Finding Placement-Relevant Clusters With Fast Modularity-Based Clustering

Mateus Fogaça[+], Andrew B. Kahng[†‡], Ricardo Reis[+*], Lutong Wang[‡]

[†]CSE and [‡]ECE Departments, UC San Diego, La Jolla, CA, USA
[+]PGMicro/[*]PPGC, Instituto de Informática, Universidade Federal do Rio Grande do Sul
{abk, luw002}@ucsd.edu, {mpfogaca, reis}@inf.ufrgs.br

ABSTRACT

In advanced technology nodes, IC implementation faces increasing design complexity as well as ever-more demanding design schedule requirements. This raises the need for new *decomposition* approaches that can help reduce problem complexity, in conjunction with new *predictive* methodologies that can help avoid bottlenecks and loops in the physical implementation flow. Notably, with modern design methodologies it would be very valuable to better predict final placement of the gate-level netlist: this would enable more accurate early assessment of performance, congestion and floorplan viability in the SOC floorplanning/RTL planning stages of design. In this work, we study a new criterion for the classic challenge of VLSI netlist clustering: how well netlist clusters "stay together" through final implementation. We propose use of several evaluators of this criterion. We also explore the use of *modularity*-driven clustering to identify natural clusters in a given graph without the tuning of parameters and size balance constraints typically required by VLSI CAD partitioning methods. We find that the netlist hypergraph-to-graph mapping can significantly affect quality of results, and we experimentally identify an effective recipe for weighting that also comprehends topological proximity to I/Os. Further, we empirically demonstrate that modularity-based clustering achieves better correlation to actual netlist placements than traditional VLSI CAD methods (our method is also 4× faster than use of *hMetis* for our largest testcases). Finally, we show a potential flow with fast "blob placement" of clusters to evaluate netlist and floorplan viability in early design stages; this flow can predict gate-level placement of 370K cells in 200 seconds on a single core.

ACM Reference Format:
Mateus Fogaça[+], Andrew B. Kahng[†‡], Ricardo Reis[+*], Lutong Wang[‡]. 2019. Finding Placement-Relevant Clusters With Fast Modularity-Based Clustering. In *ASPDAC '19: 24th Asia and South Pacific Design Automation Conference (ASPDAC '19), January 21–24, 2019, Tokyo, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3287624.3287676

ASPDAC '19, January 21–24, 2019, Tokyo, Japan

ACM ISBN 978-1-4503-6007-4/19/01…$15.00
https://doi.org/10.1145/3287624.3287676

1 INTRODUCTION

Modern Systems-on-Chip (SoCs) aggregate billions of transistors within a single die, and drivers ranging from mobility to deep learning suggest that the Moore's-Law scaling of design complexity will continue [39]. EDA tools are continually challenged to incorporate new strategies to scale tool capacity without sacrificing quality of results or overall design schedule. Moreover, despite substantial R&D investments by the EDA industry, costs of IC design (engineers, tools, schedule) continue to rise. A recent keynote by Olofsson [27] asks, "Has EDA failed to keep up with Moore's Law?"

It is well-known that the ability to predict downstream outcomes of physical implementation algorithms and tools can enable reduction of loops (iterations) in the design flow, thus saving runtime and schedule [20]. The paradigm of physical synthesis is still the major success story along such lines, but this paradigm is now over two decades old. The recent DARPA Intelligent Design of Electronic Assets (IDEA) program [38] highlights the cost crisis of modern IC design, and seeks to develop a framework capable of performing the complete RTL-to-GDSII flow without human interaction in 24 hours [27] [38]. New tools that can help avoid future failures (congestion, failed timing, etc.) while still in the early stages of floorplan definition or RTL planning appear mandatory to achieve the IDEA program goal.[1]

In this work, we seek to identify clusters of logic in a given gate-level netlist *that will remain together throughout the physical implementation flow.* (As discussed below, this is a fundamentally different criterion than the min-cut or Rent-parameter criteria of previous clustering methods in VLSI CAD). We envision that such a clustering capability will help enable new predictors of performance and congestion during early physical floorplanning and RTL planning. For example, gates within the same cluster would be known to have spatial locality, informing synthesis, budgeting and global interconnect planning optimizations. And, if combined with "blob placement" of clusters, fast evaluation of netlist and floorplan viability could be achieved.

Limitations of existing clustering approaches. Clustering is a universal strategy for problem size reduction and for helping to enforce "known-correct" structure in solutions. Clustering has been used for many years in a wide range of EDA applications, including placement [29], clock tree synthesis [30] and, more recently, grouping of instances into different power domains [3]. While many

[1]This is a long-standing challenge to design productivity and the EDA industry. That so many commercial RTL planning and "RTL signoff" efforts have been made over the past 25 years (Tera Systems, Aristo, Silicon Perspective, Atrenta SpyGlass-Physical, Oasys, etc.) indicates the difficulty of this challenge.

clustering methods for VLSI have been proposed, they have largely focused on *net cuts* (hyperedge min-cut, cluster perimeter, Rent parameter [41], etc.). Further, existing heuristics typically require design-dependent tuning and suboptimal heuristics. For instance, the well-known multilevel Fiduccia-Mattheyses [9] implementations hMetis [21] and MLPart [4] require *a priori* the target number of partitions as an input, and each aims to balance the number of vertices or total vertex area across the partitions, which conflicts with the min-cut objective.

Our approach. Among the contributions of this work, we mention two broad aspects. The first aspect is the evaluation and application of *community detection* algorithms within the VLSI CAD context. Community detection is a comparatively recent class of graph clustering methods used to find densely-connected nodes in large networks such as those arising in social media, telecommunications and bioinformatics [12]. Community detection methods rely on metrics that help identify natural clusters inside graphs, notably, the *modularity* metric [26]. Our study centers on Louvain [2], a well-known fast and efficient modularity-based graph clustering algorithm with near-linear runtime in sparse graphs. Louvain can cluster graphs with up to 700M edges within 12 minutes, using a single thread. The second aspect is our study of new measures of the correlation between a netlist clustering method and the actual placement of netlists. The absence of previous work in this vein may be due to the fact that previous clustering techniques have aimed to drive placement algorithms instead of predicting them (i.e., the final evaluation of a clustering technique was the quality of the placement itself). We study three classical concepts from computational geometry to evaluate this correlation: *convex hulls* (CH), *alpha shapes* (AS), and *Delaunay triangulations* (DT) [25]. The primary purpose of these techniques is to retrieve the geometric shape of a set of scattered points, a goal that correlates very closely to the concept of a cluster. To compare different clustering results, we apply the Davies–Bouldin index (DBi) [7], which is traditionally used to evaluate clusters given a "distance function". For spatial data, such as placement of standard-cell instances, this function can be the distance between instances in the placement.

Our contributions are summarized as follows.

(1) We employ modularity-based clustering in conjunction with VLSI-relevant graph edge-weighting to predict groups of logic gates that will remain together through the stages of physical implementation – without the need for user tuning.
(2) We explore the use of convex hulls, alpha shapes, and Delaunay triangulations to visualize and measure the correlation between the netlist clustering and the "ground-truth" actual placement. We also adopt the Davies–Bouldin index [7] to compare different clustering results.
(3) We perform experiments showing 20% better clustering quality on average for Louvain [2] versus the traditional VLSI netlist clustering tool hMetis [21], with 4× faster runtime than hMetis for our largest benchmark.
(4) We demonstrate an experimental flow that performs fast "blob placement" of clusters as a potential basis for future early-stage netlist and floorplan evaluation. Our flow can closely predict the actual gate-level placement of the leon3mp testcase (370K instances) in 200 seconds.

The remainder of this paper is organized as follows. Section 2 gives an overview of the existing literature on VLSI netlist clustering and discusses several works on modularity-based clustering. Section 3 formally defines our objective, metrics, and experimental implementation details, while Section 4 presents our experimental results. Section 5 introduces the idea of quick floorplan and placement evaluation using (modularity-based) clusters. Finally, Section 6 gives conclusions and several directions for our ongoing and future work.

2 RELATED WORKS

We now give a brief sampling of relevant works in two literatures: VLSI netlist partitioning, and community detection.

2.1 VLSI Netlist Partitioning

Netlist partitioning is a fundamental step within a broad spectrum of EDA tools. Alpert and Kahng [1] give a four-way classification of techniques according to underlying computational approach, as follows.

Move-based approaches aim to improve an initial feasible solution through iterative local perturbations such as pair-swap or shifting a single vertex to another partition. The pass-based heuristic structure of Kernighan-Lin (KL) [22] along with the vertex-shifting move structure of Fiduccia-Mattheyses (FM) [9] are at the core of such methods.

Geometric representation-based approaches exploit geometric embeddings of circuits to achieve improved cluster quality and runtime. Hall [15] gives an early spectral approach, achieving multiway partitioning solutions through quadratic placement and vertex orderings induced from eigenvectors of a netlist-derived discrete Laplacian matrix. Ou and Pedram [28] propose a two-phase min-cut strategy that comprehends timing constraints. Iterated quadratic programming is used to find an initial embedding of the design, and gate replication is subsequently applied if timing constraints are found to be too strict.

Combinatorial formulations encompass techniques such as network flows and mixed integer-linear programming that can capture complex objective functions and constraints. E.g., Yang and Wong [37] propose an iterated max-flow formulation to address the balanced bipartition problem. More recently, Blutman et al. [3] address netlist partitioning for stacked-domain designs, also using a flow-based framework.

Clustering approaches are often taxonomized as being either bottom-up or top-down. Bottom-up methods start with each module being an individual cluster, with clusters being iteratively merged until a given condition is satisfied. Top-down methods start with a single cluster and iteratively split clusters into two or more (smaller) clusters. Hybrid methods, awareness of timing and other concerns, etc. abound. E.g., Hagen and Kahng [14] perform clustering by integrating a random-walk algorithm with iterative FM. Sze and Wang [34] use a graph contraction technique to maintain delay information among different lower levels of a performance-driven clustering flow. Alternatively, Kahng and Xu [19] extend traditional FM to directly eliminate or minimize "distance-k V-shaped nodes" in the bipartitioning solution, achieving a tradeoff between cutsize and path delay.

2.2 Community Detection

The size of hypergraph and graph instances has been steadily increasing not only in VLSI netlists, but also in fields such as data science, social networks, and bioinformatics. This has led to the development of new algorithms that can quickly process and cluster huge graphs. Shiokawa and Onizuka [33] taxonomize such *community detection* algorithms into three categories: *edge-cut* based, *modularity*-based, and structural similarity. The edge-cut based category comprehends the same techniques as *move-based approaches* in the taxonomy of [1], so we do not repeat the discussion of Section 2.1.

Modularity-based clustering attempts to overcome a fundamental deficiency – namely, lack of any theoretical underpinning – of the many previous techniques that perform ad hoc minimization of the number of edges crossing between clusters, followed by improvement of clustering solutions with various heuristics. The seminal *modularity* metric proposed by Newman and Girvan [26] measures the quality of the current solution using the difference between the current solution and a random graph. Moreover, as noted above, many classical approaches assume or require a predefined and fixed number of clusters. Blondel et al. [2] propose an effective modularity-based method, called Louvain, which can *automatically* cluster a 700M-edge graph into a *natural* set of clusters, within 12 minutes. These traits make Louvain an attractive technique, applied today in many applications such as social media [33]. In [31], Shiokawa et al. describe another fast modularity-based algorithm employing incremental segregation; this method can cluster a hundred million-node graph in under five minutes.

Structural Similarity methods address a weakness of modularity-based clustering known as *resolution*, i.e., that modularity-based clustering may fail to detect small clusters in large graphs. The challenge of resolution was first noted by Fortunato and Barthélemy [11]. Xu et al. [36] propose the *structural similarity* metric which enables finding of densely connected clusters, special role nodes, hubs and outliers. The SCAN++ [32] algorithm can comprehend these topological characteristics with runtime that is linear in the number of graph edges. However, the computational cost of structural similarity is higher than that of modularity [33].

In our present work, we employ the widely-used VLSI clustering tool hMetis [21], and we apply the fast and effective modularity-based Louvain algorithm in the EDA context. In applying Louvain, a key issue is that VLSI netlists are hypergraphs, while community detection methods have been applied to graphs. As we discuss below, the success of modularity-based clustering for VLSI strongly depends on (i) the *hypergraph-to-graph mapping* used, and (ii) means of capturing structural 'hints' (I/Os, timing, etc. - cf. [5]) from the VLSI netlist structure.

3 METHODOLOGY

In this section, we first describe the problem statement and metrics for clustering evaluation. We then describe Louvain-based clustering based on a graph model of the netlist hypergraph.

3.1 Problem Definition

In this work, we use the term *cluster* to refer to a group of densely-connected instances such that the number of the interconnections among elements inside the group is much higher than the number of connections spanning different groups. The process of finding the clusters of a netlist is called *clustering*. Our goal may be stated as follows: *Given* (i) a mapped netlist and (ii) information about the standard cell library, *find* clusters containing instances that are expected to remain close to each other along the stages of the implementation flow. Since there is no formal definition of what is the nature of a good clustering to predict placement, we propose and discuss metrics below. The notations used in this section are summarized in Table 1.

Table 1: Notations.

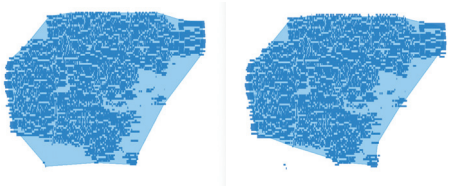
Term	Meaning
DBi	Davis-Bouldin metric
n	Number of clusters
σ_i	Average distance from the cluster elements to the centroid of cluster i
ρ_i	Centroid of cluster i
$l(\rho_i, \rho_j)$	Distance between the centroids of two clusters i and j
Q	Modularity value
A_{ij}	Sum of weights of inter-cluster edges between clusters i and j
k_i	Sum of all weights of edges connected to cluster i
c_i	Cluster of index i
$\delta(c_i, c_j)$	Function that receives as input two clusters and returns 1 if they are connected, and 0 otherwise
p_h	Number of pins of net h
w_h	Weight of net h
$w_{h,1}$	Clique weight of net h
$w_{h,2}$	Topological depth weight of net h
$d(I)$	Topological distance to the closest input
$d(O)$	Topological distance to the closest output

3.2 Metrics for Clustering Evaluation

In this subsection, we describe approaches to define cluster *shapes*, as well as clustering evaluation metrics.

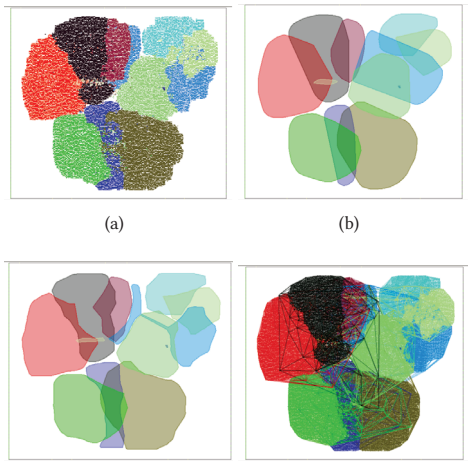
Cluster shapes. One intuitive approach to measure the correlation between the clusters and their actual placement is to retrieve their shapes for visualization and density measurement. In computational geometry, many applications need to restore the geometry from a set of scattered points. If we consider each cell as a singular point, the problems become very similar. We can represent the geometry of a given cluster using its convex hull [25], i.e., the minimum convex polygon that contains the center of all cells. Once the convex hull is computed, we calculate its *utilization* as the total cell area divided by the hull area. If the utilization is lower than a threshold, we remove the points comprising the hull and recompute the hull. In this work, we define a threshold of 64% utilization and set the maximum number of times the process can repeat as 25. We call this process "shelling" and depict an example in Figure 1. Figure 2(a) depicts a "ground-truth" placement along with a cluster, with cells colored according to their clusters. Figure 2(b) draws the corresponding convex hulls. However, if we examine the highlighted blue cluster in Figure 2(b), we see that convex hulls do not offer a compelling prospect. The hull fails to convey the bad clustering outcome and has a low utilization of 38%.

Alpha shapes [40] [8], examples of which are shown in Figure 2(c), are a type of "shape formed by a pointset" wherein a parameter *alpha* defines the squared radius of a circle that is used to carve away space around the given points. The remaining space



(a) (b)

Figure 1: The process of "shelling" the cluster shape. Figure (a) shows a cluster with total cell area equal to $24.2 \times 10^6 \mu m^2$ and shape area equal to $39.8 \times 10^6 \mu m^2$. Thus, the utilization of the cluster is equal to 60.7%. The cluster's "shell" is the set of points that compose the shape. In (b), the cluster shape is recomputed after removing the shell from (a). The final shape has area equal to $36 \times 10^6 \mu m^2$ and utilization equal to 66.7%.



(a) (b)

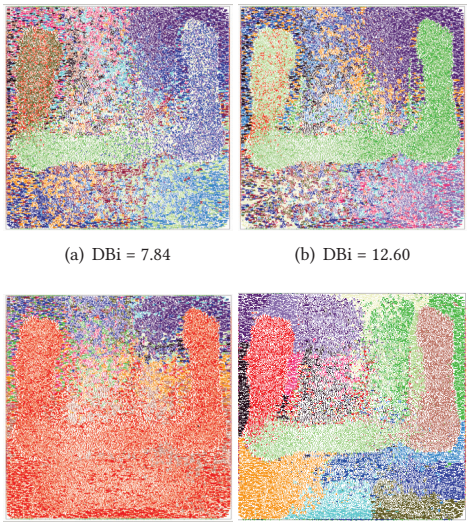
(c) (d)

Figure 2: Different approaches to correlate clusters with the placement for the circuit ispd18_test2 [24]: (a) the placement with each instance colored according to its cluster, followed by (b) the convex hulls; (c) the alpha shapes; and (d) the Delaunay triangulations of the clusters.

comprises the alpha shape of the pointset.[2] Alpha shapes are appealing in that – for appropriately chosen alpha – they provide more accurate representations of pointsets than do convex hulls. In the following, for the testcases we study, where dimensions of layout regions are in the $150\mu m$ to $500\mu m$ range, we empirically use alpha = $2500\mu m^2$. In Figure 2(c), we see that the alpha shape reveals how the blue cluster discussed earlier is clearly divided into two pieces, each of which is dense with utilization of ~66%.

Solution Evaluation. Convex hulls and alpha shapes are useful for visual and manual debugging. For solution evaluation, we propose two criteria. The first criterion is derived from the Delaunay triangulation (DT), depicted in Figure 2(d). The DT is the geometric

[2]When alpha = ∞, the alpha shape is the convex hull of the pointset (i.e., the convex hull is a special case of alpha shape). When alpha = 0, the alpha shape is the set of points of the pointset.



(a) DBi = 7.84 (b) DBi = 12.60

(c) DBi = 35.86 (d) DBi = 2.29

Figure 3: Visual comparison of different clustering solutions, with and their DBi values indicated.

dual of the Voronoi diagram over a given pointset. From the DT, we extract statistical data from the distribution of edge sizes. For our second criterion, recall that the main goal of this work is to predict groups of logic gates that will remain together through the stages of physical implementation. This goal correlates well with the goal of spatial clustering techniques. Therefore, we adopt the the Davies–Bouldin index (DBi) [7], traditionally used for spatial clustering evaluation, as a second indicator of cluster quality. The DBi is defined as:

$$DBi = \frac{1}{n} \sum_{i=1}^{n} argmax_{i \neq j} \left(\frac{\sigma_i + \sigma_j}{l(\rho_i, \rho_j)} \right) \quad (1)$$

The DBi consists of a numerical value that indicates how well-clustered is a given set of elements in a spatial region. A smaller value of DBi indicates a better clustering.

We illustrate the DT and DBi quality criteria using the four clustering solutions in Figure 3 with 18 clusters each. The distributions of DT edges are shown in Figure 4. Following its premise, DT enables a good way to analyze the edge length distribution of the clusters but fails to capture the gap in partition sizes from Figure 3(c). The largest partition in Figure 3(c) has 22K cells in contrast to the average of 5K in the other solutions. The values of DBi are as expected from the visualization perspective, where a smaller value of DBi indicates more distinct clusters as shown in Figure 3(d).

3.3 Modularity-based Clustering

The *modularity* metric [26] measures the quality of a clustering solution given a network graph and the set of clusters. It consists of a scalar value ranging from -1 to 1; higher values imply better clustering quality. The modularity metric is formally expressed as

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \quad (2)$$

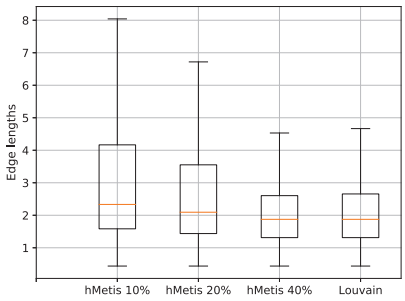


Figure 4: The boxplot of edge sizes from the DT of the clustering solutions from Figure 3.

where the value of m is computed as $m = \frac{1}{2}\sum_{ij} A_{ij}$.

Many methods, such as the Louvain algorithm, apply modularity as an objective function. As previously mentioned, our present work applies Louvain to perform modularity-based clustering of netlists.[3] This is in contrast to the existing VLSI clustering literature of essentially because of two features:

- The user does not need to calibrate the number of clusters, nor define any stopping criteria for clustering, since these are automatically captured by the modularity metric; and
- The Louvain algorithm does not impose, nor require, any area/edge balancing constraints.

3.4 Graph Model of Netlist

In most of the optimization steps, the netlist is expressed as a direct hypergraph $G = (V, E)$, where V is the set of vertices that represent the instances and E is the set of the direct hyperedges that represent the nets. Some techniques, such as Louvain, cannot handle the notion of hyperedges. Consequently, a translation method to a given netlist representing a hypergraph by a weighted graph is needed. The clique model is often used in a variety of applications. The clique model replaces the hyperedge by a complete graph, i.e., every pair of vertices is connected by a single edge. To "correctly represent" nets of different sizes, edge weighting techniques are required. Ihler et al. [17] prove that there is no perfect weighting for the clique model, but previous works frequently use edge weights as $w_h = 1/(p_h - 1)$, where p_h is the number of pins in the net. However, using the traditional clique decomposition is usually not enough to capture all the nuances necessary to match the clustering with actual placement. Our experiments show that giving higher weights to edges closer to I/O pins improves the quality of the clustering. Therefore, we also add a weighting scheme based on topological depth aiming to keep cells closer to I/Os in the same cluster. Specifically, we define the edge weights as:

$$w_{h,1} = \frac{1}{p_h - 1} \tag{3}$$

$$w_{h,2} = argmin((d(I)), (d(O))) \tag{4}$$

[3]We note that applying the modularity metric within classic VLSI partitioning methods would lose the "automatic" qualities inherent in the Louvain algorithm. In this sense, our work separately benefits from use of the modularity metric *and* use of the Louvain algorithm.

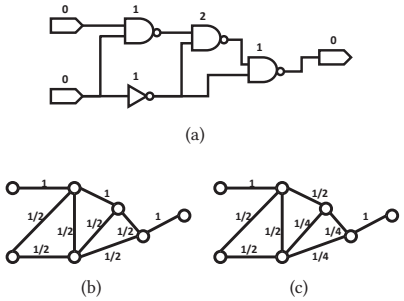


Figure 5: Netlist modeling.

Table 2: Benchmarks and attributes.

Benchmark	Insts	Nets	I/Os
jpeg_encoder	42293	46402	49
ldpc	43965	46741	4100
netcard	251306	253145	333
leon3mp	371549	370706	1849

$$w_h = \frac{1}{w_{h,1}(w_{h,2} + 1)} \tag{5}$$

Figure 5(a) depicts a netlist with two input ports, four instances, and one output port. The number above each instance represents the topological distance to the closest I/O. Figure 5(b) shows the equivalent graph using the traditional clique model, in which the number related to each edge represents its weight. Finally, Figure 5(c) integrates the notion of I/O proximity according to Equation (5). In Subsection 4.1 we present experiments discussing the impact of adding netlist information. In future works, we also plan to explore timing information in the graph modeling.

4 EXPERIMENTAL RESULTS

We implement our modularity-based clustering approach using Rsyn [10] [44] and run all experiments on an Intel Xeon E5-2695 dual-CPU server at 2.1GHz with 256GB RAM. Our analyses are performed in a set of open design blocks [42] synthesized using a standard industrial tool flow and a commercial 14nm enablement. Table 2 presents the number of instances, nets, and I/Os of each testcase. We first show how the Louvain algorithm has been enhanced to cope with our problem through the addition of design information in the netlist graph. Next, we compare the efficiency of our methodology to an existing VLSI clustering technique. Finally, we perform experiments to study the robustness of our formulation for different design floorplans.

4.1 Experiment 1: Evaluation of different Graph Models

In our first experiment, we compare the edge-weighting model from [23] with [16][35][13], described in Table 4, as replacement alternatives for Equation (3). Table 4 shows the values of DBi for each approach alone (column A) and with I/O proximity information (column I/O) of Equation (5). Tsay-Kuh is the most promising alternative alone, followed by Lengauer. We find that the addition of higher weights to nets closer to I/Os improves the quality of the solutions by 28% on average. Lengauer is the most promising of

Table 3: Description of net weighting alternatives.

Name	Weight per edge	Rationale
Lengauer [23]	$1/(p_i - 1)$	The total weight of the net cut to be at least one.
Huang [16]	$4/(p_i(p_i - 1))$	The expected weight of a net cut to be one.
Tsay-Kuh [35]	$2/p_i$	Minimizes the squared wirelength of the net.
Frankle-Karp [13]	$2/p_i^{1.5}$	Minimizes the worst deviation from the square of the spanning tree.

Table 4: Netlist tuning.

Design	[23]		[16]		[35]		[13]	
	A	I/O	A	I/O	A	I/O	A	I/O
jpeg_encoder	2.1	1.9	6.7	6.4	2.9	2.2	2.2	1.3
ldpc	3.0	2.3	44.5	47.2	2.7	2.3	17.3	3.0
netcard	4.6	2.0	11.1	2.3	1.9	2.7	8.3	3.7
leon3mp	1.2	1.0	68.0	63.8	1.1	1.0	1.4	1.2
Avg	2.7	1.8	32.6	29.9	2.1	2.1	7.3	2.3

these approaches with I/Os proximity information, outperforming Tsay-Kuh by 14%.

4.2 Experiment 2: Comparison with Traditional VLSI Clustering Methods

Here we discuss the correlation between our clustering formulation and the actual cell placement compared with the traditional min-cut clustering tool hMetis. As mentioned, one of the key advantages of modularity-based clustering is the absence of input parameters, so now we reveal details of how we run the other tool in order to have a fair comparison. hMetis requires two parameters: (i) the number of clusters and (ii) the unbalance factor.[4] First, we run hMetis targeting 16, 32, and 64 clusters using 2-way partitioning with unbalance factors of 10%, 20% and 40%. For each benchmark, we pick the runs with closest number of clusters to the Louvain result and compare with all unbalance factors. Second, we use hMetis k-way partitioning to find the same number of clusters as Louvain with unbalance factors of 10%, 20% and 40%.

We first analyze the values of DBi for both tools. Table 5 compares the number of clusters and values of DBi for Louvain and each run of hMetis. While Louvain presents an automatic behavior, hMetis shows a large gap in DBi depending on user tuning. For instance, in leon3mp there is a gap of 4× in DBi between unbalance 20% and 40% using 2-way partitioning. Louvain outperforms the best runs of hMetis in ldpc and leon3mp by 48% and 30%, respectively. The best runs of hMetis, in netcard and jpeg_encoder, outperform Louvain by 51% and 28%. On average, Louvain shows 20% better results for DBi.

One of the key advantages of Louvain is its almost linear runtime in sparse graphs. In Table 6, we present the runtimes of the experiments of Table 5. Louvain is 5.6× faster than the fastest hMetis run for the smallest benchmark, jpeg_encoder (13.5s). In the largest benchmark, leon3mp, Louvain is 4.1× faster than the fastest hMetis run (302.5s). On average, Louvain is 6× faster than hMetis.

[4]In hMetis, the unbalance factor is an integer value ranging from 1 to 49 and represents the percentage of difference allowed among its partitions in terms of number of vertices.

4.3 Experiment 3: Robustness With Respect to Design Floorplan

In this subsection, we show the robustness of Louvain using different floorplan configurations. We run the P&R flow with 1:1 and 2:1 floorplan aspect ratios and measure the difference in DBi. Note that different shapes lead to different clustering results since it affects buffering and logic restructuring. Furthermore, the DBi metric relies on the distance of the clusters, so an increased DBi in 2:1 floorplans is expected. Table 7 shows the increase in values of DBi for Louvain and hMetis in the 2:1 floorplan for the different floorplans. Furthermore, 2:1 floorplans present an average increase of 44% and 27% in DBi for Louvain and hMetis, respectively, showing that hMetis is 17% more stable. Figure 6 shows clustering results for Louvain with the different aspect ratios. Despite the increase in DBi, the number and shapes of the clusters for both aspect ratios are visually similar.

5 CLOSING THE LOOP: POTENTIAL INTEGRATION WITH 'BLOB PLACEMENT' FOR EARLY PLANNING

The results of the previous section suggest that modularity-based clustering can achieve stronger correlation with the eventual netlist placement, when compared to a traditional VLSI netlist clustering approach. In this section, we "close the loop" with placement: we demonstrate how the modularity-based clustering is a promising foundation for extremely fast placement and potential assessment of netlist and floorplan early in the physical implementation flow.

We have developed a simple experimental flow to predict final placement using (i) modularity-based clustering without any user configuration or tuning, and (ii) a "blob placement" step that performs cluster placement and shaping. The first step of our flow maps the flat gate-level netlist to a graph representation as described above, and then feeds this graph to Louvain. The output of Louvain is an initial set of clusters determined naturally according to the modularity criterion; we call these initial clusters *root blobs*.

The next step of our flow is to hierarchically break down the root blobs into smaller blobs (i.e., clusters), also using Louvain for modularity-based clustering. When the granularity is sufficiently small (in our implementation, we continue to break down the largest blobs until every blob has < 5K instances), we create a new netlist, consisting of the current set of blobs, which we refer to as *leaf blobs*. The nets of the new netlist are induced based on the cell instances that belong to each leaf blob. We assign higher weights to *intra-root blob* nets, i.e., nets that connect leaf blobs that originate from the same root blob. We also assign higher weights to nets that connect leaf blobs to I/Os. In our experiments, nets connecting inter-root blobs have weight = 1, nets connecting intra-root blobs have weight = 4, and nets that connect to I/Os have weight = 400. These values have been empirically determined.

Figure 7 depicts the outcome of our proposed flow for the two large testcases netcard and leon3mp. Netcard and leon3mp have 21 and 30 root blobs, and 187 and 405 leaf blobs, respectively. With both cases, root blobs contain an average of 12K cell instances, and leaf blobs contain an average of 850 cell instances. We adapt the open-source academic tool RePlAce [6] [43] to perform the

Table 5: Comparison among values of DBi for Louvain and hMetis.

Design	Louvain		hMetis (2-way part)				hMetis (k-way part)			
	# clusters	DBi	# clusters	10%	20%	40%	# clusters	10%	20%	40%
jpeg_encoder	52	1.94	64	2.39	1.71	2.81	52	2.18	1.51	2.17
ldpc	18	2.29	16	9.27	4.36	12.94	18	7.84	12.60	35.86
netcard	21	1.95	16	1.39	1.32	2.46	21	1.29	1.72	2.38
leon3mp	30	1.04	32	2.61	1.47	3.51	30	2.33	1.55	3.80
Avg		1.81		3.91	2.22	5.43		3.41	4.34	11.05

Table 6: Runtimes (s) of the experiments in Table 5.

Design	Louvain	hMetis (2-way part)			hMetis (k-way part)		
		10%	20%	40%	10%	20%	40%
jpeg_encoder	2.4	13.5	13.6	14.3	13.9	14.2	14.4
ldpc	2.7	33.6	28.7	35.3	30.9	28.3	43.7
netcard	54.9	208.2	227.4	227.2	207.6	215.6	232.3
leon3mp	72.9	392.6	302.5	321.1	314.4	310.4	325.7

Table 7: Increase in DBi for Louvain and hMetis with a 2:1 floorplan.

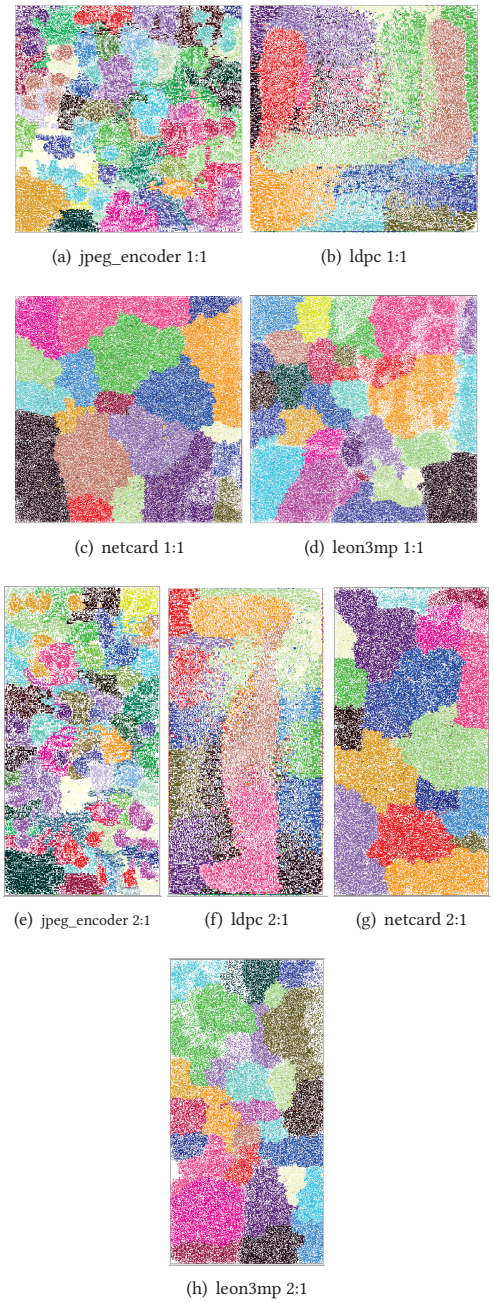
Design	Louvain	hMetis
jpeg_encoder	1.46	1.35
ldpc	1.03	1.44
netcard	2.25	1.15
leon3mp	1.04	1.15
Avg	1.44	1.27

blob placement. In doing so, we inflate the blob dimensions by 20% and 30% for netcard and leon3mp, respectively, to simulate the utilization settings from the original placement. The total runtime for the hierarchical breakdown of the gate-level netlist into leaf blobs, plus RePlAce placement, is 143s for netcard (250K instances) and 200s for leon3mp (370K instances) using a single thread of a 2.1GHz Xeon server.[5] Despite the intrinsic noise from the change of placement tool, one can easily notice the similarity between the original placements (Figures 6(c) and (d)) and our predictive blob placements. Accordingly, we believe that our new modularity-based clustering has promise as the basis of early planning steps that can improve efficiency of physical implementation.

6 CONCLUSIONS AND ONGOING WORK

In this paper, we study netlist clustering in the context of enabling early feedback at physical floorplanning and RTL planning stages of design. Our new criterion for clustering assesses whether netlist clusters "stay together" through final physical implementation. We support evaluation of this criterion via several methods, including the use of *alpha shapes*, Delaunay triangulation of a cluster's placed locations, and the Davies-Bouldin index. For the purpose of predicting cohesion in final layouts, we find that *modularity*-driven clustering, as exemplified by the Louvain [2] algorithm, is clearly superior to mincut- or Rent parameter-driven methods [21] [4] [41] that have dominated the VLSI CAD literature. Importantly, the modularity metric allows identification of "natural" clusters in a given graph without parameter tuning, and without imposition of balancing constraints; yet, it may also be applied hierarchically as needed. We also show that the hypergraph-to-graph mapping is critical to successful application of modularity-based clustering: our initial study of mapping techniques suggests that (i) a weighting approach of Lengauer [23] is effective in conjunction with Louvain, and (ii) encoding topological proximity to I/Os significantly increases

[5]The hierarchical use of Louvain could be modified to trivially exploit availability of multiple threads.



(a) jpeg_encoder 1:1 (b) ldpc 1:1 (c) netcard 1:1 (d) leon3mp 1:1 (e) jpeg_encoder 2:1 (f) ldpc 2:1 (g) netcard 2:1 (h) leon3mp 2:1

Figure 6: Visual comparison of Louvain clusters in floorplans with different aspect ratios.

(a) (b)

Figure 7: Blob placement of (a) netcard and (b) leon3mp. Compare with Figures 6(c) and (d).

clustering quality. Comparisons with traditional hMetis-based clustering [21] show that our Louvain-based approach achieves on average 20% better correlation to actual netlist placements, as well as 4× faster runtimes for our largest testcases. Last, we demonstrate the potential of using modularity-based clustering with fast "blob placement" of clusters to efficiently evaluate netlist and floorplan viability in early stages of design.

Our ongoing work is in several directions. First, we believe that our studies of the impact of hypergraph-to-graph mapping choices (Section 4.1) point out the potential value of improved mappings. We are exploring the use of machine learning to improve such mappings, potentially leading to localized variation of graph mapping strategy in the netlist. Second, we are pursuing various extensions of the "blob placement" flow described in Section 5. For example, it is necessary to be able to handle large amounts of whitespace in the given block floorplan. And, the clustering must ultimately be sensitive to details of a floorplan with embedded macro blocks: determining clusters that cohere in final placement may entail dynamic fragmentation of "blobs" (into sub-blobs that will each stay together) during the global placement iteration. Ultimately, improved cluster placement could open the door to research on predicting timing and congestion from blob placement results. Finally, we observe that the use of fast prediction of final placement – e.g., to drive floorplan changes or early (RT-level) physically-aware synthesis – would likely lead to different final placements than those we currently predict. How to manage this inherent chicken-egg loop is an open question.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, CNPq and FAPERGS.

Research at UCSD is supported by Qualcomm, Samsung, NXP Semiconductors, Mentor Graphics, DARPA (HR0011-18-2-0032), NSF (CCF-1564302) and the C-DEN center.

REFERENCES

[1] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey", *Integration, the VLSI J.* 19(1-2) (1995), pp. 1–81.
[2] V. D. Blondel, J. L. Guillaume, R. Lambiotte and E. Lefebvre, "Fast Unfolding of Communities in Large Networks", *J. of Statistical Mechanics: Theory and Experiment* 10 (2008), pp. 1–12.
[3] K. Blutman, H. Fatemi, A. B. Kahng, A. Kapoor, J. Li and J. P. de Gyvez, "Floorplan and Placement Methodology for Improved Energy Reduction in Stacked Power-Domain Design", *Proc. ASP-DAC*, 2017, pp. 444–449.
[4] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Improved Algorithms for Hypergraph Bipartitioning", *Proc. ASP-DAC*, 2000, pp. 661–666.
[5] B. Chen and M. Marek-Sadowska, "Timing-Driven Placement of Pads and Latches", *Proc. IEEE ASIC Conf.*, 1992, pp. 30-33.
[6] C.-K. Cheng, A. B. Kahng, I. Kang and L. Wang, "RePlAce: Advancing Solution Quality and Routability Validation in Global Placement", *IEEE Trans. on CAD* (2018), doi:10.1109/TCAD.2018.2859220.
Boolean Networks", *Proc. DAC*, 1994, pp. 670–675.
[7] D. L. Davies and D. W. Bouldin, "A Cluster Separation Measure", *IEEE Trans. PAMI* 1(2) (1979), pp. 224–227.
[8] H. Edelsbrunner, D. G. Kirkpatrick and R. Seidel, "On the shape of a set of points in the plane", *IEEE Trans. on Information Theory* 29(4) (1983), pp. 551–559.
[9] C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions", *Proc. DAC*, 1982, pp. 175–181.
[10] G. Flach, M. Fogaça, J. Monteiro, M. Johann and R. Reis, "Rsyn – An Extensible Physical Synthesis Framework", *Proc. ISPD*, 2017, pp. 33–40.
[11] S. Fortunato and M. Barthelemy, "Resolution Limit in Community Detection", *Proc. Proc. Nat. Acad. Sci.* 69 (2007), pp. 1–15.
[12] S. Fortunato and D. Hric, "Community Detection in Networks: A User Guide", *Physics Reports* 659 (2016), pp. 1–44.
[13] J. Frankle and R. M. Karp, "Circuit Placement and Cost Bounds by Eigenvector Decomposition", *Proc. ICCAD*, 1986, pp. 414–417.
[14] L. Hagen and A. B. Kahng, "A New Approach to Effective Circuit Clustering". *Proc. ICCAD*, 1992, pp. 422–427.
[15] K. M. Hall, "An r-Dimensional Quadratic Placement Algorithm", *Management Science* 17 (1970), pp. 219–229.
[16] D. J. H. Huang and A. B. Kahng, "When Clusters Meet Partitions: New Density-Based Methods for Circuit Decomposition", *Proc. European Design and Test Conf.*, 1995, pp. 60–64.
[17] E. Ihler, A. D. Wagner and F. Wagner, "Modeling Hypergraphs by Graphs with the Same Mincut Properties", *Inf. Process. Lett.* 45 (1993), pp. 171–175.
[18] K. Jeong, A. B. Kahng and H. Yao, *RentCon: Rent Parameter Evaluation Using Different Methods.* https://vlsicad.ucsd.edu/WLD/index.html
[19] A. B. Kahng and X. Xu, "Local Unidirectional Bias for Smooth Cutsize-Delay Tradeoff in Performance-Driven Bipartitioning", *Proc. ISPD*, 2003, pp. 81–86.
[20] A. B. Kahng, "Reducing Time and Effort in IC Implementation: A Roadmap of Challenges and Solutions", *Proc. DAC*, 2018, pp. 1–6.
[21] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Domain", *Proc. DAC*, 1997, pp. 526–529.
[22] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *The Bell System Tech. J.* 49 (1970), pp. 291–307.
[23] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, New York, Wiley-Teubner, 1990.
[24] S. Mantik, G. Posser, W. Chow, Y. Ding and W. H. Liu, "ISPD 2018 Initial Detailed Routing Contest and Benchmarks", *Proc. ISPD*, 2018, pp. 140–143.
[25] D. B. Mark, M. Overmars and O. Cheong, *Computational Geometry: Algorithms and Applications*, New York, Springer, 1997.
[26] M. E. Newman and M. Girvan, "Finding and Evaluating Community Structure in Networks", *Phys. Rev. E* 69 (2004), pp. 1–15.
[27] A. Olofsson, "Silicon Compilers - Version 2.0", keynote, *Proc. ISPD*, 2018. http://www.ispd.cc/slides/2018/k2.pdf
[28] S. Ou and M. Pedram, "Timing-Driven Bipartitioning with Replication Using Iterative Quadratic Programming", *Proc. ASP-DAC*, 1999, pp. 105–108.
[29] J. A. Roy, S. N. Adya, D. A. Papa and I. L. Markov, "Min-cut Floorplacement", *IEEE Trans. CAD* 25(7) (2006), pp. 1313–1326.
[30] R. S. Shelar, "An Efficent Clustering Algorithm for Low Power Clock Tree Synthesis", *Proc. ISPD*, 2007, pp. 181–188.
[31] H. Shiokawa, Y. Fujiwara and M. Onizuka, "Fast Algorithm for Modularity-based Graph Clustering", *Proc. AAAI*, 2013, pp. 1170–1176.
[32] H. Shiokawa, Y. Fujiwara, and M. Onizuka, "SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-scale Graphs", *Proc. VLDB Endow.*, 2015, pp. 1178–1189.
[33] H. Shiokawa and M. Onizuka, "Scalable Graph Clustering and Its Applications", *Encyclopedia of Social Network Analysis and Mining*, New York, Springer, 2017.
[34] C. N. Sze and T.-C. Wang, "Performance-Driven Multi-Level Clustering for Combinational Circuits", *Proc. ASP-DAC*, 2003, pp. 729–734.
[35] R.-S. Tsay and E. S. Kuh, "A Unified Approach to Partitioning and Placement", *IEEE Trans. Circuits and Systems* 38 (1991), pp. 521–533.
[36] X. Xu, N. Yuruk, Z. Feng and T. A. J. Schweiger, "SCAN: A Structural Clustering Algorithm for Networks", *Proc. KDD*, 2007, pp. 824–833.
[37] H. Yang and D. F. Wong, "Efficient Network Flow Based Min-cut Balanced Partitioning"', *Proc. ICCAD*, 1994, pp. 50–55.
[38] "DARPA Rolls Out Electronics Resurgence Initiative", https://www.darpa.mil/news-events/2017-09-13
[39] *International Technology Roadmap for Semiconductors*, http://www.itrs2.net/itrs-reports.html
[40] Alpha shape, https://en.wikipedia.org/wiki/Alpha_shape
[41] K. Jeong, A. B. Kahng and H. Yao, *RentCon: Rent Parameter Evaluation Using Different Methods*, https://vlsicad.ucsd.edu/WLD/index.html
[42] OpenCores: Open Source IP-Cores, http://www.opencores.org.
[43] RePlAce, https://github.com/abk-openroad/RePlAce
[44] Rsyn, https://github.com/RsynTeam/rsyn-x