wirelengths are all multiples of four.

TABLE I
ROUTING SWITCH MATRIX TYPES EXTRACTED FROM THE INTEL STRATIX IV FPGA ARCHITECTURE DESCRIPTION IN VTR [11]. THE NUMBER OF MULTIPLEXERS PER SWITCH MATRIX TYPE DIFFERS, AS THE TOPOLOGIES ARE ADAPTED FOR CONNECTING WIRES OF NONUNIFORM LENGTHS.

Multiplexer	SM1	SM2	SM3	SM4
12:1	128	128	132	132
40:1	4	8	4	8

coarse-grained clustering limits the power-gating opportunities, while the fine-grained clustering imposes higher area and power overhead. Additionally, for any granularity, the clustering can be done in many different ways, which are not all equally good.

To illustrate this on an example, let us consider a simplified FPGA routing architecture, in which all routing segments are of the length one, all switch matrices are composed of the same number and type of multiplexers, and the routing channels are 32-bit wide. Fig. 5 shows the topology of the corresponding switch matrix; one quarter of the switch matrix multiplexers are used per each side (top, bottom, left, and right) to drive unidirectional routing segments and enable connecting them to form longer wires. Then, let us use Verilog-to-Routing (VTR) to place and route a benchmark circuit [11], for example *usb-phy* from the IWLS'05 benchmark suite [18], and analyze the utilization pattern of the multiplexers in the switch matrices. Firstly, it can be observed that the utilization pattern varies from one switch matrix to another. For example, in one of the switch matrices (SM1) inside the region occupied by the benchmark circuit, only the multiplexers shaded in dark gray in Fig. 5 are in use; others not. In another switch matrix inside the same region (SM2) different multiplexers are in use. Table II lists the multiplexer utilization patterns (0-unused, 1-used) for SM1 and SM2.

One possibility is to create 16 clusters, each containing four muxes in total, where every mux in a cluster is driving the routing track of the same number [7]; In other words, all muxes marked as M1 comprise one cluster, all muxes marked as M2 comprise another cluster, etc. As a result, 25% of all clusters can be switched off in SM1 and only one cluster in SM2 (clusters that can be switched off are marked in red in Table II), which is equivalent to 36% and 11% of all **unused**

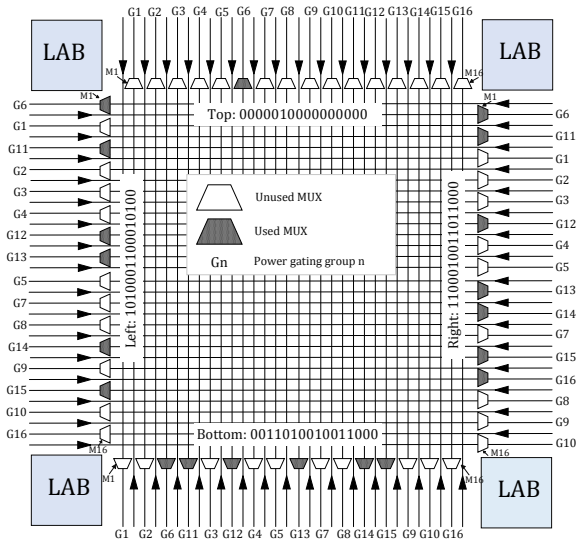

Fig. 5. Distribution of used switch-matrix multiplexers and power gating groups (sample switch matrix No. 1 in *usb-phy* benchmark circuit).

TABLE II
UTILIZATION PATTERN OF THE MULTIPLEXERS IN TWO RANDOMLY CHOSEN SWITCH MATRICES INSIDE THE FPGA REGION OCCUPIED BY USB-PHY BENCHMARK. TO P&R THE BENCHMARK, VTR [11] AND A SIMPLIFIED FPGA ARCHITECTURE DETAILED IN FIG. 5 ARE USED.

SM SIDE	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16
SM1-TOP	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
SM1-RIGHT	1	1	0	0	0	1	0	0	1	1	0	1	1	0	0	0
SM1-BOTTOM	0	0	1	1	0	1	0	0	1	0	0	1	1	0	0	0
SM1-LEFT	1	0	1	0	0	0	1	1	0	0	0	1	0	1	0	0
SM2-TOP	0	0	1	1	0	1	1	0	0	1	1	0	0	0	0	1
SM2-RIGHT	1	1	1	0	1	0	1	0	0	1	0	1	1	0	0	1
SM2-BOTTOM	0	0	1	0	1	0	1	1	1	0	0	1	1	0	1	1
SM2-LEFT	1	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0

muxes in SM1 and SM2, respectively. Another, less obvious clustering possibility is labelled as Gi in Fig. 5 and Table III. This clustering would allow to switch off many more unused muxes: ≈80% in SM1 and ≈36% in SM2. Yet, it is not certain that this clustering, if applied to all switch matrices used by the usb-phy circuit, would result in the optimal power reduction scheme. Hence, to find a clustering solution that performs well in general, we need to extend the analysis to a wide range of application domains. Since finding the optimal clustering is a known NP-complete problem, we can try using a machine-learning technique to solve it.

C. K-means Clustering

Clustering algorithms are used in many different applications, from pattern classification [19] to knowledge discovery and data mining [20]. One of the most popular and studied clustering algorithm in unsupervised machine learning is *K*-means clustering [21], [22].

TABLE III
CLUSTERING LABELED Gi IN FIGURE 5 IS SUPERIOR, AS MORE UNUSED MULTIPLEXERS ARE GROUPED IN THE SAME POWER GATING REGIONS.

SM SIDE	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16
SM1-TOP	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
SM1-RIGHT	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1
SM1-BOTTOM	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0
SM1-LEFT	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0
SM2-TOP	0	0	1	1	0	1	1	0	0	1	1	0	0	0	0	1
SM2-RIGHT	1	0	1	1	0	1	0	0	0	1	1	0	0	1	1	1
SM2-BOTTOM	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1
SM2-LEFT	0	0	1	0	0	1	0	0	0	0	0	0	1	1	0	0

Given a set of N data points $\mathbf{V} = \{v_1, v_2, ..., v_N\}$, where each data point is an m-dimensional vector, *K*-means clustering partitions the data into k clusters $\mathbf{C} = \{C_1, C_2, ..., C_k\}$, with the aim of maximizing the disparity among the clusters and minimizing the disparity within the clusters [23], [24]. The notion of disparity and what metric should be measured to calculate it depends on the application [25]. The algorithm is iterative in nature. It starts by setting the initial cluster centers and then repeats the following steps:

1) Assign each data point v_i, $1 \leq i \leq N$, to the cluster C_j, $1 \leq j \leq k$, whose center has the least squared Euclidean distance to v_i [26]–[28].
2) Compute new cluster centers as the means of the cluster members.

The algorithm stops once there are no updates to be made.

Three user-defined parameters that can significantly affect the efficiency of the algorithm are the homogeneity metric, the number of clusters k, and the initialization of the cluster centers. The homogeneity metric, which is often the distance to the cluster center, can be changed according to the clustering criteria. To find the most convenient number of clusters, the algorithm can be repeated with variable number of clusters and the obtained solutions compared.

There are several initialization procedures that affect the *K*-means algorithm sensitivity to the initial cluster center. For example, one can run *K*-means algorithm with different cluster initializations and afterwards select the grouping that results in the least squared distance. The main drawback of this approach is its time complexity in clustering large data sets. Global *K*-means clustering proposes an alternative initialization procedure, based on incremental addition of one cluster center at a time, based on the previously determined cluster centers [29]. This procedure is computationally intensive too, as it requires a number of executions of the *K*-means clustering only to find suitable initial cluster centers. Yet another, but considerably less complex, initialization approach is *k-means++* [30]. It selects the center of the first cluster randomly, and then it applies the probabilistic metric proportional to the distance to the previously selected centers to find each subsequent cluster center. A thorough comparative study by Celebi et al. [31] finds that *k-means++* performs generally well; hence, we opted for this initialization method.

IV. PROPOSED CLUSTERING ALGORITHMS

As illustrated in the motivational example (Section III-B), what may seem to be the optimal multiplexer clustering in one switch matrix, can be far from optimal in another switch matrix. Furthermore, trying all possible multiplexer clustering opportunities is impractical, due to a prohibitively large solution space. The above describes a common set of problems that can be solved using machine learning (ML) algorithms [32]–[36]. Hence, we decide to borrow from existing ML algorithms, such as *K*-means clustering, to solve the multiplexer clustering problem efficiently.

However, prior to clustering, one needs to gather learning data: create a set of multi-dimensional vectors that capture all the information relevant for determining good clusters. In our problem—assigning switch matrix multiplexers to power-gating regions—it is the utilization pattern of all multiplexers in all switch matrices, and for all benchmark circuits, that captures the desired information. Hence, on a set of L training (learning) benchmarks, we employ the following steps to gather the data:

1) Place and route all L benchmarks on the target FPGA.
2) For every benchmark, in the FPGA region occupied by it, identify all switch matrices that are in use[1]. If there are several different types of switch matrices in the target FPGA architecture, which is common in modern FPGAs where interconnects have nonuniform lengths, consider every switch matrix type as an independent clustering problem. Consequently, split all the used switch matrices in groups according to their types. Then, use these groups to create input data sets for the clustering algorithms, as described in the next steps.
3) For every switch matrix type and for every multiplexer M_i in a switch matrix, create a row vector

$$v_i = \begin{bmatrix} v_{i_1} & v_{i_2} & ... & v_{i_L} \end{bmatrix}, \quad (1)$$

composed of a sequence of L smaller vectors v_{i_m}, where m is the index of the benchmark from which the vector v_{i_m} is extracted. The length of vector v_{i_m} equals the number of times this particular switch matrix type is in use by the benchmark B_m and may not be the same across all benchmarks. An element in the column j of the vector v_{i_m} represents the utilization of the multiplexer M_i in the j^{th} instance of the switch matrix in benchmark B_m: 1 if M_i is in use by the benchmark B_m, 0 otherwise.
4) Add all vectors v_i to the learning data set V.

In the following sections, we propose two approaches for finding an efficient multiplexer clustering scheme. Their most important difference lies in the metric used when assigning objects to clusters. The first algorithm employs squared Euclidean distance metric and assigns an object to the cluster whose center is the least distant (Subsection IV-A). The second, however, quantifies the similarity in multiplexer utilization and assigns an object to the cluster that has the most similar utilization pattern (section IV-B).

A. Clustering using K-means Algorithm (KM)

The first clustering algorithm, shown in Algorithm 1, is essentially *K*-means clustering. The algorithm inputs are the data set V and the number of clusters K. Computation starts by initializing a special vector called *cluster center*. Then, the next three steps are repeated in a loop: emptying the clusters and, for every object in the data set, assigning it to the closest cluster and updating that cluster center. The loop repeats as long as the output (cluster set C) is changing (and a threshold in the number of executions is not reached).

[1]A switch matrix is in use if at least one of its multiplexers is in use.

Algorithm 1: Power gating using *K*-means clustering.

```
Input: Data set: V = {v_i}, i = 1..N
Input: Total number of clusters: K
Output: Clusters C = {C_1, C_2, ..., C_K}
Variables: Cluster centers: μ_1, μ_2, ..., μ_K
InitCenters(V, K)
while Cluster members change do
    foreach C_i ∈ C do
        C_i ← ∅
    foreach v_i ∈ V do
        k ← FindClosestCluster(v_i, C)
        C_k ← C_k ∪ v_i
        μ_k ← UpdateCenter(μ_k, v_i)
```

To find initial cluster centers we employ a well known *K*-means++ initialization method [31] (Section III-C). Algorithm 2 details the implementation of `initCenters` function. The center of the first cluster is chosen randomly from the data set V. Then, for every object v_i in V, we call `computeDistance` function to find the distance $d_{i,j}$ between v_i and an already initialized cluster center μ_j. Next, we compute the probability that v_i may become the next cluster center as $\frac{d_{i,j}^2}{\sum_{i=1..N} d_{i,j}^2}$, where N is the cardinality of the set V. The vector that maximizes this probability finally becomes the initial center of the next cluster.

To compute the distance between two objects or an object and a cluster center (in `FindClosestCenter` function), we call the `ComputeDistance` function, which returns the squared Euclidean distance between two vectors v_i and v_j:

$$d_{i,j} = ||v_i - v_j||^2 \quad (2)$$

At the end of every clustering iteration, cluster centers are recomputed in `UpdateCenter` function as the mean of all the objects in the cluster.

B. Clustering using Utilization Similarity Metric (SiM)

A power gating cluster C_i is defined by the switch matrix multiplexers it groups together, where every multiplexer is represented by a vector v_i, defined in (1). In vectors v, an element (dimension) corresponds to a switch matrix and the element value corresponds to the usage of the multiplexer in that switch matrix: 1, if in use; 0, otherwise. Therefore, two multiplexers M_i and M_j having the same utilization pattern are represented by two identical vectors v_i and v_j. Such two multiplexers should, ideally, belong to the same cluster.

Let us define the *similarity metric* $s_{i,j}$ between two vectors v_i and v_j as the number of dimensions m in which they are equal:

$$s_{i,j} = \sum_{m=1..|v_i|} \begin{cases} 1, & \text{if } v_i[m] = v_j[m] \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

Algorithm 2: Function `InitCenters` for computing the initial cluster centers.

Input: Data set: $V = \{v_i\}, i = 1..N$
Input: Total number of clusters: K
Output: Cluster centers: $\mu_1, \mu_2, ..., \mu_K$
Variables: Distance between v_i and μ_j: $d_{i,j}$; sum of distances squared: $dssq$; next cluster ID: k; probabilistic distance metric: p

```
μ_1 ← randomly chosen from V
k ← 2
while k < K do
    dssq ← 0
    foreach v_i ∈ V do
        foreach μ_j, 1 ≤ j < k do
            d_{i,j} ← ComputeDistance(v_i, μ_j)
            dssq ← dssq + d_{i,j}^2
    p ← 0
    foreach v_i ∈ V do
        p_{i,j} ← d_{i,j}^2 / dssq
        if p_{i,j} > p then
            p ← p_{i,j}
            μ_k ← v_i
    k ← k + 1
```

As a consequence, instead of assigning an object v_i to the cluster that is the closest in squared Euclidean distance (2), we should rather assign it to the cluster that maximizes the similarity between v_i and the cluster members. The cluster center, defined in Section IV-A, is not a good indication of similarity between the entire cluster and an object. Instead, we could define the *mode* of a cluster—commonly done in clustering categorical data—as the vector in which an element in dimension m takes the value of the most frequent element in the same dimension across all vectors in the cluster. However, given that there are many more unused than used multiplexers in switch matrices, this approach skews all cluster modes toward zero. Hence, instead of the cluster center and the cluster mode, we introduce here the concept of the cluster *pattern* ρ. In clusters containing a single member, those members are the cluster patterns. However, in clusters with two members and more, the cluster pattern is progressively constructed from the current cluster pattern and the newly added member v as

$$\rho[m] = \begin{cases} 1, & \text{if } \rho[m] = v[m] \text{ and } v[m] = 1 \\ 0, & \text{if } \rho[m] = v[m] \text{ and } v[m] = 0 \\ \mathbf{X} & \text{if } \rho[m] \neq v[m]. \end{cases} \quad (4)$$

Here, m is in the range $1 \leq m \leq |v|$ and $\mathbf{X}$ stands for a value that is neither 1 nor 0[2]. As a result, we can assess the power gating efficiency ε of cluster C_i as the cluster size weighted by the number of the pattern elements different than $\mathbf{X}$:

$$\varepsilon(C_i) = |C_i| \cdot \Big(|\rho_i| - CountX(\rho_i) \Big) \quad (5)$$

Here, $CountX(\rho_i)$ is the number of the pattern vector elements equal to $\mathbf{X}$. The power gating efficiency of the entire set C of K clusters then becomes:

$$\varepsilon(C) = \sum_{i=1..K} |C_i| \cdot \Big(|\rho_i| - CountX(\rho_i) \Big) \quad (6)$$

From (5) and (6), it can be inferred that the optimal clustering solution C is the one that maximizes the metric $\varepsilon(C)$.

[2]In our implementation, $\mathbf{X}$ equals -1, but it can be any other value, provided that it is neither 0 nor 1.

Algorithm 3: Our novel *SiM* clustering, which uses the similarity metric defined in (3) and the concept of cluster pattern from (4).

Input: Data set: $V = \{v_i\}, i = 1..N$
Input: Total number of clusters: K
Output: Clusters $C = \{C_1, C_2, ..., C_K\}$
Variables: Cluster patterns: $\rho_1, \rho_2, ..., \rho_K$

```
InitPatterns(V, K)
foreach v_i ∈ V do
    k ← FindTheMostSimilarCluster(v_i, C)
    C_k ← C_k ∪ v_i
    ρ_k ← UpdateClusterPattern(ρ_k, v_i)
```

1) SiM clustering: Algorithm 3 lists the steps of our clustering algorithm that uses the similarity metric and the notion of cluster pattern to cluster the multiplexers. We refer to this algorithm as *SiM*. The function `InitPatterns` is identical to the function `InitCenters` in Algorithm 2, except that, instead of calling `ComputeDistance` and returning the cluster centers, it computes the similarity using the expression in (3) and returns the cluster patterns. The function `FindTheMostSimilarCluster(`v_i, C`)` returns the cluster index k, such that the similarity metric for the cluster pattern ρ_k and the vector v_i is maximized. Finally, the function `UpdateClusterPattern` recomputes the pattern following the expression in (4).

2) SiM-PR clustering: Starting new clustering iteration without updating the cluster pattern at the end of the previous clustering iteration is highly unlikely to result in a more efficient clustering solution, because $CountX(\rho_i)$ either remains the same or increases as the iterations advance. This is why we let *SiM* clustering algorithm run a single iteration only. To overcome this limitation, we design *SiM-PR* (*SiM* with pattern reduction) clustering algorithm, shown in Algorithm 4. Unlike *SiM*, *SiM-PR* runs in iterations and, at the end of every iteration, replaces cluster patterns with randomly chosen members of the corresponding clusters. Consequently, every new iteration starts with patterns whose elements are reduced to two values only: 0 and 1.

3) SiM-IPR clustering: *SiM* and *SiM-PR* result in two extreme solutions. The former completes only one iteration, while the latter allows for more iterations but, at the end of every iteration, it reduces the cluster patterns to randomly

Algorithm 4: Our novel *SiM-PR* clustering. Unlike *SiM*, this algorithm runs multiple iterations, at the end of every iteration, replaces the cluster pattern with a randomly chosen cluster member. As a consequence, at the end of every iteration, the patterns elements are reduced back to two values only: 0 and 1.

Input: Data set: $V = \{v_i\}, i = 1..N$
Input: Total number of clusters: K
Output: Clusters $C = \{C_1, C_2, ..., C_K\}$
Variables: Cluster patterns: $\rho_1, \rho_2, ..., \rho_K$
`InitPatterns`(V, K)
while *Cluster members change* **do**
  foreach $v_i \in V$ **do**
    $k \leftarrow$ `FindTheMostSimilarCluster`(v_i, C)
    $C_k \leftarrow C_k \cup v_i$
    $\rho_k \leftarrow$ `UpdateClusterPattern`(ρ_k, v_i)
  foreach $C_i \in C$ **do**
    ρ_i = `RandomMemberFromCluster`(C_i)

chosen elements from the corresponding clusters. The randomness in *SiM-PR* can help finding a more efficient clustering solution compared to *SiM*, but it can also cause the algorithm to remain in a locally-optimal solution. Therefore, we suggest an alternative in which the patterns are reduced incrementally, depending on the efficiency metric defined in (5). We name this algorithm *SiM* with Incremental Pattern Reduction (*SiM-IPR*) and show its implementation in Algorithm 5. Unlike *SiM-PR*, this algorithm applies the pattern reduction only on R/K of clusters that are the most inefficient (have the lowest power gating efficiency metric computed as in (5)). The parameter R (rate) represents the number of clusters whose patterns are to be reduced in the current clustering iteration. The initial value and rate of decay of R may influence the results. In our implementation, we chose to set its initial value to $K/2$ and to reduce R by half in every subsequent iteration, without comparing with other possibilities for the moment.

V. Experimental Setup and Results

To find the power-gating clusters and to test their efficiency, we use Titan benchmarks [10] (industrial-size FPGA circuits covering a large range of application domains) and the Intel Stratix-IV FPGA architecture, available as part of the latest VTR package [11], [17]. This FPGA architecture faithfully illustrates heterogeneous FPGA logic and routing architectures. Section III explains it in detail. We ran two experiments (EXP1 and EXP2), both using benchmarks listed in Table IV. The experiments differ in the set of circuits used to define power gating clusters (to *learn*, hence the letter L in Table IV) and those used to test the performance (marked with T). We used VTR to place and route the benchmarks, as well as extract all the data on multiplexer utilization; routing channel width was set to 300, to allow the largest benchmark to route successfully.

Fig. 6 compares our algorithms to *K*-means clustering. It shows, side by side, the number of multiplexers that can be

Algorithm 5: Our novel *SiM-IPR* clustering. Unlike *SiM-PR*, this algorithm applies pattern reduction only on R/K of clusters that are the most inefficient, according to (5). The parameter R stands for rate. Its initial value is $K/2$. In every subsequent iteration, it is reduced by half.

Input: Data set: $V = \{v_i\}, i = 1..N$
Input: Total number of clusters: K
Output: Clusters $C = \{C_1, C_2, ..., C_K\}$
Variables: Cluster efficiencies: $E = \{E_1, E_2, ..., E_K\}$;
  set of clusters whose patterns are to be reduced: C_R; cluster reduction rate: R;
  cluster patterns: $\rho_1, \rho_2, ..., \rho_K$.
`InitPatterns`(V, K)
$R \leftarrow K/2$
while *Cluster members change* **do**
  foreach $v_i \in V$ **do**
    $k \leftarrow$ `FindTheMostSimilarCluster`(v_i, C)
    $C_k \leftarrow C_k \cup v_i$
    $\rho_k \leftarrow$ `UpdateClusterPattern`(ρ_k, v_i)
  foreach $C_i \in C$ **do**
    $E_i \leftarrow$ `Efficiency`(C_i)
  $C_R = \emptyset$
  for $1 \leq r \leq R$ **do**
    k = `LeastEfficientIndex`(E)
    $E \leftarrow E \setminus E_k$
    $C_R = C_R \cup C_k$
  foreach $C_i \in C_R$ **do**
    ρ_i = `RandomMemberFromCluster`(C_i)
  $R \leftarrow R/2$

TABLE IV
Benchmark details.

No.	Name	# Blocks	DSPs	FPGA Size	EXP1	EXP2	Application
B1	sparcT1_chip2	814,799	24	279 × 207	L	L	Multi-coreμP
B2	LU_Network	630,212	896	221 × 164	L	L	Matrix Decomposition
B3	mes_noc	548,047	0	192 × 142	L	L	On-chip Network
B4	gsm_switch	487,454	0	255 × 189	L	L	Communication Switch
B5	denoise	343,263	192	150 × 111	L	T	Image Processing
B6	sparcT2_core	287,839	0	152 × 113	L	T	μP Core
B7	cholesky_bdti	257,750	1,027	169 × 125	L	T	Matrix Decomposition
B8	minres	252,600	614	224 × 166	L	T	Control Systems
B9	stap_qrd	237,193	579	158 × 117	L	L	Radar Processing
B10	openCV	212,616	740	232 × 172	L	T	Computer Vision
B11	dart	202,414	0	138 × 102	L	L	Network Simulator
B12	bitonic_mesh	192,648	676	242 × 179	L	L	Sorting
B13	des90	109,962	352	171 × 127	L	L	MultiμP system
B14	neuron	90,779	565	129 × 96	L	L	Neural Network
B15	segmentation	174,072	104	136 × 101	T	L	Computer Vision
B16	SLAM_spheric	124,648	296	124 × 92	T	L	Control Systems
B17	cholesky_mc	108,239	452	125 × 93	T	L	Matrix Decomposition
B18	stereo_vision	92,662	152	129 × 96	T	L	Image Processing
B19	sparcT1_core	91,235	8	82 × 61	T	L	μP Core

switched off for all benchmarks, when *SiM*, *SiM-PR*, and *SiM-IPR* algorithms are applied and K is set to 4, 8, 16, 24, and 32. The results are normalized to those of *K*-means algorithm. We can observe that for low number of clusters (4, 8, 16), *K*-means clustering is superior, whereas for higher number of clusters (24, 32) *SiM-IPR* takes over, often outperforming *K*-means by 10–20%.

Then, we use COFFE [12], 22nm predictive technology model [37], and HSPICE to estimate the area overhead of our resulting FPGA architectures (Table V), as well as those ob-

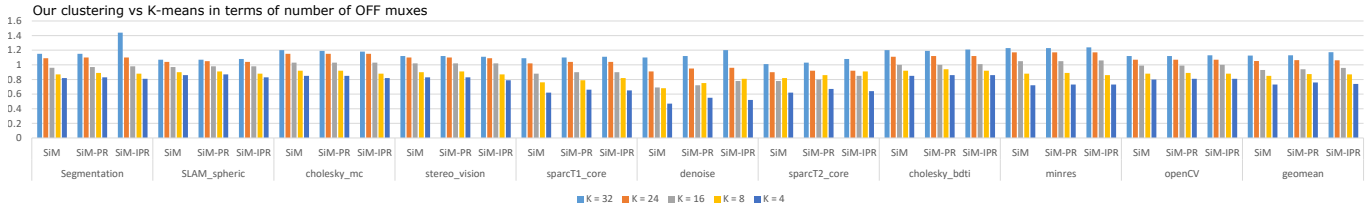

Fig. 6. Number of multiplexers that can be switched off for 4, 8, 16, 24, and 32 clusters per switch matrix, normalized to those of K-means algorithm. It can be noticed that for low number of clusters (4, 8, 16), K-means is superior, whereas for higher number of clusters (24, 32) *SiM-IPR* takes over, often outperforming K-means by 10–20%.

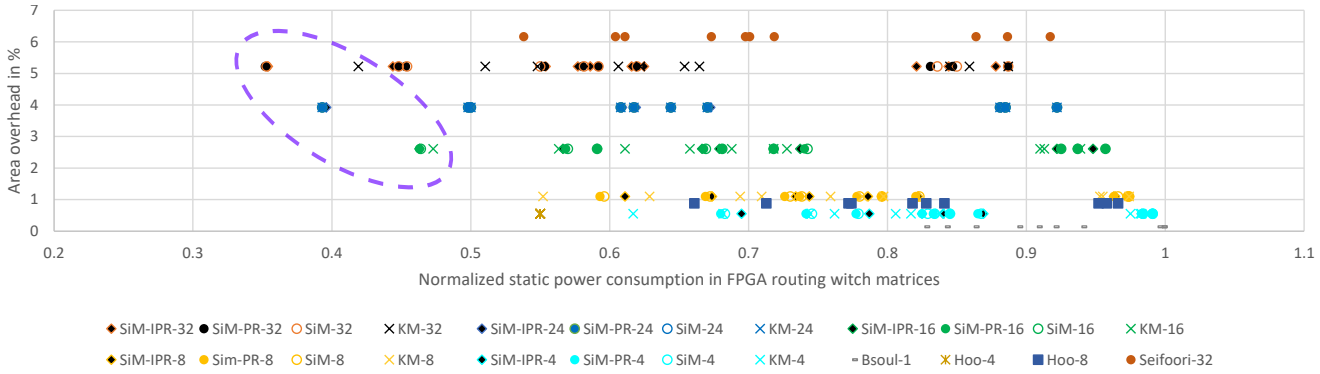

Fig. 7. Area overhead (in %) versus static power consumption in the FPGA routing network; latter is normalized to the static power consumption estimated using an FPGA routing architecture that provides no power-gating mechanisms. One marker corresponds to one benchmark. The results of both experiments (EXP1 and EXP2) are plotted here.

TABLE V
AREA OVERHEAD IN THE FPGA ROUTING SWITCH MATRICES AFTER ADDING POWER-GATING LOGIC. FOR A FIXED K, ALL OUR ALGORITHMS REPORT VERY SIMILAR OVERHEADS.

Bsoul et al. $K = 1$ [5]	**Hoo et al.** $K = 4$ [6]	**Hoo et al.** $K = 8$ [6]	**Seifoori et al.** $K = 32$ [7]	***Our*** $K = 32$	***Our*** $K = 24$	***Our*** $K = 16$	***Our*** $K = 8$	***Our*** $K = 4$
0.136%	0.55%	0.88%	6.16%	5.48%	4.12%	2.69%	1.18%	0.58%

TABLE VI
PERCENTAGE OF ALL MULTIPLEXERS THAT CAN BE TURNED OFF USING THE POWER GATING SCHEMES PROPOSED IN RELATED WORKS [5]–[7] VERSUS K-MEANS CLUSTERING (FOR $K \leq 16$) AND *SiM-IPR* CLUSTERING (FOR $K = 32$).

Experiment-Benchmark	**Bsoul et al.** $K = 1$ [5]	**Hoo et al.** $K = 4$ [6]	***KM*** $K = 4$	**Hoo et al.** $K = 8$ [6]	***KM*** $K = 8$	**Seifoori et al.** $K = 32$ [7]	***SiM-IPR*** $K = 32$
EXP1-B15	8.70%	14.41%	17.99%	16.76%	22.44%	31.09%	52.74%
EXP1-B16	15.28%	22.66%	28.37%	24.66%	32.18%	35.62%	46.63%
EXP1-B17	10.24%	17.31%	21.67%	19.50%	26.76%	36.41%	49.30%
EXP1-B18	19.17%	33.97%	42.59%	37.11%	49.75%	51.48%	71.54%
EXP1-B19	0.35%	2.42%	2.95%	4.03%	5.26%	12.61%	17.07%
EXP2-B5	0.17%	1.73%	2.06%	3.81%	3.90%	13.03%	17.11%
EXP2-B6	0.13%	1.35%	1.62%	2.74%	2.81%	9.23%	12.59%
EXP2-B7	11.46%	15.99%	20.01%	18.14%	24.23%	33.39%	45.61%
EXP2-B8	6.54%	21.22%	26.46%	24.32%	33.93%	44.10%	61.34%
EXP2-B10	17.58%	28.55%	35.34%	31.37%	41.20%	43.44%	61.30%
Geomean	3.48%	10.05%	12.41%	13.31%	16.88%	26.94%	37.48%

tained using the power gating approaches proposed by Bsoul et al. ($K = 1$, as our focus is on the static power-gating strategy and not the dynamic one), Hoo et al. [6] (two versions: $K = 4$ and $K = 8$[3]), and Seifoori et al. [7] (K = 32, multiplexers driving the same track in each direction are clustered together). It should be noted here that the area overhead is computed with respect to the area of the FPGA switch matrix and that we estimate the static power consumption of FPGA routing resources and power-gating circuitry only.

Results are plotted in Fig. 7. The vertical axis shows area overhead, while the horizontal axis shows static power consumption, normalized to the static power consumption when no power gating is used (baseline). Dashed line emphasizes the best results achieved by our algorithms showing that we do manage to outperform all other clustering strategies. Our algorithms achieve the highest static power consumption savings for $K = 32$, at an area overhead of less than 6%. Finally, Table VI compares the ratios of multiplexers that can be switched off using heuristics by other researchers vs. ours, for the same K to enable fair comparison. In all cases, our strategies report higher values.

VI. CONCLUSIONS

In this paper, we leverage machine learning to reduce static power consumption of FPGA routing resources. The experimental results show that our FPGA power-gated architectures achieve better results at an even lower overhead in area, compared to the power-gating heuristics published so far. Future work will focus on formulating a metric that would guide our clustering algorithms to consider the power consumption of the power gating logic as well. Consequently, even better results may be expected.

[3] K = 4 clusters all multiplexers in each side of the switch matrix into a single power gating region. K = 8 clusters the multiplexers in each side of a switch matrix in two power gating regions, where one region contains only *12:1* multiplexers and the other only *40:1* multiplexers.

REFERENCES

[1] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, Jan. 2007.

[2] Y. Lin, F. Li, and L. He, "Routing track duplication with fine-grained power-gating for FPGA interconnect power reduction," in *Proceedings of the Asia and South Pacific design automation conference (ASP-DAC)*. Shanghai, China: ACM, Jan. 2005, pp. 645–650.

[3] A. A. Bsoul and S. J. Wilton, "An FPGA architecture supporting dynamically controlled power gating," in *Proceeding of International Conference on Field-Programmable Technology (FPT)*. Beijing, China: IEEE, Dec. 2010, pp. 1–8.

[4] A. Gayasen, Y. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, "Reducing leakage energy in FPGAs using region-constrained placement," in *Proceedings of the 12th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, Monterey, CA, USA, Feb. 2004, pp. 51–58.

[5] A. A. Bsoul and S. J. Wilton, "An FPGA with power-gated switch blocks," in *Proceedings of International Conference on Field-Programmable Technology (FPT)*. Seoul, South Korea: IEEE, Dec. 2012, pp. 87–94.

[6] C. H. Hoo, Y. Ha, and A. Kumar, "A directional coarse-grained power gated FPGA switch box and power gating aware routing algorithm," in *Proceedings of 23rd International Conference on Field Programmable Logic and Applications (FPL)*. Porto, Portugal: IEEE, Sep. 2013, pp. 1–4.

[7] Z. Seifoori, B. Khaleghi, and H. Asadi, "A power gating switch box architecture in routing network of SRAM-based FPGAs in dark silicon era," in *Proceeding of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Lausanne, Switzerland: IEEE, Mar. 2017, pp. 1342–1347.

[8] T. Tuan and B. Lai, "Leakage power analysis of a 90nm FPGA," in *Proceedings of the Custom Integrated Circuits Conference (CICC)*. San Jose, CA, USA: IEEE, Sep. 2003, pp. 57–60.

[9] V. Degalahal and T. Tuan, "Methodology for high level estimation of FPGA power consumption," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*. New York, NY, USA: ACM, Jan. 2005, pp. 657–660.

[10] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Titan: Enabling large and complex benchmarks in academic CAD," in *Proceedings of the 23rd International Conference on Field programmable Logic and Applications (FPL)*. Porto, Portugal: IEEE, Sep. 2013, pp. 1–8.

[11] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed *et al.*, "VTR 7.0: Next generation architecture and CAD system for FPGAs," *ACM Transactions on Reconfigurable Technology and Systems TRETS*, vol. 7, no. 2, p. 6, Jun. 2014.

[12] C. Chiasson and V. Betz, "COFFE: Fully-automated transistor sizing for FPGAs," in *Proceeding of International Conference on Field-Programmable Technology (FPT)*. Kyoto, Japan: IEEE, Dec. 2013, pp. 34–41.

[13] C. Li, Y. Dong, and T. Watanabe, "New power-aware placement for region-based FPGA architecture combined with dynamic power gating by PCHM," in *Proceedings of the 17th IEEE/ACM International Symposium on Low-Power Electronics and Design (ISLPED)*. Fukuoka, Japan: IEEE Press, Aug. 2011, pp. 223–228.

[14] C. H. Hoo, Y. Ha, and A. Kumar, "A directional coarse-grained power gated FPGA switch box and power gating aware routing algorithm," in *Proceedings of the 23rd International Conference on Field programmable Logic and Applications (FPL)*. Porto, Portugal: IEEE, Sep. 2013, pp. 1–4.

[15] S. Yazdanshenas and H. Asadi, "Fine-grained architecture in dark silicon era for SRAM-based reconfigurable devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 10, pp. 798–802, Aug. 2014.

[16] "Virtex-6 FPGA configurable logic block." User Guide, Xilinx, Feb. 2012.

[17] "Stratix IV device handbook." Handbook, Altera, Jun. 2016.

[18] (2005 (accessed June 15, 2019)) IWLS 2005 benchmarks. [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

[19] R. O. Duda and P. E. Hart, "Pattern classification and scene analysis," *A Wiley-Interscience Publication, New York: Wiley*, 1973.

[20] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, *Advances in knowledge discovery and data mining*. The MIT Press, Mar. 1996.

[21] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[22] A. Gersho and R. M. Gray, *Vector quantization and signal compression*. Springer Science & Business Media, Dec. 2012, vol. 159.

[23] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, May 2005.

[24] S. Sharma and S. Rai, "Genetic k-means algorithm implementation and analysis," *International Journal of Recent Technology and Engineering*, vol. 1, no. 2, pp. 117–120, Jun. 2012.

[25] Y.-C. Chiou and L. W. Lan, "Genetic clustering algorithms," *European Journal of Operational Research*, vol. 135, no. 2, pp. 413–427, Dec. 2001.

[26] P. K. Agarwal and C. M. Procopiuc, "Exact and approximation algorithms for clustering," *Algorithmica*, vol. 33, no. 2, pp. 201–226, Jun. 2002.

[27] S. Arora, P. Raghavan, and S. Rao, "Approximation schemes for Euclidean k-medians and related problems," in *Proceedings of the 30th Annual (ACM) Symposium on Theory of Computing (STOC)*. New York, NY, USA: ACM, May 1998.

[28] S. G. Kolliopoulos and S. Rao, "A nearly linear-time approximation scheme for the Euclidean k-median problem," *(SIAM) Journal on Computing*, vol. 37, no. 3, pp. 757–782, Jul. 2007.

[29] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern Recognition*, vol. 36, no. 2, pp. 451–461, Feb. 2003.

[30] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the 18th Annual (ACM)-(SIAM) Symposium on Discrete Algorithms*. Orlando, FL, USA: Society for Industrial and Applied Mathematics, Jan. 2007, pp. 1027–1035.

[31] M. E. Celebi, H. A. Kingravi, and P. A. Vela, "A comparative study of efficient initialization methods for the K-means clustering algorithm," *Expert Systems with Applications*, vol. 40, no. 1, pp. 200–210, Jan. 2013.

[32] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition letters*, vol. 31, no. 8, pp. 651–666, Jun. 2010.

[33] Z. Cai, M. Heydari, and G. Lin, "Clustering binary oligonucleotide fingerprint vectors for DNA clone classification analysis," *Journal of Combinatorial Optimization*, vol. 9, no. 2, pp. 199–211, Mar. 2005.

[34] Z. Cai, R. Goebel, M. R. Salavatipour, Y. Shi, L. Xu, and G. Lin, "Selecting genes with dissimilar discrimination strength for sample class prediction," in *Proceedings of the 5th Asia-Pacific Bioinformatics Conference (APBC)*. Hong Kong, China: World Scientific, Jan. 2007, pp. 81–90.

[35] Z. Cai, L. Xu, Y. Shi, M. R. Salavatipour, R. Goebel, and G. Lin, "Using gene clustering to identify discriminatory genes with higher classification accuracy," in *Proceedings of 6th IEEE Symposium on BioInformatics and BioEngineering (BIBE)*. Arlington, Virginia: IEEE, Oct. 2006, pp. 235–242.

[36] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, Mar. 1982.

[37] (2013 (accessed June 30, 2019)) Predictive technology model (ptm). [Online]. Available: http://ptm.asu.edu/