Analytical Clustering Score with Application to Postplacement Register Clustering

CHANG XU, GUOJIE LUO, and PEIXIN LI, Peking University
YIYU SHI, University of Notre Dame
IRIS HUI-RU JIANG, National Chiao Tung University

Circuit clustering is usually done through discrete optimizations to enable circuit size reduction or design-specific cluster formation. In this article, we are interested in the register-clustering technique for clock-power reduction by leveraging new opportunities introduced by multibit flip-flop (MBFF). Currently, INTEGRA is the only existing postplacement MBFF clustering optimizer with a subquadratic time complexity. However, it severely degrades the wirelength, especially for realistic designs, which may nullify the benefits of MBFF clustering. In contrast, we formulate an analytical clustering score with a nonlinear programming framework, in which the wirelength objective can be seamlessly integrated and the solver has empirical subquadratic time complexity. With the MBFF library, the application of our analytical clustering method achieves comparable clock power to the state-of-the-art techniques, but further reduces the wirelength by about 25%. Even without the MBFF library, we can still achieve 30% clock wirelength reduction. In addition, the proposed method can potentially be integrated into an in-placement MBFF clustering solver and be applied to other problems that require formulating clustering scores in their objective functions.

CCS Concepts: • **Hardware → Electronic design automation**; **Physical design (EDA)**; *Placement*

Additional Key Words and Phrases: Multibit flip-flops, register clustering, placement, clock power, timing

ACM Reference Format:
Chang Xu, Guojie Luo, Peixin Li, Yiyu Shi, and Iris Hui-Ru Jiang. 2016. Analytical clustering score with application to postplacement register clustering. ACM Trans. Des. Autom. Electron. Syst. 21, 3, Article 41 (May 2016), 18 pages.
DOI: http://dx.doi.org/10.1145/2894753

1. INTRODUCTION

Circuit clustering is a useful technique in electronic design automation. Clustering problems can be classified into two categories. One is for circuit size reduction; a short survey can be found in Yan et al. [2011]. The other is for design-specific cluster formation, such as voltage island grouping [Wu et al. 2007] and register clustering [Cheon et al. 2005; Hou et al. 2009].

This work is partly supported by National Natural Science Foundation of China (NSFC) Grant 61202073, Research Fund for the Doctoral Program of Higher Education of China (MoE/RFDP) Grant 20120001120124, and Beijing Natural Science Foundation (BJNSF) Grant 4142022.
Authors' addresses: C. Xu and P. Li, Center for Energy-efficient Computing and Applications, Peking University, Beijing, 100871; emails: changxu@pku.edu.cn, Pageli328@gmail.com; G. Luo, Center for Energy-efficient Computing and Applications, Peking University; PKU-UCLA Joint Research Institute in Science and Engineering; Collaborative Innovation Center of High Performance Computing, National University of Defense Technology; email: gluo@pku.edu.cn; Y. Shi, Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556; email: yshi4@nd.edu; I. H. Jiang, Department of EE and Institute of Electronics, National Chiao Tung University, Hsinchu City, Taiwan, 300; email: huiru.jiang@gmail.com.

DOI: http://dx.doi.org/10.1145/2894753

In this article, we are interested in the register-clustering problem. The basic idea is to place a group of registers close to each other so that the capacitance of the clock network could be largely reduced, thus the clock switching power could be optimized. There are some previous works performing register clustering during the placement stage. Roughly speaking, Cheon et al. [2005] and Hou et al. [2009] first decided the leaf cluster with a heuristic method (e.g., Quick Clock Tree Synthesis or registers belong to a single vector). Then, they would place registers of the same leaf cluster close to each other either by setting a huge weight of pseudonet or through group bounds control. Usually, this process might be done several times to incrementally adjust the formation of a leaf cluster. Ward et al. [2013] propose a novel latch placement methodology to minimize local clock-tree capacitance.

As technology development improves the driving strength of inverters inside registers, it is now possible to share common inverters in several flip-flops (FFs), resulting in the multibit flip-flop (MBFF). The MBFF clustering problem is to effectively and efficiently merge several single-bit flip-flops (SBFFs) into an MBFF. Compared with traditional register-clustering techniques, the emerging MBFF technique leads to better power reduction. MBFF reduces the clock load by having a single clock input pin, reducing the corresponding wire load. In addition, shared clock inverters within an MBFF further reduce power consumption compared with SBFFs.

Existing works explore MBFF clustering in three design stages: preplacement stage [Cadence 2005; Kretchmer and Logic 2001], in-placement stage [Tsai et al. 2013; Hsu et al. 2013], and postplacement stage. Because more physical information is available after placement, postplacement MBFF clustering has attracted a lot of attention [Yan and Chen 2010; Lin et al. 2011; Wang et al. 2012; Jiang et al. 2011]. Among all these works, INTEGRA [Jiang et al. 2011] delivers the best performance in both power reduction and runtime consumption, and is the only one with subquadratic complexity. However, almost all previous works pay little attention to the signal wirelength, either regarding it as a secondary objective [Yan and Chen 2010; Lin et al. 2011; Wang et al. 2012] or ignoring it altogether [Jiang et al. 2011]. We show later that, for a series of realistic benchmarks, INTEGRA's clustering severely degrades signal wirelength, which may nullify the power benefit of MBFFs.

In this article, we propose an analytical model for the register-clustering problem at the postplacement stage. The overall idea is to generate as many register clusters as possible to maximize power reduction while trying to avoid severe signal wirelength degradation. Specifically, we first propose an exact formulation of the number of clusters based on the Dirac delta function and with another objective in signal wirelength, then smooth it by the Gaussian function. The optimization problem is solved with a well-designed Nonlinear Programming (NLP) solver based on the Nesterov method. In addition, we apply some acceleration techniques, such as customized fast Gauss transformation and a further discrete refinement to make the result practical. Compared with state-of-the-art work, we deliver high-quality register-clustering results with shorter signal wirelength in empirical subquadratic time.

Our clustering flow can be regarded as a postplacement power optimization technique, which can be applied in two scenarios: the emerging postplacement MBFF clustering problem and a general register-clustering problem without an MBFF library. We make the following contributions:

(1) We propose an analytical optimization method, which performs register clustering both effectively and efficiently in subquadratic runtime.
(2) Given an MBFF library, our method shows comparable power reduction to the state-of-the-art INTEGRA work. However, we further reduce signal wirelength by about 25%.

(3) Even without an MBFF library, our method can also reduce a clock tree's wirelength by about 30%. Compared with traditional register-clustering works, our method is much simpler and can guarantee no damage to timing performance.
(4) The proposed method can potentially be integrated in an in-placement MBFF clustering solver, and be applied to other problems that require formulating clustering scores in their objective functions.

The rest of this article is organized as follows. Section 2 formulates our problem of postplacement register clustering. Section 3 introduces our optimization flow, including analytical clustering model and discrete refinement. In Section 4, we describe how to use our register-clustering method for the postplacement MBFF merging problem and evaluate our approach with experimental data and comparisons. In Section 5, we apply this method to the general register-clustering problem. Section 6 contains our conclusions.

2. PROBLEM FORMULATION

Given the following input, we perform register clustering at the postplacement stage under the capacity constraint, density constraint, and timing constraint.

(1) Placement locations of FFs and other cells
(2) Timing slack between a pin and its connected FF
(3) The capacity of the register cluster

Capacity constraint limits the size of a cluster. Suppose that we have an MBFF library of 1b, 2b, and 4b FF, the capacity constraint is set to 4 since there is no larger MBFF to use.

To avoid routing congestion, we consider density constraint when finding locations for register clusters. The basic idea is to divide the placement region into bins. For each bin, the overlapped area between cells and the bin should be controlled by a threshold.

A timing constraint is proposed to prevent postplacement register clustering from violating clock performance achieved after placement. Given timing slacks between pins and the connected FFs, we can calculate the equivalent metal wirelength with the Elmore delay model [Chen and Yan 2010]. Thus, we can find the timing-violation-free distance (**TVFD**) between FF and driving (output) or driven (input) pins. Note that, if there are multiple output pins, we calculate TVFD based on the smallest timing slack. Taking TVFDs into account, for each FF, we can find a feasible region to move without a timing violation, which we label as the timing-violation-free region (**TVFR**) in Figure 1. Consequently, clustering a few SBFFs with overlapped TVFRs can guarantee no damage to timing performance.

The objective is to merge as many clusters as possible to maximize power reduction while avoiding severe wirelength damage.

As shown in Figure 2, we integrate the postplacement register clustering into traditional synthesis flow. After logic synthesis and placement, we get a netlist with SBFF, then cluster several SBFFs together. When given an MBFF library, we will replace such SBFFs' cluster with MBFF. If there is no MBFF library, we will place those SBFFs close to each other, preroute the clock pin for them, and build an artificial register cluster, which we will explain in Section 5. After register clustering, we perform clock-tree synthesis and routing.

3. ANALYTICAL MODEL AND THE OPTIMIZATION FLOW

3.1. The Basic Idea

In this article, we formulate the postplacement register-clustering problem as an optimization problem. The objective function optimizes a weighted sum of the number of

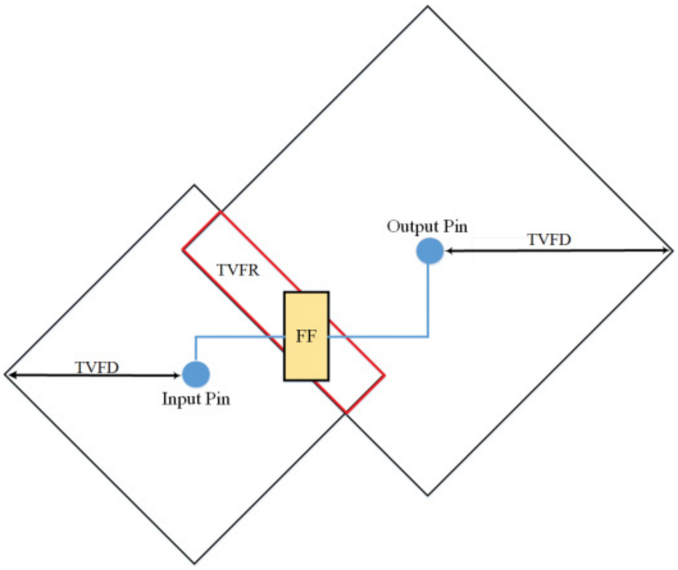

Fig. 1. Concept of TVFD and TVFR.

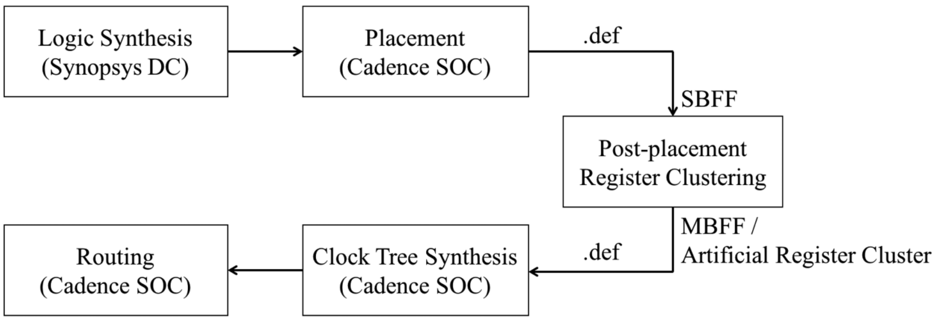
Logic Synthesis (Synopsys DC)
Placement (Cadence SOC)
.def
SBFF
Post-placement Register Clustering
MBFF / Artificial Register Cluster
.def
Clock Tree Synthesis (Cadence SOC)
Routing (Cadence SOC)

Fig. 2. Postplacement register-clustering flow.

FF clusters and signal wirelength. As shown in Figure 3, our flow consists of two steps: analytical optimization and discrete optimization. In the analytical optimization, we carefully define a continuous and differentiable objective function and exploit an effective NLP solver [Lu et al. 2014] to make our method practical. After the analytical optimization, we can get a rough clustering. Then, we invoke the discrete optimization to discretize and further improve the clustering solution. We will explain more details in the following sections.

3.2. Analytical Optimization Step

3.2.1. Definition of Clustering Score. Our objective function trades off the signal wirelength $f_l(\boldsymbol{x}, \boldsymbol{y})$ and cluster number $f_c(\boldsymbol{x}, \boldsymbol{y})$ with parameter α, where $\boldsymbol{x} = (x_1, \ldots, x_N)^T$ and $\boldsymbol{y} = (y_1, \ldots, y_N)^T$ represent coordinates for N FFs. Because of the timing constraint, FFs' locations are bounded by their feasible regions.

$$\begin{aligned} \text{minimize} \quad & \alpha \cdot f_l(\boldsymbol{x}, \boldsymbol{y}) - f_c(\boldsymbol{x}, \boldsymbol{y}) \\ \text{subject to} \quad & t(\boldsymbol{x}, \boldsymbol{y}) \leq T \end{aligned} \qquad (1)$$

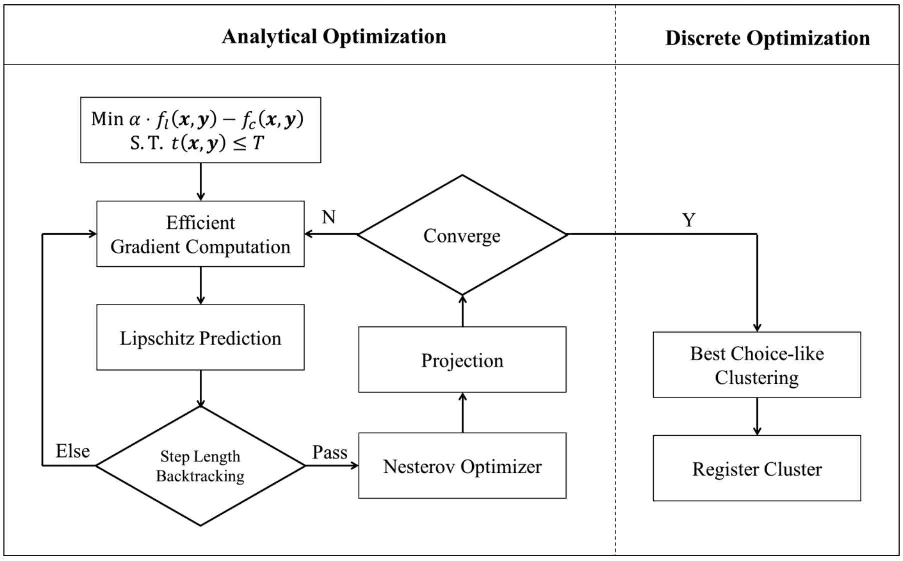
Analytical Optimization
Discrete Optimization
Min $\alpha \cdot f_l(\boldsymbol{x},\boldsymbol{y}) - f_c(\boldsymbol{x},\boldsymbol{y})$
S.T. $t(\boldsymbol{x},\boldsymbol{y}) \leq T$
Efficient Gradient Computation
N
Converge
Y
Lipschitz Prediction
Projection
Best Choice-like Clustering
Else
Step Length Backtracking
Pass
Nesterov Optimizer
Register Cluster

Fig. 3. Optimization flow.

Supposing that M-bit clusters are the most energy-efficient, we want to maximize the number of M-bit clusters. Intuitively, when two FFs have the same locations, they belong to the same cluster. Mathematically, we use the Delta function to evaluate the relationship between two FFs.

As shown in Equation (2), when the two variables are equal, the value of Delta function is one. Otherwise, it is zero. In Equation (3), for two FFs with coordinates of (a, b) and (x_j, y_j), respectively, if their Euclidean distance equals to zero, these two FFs belong to same group, and the Delta function becomes one. For each FF, we can approximate the size of the group it belongs to by checking the accumulative Delta value with other FFs. For example, if $N(a, b)$ equals to 4, we can see that the FF is placed at the same location as three other FFs. They form a 4b cluster together.

$$\delta(\omega, z) = \begin{cases} 1 & (\omega = z) \\ 0 & (\omega \neq z) \end{cases} \tag{2}$$

$$N(a, b) = \sum_{j=1}^{N} \delta(\parallel (a, b) - (x_j, y_j) \parallel, 0) \tag{3}$$

Given the target size M bit, we want to generate as many M-bit clusters as possible. In other words, we hope to maximize the number of registers that belong to M-bit clusters. As shown in Equation (4), minimizing the $-f_c(\boldsymbol{x}, \boldsymbol{y})$ term of the objective function encourages maximizing the number of FFs in M-bit clusters. In the next section, we will demonstrate that our clustering function $(f_c(\boldsymbol{x}, \boldsymbol{y}))$ is capable of generating an attractive force when FFs lie in a small-size (less than M-bit) cluster and vice versa. Consequently, we can maximize the number of M-bit clusters.

$$\min -f_c(\boldsymbol{x}, \boldsymbol{y}) = -\max f_c(\boldsymbol{x}, \boldsymbol{y}) = -\max \sum_{i=1}^{N} \delta(N(x_i, y_i), M) \tag{4}$$

The Delta function is nondifferentiable. In practice, we smooth it with the Gaussian function. As Equation (5) shows, the Gaussian function quickly converges to zero when the difference of ω and z increases to infinity. Parameters ε and d^2 are used to control the degenerate speed. ε is limited between 0 to 1. Thus, we could obtain the property shown in Equation (6). Considering Equation (3), when the distance between two FFs equals to zero, the Gaussian function gets the same value as the Delta function. When their distance is larger than d, the Gaussian function will be less than ε. Consequently, the smaller d and ε are, the more quickly Gaussian function degenerates. We will discuss the effect of parameter d in Appendix C.

Thus, the value of the Gaussian function between two FFs far apart can be neglected. We apply this property to accelerate calculating our objective function.

The signal wirelength $f_l(\boldsymbol{x}, \boldsymbol{y})$ is measured by the total half-perimeter wirelength (HPWL) between locations of FFs and their driving (output) and driven (input) pins. Since HPWL is nondifferentiable, we use the weighted-average approximation in Hsu et al. [2011].

With the smoothing technique, our constraint optimization problem can be solved by NLP solvers. In Appendix A, we will systematically illustrate our efficient and effective NLP solver.

$$\delta(\omega, z) \approx D(\omega, z) = \exp((\omega - z)^2 \ln \varepsilon / d^2) \tag{5}$$

$$\begin{cases} D(\omega, z) = 1 & \text{when} \quad w = z \\ D(\omega, z) < \varepsilon & \text{when} \quad \mid \omega - z \mid > d \end{cases} \tag{6}$$

3.2.2. Insights of the Clustering Function. In our formulation, we assume that M-bit clusters are the most energy-efficient and we want to maximize the number of M-bit clusters. The following theoretical analysis proves that our formulation has a great property of generating both attractive forces and repelling forces based on different cluster formations. Clusters with more than M FFs are pushed apart so that some extra FFs can move farther away and enable the formation of multiple M-bit clusters.

PROPERTY 1. *FF_i contributes a term $f_{c,i} = \delta(N(x_i, y_i), M)$ to the clustering score f_c. When FF_i lies in an undersized ($N(x_i, y_i) < M$) cluster, maximizing this term results in attractive forces for the neighboring FFs of FF_i. When FF_i lies in an oversized ($N(x_i, y_i) > M$) cluster, maximizing this term results in repelling forces for the neighboring FFs of FF_i.*

PROOF. The force direction of FF_j resulting from $f_{c,i}$ towards can be detected by checking the sign of Equation (7). Here, we consider only the x-direction without loss of generality.

$$\begin{cases} \text{FF}_i \text{ attracts FF}_j & \text{when } (x_i - x_j) \cdot \frac{\partial f_{c,i}}{\partial x_j} > 0 \\ \text{FF}_i \text{ repels FF}_j & \text{when } (x_i - x_j) \cdot \frac{\partial f_{c,i}}{\partial x_j} < 0 \end{cases} \tag{7}$$

When smoothing with the Gaussian function, the calculation of the partial derivative is shown in Equations (8), (9), and (10).

$$\frac{\partial f_{c,i}}{\partial x_j} = \frac{\partial f_{c,i}}{\partial N(x_i, y_i)} \cdot \frac{\partial N(x_i, y_i)}{\partial x_j} \tag{8}$$

$$\frac{\partial f_{c,i}}{\partial N(x_i, y_i)} = 2\lambda(N(x_i, y_i) - M) \exp((N(x_i, y_i) - M)^2 \lambda) \tag{9}$$

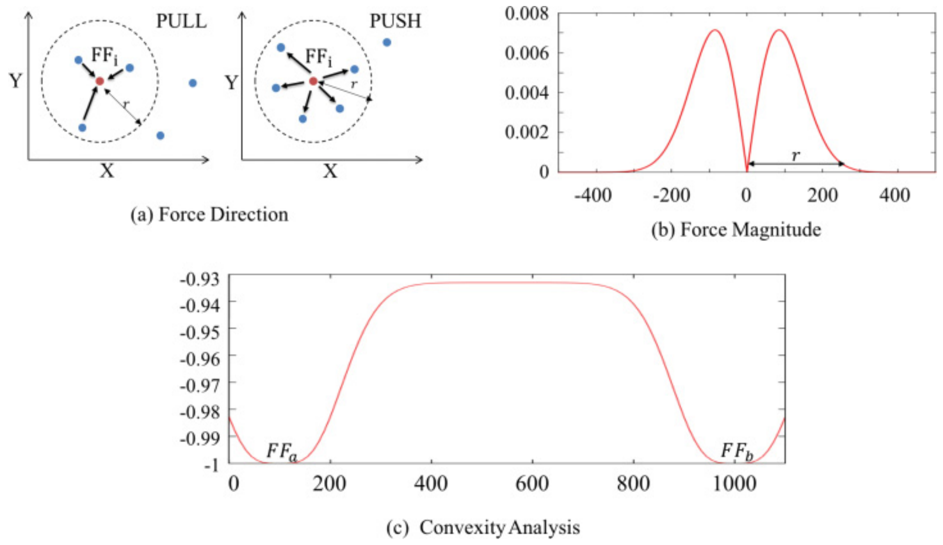

Fig. 4. Analysis of force direction, magnitude, and convexity of clustering function.

$$\frac{\partial N(x_i, y_i)}{\partial x_j} = 2\lambda \exp(\lambda((x_i - x_j)^2 + (y_i - y_j)^2))(x_j - x_i) \tag{10}$$

Note that λ equals to $\ln \varepsilon/d^2$ and ε ranges from 0 to 1. Thus, the force direction is related only to N_i, which is shown in Equation (11):

$$\begin{cases} (x_i - x_j) \cdot \frac{\partial f_{c,i}}{\partial x_j} > 0 & \text{when } N(x_i, y_i) < M \\ (x_i - x_j) \cdot \frac{\partial f_{c,i}}{\partial x_j} < 0 & \text{when } N(x_i, y_i) > M \end{cases} \quad \square \tag{11}$$

In Figure 4, we illustrate the force direction and strength from FF$_i$ toward other FFs and the convexity of the clustering function. As is shown in Figure 4(a), when FF$_i$ is adjacent to less than $M - 1$ FFs, FF$_i$ will attract more FFs to form a cluster of size M; when there are more than $M - 1$ neighbors, FF$_i$ will repel the extra FFs. Thus, it guarantees the maximization of M-bit clusters' number. Since the $f_{c,i}$ part contains Exp function, which degenerates quickly, FF$_i$ will only affect other FFs within a distance of r in Figure 4(a).

Here, we take a specific case as an example to illustrate the force strength from FF$_i$, where FF$_i$ locates at $(0, 0)$ point, ε is set to 0.5, and d is set to 100. In Figure 4(b), where the x-axis represents the change of x_j of another FF and y-axis indicates the value of $\partial f_{c,i}/\partial(x_j)$, we can see that FF$_i$ only affects other FFs in the neighborhood. We consider this feature to accelerate the gradient calculation with fast Gauss transformation (FGT) in Appendix B.

From the previous analysis, the f_l term of the objective function can pull FFs toward their "optimal locations" in terms of signal wirelength, while the f_c term of the objective function can effectively cluster FFs in the neighborhood into as many M-bit groups as possible. Thus, our objective function achieves high-quality results in both signal wirelength and clustering ratio.

In fact, our clustering function is not convex. Suppose that, in the following circumstance, for FF_i, its neighboring FFs are FF_a and FF_b, and they are aligned horizontally.

The value of the $f_{c,i}$ term changes with different locations of FF_i, which is illustrated in Figure 4(c). Although it is not convex, we will show its effectiveness in Section 4.

3.3. Discrete Optimization Step

In analytical optimization, we perform register clustering from a global scope, which considers both clock power and signal wirelength. Then, we apply discrete optimization in Algorithm 1 to complete the conversion from a continuous solution to the final discrete solution.

The discrete optimization flow looks like a two-pass, best-choice clustering [Alpert et al. 2005]. In the first pass (Lines 6–11), we extract partial clusters by bottom-up clustering based on the proximity relation after the analytical solution. In this step, the size of clusters cannot exceed cluster capacity (i.e., M-bit in our formulation).

Then, we perform further refinement to improve the ratio of M-bit clusters in the second pass (in Lines 12–20). In this step, we temporally form clusters that exceed the cluster capacity and kick out the extra FFs immediately based on the heuristic score. Note that we check the **timing constraint** during cluster formation to guarantee that FFs in the same cluster have overlapped feasible regions.

ALGORITHM 1: Discrete Optimization

```
   Input: 1) The initial FFs' locations based on analytical solution
   2) The feasible region for each FF
   Output: MBFF clustering result
   // Preprocessing
1  Construct Bin-Structure for nearest neighbor searching;
2  Search nearest neighbors on Bin-Structure and Insert tuples into Priority Queue;
   // Clustering & Refinement
3  $Pass = 1$;
4  while $Pass <= 2$ do
5      Pop top tuple($FF_i, FF_j, d$) from PQ;
6      if $Pass == 1$ then
7          if $|Group(FF_i) \bigcup Group(FF_j)| < M$ then
8              if satisfy timing constraint then
9                  Merge Group(FF$_i$) and Group(FF$_j$);
10             end
11         end
12     else
13         if $|Group(FF_i) \bigcup Group(FF_j)| < M$ then
14             if satisfy timing constraint then
15                 Merge Group(FF$_i$) and Group(FF$_j$);
16             end
17         else
18             Merge Group(FF$_i$) and Group(FF$_j$) and kick out excess FFs;
19         end
20     end
21     if PQ.empty && Pass==1 then
22         Remove the M-bit groups and insert the tuples for nearest neighbors for the
           remaining FFs;
23     end
24     $Pass++$;
25 end
```

Lines 1 and 2 extract the proximity relation based on the analytical solution. We calculate distances between FF pairs. For the purpose of maintaining a proximity

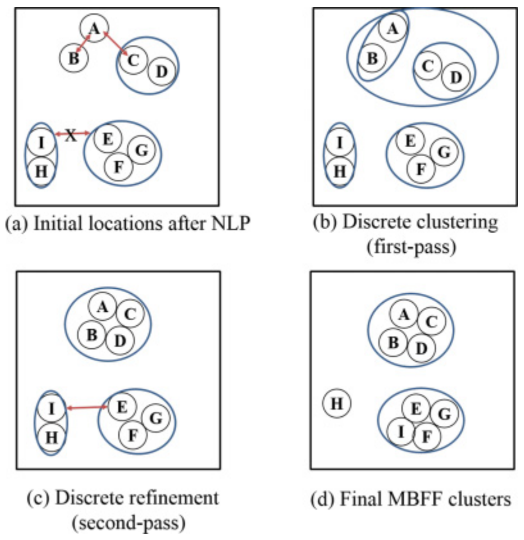
A
B
C
D
X
I
H
E
G
F
(a) Initial locations after NLP
(b) Discrete clustering (first-pass)
(c) Discrete refinement (second-pass)
(d) Final MBFF clusters

Fig. 5. Discrete optimization.

relation, the FF pair with small distance will be manipulated with high priority during the subsequent cluster formation. To identify the globally nearest FF pair, we insert tuple($\mathrm{FF}_i, \mathrm{FF}_j, d$) into a priority queue (PQ) with the distance d between FF_i and FF_j as the sort key. As we have explained in the analytical step, to avoid signal wirelength damage, we only cluster FFs in the neighborhood. Thus, it is not necessary to calculate the distance between every two FFs. In our implementation, we divide the whole region into bins and only compute the distance between FF pairs in the adjacent bins. Thus, calculating the distance of the promising FF pairs can be done in $O(N)$ time.

Lines 6 to 11 contain the clustering step (first pass). After the preprocessing stage, we pick up the top tuple($\mathrm{FF}_i, \mathrm{FF}_j, d$) from the PQ. If merging the groups that FF_i and FF_j belong to does not violate the capacity constraint and timing constraint, we commit the merge. Supposing that the cluster capacity is set to be 4 and the top two tuples are (A, B, d_1) and (A, C, d_2) (as in Figure 5(a)), after checking the capacity constraint, we will first merge A and B into a group of two. Then, the group of AB and the group of CD will be clustered into a 4b group. However, the group of HI and group of EFG cannot be merged because of capacity constraint. Figure 5(b) shows the group information after the clustering step.

Lines 12 to 20 contain the refinement step (second pass). In this step, we allow the clustering of two groups to exceed M-bit temporarily, then remove the excess FFs with a heuristic score. Given a cluster formation, the heuristic score can estimate the HPWL of the connecting signal nets based on the connecting pins' locations. We enumerate the possible removal combinations and choose one cluster formation with the smallest estimated HPWL. In Figure 5(c), we pick up the top tuple, (I, E, d_3), from PQ. Since merging the two groups results in a group size of 5, we remove one FF with the largest signal wirelength. Figure 5(d) shows the final FFs groups' formation.

4. APPLICATIONS: POSTPLACEMENT MBFF CLUSTERING

4.1. Previous Works and Limitations

Previous works manipulate postplacement MBFF clustering with discrete methods. To satisfy the timing constraint, they build an intersection graph [Yan and Chen 2010; Lin et al. 2011; Wang et al. 2012] or interval graph [Jiang et al. 2011] and search the

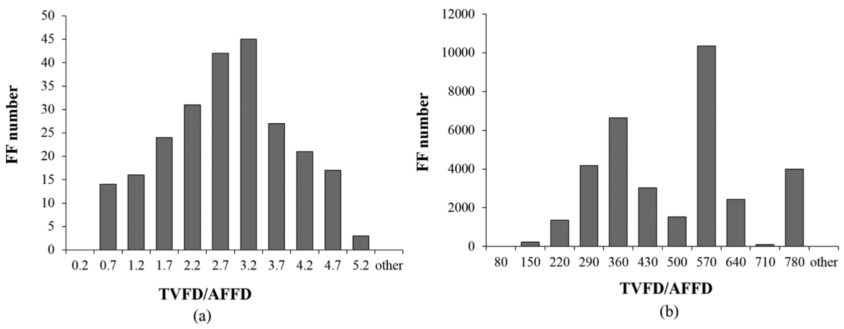

Fig. 6. The histogram of TVFD/AFFD for benchmark (a) C1 and (b) realistic design Vga.

MBFF candidates on these graphs. Among all previous works, INTEGRA shows the best performance in terms of power reduction and runtime complexity. It is the only one with subquadratic time complexity.

However, we find that the efficient runtime of INTEGRA partially benefits from the simple strategy of choosing MBFF candidates without considering the impact on signal wirelength in their objective. C1-C6 [Lin et al. 2011] benchmarks suite is widely used in evaluating MBFF clustering performance. Although the wirelength degradation on C1-C6 is acceptable (around 3%), we analyze in the following section that a series of realistic designs are quite different from C1-C6. INTEGRA's method will cause huge wirelength damage for these realistic designs.

To illustrate the difference, we define the average FF distance (AFFD) as $\sqrt{\text{ChipArea}/\#\text{FF}}$. It is obvious that the average distance between every pair of nearest FFs is approximately AFFD, assuming all the FFs are evenly placed. Then, TVFD/AFFD is calculated for every FF to estimate roughly how many FFs can be covered within the range of TVFD.

As is shown in Figure 6(a), the histogram of TVFD/AFFD for the benchmark C1 indicates that the range of TVFD is limited. The majority of FFs can reach only three other FFs, on average, within the range of TVFD.

Interestingly, all other benchmarks, C2 to C6, follow the same distributions as C1, even though the number of FFs is different. Moreover, if we look at the number of FFs and the placement region of these benchmarks, they scale in the same ratio. These facts indicate that the benchmarks C1 to C6 can represent only a single kind of circuit. They are not sufficient to evaluate the performance of MBFF-clustering algorithms.

Similarly, we pick a realistic design, vga, from the IWLS benchmark suite, and synthesize it by Synopsys DC and Cadence SOC Encounter with a tight timing constraint. Specifically, the worst negative slack (WNS) reported after placement is 0.108 ns with a clock cycle time of 3.5ns and 2.5ns for the two clock domains. Figure 6(b) reveals the TVFD/AFFD distribution.

Comparing Figures 6(a) and 6(b), we can quickly find that TVFD in realistic designs is two orders of magnitude larger than TVFD in C1 to C6. Realistic designs have much greater freedom of choosing clustering candidates. However, INTEGRA does not consider signal wirelength in their objective. Thus, INTEGRA damages the signal wirelength a lot for realistic designs. For example, the degradation of wirelength for ethernet is up to 20 times of the original wirelength.

4.2. Evaluation

We implement our two-step method of postplacement register clustering in C++, and evaluate it on an Intel Xeon machine with 16 logical threads. The MBFF library that

Table I. The Power and Area of Our MBFF Library

Bit number	Normalized power per bit	Normalized area per bit
1	1.00	1.00
2	0.86	0.96
4	0.78	0.71

Table II. Realistic Design Property

Circuit	#FF	WNS (ns)
tv80	359	0.015
wbconmax	770	0.038
paring	1338	0.094
dma	1816	0.109
ac97	2191	0.078
ethernet	10443	0.046

Table III. Comparisons Between INTEGRA and Our Method on C1 to C6

	INTEGRA			Ours			
Circuit	PWR	WLR	RT-all (s)	PWR	WLR	RT-NLP(s)	RT-all (s)
C1	82.80	96	0.01	83.50	77.40	0.42	0.42
C2	80.90	102	0.01	82.30	76.40	0.96	0.96
C3	80.80	104	0.01	82.30	74.90	3.11	3.14
C4	81.00	104	0.02	82.40	75.60	10.45	10.59
C5	80.70	105	0.05	82.10	76.40	15.98	16.66
C6	80.70	105	1.11	82.30	82.00	197.91	217.41
Avg.	1.00	1.33	1.00	1.02	1.00	244.88	251.83

we used is provided by the Faraday Company based on the UMC 55nm process. As shown in Table I, 4b FF is most efficient in terms of both power and area per bit. We evaluate our method on widely used C1 to C6 [Lin et al. 2011] benchmarks and realistic designs from the IWLS-2005 suite [Albrecht 2005]. As shown in Table II, we pick up six benchmarks from the IWLS-2005 suite and synthesize with Synopsys DC and Cadence SOC Encounter with the UMC 55nm process. The WNS can be seen from Table II.

4.2.1. Results on the Widely Used C1 to C6. We compare our results with the state-of-the-art method INTEGRA in Table III. "PWR," "WLR," "#iter-NLP," "RT-NLP," and "RT-all" represent power reduction, wirelength reduction, iterations in NLP, the runtime of NLP, and total runtime, respectively. Table III shows that our method achieves almost the same power reduction, and further reduces the wirelength by about **33%** compared with INTEGRA. The only drawback is runtime. However, the runtime is practical for realistic circuits; we show that its time complexity is subquadratic in Appendix D.

4.2.2. Results on the Realistic Designs. We use Synopsys DC and Cadence Encounter SOC to synthesize realistic designs with tight timing constraint. Properties of these benchmarks can be found in Table II, in which "#FF" represents the number of FFs in the circuit and "WNS" denotes worst negative timing slack after placement. Then, the FFs and slacks information will feed in INTEGRA to do MBFF clustering.

From the "WLR" in Table IV, we find that INTEGRA damages the wirelength about **932%**, on average. According to our previous analysis, the damage of signal wirelength is mainly due to the large timing-feasible region. To fix this defect, we shrink the TVFD with a bound factor to limit the movement of FFs for INTEGRA, which we call Bound-INTEGRA. Figure 7 compares the impacts of different bound factors to power ratio and

Table IV. Comparisons Between INTEGRA and Our Method on Realistic Designs

	INTEGRA			Bound-INTEGRA			Ours			
Circuit	PWR	WLR	RT(s)	PWR	WLR	RT(s)	PWR	WLR	RT-NLP	RT-all(s)
ac97	78.02	673.28	0.09	78.02	120.00	0.02	78.02	95.71	4.74	4.88
dma	78.03	798.29	0.06	78.04	124.71	0.05	78.02	96.00	5.39	5.43
ethernet	78.00	2038.71	1.61	78.00	216.51	0.63	78.00	87.92	20.81	24.51
pairing	78.00	931.45	0.05	78.00	132.17	0.03	78.00	109.00	6.51	6.61
tv80	78.11	350.46	0.01	78.11	109.20	0.01	78.10	95.71	0.93	0.94
wbconmax	78.02	540.81	0.02	78.26	128.00	0.03	78.02	105.00	2.29	2.30
Avg.	1.00	9.32	1.00	1.00	1.43	0.76	0.99	1.00	82.19	83.52

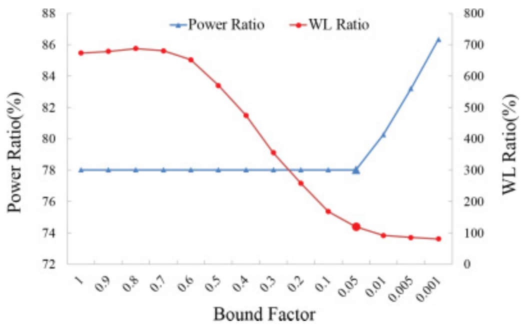

Fig. 7. Effect of different bound factors to power ratio and WL ratio.

WL ratio. We choose the one that can achievethe best WL ratio when guaranteeing the best power reduction in our implementation. For example, in Figure 7, the best bound factor is 0.05.

As shown in Table IV, Bound-INTEGRA can achieve much better signal wirelength. Even compared with Bound-INTEGRA, we can still obtain **43%** wirelength improvement, on average. In addition, our power reduction is comparable with INTEGRA. Although our runtime is longer than INTEGRA's, it is acceptable in practice.

5. APPLICATIONS: POSTPLACEMENT REGISTER CLUSTERING WITHOUT MBFF LIBRARY

5.1. Artificial Multibit Register Cluster

If there is no MBFF library, we can still use our method to optimize clock power at the postplacement stage and guarantee no timing violation. We can place several SBFFs together, route their clock pin in advance, and support one external pin for conventional clock-tree synthesis. Although we cannot benefit from capacitance reduction of the clock pin as MBFF library, we can still simplify the clock network and reduce the clock wire. As shown in Figure 8, by rotating and placing registers close to each other, we can build 2b and 4b clusters. The external clock pin that we support is labeled with "CK_out." We call this register group the "artificial multibit register cluster." In the following section, we will evaluate the impact on clock trees with the artificial multibit register cluster.

5.2. Evaluation

With our optimization method, we replace several SBFFs with artificial multibit register clusters and evaluate the clustering result on benchmark suite IWLS 2005. We synthesize the benchmarks using the Synopsys DC and Cadence SOC encounters. Refer to Figure 2 for the whole clustering flow. In our evaluation, we support the 1b cluster, 2b cluster, and 4b cluster.

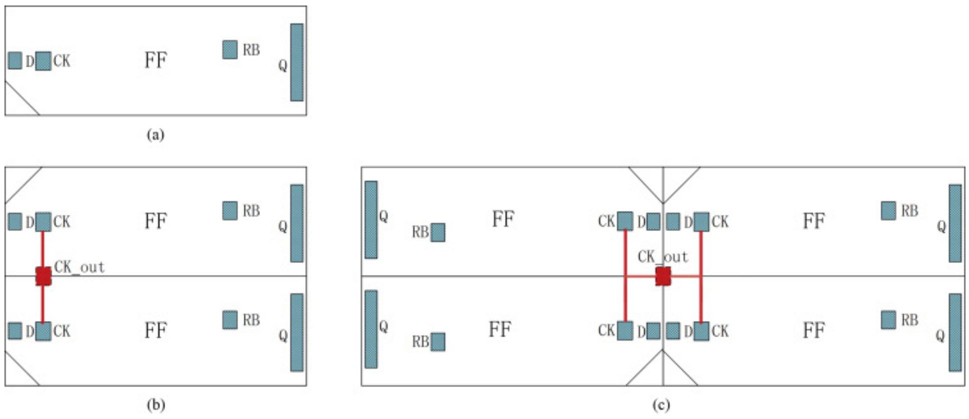

Fig. 8. Artificial multibit register cluster (a) 1-bit, (b) 2-bit, and (c) 4-bit.

Table V. Performance of Postplacement Register Clustering

Circuit	Method	# Sinks	Clk skew (ps)	Clk WL (um)	Clk switch pow (mW)	Net switch pow (mW)	Total pow (mW)
ac97	ordinary	1876	28.20	8758.73	2.13	2.97	11.26
	clustering	498	24.90	5709.18	1.17	2.10	10.62
	impr(%)	73	12	35	45	29	6
dma	ordinary	1867	32.30	8367.98	0.11	0.22	0.80
	clustering	507	15.00	5633.93	0.06	0.17	0.74
	impr(%)	73	54	33	47	24	8
ethernet	ordinary	10016	70.50	46731.54	3.98	6.01	19.34
	clustering	2888	86.40	33371.25	2.35	3.90	17.11
	impr(%)	71	−23	29	41	35	12
pairing	ordinary	1338	27.10	6496.97	0.52	1.13	3.84
	clustering	336	18.80	4077.24	0.26	0.87	3.52
	impr(%)	75	31	37	49	23	8
tv80	ordinary	359	22.00	1682.20	0.18	0.70	1.60
	clustering	91	4.10	1116.93	0.11	0.54	1.33
	impr(%)	75	81	34	37	22	17
wbconmax	ordinary	770	17.40	3739.13	0.20	2.48	6.20
	clustering	289	31.40	2600.11	0.14	2.75	7.04
	impr(%)	62	−80	30	30	−11	−14
Avg.	impr(%)	72	13	33	42	20	6

As shown in Table V, we compare our clustering method with the ordinary method, in which no register clustering is executed. "# Sinks,","Clk skew,","Clk WL,","Clk switch pow,", "Net switch pow," and "Total pow" represent the sinks of the clock tree, trigger edge skew, the clock tree's routing wirelength, the switching power of the clock tree, the switching power of all the nets, and the total power, respectively. We use the default average switching activity, which is 0.2 for Cadence, to obtain switching power.

As we have explained previously, we first connect registers inside the cluster before clock-tree synthesis and support one external clock pin. Consequently, the clock sinks shown are significantly reduced. Since the maximum cluster size is 4b, theoretical sinks reduction can be up to 75%. Due to the decrease of clock sinks, clock-tree synthesis becomes simple. Clock wirelength is also optimized. As Table V shows, the clock wirelength reduction is around **30%**. In terms of clock skew, it is not surprising to find that, in most conditions, skews of clustering flow are better than ordinary flow owing to the simplicity of the clock network. For some cases, however, clock skews become worse. One possible explanation is that Cadence SOC might stop optimizing when the clock skew value is acceptable. We have found in the user manual that the default maximum skew is 300ps.

We also check the power performance of clustering flow. As shown in Table V, the switching power of a clock network can be optimized by around **40%** because of capacitance reduction brought about by the decrease of clock wirelength. For most cases, net switching power and total power can also be reduced. An exception case is wbconmax. From Table IV, we can see that, for wbconmax, the signal wirelength after register clustering with our flow is degraded by about 5% ("WLR" for wbconmax is 105, which indicates the ratio of clustering wirelength over ordinary wirelength without clustering). Thus, the increase of net switching power probably results from the increase of signal wirelength.

6. CONCLUSION

In this article, we propose an analytical model for register clustering. The overall idea is first to propose a function that computes the number of clusters. The definition of this function relates to the nondifferentiable Dirac delta function. To make it compatible with the nonlinear program method, we smooth it using the Gaussian function. We can compute the naïve quadratic-time evaluation of the Gauss summation using the Figtree library for fast Gauss transformation. However, this general implementation still does not meet the requirement of extensive evaluations in an NLP solver. Thus, we implement a customized fast Gauss transformation using a multithreading technique. The final runtime of this approach is practical, and the overall runtime is empirically subquadratic.

We use our postplacement register-clustering method as a power optimization technique and apply it to two scenarios: (1) register clustering with an MBFF library, and (2) traditional register clustering without an MBFF library. Compared with other postplacement MBFF clustering methods, our method achieves comparable power reduction, but further reduces the wirelength by 25%. Even without an MBFF library, we can still reduce the clock tree's wirelength by about 30%.

The proposed method can potentially be integrated into an in-placement MBFF clustering solver, and be applied to other problems that require formulating the clustering scores in their objective functions.

APPENDIX

A. NONLINEAR PROGRAMMING SOLVER

As shown in Section 3.2, we formulate an analytical objective function in the constrained nonlinear programming model. Instead of using the penalty methods, we apply a projected gradient method [Rosen 1960] to handle the timing constraints.

In this work, we use Nesterov's method [Lu et al. 2014] to choose the step size to minimize the objective function efficiently. In contrast to the traditional line-search method, which is quite time-consuming, Nesterov's method leverages the Lipschitz constant to estimate the step size. As Nesterov [1983] shows, $\alpha_k = L^{-1}$ satisfies the step-length requirement, where α_k is the step size and L is the Lipschitz constant. Similar to Lu et al. [2014], we approximate the Lipschitz constant instead of precisely

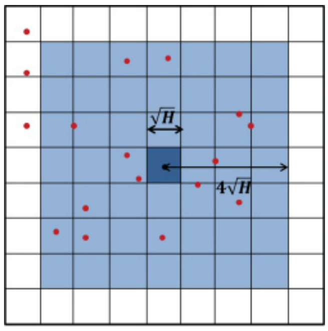

Fig. 9. Explanation of fast Gauss transformation (FGT).

ALGORITHM 2: Projected Nesterov Method

```
   Input: $\alpha_k$, $\mu_k$, $v_k$, $v_{k-1}$, $\nabla f(v_k)$, $\nabla f(v_{k-1})$
   Output: $\mu_{k+1}$, $v_{k+1}$, $\alpha_{k+1}$
1  $\alpha_k = BackTrack(v_k, v_{k-1}, \nabla f(v_k), \nabla f(v_{k-1}))$;
2  $\mu_{k+1} = v_k - \alpha_k \nabla f(v_k)$;
3  $\alpha_{k+1} = (1 + \sqrt{4\alpha_k^2 + 1}/2)$;
4  $v_{k+1} = \mu_{k+1} + (\alpha_{k-1})(\mu_{k+1} - \mu_k)/\alpha_{k+1}$;
5  $v_{k+1} = Project(v_{k+1})$;
```

calculating it. As shown in Algorithm 2, we use the backtrack method to further refine the step size at Line 1.

The solution after one step of Nesterov's method, at Line 4, may violate the timing constraint. Therefore, we apply a projection step in Line 5 to find the closest solution in the feasible space to replace the intermediate infeasible solution. Since the feasible solution space with respect to the timing constraint is a convex space, the projection is straightforward to implement. The feasible region of a flip-flop is a circle with Manhattan distance, and it becomes a rectangle after rotating the coordinates by 45 degrees. Thus, the projection can be done by rotating the coordinates by 45 degrees first, and then project the location of a timing-infeasible flip-flop into its rotated rectangular timing-feasible region.

B. CUSTOMIZED FAST GAUSS TRANSFORMATION

The computation of Equation (4), in the objective function, is expensive. A direct computation requires $O(N^2)$ complexity for all the N flip-flops. We use the fast Gauss transformation (FGT) [Odlyzko and Schönhage 1988] method to reduce the computation complexity to $O(N)$.

Figure 9 demonstrates the basic idea of the FGT. Supposing that FF$_i$ is located in a square with side length $\sqrt{H}$, which equals to $\sqrt{-d^2/\ln(\varepsilon)}$ in our problem, as demonstrated in Odlyzko and Schönhage [1988], the evaluation of exponential function with FFs within side length of $4\sqrt{H}$ is satisfied with four digits of accuracy. Consequently, instead of acquiring the precise value, we approximate gradient calculation by evaluating the objective function only with FFs in the neighborhood.

However, even using the state-of-the-art FGT library Figtree [Morariu et al. 2009], the runtime is still too slow due to the overhead of maintaining the essential data structure KD-Tree, which is more suitable for high-dimensional data. Thus, we

Table VI. Comparison of Figtree and Our Customized FGT

#FF	NLP solver with Figtree for FGT(s)	NLP solver with customized FGT(s)	SpeedUp
120	1.25	0.72	1.74
480	5.80	0.52	11.15
1920	31.43	1.59	19.76
5880	143.33	4.20	34.13
12000	413.45	7.92	52.20
192000	7817.23	207.68	34.64
Avg.	-	-	**25.60**

implemented a customized multithreaded FGT solver with an efficient bin structure instead of the KD-Tree structure for our two-dimensional data.

In our bin structure, the whole chip is partitioned into a mesh of bins. For each bin, we record a list of FFs belonging to it. By querying adjacent bins, we can quickly obtain FFs in the neighborhood. In addition, this access pattern can be easily paralleled with OpenMP. Although side length will grow as the parameter d increases, which may involve more cells during gradient calculation, in our implementation, we control the value of parameter d to avoid the extreme condition and significant damage to signal wirelength. Refer to Appendix C for the details of parameter tuning.

Based on this bin structure, statistics in Table VI show that time complexity is linear, and the practical runtime is fast. In addition, compared with Figtree-based implementation, our bin-structure-based implementation can achieve **25.6X** speedup, on average, for different amounts of FFs in benchmark C1 to C6.

C. PARAMETER TUNING

The value of α (in Equation (1)) is set to balance the quantities of the f_l part and f_c part. In fact, the magnitudes of f_l and f_c are positively related to $\text{Chip}_{\text{width}}$ and the total number of FFs (N), respectively. Thus, α is set as Equation (12).

$$\alpha = \frac{N}{\text{Chip}_{\text{width}}} \tag{12}$$

The value of ε and d (in Equation (5)) are critical for performance. If d is too small, FFs do not have enough force to affect others. However, if d is too big, each FF can influence too many FFs. Finally, FFs will reach their "balanced" states under the effects of many forces. We want to set a proper value for d to only affect a few FFs in the neighborhood. We observe the best performance when we set d as the average distance between every second-nearest FF pairs. Note that to maintain subquadratic runtime complexity, for each FF, we obtain only the second-nearest FF pair in the neighborhood. If an FF does not have the second-nearest FF, we do not consider that FF during the calculation of parameter d. In our implementation, ε is set to 0.5, meaning that the clustering score between two FFs is less than 0.5 when their distance exceeds d.

D. TIME COMPLEXITY ANALYSIS

Our method consists of two steps: the analytical optimization step and the discrete optimization step.

Analytical optimization requires evaluating the derivative of the objective function. The derivative evaluation of wirelength part is $O(N)$. Although initial register distribution affects runtime performance, for the roughly uniform distribution, the derivative evaluation of the clustering score is $O(N)$ with the customized FGT accelerating technique in Appendix B. Though it is difficult to analyze how many iterations it takes for the NLP solver, experience shows that the quadratic placement problem has a time complexity of $O(N^{1.18})$ [Spindler et al. 2008]. In addition, the state-of-the-art quadratic placer simPL [Kim et al. 2012] and nonlinear placer ePlace [Lu et al. 2014] show a comparable runtime performance. We believe that our placement-like NLP problem has subquadratic complexity.

For discrete optimization, the most time-consuming part is the pairwise distance calculation at Lines 2 and 22 in Algorithm 1. We partition the placement region into bins; the searching process will only be performed in the neighborhood and delivers $O(N)$ time.

Experimental results in Table II and Table III show a subquadratic performance. Therefore, our method shows empirical subquadratic time complexity.

ACKNOWLEDGMENTS

The author would like to thank Tao Wang from MST and Chih-Long Chang from NCTU for their help with using Cadence and INTEGRA, respectively.

REFERENCES

Christoph Albrecht. 2005. IWLS 2005 benchmarks. Retrieved March 24, 2016 from http://iwls/org/2005/benchmark_presentation.pdf.

Charles Alpert, Andrew Kahng, Gi-Joon Nam, Sherief Reda, and Paul Villarrubia. 2005. A semi-persistent clustering technique for VLSI circuit placement. In *Proceedings of the 2005 International Symposium on Physical Design (ISPD'05)*. ACM, New York, NY, 200–207.

Cadence. 2005. Cadence Encounter user manual. http://www.ece.utep.edu/courses/web5375/Links_files/4.1.pdf.

Zhi-Wei Chen and Jin-Tai Yan. 2010. Routability-driven flip-flop merging process for clock power reduction. In *2010 IEEE International Conference on Computer Design (ICCD'10)*. 203–208.

Yongseok Cheon, Pei-Hsin Ho, Andrew B. Kahng, Sherief Reda, and Qinke Wang. 2005. Power-aware placement. In *Proceedings of the 2005 Annual Design Automation Conference (DAC'05)*. ACM, New York, NY, 795–800.

Wenting Hou, D. Liu, and Pei-Hsin Ho. 2009. Automatic register banking for low-power clock trees. In *Quality of Electronic Design, 2009 (ISQED'09). Quality Electronic Design*. 647–652.

Chih-Cheng Hsu, Yu-Chuan Chen, and M. P.-H. Lin. 2013. In-placement clock-tree aware multi-bit flip-flop generation for power optimization. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'13)*. 592–598.

Meng-Kai Hsu, Yao-Wen Chang, and V. Balabanov. 2011. TSV-aware analytical placement for 3d IC designs. In *48th ACM/EDAC/IEEE Design Automation Conference (DAC'11)*. 664–669.

Iris Hui-Ru Jiang, Chih-Long Chang, Yu-Ming Yang, Evan Y.-W. Tsai, and Lancer S.-F. Chen. 2011. INTEGRA: Fast multi-bit flip-flop clustering for clock power saving based on interval graphs. In *Proceedings of the 2011 International Symposium on Physical Design (ISPD'11)*. ACM, New York, NY, 115–122.

Myung-Chul Kim, Dong-Jin Lee, and Igor L. Markov. 2012. SimPL: An effective placement algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 1, 50–60.

Yaron Kretchmer and LSI Logic. 2001. Using multi-bit register inference to save area and power: The good, the bad, and the ugly. *EE Times Asia*.

M. P.-H. Lin, Chih-Cheng Hsu, and Yao-Tsung Chang. 2011. Post-placement power optimization with multi-bit flip-flops. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 12, 1870–1882.

Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis J.-H. Huang, Chin-Chi Teng, and Chung-Kuan Cheng. 2014. ePlace: Electrostatics based placement using Nesterov's method. In *Proceedings of the 51st Annual Design Automation Conference on Design Automation Conference (DAC'14)*. ACM, New York, NY, Article 121, 6 pages.

Vlad I. Morariu, Balaji V. Srinivasan, Vikas C. Raykar, Ramani Duraiswami, and Larry S. Davis. 2009. Automatic online tuning for fast Gaussian summation. In *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou (Eds.). Curran Associates, Inc., Red Hook, NY. 1113–1120.

Yurii Nesterov. 1983. A method of solving a convex programming problem with convergence rate O(1/k2). In *Soviet Mathematics Doklady*, Vol. 27. 372–376.

Andrew M. Odlyzko and Arnold Schönhage. 1988. Fast algorithms for multiple evaluations of the Riemann zeta function. *Transactions of the American Mathematical Society* 309, 2, 797–809.

Jo Bo Rosen. 1960. The gradient projection method for nonlinear programming. Part I. Linear constraints. *Journal of the Society for Industrial and Applied Mathematics* 8, 1, 181–217.

P. Spindler, U. Schlichtmann, and F. M. Johannes. 2008. A fast force-directed quadratic placement approach using an accurate net model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 8, 1398–1411.

Chang-Cheng Tsai, Yiyu Shi, Guojie Luo, and Iris Hui-Ru Jiang. 2013. FF-bond: Multi-bit flip-flop bonding at placement. In *Proceedings of the 2013 ACM International Symposium on International Symposium on Physical Design (ISPD'13)*. ACM, New York, NY, 147–153.

Shao-Huan Wang, Yu-Yi Liang, Tien-Yu Kuo, and Wai-Kei Mak. 2012. Power-driven flip-flop merging and relocation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 2, 180–191.

Samuel I. Ward, Natarajan Viswanathan, Nancy Y. Zhou, Cliff C. N. Sze, Zhuo Li, Charles J. Alpert, and David Z. Pan. 2013. Clock power minimization using structured latch templates and decision tree induction. In *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 599–606.

Huaizhi Wu, M. D. F. Wong, I.-Min Liu, and Yusu Wang. 2007. Placement-proximity-based voltage island grouping under performance requirement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 7, 1256–1269.

J. Z. Yan, C. Chu, and Wai-Kei Mak. 2011. SafeChoice: A novel approach to hypergraph clustering for wirelength-driven placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 7, 1020–1033.

Jin-Tai Yan and Zhi-Wei Chen. 2010. Construction of constrained multi-bit flip-flops for clock power reduction. In *International Conference on Green Circuits and Systems (ICGCS'10)*. 675–678.

Received July 2015; revised December 2015; accepted February 2016