UTPlaceF: A Routability-Driven FPGA Placer With Physical and Congestion Aware Packing

Wuxi Li, Shounak Dhar, and David Z. Pan, *Fellow, IEEE*

***Abstract*—Field programmable gate array (FPGA) packing and placement without routability consideration could lead to unroutable results for high-utilization designs. Conventional FPGA packing and placement approaches are shown to have severe difficulties to yield good routability. In this paper, we propose an FPGA packing and placement engine called UTPlaceF that simultaneously optimizes wirelength and routability. A novel physical and congestion aware packing algorithm and a hierarchical detailed placement technique are proposed. UTPlaceF outperforms state-of-the-art FPGA placers simultaneously in runtime and solution quality on International Symposium on Physical Design (ISPD) 2016 benchmark suite. Compared with the top three winners of ISPD'16 FPGA placement contest, UTPlaceF can deliver 6.2%, 11.6%, and 29.1% better routed wirelength with shorter runtime.**

***Index Terms*—Algorithms, FPGA, layouts, physical design, placement.**

I. Introduction

THE FIELD programmable gate array (FPGA) is a type of premanufactured integrated circuit designed to be configured by customers or designers. FPGAs are becoming more and more popular nowadays because of their ability to reprogram in the field to fix bugs, shorter time to market, and lower nonrecurring engineering costs. Historically, FPGAs were only used for fast realization of small digital circuits. However, in recent years, the gate count of commercial FPGAs has reached the scale of millions [1], so much more complex digital systems are moving toward FPGA-based design methodologies. Fig. 1 illustrates a simplified island-style heterogeneous FPGA. A traditional island-style FPGA typically contains a 2-D array of configurable logic blocks (CLBs) and surrounded by peripheral I/O blocks. Nowadays, heterogeneous blocks, such as digital signal processors (DSPs) and random access memories (RAMs), are also becoming prevalent in modern FPGAs.

A representative FPGA computer-aided design (CAD) flow is shown in Fig. 2. During logic synthesis and technology mapping, a circuit is translated into a netlist composed of lookup tables (LUTs) and flip-flops (FFs). In the packing stage, several LUTs and FFs together form a basic logic element (BLE)

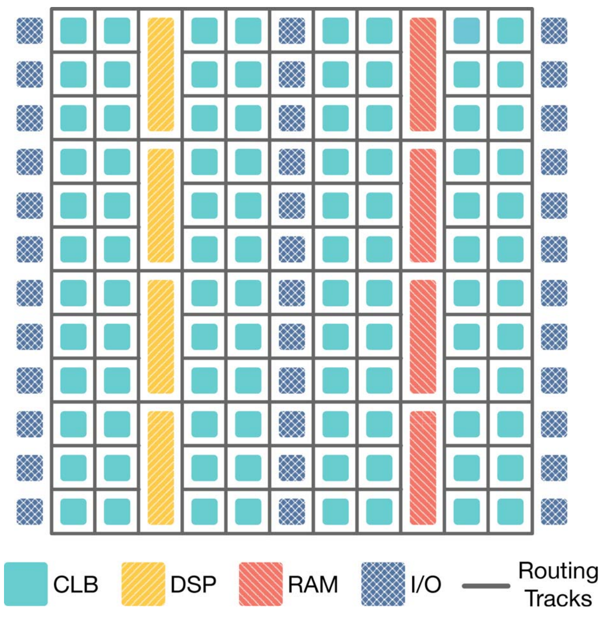


Fig. 1. Typical island-style FPGA.

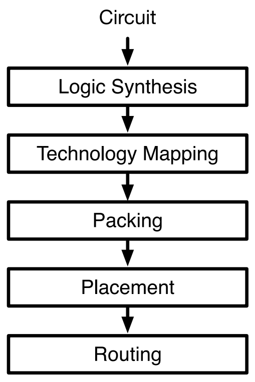


Fig. 2. Representative FPGA CAD flow.

and then several BLEs are grouped into a CLB. After packing, placement is responsible for determining the physical locations of all CLBs and complex blocks (DSPs/RAMs) while optimizing some metrics (e.g., wirelength, routability, timing, and power). Finally, routing is performed to physically connect CLBs.

As design size and complexity continue to increase dramatically, routability has become an important metric in FPGA domain. Traditional pure wirelength-driven optimizations without routability consideration often failed to map circuits into FPGA devices. Among all CAD stages, packing

Manuscript received February 8, 2017; revised May 26, 2017; accepted July 7, 2017. Date of publication July 19, 2017; date of current version March 29, 2018. The preliminary version has been presented at the International Conference on Computer-Aided Design (ICCAD) in 2016. This paper was recommended by Associate Editor P. Gupta. *(Corresponding author: Wuxi Li.)*

The authors are with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712 USA (e-mail: wuxi.li@utexas.edu; shounak.dhar@utexas.edu; dpan@ece.utexas.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2729349

and placement play key roles in optimizing various metrics, particularly routability.

Packing algorithms typically can be divided into three different categories: 1) seed-based approaches; 2) partitioning-based approaches; and 3) placement-guided and cluster-merging-based approaches. Seed-based packing approaches iteratively choose a BLE to form an initial CLB, then keep adding other unpacked BLEs into the CLB based on an attraction function until no more BLEs can be added. VPack [2], T-VPack [3], RPack [4], iRAC [5], T-NDPack [6], and MO-Pack [7] are representative examples of seed-based algorithms with different objectives and attraction functions. Partitioning-based approaches, like [8] and PPack [9], first apply a *k*-way partitioning to get a set of potential CLBs, and then perform a sequence of interpartition moves to legalize the packing solution. HDPack [10] is an example of placement-guided and cluster-merging-based methods. It incorporates physical information using a min-cut-partitioning-based global placement, and applies the idea of hybrid first choice clustering from [11] to recursively group clusters with the highest attraction until no more merging could be performed.

To improve routability, packing algorithms like [12], iRAC, [13], Un/DoPack [14], and T-NDPack proposed several different depopulation techniques to prevent CLBs from being fully filled. Depopulation can be classified into two categories: 1) uniform depopulation and 2) nonuniform depopulation. iRAC is a good example of uniform depopulation, it limits cell utilization of all CLBs based on Rent's rule. Un/DoPack is a example of nonuniform depopulation. It first runs through a regular CAD flow, then depopulates CLBs in the congested regions based on the routing result.

FPGA placement algorithms are very similar to ASIC's placement and typically fall into one of the following three categories: 1) simulated-annealing-based approaches; 2) min-cut-partitioning-based approaches; and 3) analytical approaches. Simulated annealing-based placers, like VPR [15], SCPlace [16], and [17], apply a probabilistic searching to approximate the global optimal solution. Min-cut-partitioning-based placers, e.g., [18] and [19], recursively apply min-cut partitioning until cells are fully spread out. Analytical placers typically approximate cost metrics, like wirelength and bin density, with a smooth objective function, then use numeric solvers to find the optimal solution. Different analytical placement approaches have been extensively studied in [20]–[28].

A. Motivation

Packing and placement are two key steps to achieve high-quality physical implementation with good routability. However, we found that existing academic packing and placement approaches have the following limitations.

1) Existing packing algorithms do not have good knowledge of cells' physical locations. Logical packing, which performs packing-based only on logical connectivity, could cluster cells that are physically far apart. As a result, it may lead to wirelength-unfriendly netlists and worsen routability. Most seed-based and partitioning-based packers do not have physical location information and only perform logical clustering. Existing placement-guided packers only have rough global placements, which is of poor quality compared to state-of-the-art placement engines. We believe that accurate physical information could better guide packing and yield placement-friendly CLB level netlists.
2) Existing packing algorithms are unaware of actual congestion information, which is crucial for efficient and high-quality depopulation. Blindly applying uniform depopulation would inevitably worsen wirelength and area, and it is more efficient to only avoid overpacking in routing congested regions. Therefore, accurate routing congestion information is of great importance in the packing stage.
3) In recent years, the gate count in modern FPGAs already reached the scale of millions. Therefore, packing and placement algorithms with high-quality and good scalability are highly desirable for large scale FPGAs.

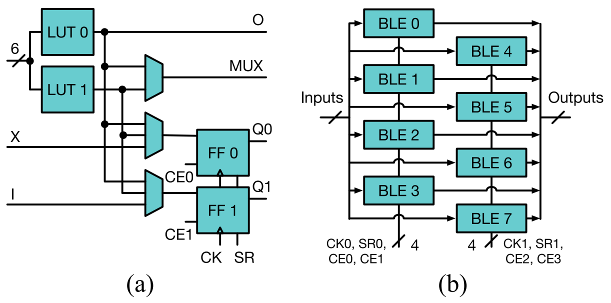


Fig. 3. (a) BLE and (b) CLB in Xilinx ultrascale architecture.

B. Contributions

In this paper, we propose a new routability-driven FPGA packing and placement engine called UTPlaceF. Our main contributions are listed as follows.

1) We propose a novel packing algorithm that incorporates accurate physical information based on a high-quality analytical global placement.
2) We propose a routing congestion-aware depopulation technique to efficiently balance wirelength and routability in a correct by construction manner.
3) We propose a hierarchical congestion-aware detailed placement technique to improve wirelength without degrading routability.
4) We perform experiments on the *International Symposium on Physical Design (ISPD)'16 Routability-Driven FPGA Placement Contest* [29] benchmark suite released by Xilinx. Compared to the ISPD'16 contest top three winners and other state-of-the-art FPGA placers, UTPlaceF achieves better routed wirelength with shorter runtime.

The rest of this paper is organized as follows: Section II reviews the preliminaries and presents the UTPlaceF framework overview. Sections III–V give the details of UTPlaceF packing and placement algorithms. Section VI shows the experimental results, followed by the conclusion in Section VII.

II. Preliminaries and Overview

A. FPGA Architecture

The ISPD'16 benchmark suite is targeted to Xilinx ultrascale VU095 [1]. The architecture of BLE and CLB in this

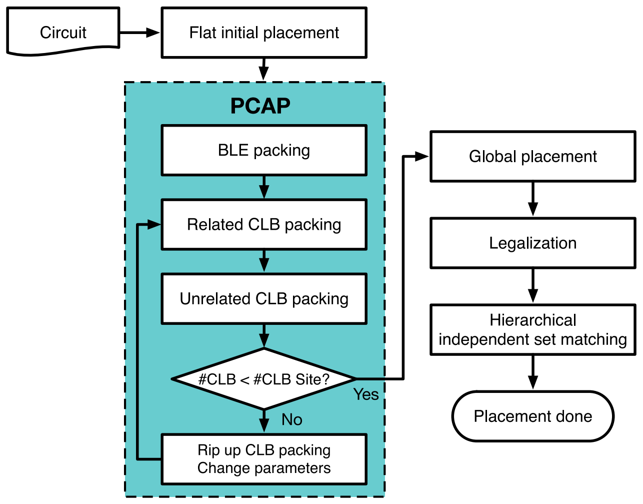


Fig. 4. Overview of UTPlaceF.

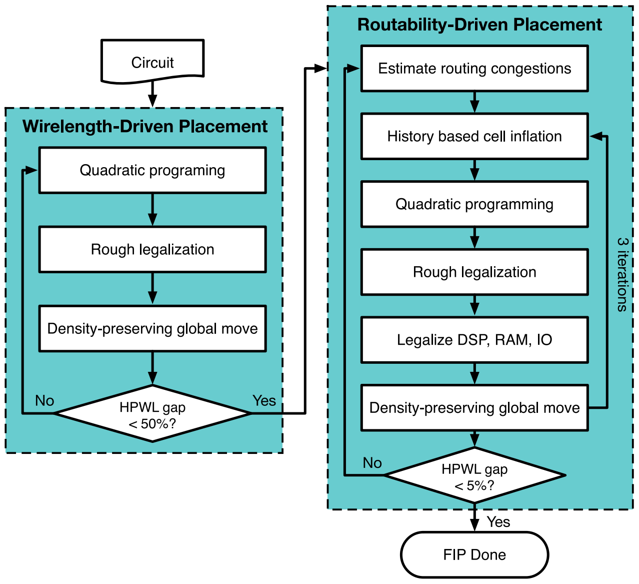

Circuit
Wirelength-Driven Placement
Quadratic programing
Rough legalization
Density-preserving global move
No
HPWL gap < 50%?
Yes
Routability-Driven Placement
Estimate routing congestions
History based cell inflation
Quadratic programming
Rough legalization
Legalize DSP, RAM, IO
Density-preserving global move
3 iterations
No
HPWL gap < 5%?
Yes
FIP Done

Fig. 5. Overall flow of FIP.

FPGA are shown in Fig. 3. Each CLB has eight BLE sites, and each BLE contains two LUT sites and two FF sites. The two LUT sites in a BLE could be implemented as a single 6-input LUT or two smaller LUTs with the total number of different input pins less than six. The two FFs in a BLE must share the same clock (CK) and set/reset (SR) pins, however, their clock enable (CE) pins could be different. There are two clock pins, two SR pins, and four CE pins available for each CLB. Each clock, SR and each two CEs are dedicated to four BLEs. More details can be found in [29].

B. Quadratic Placement

An FPGA netlist can be represented as a hypergraph $H = (V, E)$, where $V = \{v_1, v_2, \ldots, v_{|V|}\}$ is the set of cells, and $E = \{e_1, e_2, \ldots, e_{|E|}\}$ is the set of nets. Let $\boldsymbol{x} = \{x_1, x_2, \ldots, x_{|V|}\}$ and $\boldsymbol{y} = \{y_1, y_2, \ldots, y_{|V|}\}$ be the x and y coordinates of all cells. The wirelength-driven global placement problem is to determine position vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ that minimize the total wirelength and obey bin density constraint. Wirelength is measured by the half-perimeter wirelength (HPWL)

$$\text{HPWL}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{e \in E} \left\{ \max_{i,j \in e} |x_i - x_j| + \max_{i,j \in e} |y_i - y_j| \right\}. \quad (1)$$

As HPWL is not differentiable everywhere, quadratic placers approximate it by squared Euclidean distance between cells. Therefore, the wirelength cost function in quadratic placer is defined as

$$W(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}\boldsymbol{x}^T Q_x \boldsymbol{x} + \boldsymbol{c}_x^T \boldsymbol{x} + \frac{1}{2}\boldsymbol{y}^T Q_y \boldsymbol{y} + \boldsymbol{c}_y^T \boldsymbol{y} + \text{const.} \quad (2)$$

C. UTPlaceF Overview

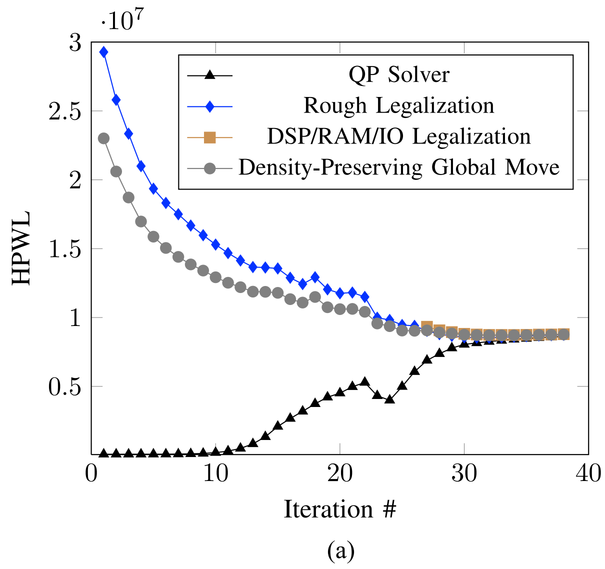
Fig. 4 shows the flowchart of UTPlaceF. The overall flow is composed of four parts: 1) flat initial placement (FIP); 2) physical and congestion aware packing (PCAP); 3) global placement; and 4) legalization and detailed placement.

FIP is responsible for generating cells' physical locations, and detecting cells that are likely to be placed into routing congested regions to better guide packing. In the packing stage (PCAP), LUTs and FFs are first grouped into BLEs, then BLEs are clustered into CLBs. We assume that FIP yields the optimal cell relative position, and packing should not perturb it too much. Therefore, PCAP, with cell physical locations information, disallows long-distance packing and prefers close packing. Similar to iRAC, absorbing small nets is treated as the main objective during packing stage of PCAP to reduce channel width and routing demand, which in turn improves wirelength and routability. Besides considering grouping connected cells, PCAP also packs unconnected cells based on their physical locations to further reduce the number of CLBs. Leveraged by routing congestion information from FIP, PCAP can perform loose packing only for cells that are likely to be placed into routing hotspots, and avoid blindly depopulating throughout whole netlists for routability enhancement. By using this congestion-aware depopulation technique, PCAP is able to achieve both good wirelength and routability.

Our global placement basically shares the same framework with FIP but handles CLBs instead of LUTs and FFs. In detailed placement stage, a bipartite-matching-based minimum-pin-movement legalization is applied first. Then a hierarchical independent set matching (ISM) is performed to further reduce wirelength. To preserve the routability optimized global placement solution, white spaces and cells in congested regions are handled specially throughout the detailed placement stage.

III. Flat Initial Placement

Our FIP adopts the main framework of an ASIC placer *POLAR 2.0* [30]. Its overall flow is shown in Fig. 5. In each iteration of wirelength-driven placement, a quadratic program is solved followed by rough legalization [31] for reducing cells overlaps. Then the density preserving global move [32] is applied to improve the wirelength of the rough-legalized placement while preserving bin densities. A sequence of pure wirelength-driven placement iterations is performed until the gap between the upper bound wirelength and the



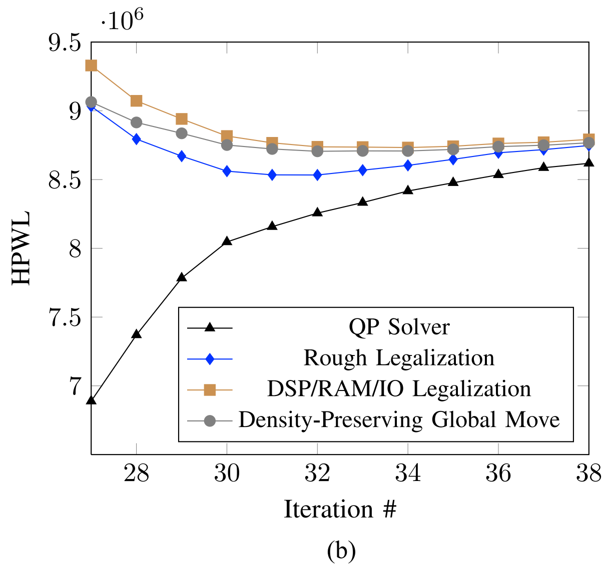


Fig. 6. HPWL at different steps in FIP of benchmark FPGA-5. (a) All placement iterations. (b) Placement iterations in routability-driven stage.

lower bound wirelength is less than a certain number, which is 50% in PCAP. In the routability-driven placement stage, after a certain number of placement iterations, a fast global router NCTUgr [33] is called for routing congestion estimation, then similar to POLAR 2.0, cells in congested regions are inflated by a small ratio, and the inflation accumulates to the end of FIP. Different from the first stage, DSPs, RAMs, and I/O blocks are immediately legalized after rough legalization in this stage. Since sites for these cells are typically discrete and scattered on FPGAs, if we handle them like LUTs and FFs in rough legalization, they might be far away from their legal positions in the final FIP solution. This discrepancy would introduce inaccuracy of cell relative positions into FIP. To eliminate this discrepancy, UTPlaceF performs an extra legalization step for DSPs, RAMs, and I/Os right after the conventional rough legalization in the second stage of FIP. By simply doing this, DSPs, RAMs, and I/Os will use their legal positions as their anchor points in placement iterations, and the discrepancy will be eliminated in the final FIP solution. The legalization approach here will be further discussed in Section V-B. Fig. 6(a) illustrates the progression of the placement solution with respect to HPWL in different steps. A zoomed-in view of the last few iterations, with the legalization for DSPs, RAMs, and I/Os enabled, is shown in Fig. 6(b).

The two objectives of FIP are: 1) generating physical locations for each LUT and FF and 2) detecting LUTs and FFs that are in routing congested regions. Cell physical locations are explicitly generated by the wirelength and routability co-optimized placement. Congestion information associated with cells is implicitly obtained from history-based cell inflation. The insight of cell inflation is quantifying the possibility of a cell lying in routing congested regions using its area—a larger cell area indicates a larger possibility of being placed into congested regions. After FIP, each LUT and FF would have a physical location and a cell area, which indicates the congestion level associated with it.

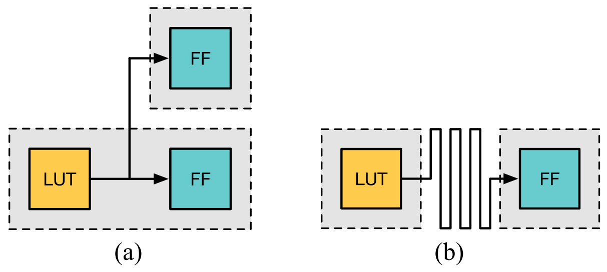


Fig. 7. Different LUT and FF pairing scenarios. (a) LUT fanouts to multiple FFs, and group with the closest one. (b) LUT fanouts to one FF that is far away and the grouping is rejected.

IV. Physical and Congestion Aware Packing

A. Max-Weighted-Matching-Based BLE Packing

As a BLE in our target FPGA architecture, Xilinx ultrascale, contains two LUTs and two FFs, existing VPR-style BLE packing algorithms cannot be directly applied. To address this new BLE architecture, we propose a two-step BLE packing algorithm that comprises: 1) LUT and FF pairing and 2) LUT-FF pairs matching.

In PCAP, we apply the LUT and FF pairing in a similar manner to the BLE packing in VPack. As shown in Fig. 7, we group each LUT to the closest FF in its fanout. Besides that, long-distance packing is rejected, and only packing within *maximum packing distance of BLE* ($\overline{\lambda_b}$) is allowed. This step is mainly to make full use of fast connections between LUTs and FFs that are in the same BLEs.

In the second step, *max-weighted matching* is used for finalizing the BLE packing. We construct an undirected weighted graph UWG $= (V, E)$, where each v_i in $V = \{v_1, v_2, \ldots, v_{|V|}\}$ is an LUT-FF pair, a single LUT, or a single FF. $E = \{e_1, e_2, \ldots, e_{|E|}\}$ represents the set of legal mergings. To apply high-attraction and close packing, we say a merging (v_i, v_j) is legal if and only if:

1) v_i and v_j are connected in the netlist;

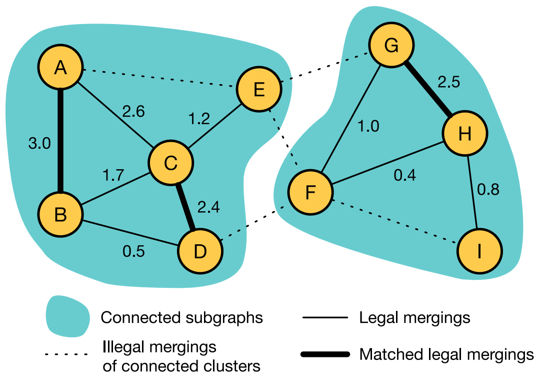


Fig. 8. Simple max-weighted cluster matching example.

2) merging v_i and v_j into the same BLE does not violate any packing rules;
3) the merging attraction is greater than the *minimum packing attraction for BLEs* ($\underline{\phi_b}$).

In the UWG, edge weights are set as merging attractions. The attraction value for a merging is defined as

$$\phi_b(v_i, v_j) = \left(1 - e^{\gamma_b\left(\text{dist}(v_i,v_j)-\overline{\lambda_b}\right)}\right) \sum_{p\in Net(v_i\cap v_j)} \frac{k_b}{\deg(p)-1}. \tag{3}$$

where γ_b is a constant value being experimentally set as 0.2, $\text{dist}(v_i, v_j)$ is the Manhattan distance between v_i and v_j, $\overline{\lambda_b}$ is the maximum packing distance of BLE which is 4 in PCAP, $Net(c_i \cap c_j)$ is the set of nets shared between v_i and v_j, $\deg(p)$ is the total number of pins of net p that is exposed in cluster level, and k_b is 2 for 2-pin nets and 1 for other nets.

The first term, $1 - e^{\gamma_b(\text{dist}(v_i,v_j)-\overline{\lambda_b})}$, is a packing distance penalty factor in range $(-\infty, 1 - e^{-\gamma_b\overline{\lambda_b}})$. This factor is very close to 1 for mergings of distance much less than $\overline{\lambda_b}$. It drops quickly as $\text{dist}(v_i, v_j)$ gets close to $\overline{\lambda_b}$, and becomes negative once $\text{dist}(v_i, v_j)$ is greater than $\overline{\lambda_b}$. By using the first term, short-distance mergings in PCAP are always preferred. The second term, $\sum_{p\in Net(v_i\cap v_j)}(k_b/\deg(p) - 1)$, is introduced for reducing the number of nets exposed in cluster level, which in turn improves wirelength and routability. With the second term, merging two clusters that share more small nets is of high priority, and 2-pin nets are given even higher weight by the factor $k_b = 2$. In PCAP, the minimum packing attraction for BLEs ($\underline{\lambda_b}$) is set to 0 by default.

Fig. 8 shows a simple cluster matching example. Due to our rules for legal mergings, the constructed UWG typically is not connected and comprises many small connected subgraphs. Mergings in different connected subgraphs are independent, so PCAP performs a max-weighted matching algorithm on each of these subgraphs and all matched cluster pairs would be merged. The location of a merged cluster is set as the average location of all cells (LUTs and FFs) it contains, and the cluster area is simply the sum of cell areas. After each pass of matching and merging, PCAP rebuilds the UWG for clusters generated from the previous stage and resolves the max-weighted matching for each new connected subgraph until no more legal merging exists.

Algorithm 1 Max-Weighted-Matching-Based BLE Packing

```
Require: FIP and LUT-FF pairing are done.
Ensure: BLE level netlist with external nets reduced.
 1: while true do
 2:     $numMatching \leftarrow 0$
 3:     $status[u] \leftarrow Untouched\ \forall u \in V$
 4:     for each $s \in V$ do
 5:         if $status[s] \neq Untouched$ then
 6:             continue
 7:         end if
 8:         $g \leftarrow$ ConstructConnectedUWG($s$)
 9:         Run max-weighted matching on $g$
10:         for each matched edge $(u, v) \in g$ do
11:             $c \leftarrow \{u, v\}$
12:             $V \leftarrow V \setminus \{u, v\} \cup \{c\}$
13:             $status[c] \leftarrow Popped$
14:             $numMatching \leftarrow numMatcahing + 1$
15:         end for
16:     end for
17:     if $numMatching == 0$ then
18:         return
19:     end if
20: end while
21:
22: function ConstructConnectedUWG(s)
23:     Initialize an empty UWG $g$
24:     Initialize an empty queue $q$
25:     Push $s$ into $q$
26:     $status[s] \leftarrow InQueue$
27:     Add $s$ into $g$
28:     while $q$ is not empty do
29:         $t \leftarrow$ fetch and pop $q$ top
30:         for each $v$ connected to $t$ do
31:             if $status[v] \neq Popped$ and $\phi_b(t, v) \geq \underline{\phi_b}$ then
32:                 if $status[v] == Untouched$ then
33:                     Push $v$ into $q$
34:                     $status[v] \leftarrow InQueue$
35:                     Add $v$ into $g$
36:                 end if
37:                 Add edge $(t, v, \phi_b(t, v))$ into $g$
38:             end if
39:         end for
40:     end while
41:     return $g$
42: end function
```

The pseudo-code of the max-weighted-matching-based BLE packing is summarized in Algorithm 1. Each connected subgraph is constructed by calling the function `ConstructConnectedUWG` from lines 22 to 42. Nodes and edges are added into the subgraph in a breadth-first search manner through a queue from lines 28 to 40. When the subgraph stops growing, the max-weighted matching would be run on the constructed subgraph at line 9, and mergings corresponding to matched edges would be committed to the netlist from lines 10 to 15. The loop of constructing subgraphs, solving max-weighted matching, and committing matched mergings are iteratively executed from lines 1 to 20, and stops at line 18 when no more merging can be performed. In PACP, it is very time-consuming to consider all neighbors of a cell connected by high-degree nets at line 30. So for each cell, we only consider its neighbors connected by nets containing at most 16 pins.

The time complexity of Algorithm 1 turns out to be $\mathcal{O}(|V|(k^3(|V|/|E|) + |E_c|\log|V_c|))$, where V and E denote the set of clusters and nets in the netlist, respectively, k denotes the average number of pins in each cluster in V, and $|V_c|$ and

$|E_c|$ are the average numbers of vertices and edges in each connected subgraph in the UWG.

On average, each cluster is incident to k nets and each net contains $k(|V|/|E|)$ clusters, so each cluster in V has $\mathcal{O}(k^2(|V|/|E|))$ connected neighbors at line 30. Since each attraction calculation (3) at line 31 takes $\mathcal{O}(k)$ time, the graph construction (`ConstructConnectedUWG`) for each connected subgraph, $G_c = (V_c, E_c)$, has time complexity of $\mathcal{O}(k^3(|V|/|E|)|V_c|)$. Besides, solving the max-weighted matching at line 9 takes $\mathcal{O}(|V_c||E_c|\log|V_c|)$ time and the cell mergings from lines 10 to 15 can be done in $\mathcal{O}(|E_c|)$ time. The time complexity of each connected subgraph (lines 8–15) can be bounded by $\mathcal{O}(|V_c|(k^3(|V|/|E|) + |E_c|\log|V_c|))$. Considering we need to handle $\mathcal{O}(|V|/|V_c|)$ such connected subgraphs, Algorithm 1, therefore, has total time complexity of $\mathcal{O}(|V|(k^3(|V|/|E|) + |E_c|\log|V_c|))$. In practice, both $|V_c|$ and $|E_c|$ are relatively small (less than 200) and independent to the netlist size.

B. Related CLB Packing With Congestion-Aware Depopulation

After BLE packing, we create a CLB for each single BLE. CLB packing is done by successively merging smaller CLBs into larger ones. CLBs that share common nets are said to be related, and in this stage, only related CLBs mergings are considered.

The bestchoice clustering (BC) [34] is used as our main engine for related CLB packing. In BC, the attractions of all legal CLB mergings are calculated first, then the algorithm iteratively merges CLB pairs with the highest attraction using a priority queue (PQ). The location and area of a merged CLB is set as the average location and total area of cells it contains, respectively. After each merging, the legality and attraction of mergings related to the new CLB are updated accordingly.

In PCAP, a related CLB merging (c_i, c_j) is said to be legal if and only if:

1) c_i and c_j are connected in the netlist;
2) merging c_i and c_j into the same CLB does not violate any packing rules;
3) the merging attraction is greater than the *minimum packing attraction for related CLBs* ($\underline{\phi_{rc}}$);
4) total area of c_i and c_j is no greater than the *maximum CLB area* ($\overline{a_c}$).

The first three rules are inherited from our BLE packing. The fourth rule is introduced to perform congestion-aware depopulation and avoid overpacking in routing congested regions. Note that all the cell areas used in the fourth rule are from the accumulated cell inflation in FIP. As discussed in Section III, cells with larger areas indicate higher possibility to be placed into routing congested regions. By constraining area of each CLB, PCAP would apply loose packing in routing congested regions, and tight packing in other regions. This congestion awareness makes PCAP able to achieve a good tradeoff between wirelength and routability. Fig. 9 illustrates our congestion-aware depopulation technique. Note that BLEs with larger areas are in routing congested regions.

The attraction function of related CLB mergings (c_i, c_j) is defined as

$$\phi_{rc}(c_i, c_j) = \left(1 - e^{\gamma_{rc}\left(\text{dist}(c_i,c_j) - \overline{\lambda_{rc}}\right)}\right) \sum_{p \in Net(c_i \cap c_j)} \frac{k_{rc}}{\deg(p) - 1}. \tag{4}$$

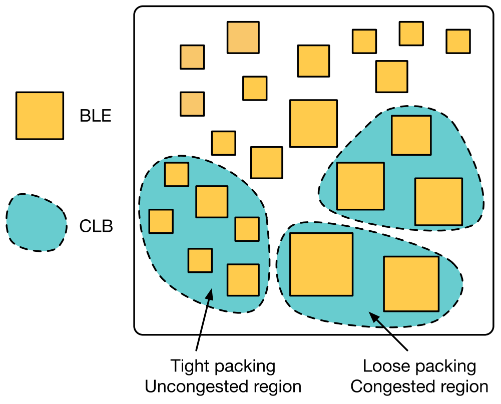


Fig. 9. Congestion-aware depopulation of PCAP during CLB packing.

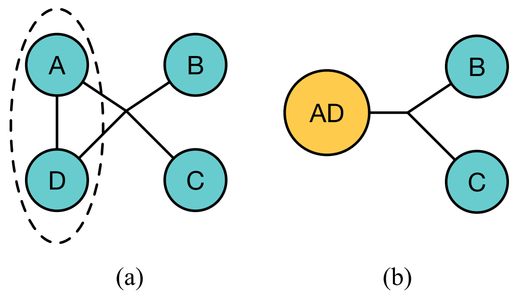


Fig. 10. Merging a pair of clusters A and D, assuming $\text{dist}(B, C) = 1$, $\gamma_{rc} = 0.2$, $\overline{\lambda_{rc}} = 6$. (a) Before merging A and D, $\phi_{rc}(B, C) \approx 0.211$ and (b) after merging A and D, $\phi_{rc}(B, C) \approx 0.316$.

Equation (4) is basically a replica of our BLE packing attraction function defined in (3), but differs only by some constant parameters. We experimentally set γ_{rc} to 0.2, and $\overline{\lambda_{rc}}$ to 6. k_{rc} is 2 for 2-pin nets and 1 for other nets. The minimum packing attraction for related CLBs ($\underline{\phi_{rc}}$) is set to 0.1, and the maximum CLB area ($\overline{a_c}$) is set as 1.8 times average CLB area by default.

Our BC-based related CLB packing algorithm has several major differences compared with the traditional BC clusterings. In traditional BC, a speedup technique called *lazy update* [34] is widely adopted, and this technique relies on their observation that the vast majority of attraction updates are decreasing their ranking in the PQ. However, this observation does not apply to our packing algorithm. Fig. 10 shows a simple example, before merging clusters A and D, B and C only share a 4-pin net, and after the merging, the 4-pin net becomes a 3-pin net in the cluster level netlist, therefore the attraction between B and C increases due to the second term in (4). We can see that this kind of attraction increases could happen to any cluster pairs that share nets with more than 3 pins and all attraction updates in our BC-based related CLB packing would increase their ranking in the PQ. Therefore, lazy update cannot be adopted here. To guarantee the globally best merging candidate can always be fetched in each iteration, instead of only considering the best neighbor for each cluster in the traditional BC, we push all possible mergings for each cluster into the PQ and dynamically update all merging attractions affected by each committed merging.

The pseudo-code of our BC-based related CLB packing is summarized in Algorithm 2. Initially, all legal merging candidates are pushed into a PQ from lines 3 to 9. Each time

Algorithm 2 BC-Based Related CLB Packing

```
Require: BLE packing is done.
Ensure: CLB level netlist with external nets reduced.
Ensure: All CLB packing rules are satisfied.
Ensure: Cells in routing congested regions are not overpacked.
 1: Create an empty priority queue $PQ$
 2: $valid[u] \leftarrow true \ \forall u \in V$
 3: for each $u \in V$ do
 4:     for each $v$ connected to $u$ do
 5:         if $\phi_{rc}(u, v) \geq \underline{\phi_{rc}}$ then
 6:             Push $(u, v, \phi_{rc}(u, v))$ into $PQ$
 7:         end if
 8:     end for
 9: end for
10: while $PQ$ is not empty do
11:     $(c_i, c_j, \phi_{rc}(c_i, c_j)) \leftarrow$ fetch and pop $PQ$ top
12:     if $valid[c_i]$ and $valid[c_j]$ then
13:         $c_{ij} \leftarrow \{c_i, c_j\}$
14:         $V \leftarrow V \setminus \{c_i, c_j\} \cup \{c_{ij}\}$
15:         $valid[c_i], valid[c_j] \leftarrow false$
16:         $valid[c_{ij}] \leftarrow true$
17:         for each $c_k$ connected to $c_{ij}$ do
18:             if $\phi_{rc}(c_k, c_{ij}) \geq \underline{\phi_{rc}}$ then
19:                 Push $(c_k, c_{ij}, \phi_{rc}(c_k, c_{ij}))$ into $PQ$
20:             end if
21:         end for
22:         for each net $e$ shared by $c_i$ and $c_j$ do
23:             for each $(c_p, c_q, \phi_{rc}^{pq})$ such that that $c_p$ and $c_q$ share $e$ do
24:                 if $valid[c_p]$ and $valid[c_q]$ then
25:                     $\phi_{rc}^{pq} \leftarrow \phi_{rc}(c_p, c_q)$
26:                     Update the ranking of $(c_p, c_q, \phi_{rc}^{pq})$ in $PQ$
27:                 end if
28:             end for
29:         end for
30:     end if
31: end while
```

the merging candidate with the highest attraction is popped from the PQ top at line 11 and is committed to the netlist from lines 13 to 14. New merging candidates related to the new cluster are pushed into the PQ from lines 17 to 21. All attractions that are affected by the new merging are updated from lines 22 to 29. The loop from lines 10 to 31 keeps executing until no more merging candidates exist in the PQ. Similar to our BLE packing, we ignore nets that have more than 16 pins in Algorithm 2 lines 4 and 17 to speed up the runtime.

Algorithm 2 turns out to have $\mathcal{O}((1 + (|V|/|E|))k^2(|V|^2/|E|)(k + \log(k^2(|V|^2/|E|))))$ time complexity, where V and E denote the set of clusters and nets in the BLE-level netlist, respectively, and k is the average number of pins in each cluster in V.

We first consider the PQ initialization from lines 1 to 9. Each cluster in V has $\mathcal{O}(k^2(|V|/|E|))$ connected neighbors, since, on average, each cluster is incident to k nets and each net contains $k(|V|/|E|)$ clusters. Thus, initially, there are $\mathcal{O}(k^2(|V|^2/|E|))$ merging candidates being pushed into the PQ and the initialization time can be bounded by $\mathcal{O}(k^2(|V|^2/|E|)\log(k^2(|V|^2/|E|)))$.

Then, we analyze the loop from lines 10 to 31. The number of mergings performed at line 13 is bounded by $\mathcal{O}(|V|)$, and after each merging, $\mathcal{O}(k^2(|V|/|E|))$ PQ push operations are executed from lines 17 to 21. Therefore, from lines 10 to 31, the total number of new merging candidates being pushed into the PQ is bounded by $\mathcal{O}(k^2(|V|^2/|E|))$. Considering there are $\mathcal{O}(k^2(|V|^2/|E|))$ merging candidates in the initial PQ, the PQ size then can be bounded by $\mathcal{O}(k^2(|V|^2/|E|))$ at any time during Algorithm 2 execution and the total time taken by PQ pop at line 11 can be bounded by $\mathcal{O}(k^2(|V|^2/|E|)\log(k^2(|V|^2/|E|)))$. Since each attraction evaluation (4) at line 18 takes $\mathcal{O}(k)$ time and each PQ push at line 19 takes $\mathcal{O}(\log(k^2(|V|^2/|E|)))$ time, the loop from lines 17 to 21 has time complexity of $\mathcal{O}(k^2(|V|/|E|)(k + \log(k^2(|V|^2/|E|))))$. By assuming any pair of merging clusters shares $\mathcal{O}(1)$ common nets, the PQ update at lines 25 and 26 needs to be performed $\mathcal{O}(k^2(|V|^2/|E|^2))$ times for each c_{ij}. So the time complexity of the loop from lines 22 to 29 is $\mathcal{O}(k^2(|V|^2/|E|^2)(k + \log(k^2(|V|^2/|E|))))$. Combining all these time complexities and the $\mathcal{O}(|V|)$ bound of the execution counts for the two loops (lines 17–21 and lines 21–29), the main loop from lines 10 to 31 turns out to have time complexity of $\mathcal{O}((1 + (|V|/|E|))k^2(|V|^2/|E|)(k + \log(k^2(|V|^2/|E|))))$.

The final $\mathcal{O}((1 + (|V|/|E|))k^2(|V|^2/|E|)(k + \log(k^2(|V|^2/|E|))))$ time complexity of Algorithm 2 then can be derived by summing the analysis results of the PQ initialization and the loop from lines 10 to 31.

C. Size-Prioritized K-Nearest-Neighbor Unrelated CLB Packing

CLBs without common nets are said to be unrelated. After related CLB packing stage, unrelated CLB mergings are considered. Different from related CLB packing, in which reducing external nets is the main objective, unrelated CLB packing aims to reduce the number of CLBs.

BC-based approaches typically could yield very good packing solutions for given attraction functions. However, they have an inherent drawback—inability of making tight packing. Generally, BC would generate a large number of medium-sized clusters that are difficult to merge further due to the cluster capacity constraint. To mitigate this issue, we proposed a size-prioritized BC-based unrelated CLB packing. By assigning higher priority to mergings producing larger CLBs, medium-sized CLBs would be promoted quickly. As a result, much tighter packing solutions can be achieved.

Unlike the related CLB packing technique in Algorithm 2, where all merging candidates are in one single PQ, we have a separate PQ for each merging size in the unrelated CLB packing. In other words, merging candidates are separated by the number of BLEs in their resulting CLBs, and only mergings result in same BLE count could be placed into the same PQ. The PQ corresponding to larger merging size is grant higher priority and always be processed first.

Within each PQ, BC-based unrelated CLB packing is performed in a manner similar to our related CLB packing. For a CLB, however, instead of considering all its connected CLBs, its K-nearest neighbors (in terms of physical distance) within distance $\overline{\lambda_{uc}}$ would be considered in our unrelated CLB packing. Besides, a different attraction function defined in (5) is used

$$\phi_{uc}(c_i, c_j) = 1 - e^{\gamma_{uc}(\mathrm{dist}(c_i, c_j) - \overline{\lambda_{uc}})}. \quad (5)$$

The attraction function ϕ_{uc} is a packing distance penalty factor similar to the first terms of (3) and (4). In UTPlaceF, we set γ_{uc} to 0.2, $\overline{\lambda_{uc}}$ to 8, and K to 30 by default. Note that, although the objective of our unrelated CLB packing

Algorithm 3 Size-Prioritized *K*-Nearest-Neighbor Unrelated CLB Packing

```
Require: Related CLB packing is done.
Ensure: CLB level netlist with total number of CLBs reduced.
Ensure: All CLB packing rules are satisfied.
Ensure: Cells in routing congested regions are not overpacked.
 1: // Create an empty priority queue for each merging size
 2: $pq[i] \leftarrow \emptyset \ \forall i \in 2, 3, \ldots, N$
 3: $valid[u] \leftarrow true \ \forall u \in V$
 4: $numMergings[u] \leftarrow 0 \ \forall u \in V$
 5: for each $u \in V$ do
 6:     AddKNearestNeighbors($u$)
 7: end for
 8: while $\exists i \in 2, 3, \ldots, N : pq[i] \neq \emptyset$ do
 9:     $PQ \leftarrow pq[i]$ where $pq[i] \neq \emptyset$ and $pq[j] == \emptyset \ \forall j \in i+1, \ldots, N$
10:     $(c_i, c_j, \phi_{uc}(c_i, c_j)) \leftarrow$ fetch and pop $PQ$ top
11:     if $valid[c_i]$ and $valid[c_j]$ then
12:         $c_{ij} \leftarrow \{c_i, c_j\}$
13:         $V \leftarrow V \setminus \{c_i, c_j\} \cup \{c_{ij}\}$
14:         $valid[c_{ij}] \leftarrow true$
15:         $valid[c_i], valid[c_j] \leftarrow false$
16:         $numMergings[c_{ij}] \leftarrow 0$
17:         AddKNearestNeighbors($c_{ij}$)
18:         for each $u \in c_i, c_j$ do
19:             for each $v$ such that $(u, v, \phi_{uc}(u, v)) \in pq[*]$ do
20:                 if $valid[v]$ then
21:                     $numMergings[v] \leftarrow numMergings[v] - 1$
22:                     if $numMergings[v] == 0$ then
23:                         AddKNearestNeighbors($v$)
24:                     end if
25:                 end if
26:             end for
27:         end for
28:     end if
29: end while
30:
31: function AddKNearestNeighbors(u)
32:     for each $v \in \{v \in V | dist(u, v) \leq \overline{\lambda_{uc}}\}$ sorted by $dist(u, v)$ do
33:         if $\phi_{uc}(u, v) \geq \overline{\phi_{uc}}$ then
34:             Push $(u, v, \phi_{uc}(u, v))$ into $pq[numBLEs(u \cup v)]$
35:             $numMergings[u] \leftarrow numMergings[u] + 1$
36:             $numMergings[v] \leftarrow numMergings[v] + 1$
37:             if $numMergings[u] == K$ then
38:                 return
39:             end if
40:         end if
41:     end for
42: end function
```

is to reduce the number of CLBs and deliver tight packing, the congestion-aware depopulation technique described in Section IV-B is still applied in this stage to maintain good routability.

The pseudo-code of our size-prioritized *K*-nearest-neighbor unrelated CLB packing is summarized Algorithm 3. Initially, all merging candidates are pushed into a PQ array ($pq[*]$) from lines 2 to 7. The merging candidates of each cluster are added by calling the function AddKNearestNeighbors from lines 31 to 42. Each time, among all nonempty PQs in $pq[*]$, the one corresponding to the largest merging size is fetched at line 9. The merging candidate with the highest attraction in the fetched PQ is popped from the PQ top at line 10 and is committed to the netlist from lines 12 to 13. New merging candidates that include the new cluster are pushed into $pq[*]$ by calling function AddKNearestNeighbors at line 17. Lines 18 to 27 guarantee that each cluster has merging candidates in $pq[*]$ if legal merging exists. The loop from lines 8 to 29 keeps executing until no more merging candidates exist in $pq[*]$.

The time complexity of Algorithm 3 turns out to be $\mathcal{O}(|V|\overline{\lambda_{uc}}^2 + K|V|\log(K|V|))$, where V denotes the set of clusters in the netlist and K is the maximum number of neighbors being considered for each cluster during the packing.

It is reasonable to assume that each cluster has average of $\mathcal{O}(K)$ valid mergings candidates in $pq[*]$, since AddKNearestNeighbors is only called during $pq[*]$ initialization (at line 6) and when a certain cluster does not have any valid merging candidates in $pq[*]$ (at lines 17 and 23). Given this assumption, each merging performed at line 12 would invalidate $\mathcal{O}(K)$ merging candidates in $pq[*]$ on average, which could trigger average of $\mathcal{O}(1)$ calls of AddKNearestNeighbors at line 23 for clusters that lose all of their valid merging candidates in $pq[*]$. Therefore, AddKNearestNeighbors is only called $\mathcal{O}(1)$ times (at lines 17 and 23) on average for each merging performed from lines 12 to 27. Considering the number of merging performed is bounded by $\mathcal{O}(|V|)$ and there are $\mathcal{O}(|V|)$ calls of AddKNearestNeighbors during $pq[*]$ initialization at line 6, we can get the following three intermediate bounds: 1) there are total of $\mathcal{O}(|V|)$ function calls of AddKNearestNeighbors in Algorithm 3; 2) there are total of $\mathcal{O}(K|V|)$ merging candidates being pushed into and popped out of $pq[*]$ in Algorithm 3; and 3) the size of $pq[*]$ is bounded by $\mathcal{O}(K|V|)$ at any time during Algorithm 3 execution.

We first analyze the total time consumed by AddKNearestNeighbors. The time complexity of finding the *K*-nearest feasible neighbors within the distance of $\overline{\lambda_{uc}}$ for a given cluster u can be bounded by $\mathcal{O}(\log|V| + \overline{\lambda_{uc}}^2)$, since collecting feasible clusters within the distance of $\overline{\lambda_{uc}}$ to u takes $\mathcal{O}(\log|V| + \overline{\lambda_{uc}}^2)$ time if all cluster locations are stored in an R-tree or a k-d tree, and getting the *K*-nearest neighbors of u in $\mathcal{O}(\overline{\lambda_{uc}}^2)$ clusters can be achieved in amortized $\mathcal{O}(\overline{\lambda_{uc}}^2)$ time. Besides, each of the $\mathcal{O}(K)$ PQ pushes at line 34 takes $\mathcal{O}(\log(K|V|))$ time, the time complexity of AddKNearestNeighbors turns out to be $\mathcal{O}(\overline{\lambda_{uc}}^2 + K\log(K|V|))$. Considering there are $\mathcal{O}(|V|)$ calls of AddKNearestNeighbors in Algorithm 3, the total time taken by AddKNearestNeighbors then can be bounded by $\mathcal{O}(|V|\overline{\lambda_{uc}}^2 + K|V|\log(K|V|))$.

As for non-AddKNearestNeighbors parts, the PQ pop at line 9 and line 10 takes total of $\mathcal{O}(K|V|\log(K|V|))$ time by assuming $pq[*]$ contains a constant number of PQs. Since the number of mergings performed is bounded by $\mathcal{O}(|V|)$, the code from lines 12 to 27 can only be executed $\mathcal{O}(|V|)$ times. In each of these $\mathcal{O}(|V|)$ executions, by ignoring the AddKNearestNeighbors at line 23, the loop from lines 18 to 27 takes $\mathcal{O}(K)$ time. Therefore, without considering the AddKNearestNeighbors at lines 17 and 23, the time complexity of the loop from lines 8 to 29 turns out to be $\mathcal{O}(K|V|\log(K|V|) + K|V|)$, which can be simplified to $\mathcal{O}(K|V|\log(K|V|))$.

Combining the results of the AddKNearestNeighbors part and the non-AddKNearestNeighbors parts, we can finally bound the time complexity of Algorithm 3 by $\mathcal{O}(|V|\overline{\lambda_{uc}}^2 + K|V|\log(K|V|))$.

For high-utilization designs, our default unrelated CLB packing might still not be able to generate tight enough

Algorithm 4 CLB Packing and Rip-Up and Repacking

```
Require: BLE packing is done.
Require: Maximum FPGA CLB utilization $U_{max}$, maximum unrelated CLB
    packing distance incrasing rate $\beta_{\overline{\lambda_{uc}}}$, and minimum related CLB packing
    attraction increasing rate $\Delta\underline{\phi_{rc}}$.
Ensure: Maximum FPGA CLB utilization constraint is satisfied.
1: while true do
2:     Perform related CLB packing
3:     Perfrom unrelated CLB packing
4:     if CLB utilization $\leq U_{max}$ then
5:         return
6:     end if
7:     $\overline{\lambda_{uc}} \leftarrow \overline{\lambda_{uc}} \cdot \beta_{\overline{\lambda_{uc}}}$
8:     $\underline{\phi_{rc}} \leftarrow \underline{\phi_{rc}} + \Delta\underline{\phi_{rc}}$
9: end while
```

packing solutions that satisfy FPGA capacity constraint. In this case, the existing packing solution will be ripped up, and new related and unrelated CLB packing parameters will be adopted to generate a tighter packing solution in the next packing pass. PCAP would iteratively perform this rip-up and repacking loop until the FPGA capacity constraint is satisfied. The details of this rip-up and repacking phase will be further discussed in Section IV-D.

D. Net Reduction and Packing Tightness Tradeoff

Our related CLB packing works effectively to reduce the number of external nets, however, it often yields relatively loose packing due to the inherent shortcoming of BC mentioned in Section IV-C. In contrast, our unrelated CLB packing is capable of aggressively reducing the number of CLBs and achieving tight packing. Therefore, if more packing is performed in the related CLB packing stage, a loose packing solution with less external nets would be delivered. However, if we only do a small portion of packing in the related CLB packing stage and leave most of the work to unrelated CLB packing, the final packing would be more inclined to the "tight" side with more external nets.

In PCAP, the minimum related CLB packing attraction ($\underline{\phi_{rc}}$) and the maximum unrelated CLB packing distance ($\overline{\lambda_{uc}}$) are used to control the amount of packing work for each (related/unrelated) CLB packing stage. Initially, $\underline{\phi_{rc}}$ is set as 0.1 to aggressively reduce the number of external nets, and $\overline{\lambda_{uc}}$ is set as 8 to only allow close packing in unrelated CLB packing stage. This initial setting typically results in a loose packing with a large amount of net reduction. For high-utilization designs, however, the packing solution generated by the initial setting could be sparse to the extent that number of CLBs exceeds the FPGA capacity. To address this problem, PCAP would discard the existing CLB packing solution (but respect BLE packing solution) and perform a repacking step, which applies related and unrelated CLB packing again. In the repacking phase, however, $\underline{\phi_{rc}}$ is increased to reduce related CLB packing, and $\overline{\lambda_{uc}}$ is also increased to allow unrelated CLB packing of longer distance. As results, the repacking step would achieve tighter packing but sacrifice net reduction. The repacking step is repeated until the CLBs utilization target is satisfied.

The pseudo-code of our rip-up and repacking is summarized in Algorithm 4. In UTPlaceF, we experimentally set $U_{\max}$ to 0.999, $\Delta\underline{\phi_{rc}}$ to 0.3, and $\beta_{\overline{\lambda_{uc}}}$ to 1.414.

V. Post-Packing Placement

A. Global Placement

After PCAP, the global placement is performed immediately to further optimize wirelength and routability. Our global placement shares the same framework and parameter settings with FIP, but instead of optimizing flat LUT/FF netlist, it considers each CLB as a whole. It is an incremental placement using the FIP solution as the starting point to speed up wirelength convergence. Since the initial CLB-level placement induced from FIP is more or less close to the optimal solution, we skip the wirelength-driven phase in Fig. 5 and directly apply the routability-driven phase to further reduce the runtime. To avoid global placement being stuck in the local optimal around FIP, the weight of pseudo-nets for cell spreading is reduced at the beginning of global placement.

B. Min-Cost Bipartite Matching-Based Legalization

A notable difference between ASIC and FPGA legalization is that ASIC standard cells have different dimensions, whereas FPGA CLBs have the same size. Because of this special property, FPGA legalization problem can be formulated as a min-cost-max-cardinality bipartite matching problem with pin movement as cost. By solving the corresponding bipartite matching problem, global placement can be legalized with minimum total pin movement. However, solving a complete bipartite matching for large designs is impractical in terms of runtime. To address this problem, we partition the placement region into a set of uniform rectangle partitions, then apply a min-cost bipartite matching for each partition. To further speed up runtime, edges in bipartite graphs are pruned based on Manhattan distance and are incrementally added when necessary.

Our legalization approach is summarized in Algorithm 5. The placement region is partition into a set of uniform rectangle regions at line 1. For each partition, we put all cells and all unoccupied sites into two sets at lines 3 and 4. To make sure each min-cost bipartite matching has enough sites to accommodate all cells, neighbor available sites are added when necessary from lines 5 to 7. The bipartite graph is constructed at line 8 with all the cells as left vertices and all the sites as right vertices in the graph. Edges between left vertices (cells) and right vertices (sites) are added from lines 12 to 17, and the edge pruning based on Manhattan distance is applied at the same time. The min-cost bipartite matching is solved at line 18, and cells are moved to their matched sites from lines 19 to 26. If no feasible solution is found, the maximum displacement constraint is increased at lines 27 and 28, and the loop from lines 11 to 29 is repeated. In UTPlaceF, we set partition width and height to 42 and 60, respectively, initial maximum displacement $D_{\max}$ to 4, and maximum displacement increasing rate $\Delta D_{\max}$ to 2.

Similar to CLBs, heterogeneous blocks like DSPs, RAMs, and I/Os also have the regularity of sizes, so they are legalized separately using Algorithm 5 with minor variations in UTPlaceF as well.

The time complexity of Algorithm 5 is $\mathcal{O}((|V|^3/|P|^2)\log(|V|/|P|)\log(|V|/|P|C))$, where V denotes the set of cells, P denotes the set of partitions, and C is the maximum cost value returned at line 14.

Algorithm 5 Min-Cost Bipartite Matching-Based Legalization

```
Require: Packing and global placement is done.
Require: Initial max displacement $D_{max}$ and max displacement
    increasing rate $\Delta D_{max}$
Ensure: Legalized placement with minimum total pin movement.
 1: Partition the placement region into a set of rectangle regions $P$
 2: for each $p \in P$ do
 3:     $L \leftarrow$ unlegalized CLBs in $p$
 4:     $R \leftarrow$ unoccupied sites in $p$
 5:     while $|L| > |R|$ do
 6:         Add the closest unoccupied site to $p$ into $R$
 7:     end while
 8:     Construct a $|L| \times |R|$ bipartite graph $g$ with left vertex set $L$
    and right vertex set $R$
 9:     $d_{min} \leftarrow 0$
10:     $d_{max} \leftarrow D_{max}$
11:     while true do
12:         for each $l \in L, r \in R$ do
13:             if $d_{min} \leq dist(l, r) < d_{max}$ then
14:                 $cost \leftarrow dist(l, r) \cdot numPins(l)$
15:                 Add edge $(l, r, cost)$ into $g$
16:             end if
17:         end for
18:         Run min-cost bipartite matching on $g$
19:         if number of matched edges $== |L|$ then
20:             for each matched edge $(l, r)$ do
21:                 Move $l$ to $r$'s location
22:                 Mark $l$ as legalized
23:                 Mark $r$ as unoccupied
24:             end for
25:             return
26:         end if
27:         $d_{min} \leftarrow d_{max}$
28:         $d_{max} \leftarrow d_{max} + \Delta D$
29:     end while
30: end for
```

The number of cells in each partition is $\mathcal{O}(|V|/|P|)$ on average. By assuming the loop from lines 5 to 7 can also be finished in $\mathcal{O}(|V|/|P|)$ time, the bipartite graph initialization from lines 3 to 8 can be done in $\mathcal{O}(|V|/|P|)$ time. Since the edge pruning complicates the complexity analysis, here we only consider the worst case, where $d_{\min} = 0$ and $d_{\max} =$ infinity at lines 9 and 10, respectively. Given this assumption, each bipartite graph has $\mathcal{O}(|V|/|P|)$ vertices and $\mathcal{O}(|V|^2/|P|^2)$ edges. As results, the time complexity of the edge construction from lines 12 to 17 is $\mathcal{O}(|V|^2/|P|^2)$. By applying network simplex algorithm [35], the min-cost bipartite matching at line 18 can be solved in $\mathcal{O}((|V|^3/|P|^3)\log(|V|/|P|)\log(|V|/|P|C))$ time. After that, moving cells to their matched sites from lines 19 to 26 takes $\mathcal{O}(|V|^2/|P|^2)$ time. Assembling all pieces together, the time complexity of legalizing one partition can be bounded by $\mathcal{O}((|V|^3/|P|^3)\log(|V|/|P|)\log((|V|/|P|)C))$ and the total time complexity of handling $|P|$ partitions turns out to be $\mathcal{O}((|V|^3/|P|^2)\log(|V|/|P|)\log((|V|/|P|)C))$.

It should be noted that, in practice, the number of edges in each bipartite graph is far less than $\mathcal{O}(|V|^2/|P|^2)$ due to our pruning technique, so the empirical runtime of Algorithm 5 is much faster than the above theoretical complexity bound.

C. Congestion-Aware Hierarchical Independent Set Matching

The idea of bipartite matching can also be applied to optimize wirelength. For a given set of legalized cells, a wirelength optimization problem can be formulated as a min-cost bipartite matching with edge weights as HPWL increase of moving cells to different sites. However, solving this matching problem cannot guarantee the optimal HPWL improvement, since the edge weight of a cell depends on the positions of other connected cells in the same matching set. To overcome this drawback, we adopt the ISM idea from NTUPlace3 [36] and only apply matching within a set of cells that do not share any nets. Besides, white spaces are also considered in our matching to further increase the solution space.

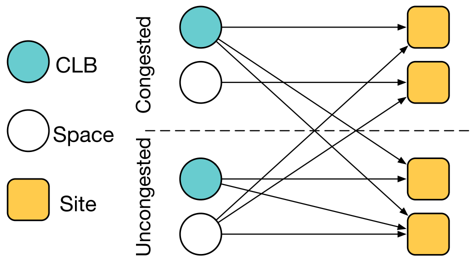


Fig. 11. Illustration of our congestion-aware ISM.

In UTPlaceF, ISM is hierarchically applied to CLBs, BLEs, and LUT pairs. One main objective of our packing stage (PCAP) is to absorb small nets into clusters (BLEs, CLBs). Therefore, most CLBs essentially are clusters of LUTs and FFs that have strong connectivity. One of our key observation is that moving cells with strong connectivity together helps to jump out of local optima in terms of wirelength, so ISM is applied to CLBs first in UPlaceF. However, even though most CLBs contain strongly connected cells, they are clusterd only based on physical distance and connectivity but are not aware of wirelength. Thus, ISM for BLEs and LUT pairs are introduced to fix our CLB packing and BLE packing, respectively, after CLB level ISM.

The ISM works effectively for optimizing HPWL. However, it could ruin the local cell density optimized for routability, especially when spaces are considered in our ISM. To mitigate this problem, we propose a congestion-aware ISM with three extra constraints introduced: 1) cells can be moved out of but not into routing congested regions; 2) spaces can be moved into but not out of congested regions; and 3) moves within congested regions are disallowed. Fig. 11 shows a simple matching example with the extra constraints applied. To get accurate congestion information, the routing congestion map is updated after a certain number of ISM iterations. By applying our congestion-aware ISM, HPWL can be optimized without routability degradation.

The pseudo-code of our congestion-aware ISM is summarized in Algorithm 6. Each independent set is generated at line 2 by calling function `GenerateIndepSet` from lines 16 to 29. The bipartite graph is constructed from lines 3 to 9. The cost of each edge is calculated by function `GetMovingCost` from 31 to 39, the extra routability rules are applied here as well. The min-cost bipartite matching is solved at line 10, and cells are moved to their matched locations from lines 11 to 13. This algorithm is sequentially executed for CLBs, BLEs, and LUT pairs to successively optimize wirelength in different levels. Note that for BLE ISM, the clock legality of CLBs must be preserved, so each independent

Algorithm 6 Congestion-Aware ISM

```
Require: Placement is legal.
Require: Routing utilization threshold $U_{th}$, maximum independent set size $N_{is}$, and maximum independent set radius $D_{is}$.
Ensure: Legalized placement with shorter wirelength and no routability degradation.
 1: for each $u \in V$ do
 2:     $S \leftarrow$ GenerateIndepSet($u$)
 3:     Construct a $|S| \times |S|$ min-cost bipartite graph $g$
 4:     for each $l \in$ left vertex set of $g$ do
 5:         for each $r \in$ right vertex set of $g$ do
 6:             $cost \leftarrow$ GetMovingCost($l, r$)
 7:             Add the edge $(l, r, cost)$ into $g$
 8:         end for
 9:     end for
10:     Run min-cost bipartite matching on $g$
11:     for each matched edge $(l, r)$ do
12:         Move $l$ to $r$'s location
13:     end for
14: end for
15:
16: function GENERATEINDEPSET($s$)
17:     $isIndep[u] \leftarrow true\ \forall u \in V$
18:     for each $u \in \{s\} \cup \{x \in V | dist(s, x) \le D_{is}\}$ do
19:         if $isIndep[u]$ then
20:             $S \leftarrow S \cup \{u\}$
21:             if $|S| \ge N_{is}$ then
22:                 return $S$
23:             end if
24:             for each $v$ connected to $u$ do
25:                 $isIndep[v] \leftarrow false$
26:             end for
27:         end if
28:     end for
29: end function
30:
31: function GETMOVINGCOST($l, r$)
32:     if $l$ is a cell and routing utilization at $r > U_{is}$ then
33:         return infinity
34:     end if
35:     if $l$ is a white space and routing utilization at $l > U_{is}$ then
36:         return infinity
37:     end if
38:     return HPWL increase of moving $l$ to $r$'s location
39: end function
```

set can only contain BLEs belong to the same clock net (and may also contain BLEs without clocks). In UTPlaecF, U_{th} is set to 0.7, N_{is} is set to 50, and D_{is} is set to 10 by default.

The time complexity of Algorithm 6 turns out to be $\mathcal{O}(|V|(D_{\text{is}}^2 + k^2(|V|/|E|)N_{\text{is}} + kN_{\text{is}}^2 + N_{\text{is}}^3\log N_{\text{is}}\log(N_{\text{is}}C)))$, where V and E are the set of cells and nets, respectively, k is the average number of pins in each cell in V, and C is the maximum noninfinity cost returned by GetMovingCost.

In each call of GenerateIndepSet, the number of iterations of the loop from lines 18 to 28 is bounded by $\mathcal{O}(D_{\text{is}}^2)$, but no more than N_{is} of them can execute the loop from lines 24 to 26. In addition, for each of these N_{is} iterations, line 25 is executed for $\mathcal{O}(k^2(|V|/|E|))$ times, since, on average, each cell is connected to k nets and each net have $k(|V|/|E|)$ cells. Thus, each GenerateIndepSet can be finished in $\mathcal{O}(D_{\text{is}}^2 + k^2(|V|/|E|)N_{\text{is}})$ time.

In our implementation, each call of GetMovingCost takes amortized $\mathcal{O}(k^2(|V|/|E|N_{\text{is}}) + k)$ time. During each bipartite matching run, the HPWL of $\mathcal{O}(kN_{\text{is}})$ nets could be changed. We first use brute-force approach to precompute the bounding boxes of these $\mathcal{O}(kN_{\text{is}})$ nets without considering cells in the independent set S, which takes $\mathcal{O}(k^2(|V|/|E|)N_{\text{is}})$ time. Then, for each of these nets, the HPWL change of moving any cell in S can be obtained in $\mathcal{O}(1)$ time. Consequently, the total HPWL change of moving any cell in S can be computed in $\mathcal{O}(k)$ time. Note that, in each bipartite graph construction from lines 3 to 9, the precomputation for net bounding boxes only needs to be done once but GetMovingCost is called $\mathcal{O}(N_{\text{is}}^2)$ times, so the amortized time complexity of GetMovingCost is $\mathcal{O}(k^2(|V|/|E|N_{\text{is}}) + k)$. The time complexity of each bipartite graph construction from lines 3 to 9 then can be bounded by $\mathcal{O}(k^2(|V|/|E|)N_{\text{is}} + kN_{\text{is}}^2)$ time.

By applying network simplex algorithm [35], each min-cost bipartite matching at line 10 can be solved in $\mathcal{O}(N_{\text{is}}^3\log N_{\text{is}}\log(N_{\text{is}}C))$ time. After that, moving cells to their matched sites (lines 11–13) can be done in $\mathcal{O}(N_{\text{is}}^2)$ time.

Combining all the analysis results, the time complexity of each iteration of the main loop (lines 1–14) is $\mathcal{O}(D_{\text{is}}^2 + k^2(|V|/|E|)N_{\text{is}} + kN_{\text{is}}^2 + N_{\text{is}}^3\log N_{\text{is}}\log(N_{\text{is}}C))$. Considering the main loop are executed $\mathcal{O}(|V|)$ times, we can conclude that the time complexity of Algorithm 6 is $\mathcal{O}(|V|(D_{\text{is}}^2 + k^2(|V|/|E|)N_{\text{is}} + kN_{\text{is}}^2 + N_{\text{is}}^3\log N_{\text{is}}\log(N_{\text{is}}C)))$.

TABLE I
ISPD'16 PLACEMENT CONTEST BENCHMARKS STATISTICS

Benchmark	#LUT	#FF	#RAM	#DSP	#Ctrl Set
FPGA-1	50K	55K	0	0	12
FPGA-2	100K	66K	100	100	121
FPGA-3	250K	170K	600	500	1281
FPGA-4	250K	172K	600	500	1281
FPGA-5	250K	174K	600	500	1281
FPGA-6	350K	352K	1000	600	2541
FPGA-7	350K	355K	1000	600	2541
FPGA-8	500K	216K	600	500	1281
FPGA-9	500K	366K	1000	600	2541
FPGA-10	350K	600K	1000	600	2541
FPGA-11	480K	363K	1000	400	2091
FPGA-12	500K	602K	600	500	1281
Resources	538K	1075K	1728	768	N/A

VI. Experimental Results

UTPlaceF was implemented in C++ and tested on a Linux machine with 3.40 GHz CPU and 32 GB RAM. The benchmark suite released by Xilinx for ISPD'16 FPGA placement contest was used to validate the effectiveness of UTPlaceF. Related executables, placement solutions, and benchmarks are released at link (http://www.wuxili.net/UTPlaceF.html).

A. Benchmark Characteristics

The characteristics of ISPD'16 benchmark suite are listed in Table I. This benchmark suite has cell count ranging from 0.1 to 1.1 million, which is much larger than existing academic FPGA benchmarks. Note that several benchmarks have extremely high cell utilization, which raises two requirements to FPGA placement packing and placement engines: 1) the capability to yield tight packing solutions to satisfy the CLB capacity constraint and 2) the capability to reduce routing resource demand, since little white space is available for cell and routing demand spreading.

TABLE II
COMPARISON WITH ISPD'16 CONTEST WINNERS ON ISPD 2016 BENCHMARK SUITE

Benchmark	1st Place		2nd Place		3rd Place		UTPlaceF	
	Routed WL	Runtime(s)	Routed WL	Runtime(s)	Routed WL	Runtime(s)	Routed WL	Runtime(s)
FPGA-1	PE*	N/A	379932	118	581975	97	356769	185
FPGA-2	677877	435	679878	208	1046859	191	642108	305
FPGA-3	3223042	1527	3660659	1159	5029157	862	3215087	831
FPGA-4	5628519	1257	6497023	1149	7247233	889	5409765	824
FPGA-5	10264769	1266	UR	N/A	UR	N/A	9659958	1237
FPGA-6	6630179	2920	7008525	4166	6822707	8613	6487628	1041
FPGA-7	10236827	2703	10415871	4572	10973376	9169	10104837	1721
FPGA-8	8384338	2645	8986361	2942	12299898	2741	7879022	1686
FPGA-9	UR†	N/A	13908997	5833	UR	N/A	12369055	2537
FPGA-10	PE	N/A	PE	N/A	UR	N/A	8794515	3182
FPGA-11	11091383	3227	11713479	7331	UR	N/A	10196038	2151
FPGA-12	9021769	4539	PE	N/A	UR	N/A	7755443	2944
Norm.	+6.2%	1.55×	+11.6%	2.30×	+29.1%	3.10×	+0.0%	1.00×

* PE: Placement error
† UR: Unroutable placement

TABLE III
COMPARISON WITH STATE-OF-THE-ART ACADEMIC FPGA PLACERS ON ISPD 2016 BENCHMARK SUITE

Benchmark	[26] *		RippleFPGA [27]		GPlace [28]		UTPlaceF	
	Routed WL	Runtime(s)	Routed WL	Runtime(s)	Routed WL	Runtime(s)	Routed WL	Runtime(s)
FPGA-1	384709	215	362563	74	493788	30	356769	185
FPGA-2	652690	399	677563	167	903099	61	642108	305
FPGA-3	3181331	1555	3617466	1037	3908244	289	3215087	831
FPGA-4	5504083	1289	6037293	621	6277878	280	5409765	824
FPGA-5	10068879	1237	10455204	1012	UR	N/A	9659958	1237
FPGA-6	6411247	2827	6960037	2772	7643382	600	6487628	1041
FPGA-7	10040562	2588	10248020	2170	11255351	691	10104837	1721
FPGA-8	8113483	2705	8874454	1426	9323360	734	7879022	1686
FPGA-9	13616625	3407	12954350	2683	14002965	974	12369055	2537
FPGA-10	8866049	4091	8564363	5555	UR	N/A	8794515	3182
FPGA-11	10834629	3267	11226088	3636	12367773	923	10196038	2151
FPGA-12	8246410	4625	8928528	9748	UR	N/A	7755443	2944
Norm.	+3.7%	1.47×	+7.3%	1.61×	+16.8%	0.38×	+0.0%	1.00×

* This is the preliminary version of UTPlcaeF that was published on ICCAD'16.

B. Comparison With Previous Works

We compare our results with the top three winners of ISPD'16 placement contest and other state-of-the-art FPGA placers. The results are shown in Tables II and III. All routed wirelength are reported by Xilinx Vivado v2015.4, and runtime of the contest winners are evaluated on a Linux Machine with 3.20 GHz CPU and 32 GB RAM. Normalized results in the last row of Tables II and III are based on comparisons with our results, and only benchmarks that other placers completed are considered in each comparison. It can be seen that UTPlaceF achieves the best overall routed wirelength. On average UTPlaceF outperforms by 6.2%, 11.6%, 29.1%, 3.7%, 7.3%, and 16.8% in routed wirelength compared with the top three contest winners, [26], RippleFPGA [27], and GPlace [28], respectively. It should be noted that only UTPlaceF and RippleFPGA are able to route all 12 benchmarks. In terms of runtime, as all placers are evaluated on different machines, it is not fair to compare them directly. However, we still can see that the runtime of UTPlaceF is only worse than GPlace, and is about 1.5× to 3.1× faster than other placers.

C. Runtime Analysis

The runtime breakdown of UTPlaceF is shown in Table IV. On average, 55.0% of the total runtime is taken by FIP, while PCAP, global placement, and hierarchical ISM, respectively, take 16.2%, 5.1%, and 21.4% of the total runtime, and legalization only takes 1.6% of the total runtime. PCAP is further divided into three components: 1) BLE packing takes 0.5% of the total runtime; 2) related CLB packing takes 2.3% of the total runtime; and 3) the remaining 13.4% is taken by the unrelated CLB packing. In hierarchical ISM, CLB, BLE, and LUT-pair level ISM, respectively, take 2.6%, 6.7%, and 12.1% of the total runtime.

D. Congestion-Aware Hierarchical Independent Set Matching Effectiveness Validation

To show the effectiveness of our proposed congestion-aware hierarchical ISM, we compared the routed wirelength of the intermediate placement solution after each ISM steps. Table V summarizes the experimental results. Compared with post-legalization solutions, placements after CLB ISM, BLE ISM, and LUT-Pair ISM are 2.4%, 3.7%, and 5.5% shorter in routed wirelength, respectively.

TABLE IV
RUNTIME BREAKDOWN OF UTPLACEF

Benchmark	FIP	PCAP			GP	Legalization	Hierarchical ISM			Others	Total
		BLE	Related CLB	Unrelated CLB			CLB	BLE	LUT Pair		
FPGA-1	115	1	2	3	9	1	5	26	22	1	185
FPGA-2	187	2	4	4	17	1	8	38	42	2	305
FPGA-3	528	5	12	13	58	1	28	57	120	9	831
FPGA-4	500	6	12	17	56	2	33	61	127	10	824
FPGA-5	663	7	13	23	77	3	41	67	136	11	1041
FPGA-6	1029	8	44	171	101	30	49	77	197	15	1721
FPGA-7	974	9	52	195	126	19	54	81	205	16	1731
FPGA-8	1004	9	26	39	91	16	49	174	274	4	1686
FPGA-9	1217	13	100	538	119	41	60	116	314	19	2537
FPGA-10	1343	11	65	1154	95	59	60	148	229	18	3182
FPGA-11	1248	12	50	133	115	23	55	203	308	4	2151
FPGA-12	1720	12	66	279	116	107	54	226	346	18	2944
Norm.	55.0%	0.5%	2.3%	13.4%	5.1%	1.6%	2.6%	6.7%	12.1%	0.7%	100.0%

TABLE V
ROUTED WIRELENGTH AT DIFFERENT STAGES OF THE CONGESTION-AWARE HIERARCHICAL ISM

Benchmark	Post Legalization Routed WL	Post CLB ISM Routed WL	Post BLE ISM Routed WL	Post LUT-Pair ISM Routed WL
FPGA-1	406555	394462	359273	356769
FPGA-2	702458	676831	646553	642108
FPGA-3	3339112	3264313	3236204	3215087
FPGA-4	5602968	5484237	5433021	5409765
FPGA-5	9908463	9696556	9641323	9659958
FPGA-6	6823796	6685567	6613140	6487628
FPGA-7	10499688	10322086	10252708	10104837
FPGA-8	8155719	7992101	7913338	7879022
FPGA-9	13181397	12936690	12834049	12369055
FPGA-10	9916422	9633335	9284655	8794515
FPGA-11	10687229	10332421	10306784	10196038
FPGA-12	8479546	8208020	7956163	7755443
Norm.	+0.0%	-2.4%	-3.7%	-5.5%

VII. CONCLUSION

With the utilization of FPGA designs being pushed to the upper limit, routability optimization is becoming a fundamental issue in modern physical design flow for FPGA. In this paper, we have proposed a routability-driven FPGA packing and placement engine called UTPlaceF. A novel packing algorithm PCAP and a congestion-aware detailed placement techniques for wirelength and routability co-optimization are proposed. The experimental results show that UTPlaceF achieves high-quality packing and placement solutions, which outperform the top three winners of the ISPD'16 placement contest and other state-of-the-art FPGA placers. Our future work includes two directions: 1) exploring techniques to parallelize and speed up UTPlaceF framework and 2) enhancing UTPlaceF to accommodate more complicated placement and packing constraints imposed by modern FPGA architectures, e.g., clock network and I/O block constraints.

REFERENCES

[1] *Xilinx Inc.* Accessed on Mar. 17, 2017. [Online]. Available: http://www.xilinx.com

[2] V. Betz and J. Rose, "Cluster-based logic blocks for FPGAs: Area-efficiency vs. input sharing and size," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Santa Clara, CA, USA, 1997, pp. 551–554.

[3] A. S. Marquardt, V. Betz, and J. Rose, "Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density," in *Proc. ACM Symp. FPGAs*, Monterey, CA, USA, 1999, pp. 37–46.

[4] E. Bozorgzadeh, S. Ogrenci-Memik, and M. Sarrafzadeh, "RPack: Routability-driven packing for cluster-based FPGAs," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, Yokohama, Japan, 2001, pp. 629–634.

[5] A. Singh, G. Parthasarathy, and M. Marek-Sadowska, "Efficient circuit clustering for area and power reduction in FPGAs," *ACM Trans. Design Autom. Electron. Syst.*, vol. 7, no. 4, pp. 643–663, 2002.

[6] H. Liu and A. Akoglu, "Timing-driven nonuniform depopulation-based clustering," *Int. J. Reconfig. Comput.*, vol. 2010, Jan. 2010, pp. 1–11.

[7] S. T. Rajavel and A. Akoglu, "MO-pack: Many-objective clustering for FPGA CAD," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, New York, NY, USA, 2011, pp. 818–823.

[8] Z. Marrakchi, H. Mrabet, and H. Mehrez, "Hierarchical FPGA clustering based on multilevel partitioning approach to improve routability and reduce power dissipation," in *Proc. Int. Conf. Reconfig. Comput. FPGAs (ReConFig)*, 2005, pp. 25–28.

[9] W. Feng, "K-way partitioning based packing for FPGA logic blocks without input bandwidth constraint," in *Proc. IEEE Int. Conf. Field Program. Technol. (FPT)*, Seoul, South Korea, 2012, pp. 8–15.

[10] D. T. Chen, K. Vorwerk, and A. Kennings, "Improving timing-driven FPGA packing with physical information," in *Proc. IEEE Int. Conf. Field Program. Logic Appl. (FPL)*, Seoul, South Korea, 2007, pp. 117–123.

[11] K. Vorwerk and A. Kennings, "An improved multi-level framework for force-directed placement," in *Proc. IEEE/ACM Design Autom. Test Europe (DATE)*, Munich, Germany, 2005, pp. 902–907.

[12] R. Tessier and H. Giza, "Balancing logic utilization and area efficiency in FPGAs," in *Proc. IEEE Int. Conf. Field Program. Logic Appl. (FPL)*, 2000, pp. 535–544.

[13] M. Tom and G. Lemieux, "Logic block clustering of large designs for channel-width constrained FPGAs," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Anaheim, CA, USA, 2005, pp. 726–731.

[14] M. Tom, D. Leong, and G. Lemieux, "Un/DoPack: Re-clustering of large system-on-chip designs with interconnect variation for low-cost FPGAs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2006, pp. 680–687.
[15] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Proc. IEEE Int. Conf. Field Program. Logic Appl. (FPL)*, 1997, pp. 213–222.
[16] G. Chen and J. Cong, "Simultaneous timing driven clustering and placement for FPGAs," in *Proc. IEEE Int. Conf. Field Program. Logic Appl. (FPL)*, 2004, pp. 158–167,
[17] G. Chen and J. Cong, "Simultaneous placement with clustering and duplication," *ACM Trans. Design Autom. Electron. Syst.*, vol. 11, no. 3, pp. 740–772, 2006.
[18] P. Maidee, C. Ababei, and K. Bazargan, "Fast timing-driven partitioning-based placement for island style FPGAs," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Anaheim, CA, USA, 2003, pp. 598–603.
[19] P. Maidee, C. Ababei, and K. Bazargan, "Timing-driven partitioning-based placement for island style FPGAs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 3, pp. 395–406, Mar. 2005.
[20] Y. Xu and M. A. S. Khalid, "QPF: Efficient quadratic placement for FPGAs," in *Proc. IEEE Int. Conf. Field Program. Logic Appl. (FPL)*, Tampere, Finland, 2005, pp. 555–558.
[21] P. Gopalakrishnan, X. Li, and L. Pileggi, "Architecture-aware FPGA placement using metric embedding," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2006, pp. 460–465.
[22] M. Xu, G. Gréwal, and S. Areibi, "StarPlace: A new analytic method for FPGA placement," *Integr. VLSI J.*, vol. 44, no. 3, pp. 192–204, 2011.
[23] M. Gort and J. H. Anderson, "Analytical placement for heterogeneous FPGAs," in *Proc. IEEE Int. Conf. Field Program. Logic Appl. (FPL)*, Oslo, Norway, 2012, pp. 143–150.
[24] T.-H. Lin, P. Banerjee, and Y.-W. Chang, "An efficient and effective analytical placer for FPGAs," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, 2013, pp. 1–6.
[25] Y.-C. Chen, S.-Y. Chen, and Y.-W. Chang, "Efficient and effective packing and analytical placement for large-scale heterogeneous FPGAs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2014, pp. 647–654.
[26] W. Li, S. Dhar, and D. Z. Pan, "UTPlaceF: A routability-driven FPGA placer with physical and congestion aware packing," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Austin, TX, USA, 2016, pp. 1–7.
[27] C.-W. Pui *et al.*, "RippleFPGA: A routability-driven placement for large-scale heterogeneous FPGAs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Austin, TX, USA, 2016, pp. 1–8.
[28] R. Pattison, Z. Abuowaimer, S. Areibi, G. Gréwal, and A. Vannelli, "GPlace: A congestion-aware placement tool for ultrascale FPGAs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Austin, TX, USA, 2016, pp. 1–7.
[29] *ISPD 2016 Routability-Driven FPGA Placement Contest*. Accessed on Mar. 17, 2017. [Online]. Available: http://www.ispd.cc/contests/16/ispd2016_contest.html
[30] T. Lin and C. C. Chu, "POLAR 2.0: An effective routability-driven placer," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2014, pp. 1–6.
[31] M.-C. Kim, D.-J. Lee, and I. L. Markov, "SimPL: An effective placement algorithm," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 1, pp. 50–60, Jan. 2012.
[32] T. Lin, C. C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev, "POLAR: Placement based on novel rough legalization and refinement," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2013, pp. 357–362.
[33] W.-H. Liu, Y.-L. Li, and C.-K. Koh, "A fast maze-free routing congestion estimator with hybrid unilateral monotonic routing," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2012, pp. 713–719.
[34] G.-J. Nam, S. Reda, C. J. Alpert, P. G. Villarrubia, and A. B. Kahng, "A fast hierarchical quadratic placement algorithm," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 4, pp. 678–691, Apr. 2006.
[35] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, 1993.
[36] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 7, pp. 1228–1240, Jul. 2008.

Wuxi Li received the B.S. degree in microelectronics from Shanghai Jiao Tong University, Shanghai, China, in 2013. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA.

His current research interest includes physical design automation for FPGAs.

Mr. Li was a recipient of the 1st-place Awards in the FPGA placement contests of ISPD 2016 and 2017.

Shounak Dhar received the B.Tech. degree in electrical engineering from the Indian Institute of Technology Bombay, Mumbai, India, in 2014. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA.

He has published two papers as first author and holds one U.S. patent. His current research interests include placement, routing, and CAD for FPGAs.

Mr. Dhar was a recipient of the 1st-place Award in the FPGA placement contest of ISPD 2016.

David Z. Pan (S'97–M'00–SM'06–F'14) received the B.S. degree from Peking University, Beijing, China, and the M.S. and Ph.D. degrees from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA.

From 2000 to 2003, he was a Research Staff Member with IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA. He is currently the Engineering Foundation Professor with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA. He has published over 280 technical papers, and is the holder of eight U.S. patents. He has graduated 21 Ph.D. students who are now holding key academic and industry positions. His current research interests include cross-layer nanometer IC design for manufacturability, reliability, security, physical design, analog design automation, and CAD for emerging technologies.

Dr. Pan was a recipient of number of awards for his research contributions, including the Semiconductor Research Corporation (SRC) 2013 Technical Excellence Award, the Design Automation Conference (DAC) Top 10 Author in Fifth Decade, the DAC Prolific Author Award, the ASP-DAC Frequently Cited Author Award, the 14 Best Paper Awards at Premier Venues [International Symposium on Hardware Oriented Security and Trust 2017, International Society for Optics and Photonics (SPIE) 2016, ISPD 2014, International Conference on Computer-Aided Design (ICCAD) 2013, Asia and South Pacific Design Automation Conference (ASPDAC) 2012, ISPD 2011, IBM Research 2010 Pat Goldberg Memorial Best Paper Award, ASPDAC 2010, Design Automation and Test in Europe 2009, International Conference on Integrated Circuit Design and Technology 2009, and SRC Techcon in 1998, 2007, 2012, and 2015] plus 11 additional Best Paper Award nominations at DAC/ICCAD/ASPDAC/ISPD, Communications of the ACM Research Highlights in 2014, the ACM/SIGDA Outstanding New Faculty Award in 2005, the NSF CAREER Award in 2007, the SRC Inventor Recognition Award three times, the IBM Faculty Award four times, the UCLA Engineering Distinguished Young Alumnus Award in 2009, the UT Austin Recognizing Asian and Asian American Faculty and Staff Instilling Strength and Excellence Faculty Excellence Award in 2014, and many international CAD contest awards, among others. He has served as a Senior Associate Editor for *ACM Transactions on Design Automation of Electronic Systems*, an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II, EEE DESIGN & TEST, *Science China Information Sciences*, the *Journal of Computer Science and Technology*, and IEEE Circuits and Systems Society Newsletter. He has served in the Executive and Program Committees of many major conferences, including DAC, ICCAD, ASPDAC, and ISPD. He is the ASPDAC 2017 Program Chair, ICCAD 2018 Program Chair, DAC 2014 Tutorial Chair, and ISPD 2008 General Chair. He is a fellow of SPIE.