Integration, the VLSI Journal 74 (2020) 32–44

Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

On the superiority of modularity-based clustering for determining placement-relevant clusters

Mateus Fogaça [a,c,*], Andrew B. Kahng [d,e], Eder Monteiro [c], Ricardo Reis [a,b,c], Lutong Wang [e], Mingyu Woo [e]

[a] PGMicro, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brazil
[b] PPGC, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brazil
[c] Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brazil
[d] CSE, University of California at San Diego, La Jolla, CA, USA
[e] ECE departments, University of California at San Diego, La Jolla, CA, USA

ARTICLE INFO

Keywords:
EDA
Physical design
Floorplanning
Placement
Modularity-based clustering

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 the 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 2× faster than use of *hMetis* for our largest testcases). Finally, we propose a flow with fast "blob placement" of clusters. The "blob placement" is used as a seed for a global placement tool that performs placement of the flat netlist. With this flow we achieve 20% speedup on the placement of a netlist with 4.9 M instances with less than 3% difference in routed wirelength.

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 [1]. 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 [2] 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 tool runtime and overall design schedule [3]. 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 [4] 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 h [2,4]. New tools that can help to avoid future failures (congestion, failed timing, etc.) while still in the early stages of floorplan definition or RTL planning appear

* Corresponding author. PGMicro, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brazil.
E-mail addresses: mpfogaca@inf.ufrgs.br (M. Fogaça), abk@ucsd.edu (A.B. Kahng), emrmonteiro@inf.ufrgs.br (E. Monteiro), reis@inf.ufrgs.br (R. Reis), luw002@eng.ucsd.edu (L. Wang), mwoo@eng.ucsd.edu (M. Woo).

https://doi.org/10.1016/j.vlsi.2020.03.007
Received 17 November 2019; Received in revised form 21 February 2020; Accepted 31 March 2020
Available online 23 April 2020

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; this knowledge would then inform 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 [5], clock tree synthesis [6] and, more recently, grouping of instances into different power domains [7]. While many clustering methods for VLSI have been proposed, they have largely focused on *net cuts* (hyperedge min-cut, cluster perimeter, Rent parameter [8], etc.). Further, existing heuristics typically require design-dependent tuning and sub-optimal heuristics. For instance, the well-known multilevel Fiduccia-Mattheyses [9] implementations hMetis [10] and MLPart [11] 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 contributions. 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* criterion [13]. Our study centers on Louvain [14], 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 700 M edges within 12 min, 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) [15]. 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) [16], Variance Ratio Criterion [17] and Silhouette Coefficient [18], which are traditionally used to evaluate how "well-separated" clusters are. For spatial data, such as placements of standard-cell instances, our evaluation criteria measure (i) the distances from instances to the center of gravity of the clusters they belong and (ii) the distance among the center of gravity of clusters. In a "good" clustering solution, the ratio between (i) and (ii) is a small numeric value.

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.
3. We adopt Davies–Bouldin index [16], Variance Ratio Criterion [17] and Silhouette Coefficient [18] as criteria to compare clustering results. These criteria are extensively used for evaluation of spatial clustering but have not been explored by the EDA community.[2]
4. We perform experiments showing 50% better clustering quality on average for Louvain [14] versus the traditional VLSI netlist clustering tool hMetis [10], with 2× faster runtime than hMetis for our largest benchmark.
5. 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 instances that remain together in the actual gate-level placement with a speed-up of 50% compared to flat placement for a testcase with 1.2 M instances and 20% speed-up for a testcase with 4.5 M instances.[3]

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 overview of relevant works in two literatures: VLSI netlist partitioning, and community detection. Since the modularity criterion is not previously addressed in the VLSI CAD literature, we present a somewhat expansive review of its development and usage in the machine learning literature.

2.1. VLSI netlist partitioning

Netlist partitioning is a fundamental step within a broad spectrum of EDA tools. Alpert and Kahng [19] 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) [20] along with the vertex-shifting move structure of Fiduccia-Mattheyses (FM) [9] are at the core of such methods. Two FM tools, hMetis [10] and MLPart [11] are widely used in academic and commercial flows today.

Geometric representation-based approaches exploit geometric embeddings of circuits to achieve improved cluster quality and runtime. Hall [21] gives an early spatial approach, achieving multi-way partitioning solutions through quadratic placement and vertex orderings induced from eigenvectors of a netlist-derived discrete Laplacian matrix. Barnes [22] extends this strategy to perform k-way partitioning.

[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.

[2] The silhouette metric has not been widely used in the VLSI CAD clustering literature, with [61] being the only example of which we are aware.

[3] Note that the core motivation and contribution of our current work is to *rapidly predict* the placement. If instance placements can be quickly known, an expert designer is able to tune the flow setup (e.g., with small modifications to floorplan, density screens, grouping, etc.) to improve the quality of results or to fix timing and routability issues. In this context, our work provides a methodology to improve the outcomes and/or the turnaround time of the netlist-to-placement phase of the implementation flow.

Ou and Pedram [23] 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 [24] propose an iterated max-flow formulation to address the balanced bipartition problem. More recently, Blutman et al. [7] 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, abound. E.g., Hagen and Kahng [25] perform clustering by integrating a random-walk algorithm with iterative FM. Sze and Wang [26] use a graph contraction technique to maintain delay information among different lower levels of a performance-driven clustering flow. Alternatively, Kahng and Xu [27] 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. Finally, Jindal et al. [28] propose a bottom-up clustering technique to find tangled logic using the Rent parameter, and demonstrate how to alleviate routing hotspots using their clusters.

2.2. Modularity-driven clustering

Before diving into *modularity-driven clustering*, we provide some background on *community detection*. Community detection is a subset of the machine learning literature comprising algorithms closely related to the partitioning methods described in Section 2.1 and hierarchical clustering studied in sociology [13]. Communities (also referred to as clusters) correspond to the division of the vertices of the graph into groups, where the edges within a given group are denser and between them are sparser [13]. The algorithms in the community detection field are taxonomized into three categories: (i) *divisive methods* [29,30] find communities by iteratively removing edges from the graph, (ii) *agglomerative methods* [14,31] iteratively merge vertices and communities, and (iii) *optimization methods* [32,33,34] maximize an objective function using heuristic methods (e.g., simulated annealing). *Modularity* is a criterion used to evaluate the division of the graph into communities. Modularity is a numeric value ranging from −1 to 1, and higher values indicate a better division of the graph. Methods that adopt modularity as an objective function are called *modularity-driven clustering*.

The modularity criterion is proposed by Newman and Girvan in Refs. [13]. Newman and Girvan propose three divisive methods for community detection and evaluate them using artificial and real-world graphs whose community structure is well-known. However, Newman and Girvan highlight that in most real-world problems, the ground-truth communities are not known. To tackle this problem, the authors propose a numeric criterion, called modularity. The correlation between modularity and the ground-truth communities for a given graph has been assessed using artificially-generated testcases. Newman and Girvan have noticed peaks of modularity for solutions whose communities have correlated well with the ground-truth.

Later, Newman finds the answer to the question, "If a high value of modularity represents a good community division why not simply optimize modularity over all possible divisions to find the best one?" [33]. Newman proposes a greedy agglomerative algorithm that starts with every vertex as a sole member of a community. Then, the algorithm repeatedly merges communities together in pairs. At each step, the algorithm chooses the pair of communities that represent the best increase in modularity. Newman's algorithm has since been improved by Clauset et al. [32] (also known as the *CNM algorithm*) and Wakita and Tsurumi [35].

In [14], Blondel et al. propose an agglomerative modularity-driven method, called Louvain algorithm. Initially, each vertex is considered a community. The algorithm is iterative and each iteration is composed of two phases. The first phase performs swaps of vertices between neighboring communities that produce gains in modularity. The second phase builds a new graph in which the vertices are the communities found in the first phase. The edges among the vertices of the new graph are the sum of the edges between the vertices of the corresponding communities on the old graph. Unlike previous approaches, Louvain is fast and scalable. Experiments performed in Ref. [14] show linear runtime complexity with respect to the number of vertices in sparse graphs. For instance, Louvain is able to perform community detection in a graph with 118 M vertices in 152 min. Since each pass reduces the size of the graph, most of the runtime is spent on the first iteration. The Louvain code is available in Ref. [36].

2.3. Modularity-driven clustering for hypergraphs

In Section 2.2, we have provided an overview about modularity-driven clustering of *graphs*. Nevertheless, the data of many practical applications, such as social networks and VLSI netlists, are described using *hypergraphs*. One way to tackle problem instances arising in such applications is to use a *hypergraph-to-graph mapping* method [37]. In doing so, some information in hyperedges with degrees greater than two may be lost. Some research has intended to enable the modularity criterion to hypergraphs. Neubauer et al. [38,39] propose a modularity criterion and optimization method for *k*-partite *k*-uniform hypergraphs. Kumar et al. [40,41] propose a modularity criterion for hypergraphs of any degree and a method to integrate the proposed criterion into the Louvain algorithm. Additionally, Kumar et al. devise an incremental weighting scheme to balance the number of vertices per cluster. Finally, Kamiński et al. [42] adapt the modularity criterion for hypergraphs using the Chung-Lu model [43]. Kamiński et al. show that their criterion correlates well with hyperedge cut and adjust the CNM algorithm to use the proposed criterion. The CNM code is available as Julia scripts on GitHub [44].

Despite the above-mentioned efforts to extend modularity-driven clustering to hypergraphs, to the best of our knowledge there is no available, open-source and *scalable* tool that serves the hypergraph clustering context in the way that Louvain presently serves the modularity-driven graph clustering context. For instance, we have tried to cluster our testcases from Section 4 using the [44]. However, a design with 8 K cells, which is much smaller than the netlists arising in our present work, takes an average of 25 min when we sweep the number of iterations of the algorithm from 500 to 10,000 with step of 500. In contrast, Louvain can cluster a design with 1.4 M instances in 7 min. On the small testcase *jpeg_encoder_14* with 44 K instances, the scripts from Ref. [44] crash due to stack overflow.

In our present work, we employ the widely-used VLSI clustering tool hMetis [10], 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) the means of capturing structural 'hints' (I/Os, timing, etc. - cf [45]) 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.

Table 1
Notations.

Term	Meaning
DBi	Davis-Bouldin index
VRC	Variance ratio criterion
SC	Silhouette coefficient
N	Total number of elements being clustered
n	Number of clusters
n_i	Number of elements in cluster i
σ_i	Average distance from the cluster elements to the centroid of cluster i
ρ	Centroid of all elements being clustered
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	i^{th} cluster
$\delta(c_i, c_j)$	Function that receives two clusters as input 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
B	Between-clusters dispersion
W	Within-clusters dispersion
x	Coordinate (x, y) of an element of a cluster
a_i	Average pairwise distance over all elements of a given cluster c_i
b_i	Average pairwise distance of an element given cluster c_i and all other elements of the nearest cluster
$\Delta DBi/VRC/\Delta SC$	Normalized variations of DBi, VRC and SC
$DBi_{tech}/VRC_{tech}/SC_{tech}$	Value of DBi, VRC and SC for the technology node "*tech*"

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.

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 [15], 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 our 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 Fig. 1. Fig. 2(a) depicts a "ground-truth" placement along with a cluster, with cells colored according to their clusters. Fig. 2(b) draws the corresponding convex hulls. However, if we examine the highlighted blue cluster in Fig. 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 [47,48], examples of which are shown in Fig. 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 comprises the alpha shape of the pointset.[4] 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 μm–500 μm range, we empirically use alpha = 2500 μm^2. In Fig. 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%.

Our last approach to retrieve cluster shapes is derived from the Delaunay triangulation (DT), depicted in Fig. 2(d). The DT is the geometric dual of the Voronoi diagram over a given pointset. One can infer the cluster shape and outliers by analyzing the sizes and density of DT edges of a given cluster. Statistical data may also be extracted from the distribution of edge lengths to assess the clustering solution.

Solution Evaluation. Convex hulls, alpha shapes and DT are useful for visual and manual debugging. For solution evaluation, we propose three criteria. Recall that the main goal of our 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. For our experiments, we adopt the Davies–Bouldin index (DBi) [16], Variance Ratio Criterion (VRC) [17] and Silhouette Coefficient (SC) [18], traditionally used for spatial clustering evaluation, as indicators of cluster quality.[5] The DBi is defined as:

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

[4] 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.

[5] To ensure a correct comparison, we implemented DBi, VRC and SC in the same way as in Ref. [62].

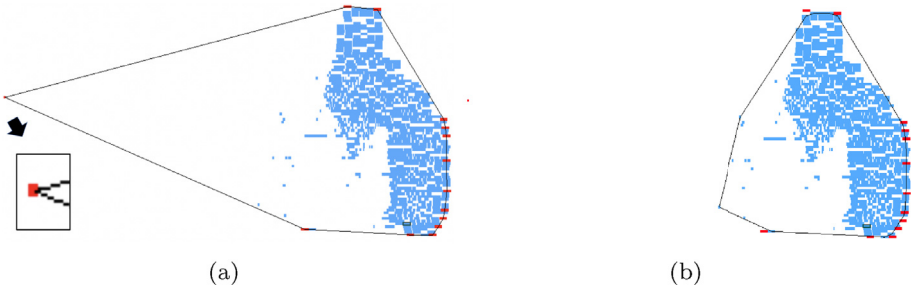


Fig. 1. The process of "shelling" the cluster shape. Figure (a) shows a cluster with total cell area equal to $4.6 \times 10^3 \mu m^2$ and shape area equal to $23.0 \times 10^3 \mu m^2$. Thus, the utilization of the cluster is equal to 20.2%. The cluster's "shell" is the set of red instances that are on the boundary of the shape. In (b), the cluster shape is recomputed after removing the shell from (a). The final shape has area equal to $11.6 \times 10^3 \mu m^2$ and utilization equal to 40.1%. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

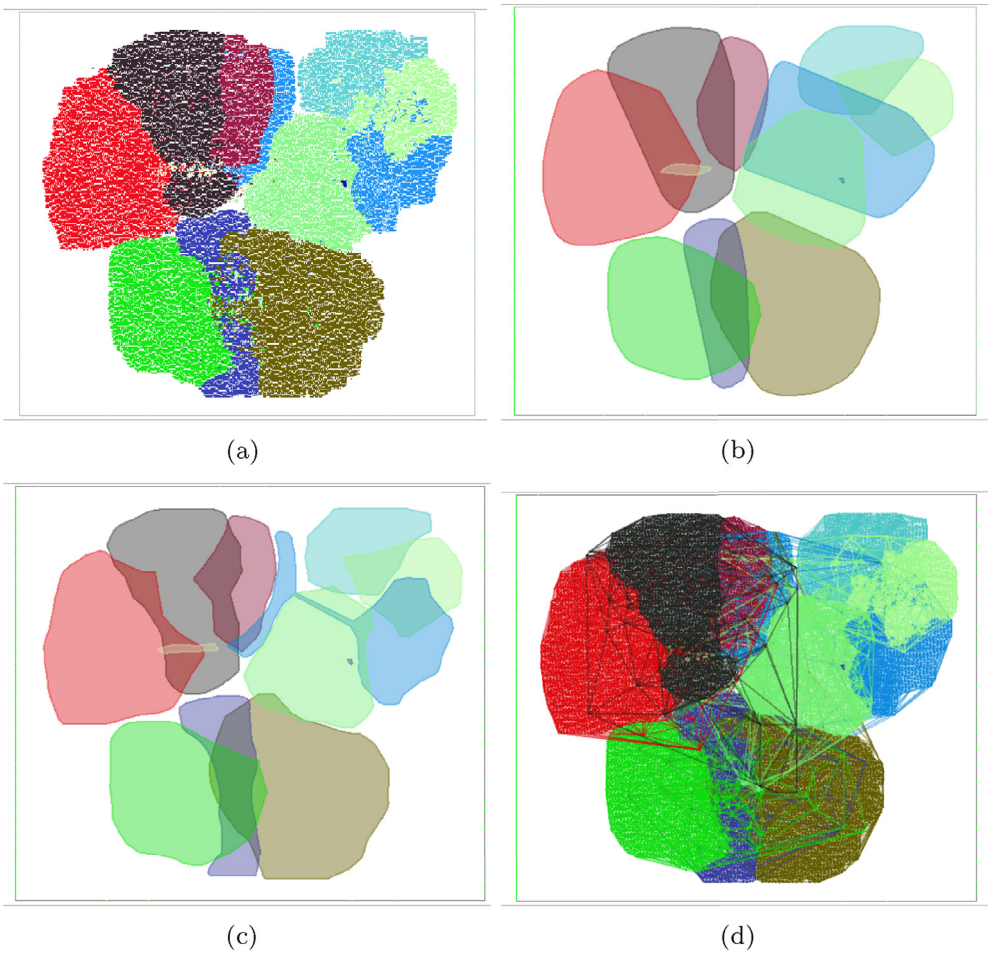


Fig. 2. Different approaches to correlate clusters with the placement for the circuit ispd18_test2 [46]: (a) the placement with each instance colored according to its cluster, followed by (b) the convex hulls; (c) the alpha shapes; and (d) the Delaunay triangulations of the clusters.

where n is the number of clusters, σ_i is the average distance from the cluster elements to the centroid of cluster i, ρ_i is the centroid of cluster i and $l(\rho_i, \rho_j)$ is the distance between centroids ρ_i and ρ_j. In DBi, smaller values of $\sigma_i(\sigma_j)$ indicate more denser clusters and higher values of $l(\rho_i, \rho_j)$ indicate well-separated clusters. Therefore, smaller values of DBi indicate a better clustering solution. VRC is defined as:

$$VRC = \frac{Tr(B)}{Tr(W)} \times \frac{N-n}{n-1} \tag{2}$$

where Tr(B) is the trace of matrix B, and N is the total number of elements being clustered. The *between-clusters* dispersion (B) and *within-clusters* dispersion (W) are computed as

$$B = \sum_{i}^{n} n_i(\rho_i - \rho)(\rho_i - \rho)^T \tag{3}$$

$$W = \sum_{i}^{n} \sum_{x \in c_i} (x - \rho_i)(x - \rho_i)^T \tag{4}$$

where n_i is the number of elements in cluster i, ρ is the centroid of all elements being clustered and x is the coordinate (x, y) of an element in cluster c_i. Higher values of B indicate well-separated clusters and smaller values of W indicate denser clusters. Therefore, higher values of VRC indicate a better clustering solution. Additionally, the VCR criterion tends to be higher in solutions with smaller number of clusters. Finally, SC is defined as:

$$SC = \sum_{i=1}^{n} \frac{b_i - a_i}{max(a_i, b_i)} \tag{5}$$

where a_i is the average pairwise distance over all elements of a given

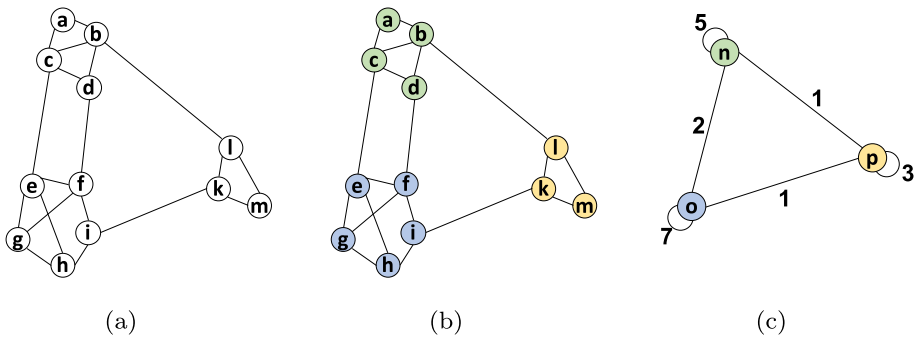


Fig. 3. Representation of the Louvain algorithm steps. Figure (a) depicts a graph with 12 vertices and 19 edges. For simplicity, we assume unitary edge weights. Figure (b) depicts the result of the first step, called *modularity optimization*, which moves vertices among clusters trying to optimize modularity. Vertices are colored according to their cluster. Figure (c) illustrates the second step, called *community aggregation*, in which vertices belonging to the same cluster are merged into a single vertex. Edges connecting the same pair of clusters are merged and their weights are summed.

cluster i and b_i is the average pairwise distance of an element of a given cluster i and all elements of the nearest cluster. SC is a numeric value ranging from −1 to 1. In SC, smaller values of a_i indicate denser clusters and higher values of b_i indicate well-separated clusters. Values of SC closer to 1 indicate a better clustering solution.

3.3. Modularity-based clustering

The *modularity* criterion [13] 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 criterion is formally expressed as

$$Q = \frac{1}{2m}\sum_{i,j}[A_{ij} - \frac{k_i k_j}{2m}]\delta(c_i, c_j) \tag{6}$$

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.[6] This is in contrast to the existing VLSI clustering literature 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 criterion; and
- The Louvain algorithm does not impose, nor does it require, any area/edge balancing constraints.

Louvain is a bottom-up modularity-based clustering algorithm. In the initial solution, each vertex of the input graph is considered as a cluster. The main loop of Louvain tries to merge clusters aiming to optimize modularity. The main loop is composed of two phases: *modularity optimization* and *community aggregation*. In the *modularity optimization*, Louvain traverses all vertices of the graph and computes the modularity delta of moving the vertex from the current cluster to all topological neighbor clusters. The vertex is moved to the neighbor that produces the largest increase in modularity. If no movement provides an increase in modularity, the vertex remains in its current cluster. In the *community aggregation*, all vertices belonging to the same cluster are merged into a single vertex. The edges connecting the same pair of vertices are merged into a single edge and their weights are summed. Edges connecting vertices that belong to the cluster create a self-loop edge. We depict the two phases of Louvain in Fig. 3. Each iteration of the loop is called a *pass*. The loop is repeated until no increase in modularity is observed.

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 represent a hypergraph by a weighted graph is needed. The *clique and star decompositions* are often used in a variety of applications. The clique decomposition 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. [49] prove that there is no perfect weighting for the clique decomposition. In this work, we evaluate the five different edge weighting schemes for the clique decomposition presented in Table 2. The star decomposition replaces the hyperedges with edges connected to a virtual node. In this work, the edges created by the star decomposition have weight equal to 1.

The traditional clique or star decompositions are 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 in a subset of testcases.[7] 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:

$$w_{h,2} = argmin((d(I)), (d(O))) \tag{7}$$

$$w_h = w_{h,1}\frac{1}{(w_{h,2}+1)} \tag{8}$$

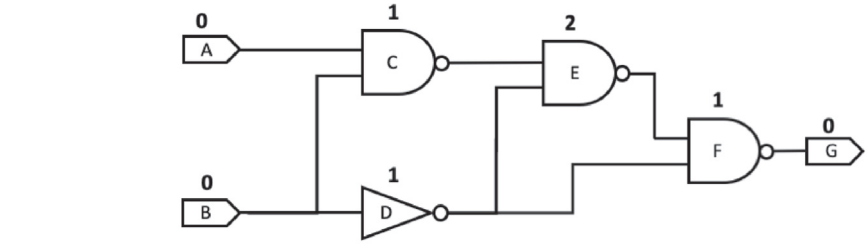
Fig. 4(a) depicts a netlist with two input ports, four instances, and one output port. The number above each instance represents the topological distance to the closest I/O. Fig. 4(b) shows the equivalent graph using the traditional clique decomposition, in which the number related to each edge represents its weight using the Lengauer weighting scheme. Fig. 4(c) shows the equivalent graph using the star decomposition (virtual nodes are drawn as circles with a dashed outline). Finally, Fig. 4(d) integrates the notion of I/O proximity according to Equation (8). In Subsection 4.1 we present experiments discussing the impact of adding netlist information. We note in Section 6 that incorporation of timing information (slack, etc.) in the graph modeling remains an open issue for future work.

[6] We note that applying the modularity criterion 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 criterion *and* use of the Louvain algorithm.

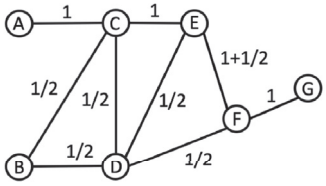
[7] In the experiments of Fogaça et al. [63], the addition of I/O proximity weights improves the quality of results in terms of DBi by 28%, on average. In Section 4.1, we extend the experiments of [63] and observe that I/O proximity weights only improve DBi in when combined with Tsay-Kuh-2 edge weights and the Star decomposition.

Table 2
Description of net weighting alternatives.

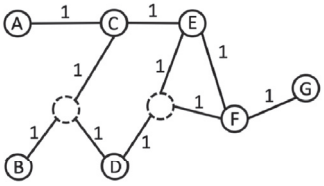
Name	Weight per edge	Rationale
Lengauer [50]	$1/(p_h - 1)$	Set the total weight of the net cut to be at least one.
Huang [51]	$4/(p_h(p_h - 1))$	Set the expected weight of a net cut to be one.
Tsay-Kuh [52]	$2/p_h$	Minimize the squared wirelength of the net.
Tsay-Kuh-2 [52]	$(2/p_h)^3$	Minimize the Manhattan wirelength of the net.
Frankle-Karp [53]	$2/p_h^{1.5}$	Minimize the worst deviation from the square of the spanning tree.



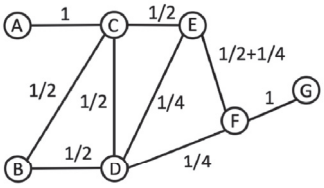
(a) Netlist



(b) Clique decomposition using Lengauer weighting scheme



(c) Star decomposition



(d) Clique decomposition with I/O proximity weights

Fig. 4. Netlist decomposition.

4. Experimental results

We implement our modularity-based clustering approach using Rsyn [54,55] and run all experiments on an Intel Xeon E5-2695 dual-CPU server at 2.1 GHz with 256 GB RAM. Our analyses are performed in a set of open design blocks [56]. We use a commercial synthesis tool to generate our gate-level netlists. We run synthesis to reach the smallest clock period that does not generate timing violation based on TT corner on each PDK. Our testcases are synthesized in three industrial technologies: 14 nm, 28 nm and 65 nm. We then perform I/O placement and remove all buffers using a commercial place-and-route tool. The placement we use in our experiments is generated by the academic global placement tool RePlAce v1.1.1 [57,58] and the legalization tool OpenDP v0.1.0 [59,60].

Table 3 presents the number of instances, nets, and I/Os of each testcase. The number after each testcase name indicates the testcase enablement, i.e., technology node. We conduct four experiments. Our first experiment evaluates the results of Louvain using different graph models of the netlist. Our second experiment compares the efficiency of our methodology to an existing VLSI clustering technique. Our third experiment studies the robustness of our formulation for different design floorplans. Finally, our forth experiment compares the performance of Louvain clustering across all the enablements.

Table 3
Benchmarks and attributes.

Benchmark	Insts	Nets	I/Os
jpeg_encoder_14	44,083	45,018	49
ldpc_decoder_14	38,559	40,610	4100
netcard_14	272,865	274,704	1849
leon3mp_14	316,537	316,791	333
MegaBoom_14	1,249,594	1,254,352	945
MegaBoom_X2_14	2,492,643	2,502,392	1888
MegaBoom_X3_14	3,734,334	3,749,542	2831
MegaBoom_X4_14	4,979,581	4,999,882	3774
jpeg_encoder_28	46,962	47,775	49
ldpc_decoder_28	40,402	42,506	4100
netcard_28	235,277	237,122	1849
leon3mp_28	400,836	401,091	333
MegaBoom_28	1,419,923	1,425,174	945
netcard_65	239,901	241,740	1849
leon3mp_65	325,041	325,295	333
MegaBoom_65	1,169,564	1,174,669	945

4.1. Experiment 1: evaluation of different graph models

In our first experiment, we compare the clique decomposition using different weighting schemes and the star decomposition. Table 4 shows the values of DBi, VRC and SC for each approach alone and with

I/O proximity information of Equation (8). To compare the weighting schemes in the "Average" row, we first normalize DBi, VRC and SC per benchmark using the values of Lengauer without I/O proximity weights as the reference for normalization. We then compute the average of the normalized values for each column.[8] We find that the addition of I/O proximity weights significantly improves DBi and VRC for Huang and Tsay-Kuh-2 weighting schemes and improves DBi for the Star decomposition. Lengauer, Tsay-Kuh and Frank-Karp show better DBi, VRC and SC without I/O proximity weights. Lengauer without I/O proximity weights presents the best results in terms of DBi, VRC and SC. Therefore, all of our following experiments are performed using Lengauer without I/O proximity weights.

4.2. Experiment 2: comparison with traditional VLSI clustering methods

We now discuss the correlation between our clustering formulation and the actual cell placement as compared with the traditional min-cut clustering tool hMetis. As mentioned, one of the key advantages of modularity-based clustering is its absence of input parameters; this itself presents a challenge when we seek a fair comparison versus hMetis. hMetis requires two parameters: (i) the number of clusters and (ii) the unbalance factor.[9] Additionally, hMetis performs *2-way partitioning* if the target number of clusters is a power of 2 and *k-way partitioning* otherwise. In our experiments, we first run hMetis in *2-way partitioning* mode targeting the nearest power of 2 to the number of clusters found by Louvain. We run hMetis in *k-way partitioning* mode targeting the same number of clusters found by Louvain. We execute each mode with three settings of unbalance factor: 10%, 20% and 40%, and report the best of the three.

Table 5 compares Louvain and hMetis using DBi, VRC, SC and runtime. Our criteria are normalized using Louvain values as the reference before computing the average numbers. The normalization follows the same procedure adopted in Table 4. For each criterion, we show the best value among the runs with 10%, 20% and 40% unbalance factor. Louvain outperforms hMetis for our largest testcases, leon3mp and MegaBoom, in terms of DBi, VRC and SC. In ldpc_decoder, Louvain outperforms hMetis in terms of DBi. The results of hMetis vary considerably depending on the input configuration. In jpeg_encoder, there is a 1.4× DBi gap and a 2.2× SC gap between hMetis in 2-way and *k*-way partitioning modes. On average, Louvain shows 6.7× and 5.2× better DBi than hMetis in 2-way and *k*-way partitioning modes, respectively. hMetis shows better VRC results by 1.6× and 1.7× in 2-way and *k*-way partitioning modes. Similarly, hMetis shows better SC by 1.2× compared to Louvain in both 2-way and *k*-way partitioning modes. However, Louvain outperforms hMetis for our largest testcases. For instance, in MegaBoom, Louvain shows 9.7× and 11.4× better VRC than hMetis in 2-way and *k*-way partitioning modes, respectively.

One of the key advantages of Louvain is its almost linear runtime in sparse graphs. Louvain is 6× faster than the fastest hMetis run for the smallest benchmark, ldpc_decoder (18s). In the largest benchmark, MegaBoom, Louvain is 1.8× faster than the fastest hMetis run (826s). On average, Louvain is 4.5× faster than hMetis.

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 placement tool with 1:1, 1.5:1, 2:1, 2.5:1 and 3:1 floorplan aspect ratios and measure the difference in DBi, VRC and SC. Table 6 shows the delta from the floorplan with

Table 4
Netlist tuning.

	Design	Lengauer			Huang			Tsay-Kuh			Tsay-Kuh-2			Frankle-Karp			Star decomposition		
		DBi	VRC	SC	DBi	VRC	SC	DBi	VRC	SC	DBi	VRC	SC	DBi	VRC	SC	DBi	VRC	SC
w/o I/O weights	jpeg_encoder_28_55	3.5	5495	−0.057	4.9	3622	−0.144	5.0	2734	−0.168	5.8	3522	−0.192	3.9	3668	−0.099	3.6	1901	−0.164
	jpeg_encoder_28_70	3.6	4988	−0.096	4.7	3080	−0.171	4.0	4122	−0.106	3.7	3476	−0.188	4.1	3635	−0.101	3.1	2548	−0.151
	ldpc_decoder_28_55	38.3	8	−0.614	63.2	9	−0.701	41.9	8	−0.618	56.5	11	−0.756	32.4	8	−0.618	76.6	10	−0.343
	ldpc_decoder_28_70	40.1	9	−0.616	42.0	8	−0.710	42.5	8	−0.609	39.0	11	−0.753	38.6	8	−0.622	49.6	17	−0.330
	netcard_28_55	3.2	24,745	0.006	22.1	2574	−0.283	15.6	8396	−0.211	19.0	2821	−0.283	23.3	5322	−0.190	8.9	6969	−0.231
	netcard_28_70	3.7	24,921	−0.013	23.7	2596	−0.282	11.2	8066	−0.246	16.8	2620	−0.267	13.3	5113	−0.207	7.2	6657	−0.213
	leon3mp_28_55	1.7	120,483	0.074	8.0	31,247	−0.245	4.0	78,663	−0.029	53.0	9592	−0.308	3.5	73,884	−0.054	3.9	73,311	−0.091
	leon3mp_28_70	1.6	122,899	0.066	11.6	29,156	−0.248	3.2	74,776	−0.027	55.2	8976	−0.342	1.9	81,224	−0.025	2.3	76,294	−0.101
	MegaBoom_28_55	2.1	375,126	0.065	21.0	21,945	−0.352	2.2	404,678	0.031	30.7	13,496	−0.358	2.1	370,853	−0.017	2.9	476,911	0.118
	MegaBoom_28_70	1.5	329,231	0.043	21.6	11,939	−0.434	3.1	365,299	0.056	33.2	10,759	−0.387	2.1	354,274	0.016	3.1	210,473	0.015
	Average	1.0	1.0	1.0	5.5	0.4	0.7	2.0	0.7	0.9	11.7	0.4	0.7	2.0	0.7	0.9	1.7	0.8	1.1
w/I/O weights	jpeg_encoder_28_55	4.4	1922	−0.266	4.1	1529	−0.350	7.6	1593	−0.282	4.2	1300	−0.345	3.8	2148	−0.252	5.0	1294	−0.387
	jpeg_encoder_28_70	4.7	1859	−0.248	5.1	1598	−0.358	5.3	1493	−0.289	5.4	1238	−0.340	3.4	1882	−0.241	5.2	1355	−0.374
	ldpc_decoder_28_55	40.7	8	−0.660	48.3	8	−0.738	53.1	7	−0.639	52.8	10	−0.747	36.6	8	−0.661	29.8	9	−0.608
	ldpc_decoder_28_70	38.7	8	−0.661	38.6	8	−0.738	37.0	8	−0.654	48.7	11	−0.747	41.0	7	−0.642	62.0	8	−0.637
	netcard_28_55	17.4	2266	−0.417	29.2	1959	−0.449	11.1	3423	−0.392	21.2	1475	−0.552	32.7	1860	−0.477	7.6	9135	−0.176
	netcard_28_70	18.5	2114	−0.413	15.0	2061	−0.437	10.7	3394	−0.388	13.6	1370	−0.579	22.9	2079	−0.467	8.4	9680	−0.156
	leon3mp_28_55	3.7	69,923	−0.079	5.4	20,891	−0.273	3.4	60,302	−0.109	8.9	9724	−0.415	4.6	58,482	−0.087	2.0	97,302	−0.073
	leon3mp_28_70	2.5	81,649	−0.039	5.0	21,097	−0.259	3.1	62,439	−0.093	10.4	9576	−0.408	5.0	59,640	−0.085	2.6	94,111	−0.039
	MegaBoom_28_55	1.7	307,569	−0.036	5.3	96,128	−0.224	2.0	290,638	−0.090	5.0	63,940	−0.237	2.7	198,454	−0.192	1.7	397,838	0.005
	MegaBoom_28_70	1.8	287,494	−0.001	14.2	73,202	−0.238	1.9	232,260	−0.093	8.2	45,159	−0.297	2.3	206,758	−0.178	2.9	171,724	−0.061
	Average	2.1	0.6	0.8	3.6	0.3	0.7	1.8	0.5	0.8	3.5	0.4	0.6	2.9	0.5	0.8	1.5	0.6	0.9

[8] Since SC are values in the range −1 to 1, we add 1 to the values of SC before normalization.

[9] 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.

Table 5
Comparison among number of clusters (CL) and values of DBi, VRC, SC and runtime (CPU) for Louvain and hMetis. We highlight the best result for each evaluation criterion in each design.

Design	Louvain					hMetis 2-way					hMetis k-way				
	#CL	DBi	VRC	SC	CPU(s)	#CL	DBi	VRC	SC	CPU(s)	#CL	DBi	VRC	SC	CPU(s)
jpeg_encoder_28	84	3.6	4987.89	−0.096	2	64	3.1	10,171	0.042	13	84	**2.2**	**12,987**	**0.096**	14
ldpc_decoder_28	73	**40.1**	8.67188	−0.616	3	64	72.8	26	**−0.210**	18	73	95.3	**28**	−0.264	19
netcard_28	72	3.7	24920.5	−0.013	25	64	**3.5**	**67,680**	**0.122**	145	72	3.6	55,675	0.101	138
leon3mp_28	70	**1.6**	**122,899**	**0.066**	75	64	10.2	33,347	−0.053	191	70	21.4	28,753	−0.088	179
MegaBoom_28	40	**1.5**	**329,231**	**0.043**	448	70	35.1	33,897	−0.119	826	40	13.0	28,793	−0.137	912
Average		**1**	1	1	1		6.7	1.6	**1.2**	4.5		5.2	**1.7**	1.2	4.6

Table 6
Variation of DBi, SC and VRC with aspect ratios 1.5:1, 2:1, 2.5:1 and 3:1 compared to their implementation with aspect ratio 1:1. Values are normalized according to Equations (9)-(11).

Design	1.5:1			2.0:1			2.5:1			3.0:1		
	DBi	VRC	SC	DBi	VRC	SC	DBi	VRC	SC	DBi	VRC	SC
jpeg_encoder_14	0.20	−0.03	−0.25	−0.02	0.17	−0.50	0.06	0.17	−0.18	−0.09	0.60	0.25
ldpc_decoder_14	−0.20	−0.12	0.02	0.16	0.22	−0.13	−0.13	0.22	−0.01	0.18	0.13	−0.24
netcard_14	−0.13	−0.32	−0.35	0.02	−0.18	−0.26	−0.61	−0.18	0.19	−0.06	0.52	0.41
leon3mp_14	−0.31	−0.04	−0.38	−0.32	0.10	0.02	−0.01	0.10	−0.62	0.10	0.12	−1.41
MegaBoom_14	0.22	−0.01	−0.45	0.22	0.93	−0.22	0.19	0.93	−0.22	0.02	2.03	−0.13
Average	−0.04	−0.10	−0.28	0.01	0.25	−0.22	−0.10	0.25	−0.17	0.03	0.68	−0.22
Std. dev.	0.24	0.13	0.19	0.21	0.41	0.19	0.31	0.41	0.30	0.11	0.79	0.71

aspect ratio 1:1 to aspect ratios 1.5:1, 2:1, 2.5:1 and 3:1. The values of our evaluation criteria are normalized using Equations (9)-(11), so that 0 means no change with respect to 14 nm, positive values mean improvement and negative values mean degradation. We observe a significant variation in every criterion when we change the floorplan. For instance, we see a 93% improvement in VRC for MegaBoom considering the aspect ratio 2:0 and 61% degradation of DBi in netcard considering aspect ratio 2.5:1. The numbers do not follow any trend and the standard deviation can be as large as 79%. The reason for this behavior can be seen in Fig. 5. The significant variation in our evaluation criterion comes from the chaotic behavior of the placement tool. The neighborhood and shape of the clusters determine our evaluation criteria. We highlight four clusters to compare the different placement solutions. Clusters 1 and 2 are placed next to each other in all the five solutions. However, in aspect ratio 1:1, cluster 1 is at the core boundary, while in aspect ratio 2.5:1 both clusters are not in the core boundary. We observe similar behavior in clusters 3 and 4. Clusters 3 and 4 are placed next to clusters 1 and 2 in aspect ratio 1.5:1 and 2.5:1, but are placed far apart in the other aspect ratios. We may conclude that DBi, VRC and SC are good metrics by which to compare clustering solutions for the same ground-truth placement, but not by which to compare the same clustering for different placements.

$$\Delta DBi = 1 - \frac{DBi_{tech}}{DBi_{14}} \tag{9}$$

$$\Delta VRC = \frac{VRC_{tech}}{VRC_{14}} - 1 \tag{10}$$

$$\Delta SC = \frac{SC_{tech} - SC_{14}}{|SC_{14}|} \tag{11}$$

4.4. Experiment 4: validation across technologies

Ideally, Louvain would find a similar number of clusters for different gate-level netlists originated from the same RTL (e.g., two netlists, one synthesized in a 14 nm enablement and the other in a 28 nm enablement.) However, the features of the netlist graph (e.g., average cardinality of the vertices) may vary depending on the enablement used in the synthesis. The difference happens due to the number and types of logic functions available in the standard cell library and their implementation (e.g., number of available VTs and drive strengths.) In this experiment, we assess the robustness of Louvain by comparing leon3mp, MegaBoom and netcard synthesized using three enablements: 14 nm, 28 nm and 65 nm. From Table 3, the reader can see the impact of these details in synthesis – e.g., MegaBoom_14 has 6.8% more instances than MegaBoom_65 and 13% fewer instances than MegaBoom_28. netcard_14 has 12% more instances than netcard_28 and netcard_65. The difference is more significant between leon3mp_14 and leon3mp_28 (27%). The difference in synthesis affects the values of DBi, VRC and SC, as shown in Table 7. For example, MegaBoom_28 presents 3× better VRC and 1.48× better SC than MegaBoom_14, and MegaBoom_65 has 1.19× better VRC than MegaBoom_14. However, netcard_65 presents 2.35× worse SC than netcard_14. Fig. 6 shows the instances of MegaBoom_28 and MegaBoom_65, colored according to Louvain clustering. The visualization for MegaBoom_14 can be seen in Fig. 5(a). The number of clusters found by Louvain also changes significantly. MegaBoom_28 and MegaBoom_65 have 40 and 37 clusters, respectively, in contrast to 27 clusters of MegaBoom_14. The numerical and visual results of Louvain for a given RTL synthesized in 14 nm, 28 nm and 65 nm technology nodes vary significantly. Therefore, we conclude that the current implementation of Louvain is not “robust” with respect to changes of technology nodes. As a possible extension of this work, we believe the “robustness” of Louvain could be improved using hints from the RTL hierarchy. For instance, Louvain could assign higher weights to edges that connect instances belonging to the same RTL hierarchy.

5. Closing the loop: integration with ‘blob placement’ and ‘seeded placement’

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 con-

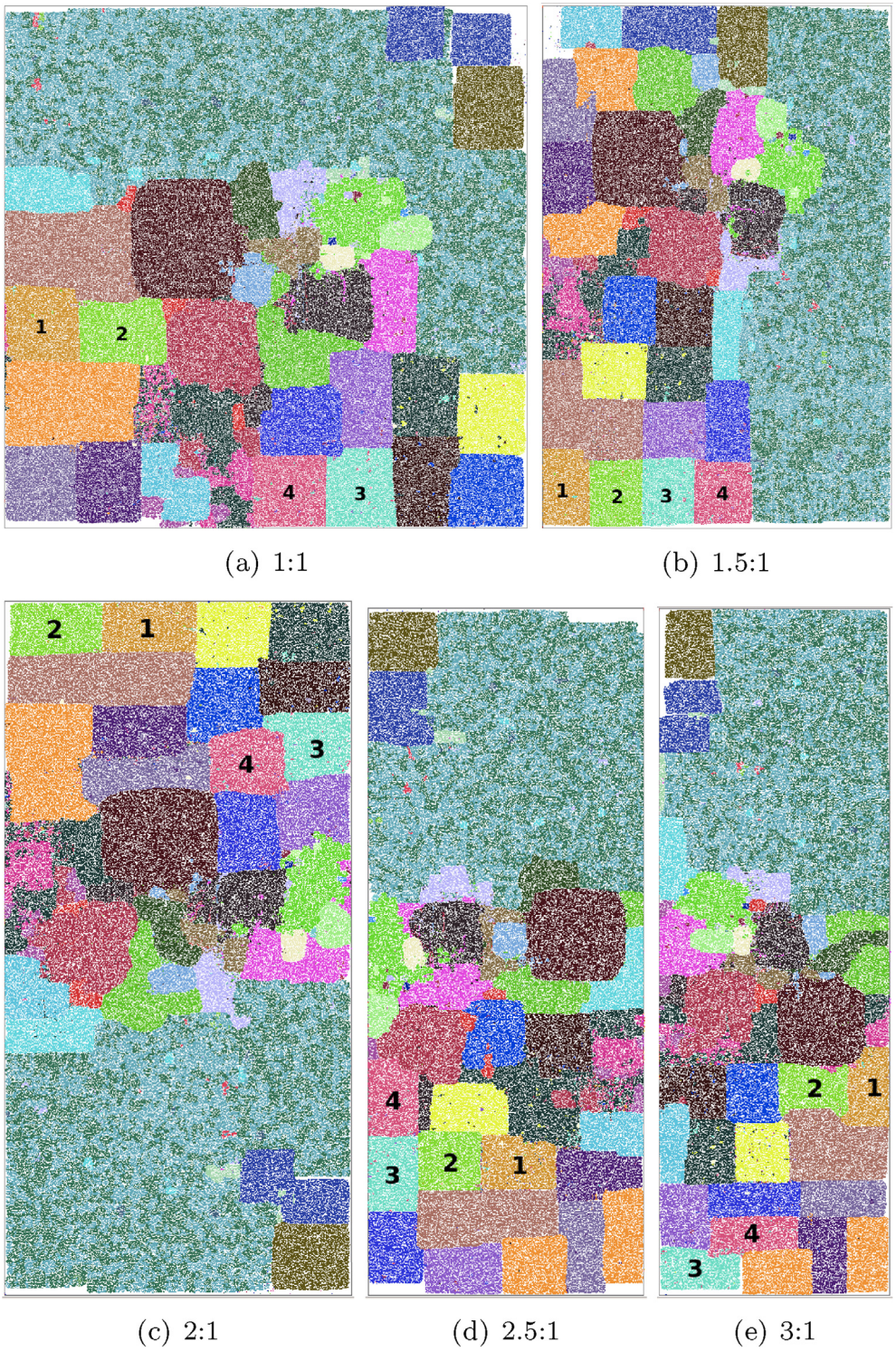
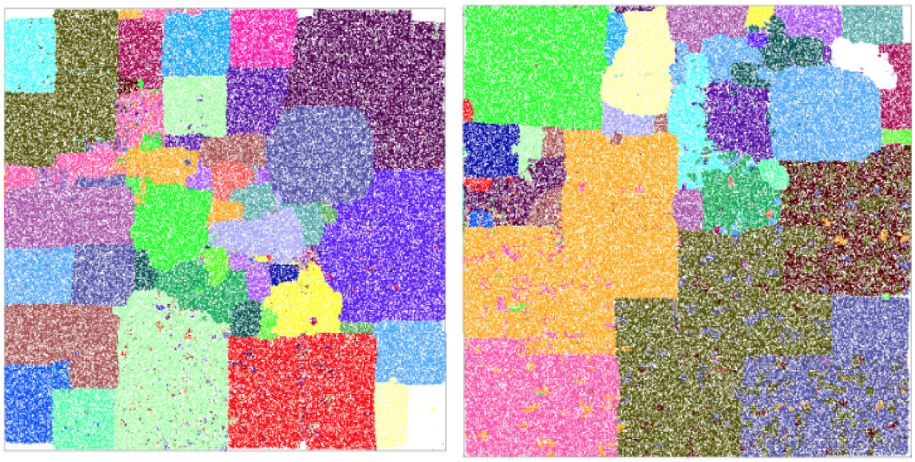


Fig. 5. Visual comparison of MegaBoom_14 with different aspect ratios and same utilization. The images have been scaled for a better visualization.

figuration or tuning, (ii) a "blob placement" step that performs cluster placement and shaping, and (iii) a fast placement of the flat netlist using a "seeded placement" originated from the "blob placement". The flow is depicted in Fig. 7.

The initial 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. In our experiments, a single iteration of hierarchical clustering is sufficient to produce small blobs. Then, 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



(a) (b)

Fig. 6. Clustering results for (a) MegaBoom_28 and (b) MegaBoom_65. Compare with MegaBoom_14 from Fig. 5(a).

Table 7
Variation of DBi, VRC and SC for leon3mp and MegaBoom when compared to their implementation in 14 nm. Values are normalized according to Equations (9)-(11).

Design	DBi	VRC	SC
netcard_28	−0.03	−0.07	0.82
netcard_65	−1.87	−0.71	−2.35
leon3mp_28	0.32	0.30	0.48
leon3mp_65	0.28	−0.01	0.64
MegaBoom_28	0.55	3.23	1.48
MegaBoom_65	0.39	1.19	0.25

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. Furthermore, we note that our clusters are not loosely-connected. In MegaBoom_14, MegaBoom_28 and MegaBoom_65, we observe 21 K, 12 K and 19 K clusters, and 72 K, 59 K and 74 K inter-cluster nets, respectively. The same behavior is observed in other testcases, such as netcard_14 and leon3mp_14 that have 2.8 K and 2.6 K clusters, and 27 K and 37 K nets, respectively.

Fig. 8 depicts the outcome of "blob placement" for the MegaBoom_14 that has 27 root blobs and 21 K leaf blobs. The root blobs contain an average of 46 K instances and leaf blobs contain an average of 59 instances. We adapt the open-source academic tool RePlAce to perform the blob placement. In doing so, we inflate the blob dimensions by 20%, 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 12min for MegaBoom_14 (1.2 M instances), using a single thread of a 2.1 GHz Xeon server.[10]

Next, we create a "seeded placement" based on the blob locations. In the "seeded placements", we restore the initial flat netlist and place each instance in the center coordinate of the blob that represents the instance cluster. Finally, we feed the "seeded placement" to RePlAce which spreads the instances while minimizing the wirelength. Fig. 8(a) shows the blob placement and Fig. 8(b) shows the final flat placement for MegaBoom_14.

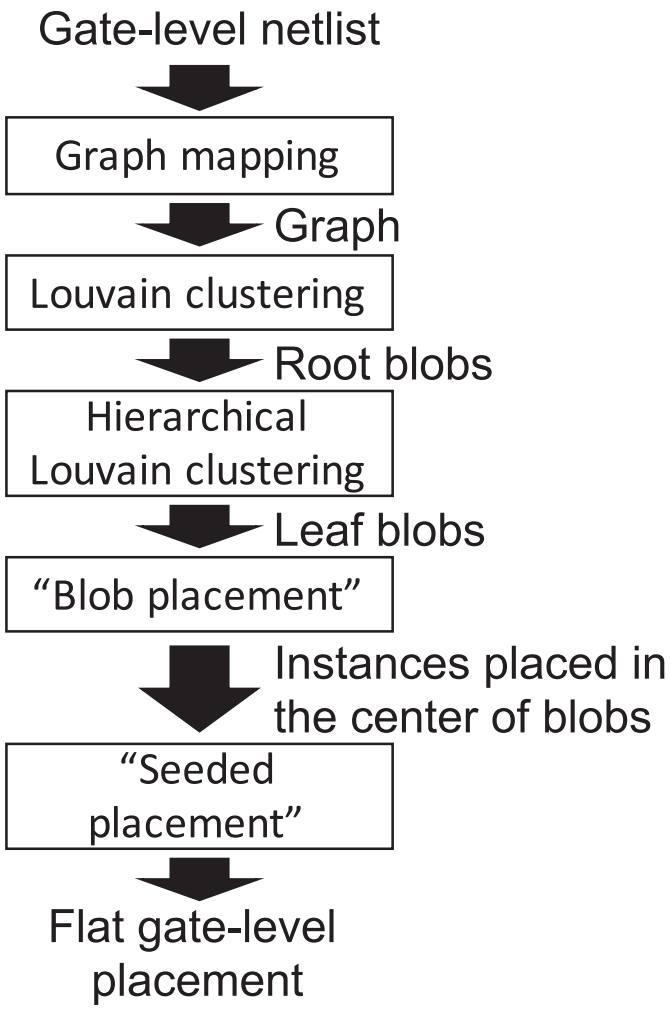


Fig. 7. Experimental fast placement flow.

Table 8 shows the routed wirelength and runtime of the flat placement and fast placement to our six largest testcases in 14 nm.[11] The

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

[11] We use a commercial tool to perform global routing and extract routed estimated wirelength information.

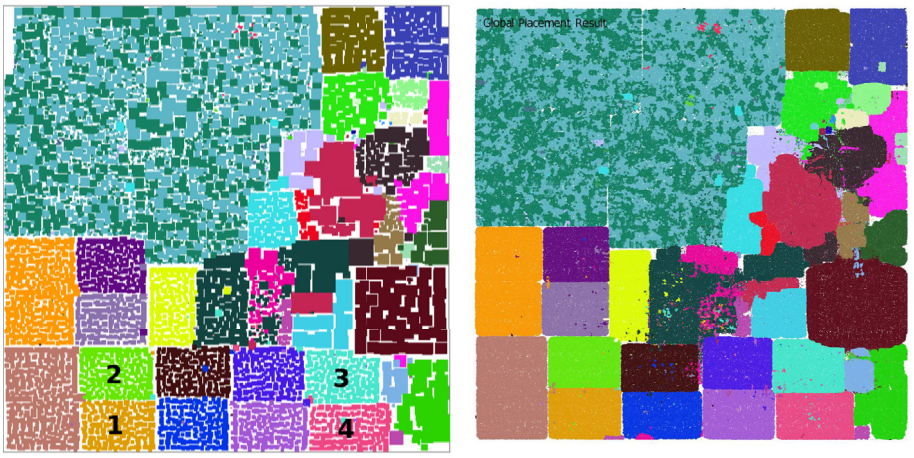


Fig. 8. MegaBoom_14: (a) "blob placement" and (b) "seeded placement". Compare to the flat placement of Fig. 5(a).

Table 8
Results of fast placement using "seeded placement." The runtime of fast placement is broken into Louvain clustering (LC), hierarchical Louvain clustering (HLC), "blob placement" (BP) and "seeded placement" (SP.) Refer back to Table 3 for the instance complexities.

Design	Flat placement		Fast "seeded" placement					
	WL(m)	CPU(s)	WL(m)	LCCPU(s)	HLCCPU(s)	BPCPU(s)	SPCPU(s)	TotalCPU(s)
MegaBoom_14	21.0	1941	21.3	268	266	184	437	1156
MegaBoom_28	36.0	1623	36.2	335	108	116	408	967
MegaBoom_65	57.5	1613	57.0	164	100	104	432	800
MegaBoom_14_X2	41.6	3214	41.8	705	271	440	1299	2714
MegaBoom_14_X3	62.3	5211	64.1	1020	415	943	1933	4311
MegaBoom_14_X4	83.4	8642	85.6	1418	572	1400	3492	6882

total runtime of fast placement is broken into Louvain clustering, hierarchical Louvain clustering, "blob placement" and "seeded placement". Our experimental flow presents runtime speed-ups that range from 20% (MegaBoom_X4_14) to 50% (MegaBoom_14 and MegaBoom_65). The largest chunk of runtime in our experimental flow comes from the "seeded placement" itself, followed by Louvain clustering. The hierarchical Louvain clustering is the step that scales best. We also note that hierarchical Louvain clustering is parallelizable because the step consists of applying clustering hierarchically in the root blobs – for an 8-thread CPU, the runtime can be potentially reduced between 6× and 8×. The use of "seeded placement" causes a minimal degradation in wirelength that ranges from 0.4% (MegaBoom_X2_14) to 2.8% (MegaBoom_X3_14). We observe a slight improvement of 0.9% in the wirelength of MegaBoom_65.

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 (i) *alpha shapes* and Delaunay triangulation of a cluster's placed locations for manual debug and visualization and (ii) the Davies-Bouldin index, Variance Ratio Criterion and Silhouette Coefficient as numerical criteria.

For the purpose of predicting cohesion in final layouts, we find that *modularity*-driven clustering, as exemplified by the Louvain [14] algorithm, is clearly superior to mincut- or Rent parameter-driven methods [11,8,10] that have dominated the VLSI CAD literature. Importantly, the modularity criterion 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 a weighting approach of Lengauer [50] is effective in conjunction with Louvain. Comparisons with traditional hMetis-based clustering [10] show that our Louvain-based approach achieves on average 50% better correlation to actual netlist placements, as well as 2× 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 work leaves a number of open directions for future research. First, we believe that much richer tuning of the hypergraph-to-graph mapping is possible according to additional instance and netlist attributes (stage within logic cone, special status of buffer/inverter instances, etc.). Second, static timing analysis can be used to inject timing information into the net weighting, likely improving the clustering results. Third, design hierarchy and structure may also help clustering to "avoid mistakes", i.e., by providing name or clock contexts to additionally guide the graph construction and hence the clustering. Finally, we believe that parallelism can be exploited to speed up our prototype "seeded placement" flow.

CRediT authorship contribution statement

Mateus Fogaça: Software, Investigation, Writing - original draft. **Andrew B. Kahng:** Conceptualization, Writing - review & editing, Supervision. **Eder Monteiro:** Investigation, Writing - original draft. **Ricardo Reis:** Writing - review & editing, Supervision. **Lutong Wang:** Software, Writing - review & editing. **Mingyu Woo:** Software, Investigation, Writing - original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

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] International technology roadmap for Semiconductors. http://www.itrs2.net/itrs-reports.html.
[2] A. Olofsson, Silicon compilers - version 2.0, keynote, in: Proc. ISPD, 2018, http://www.ispd.cc/slides/2018/k2.pdf.
[3] A.B. Kahng, Reducing time and effort in IC implementation: a roadmap of challenges and solutions, in: Proc. DAC, 2018, pp. 1–6.
[4] DARPA rolls out electronics resurgence initiative. https://www.darpa.mil/news-events/2017-09-13.
[5] J.A. Roy, S.N. Adya, D.A. Papa, I.L. Markov, Min-cut floorplacement, IEEE Trans. CAD 25 (7) (2006) 1313–1326.
[6] R.S. Shelar, An efficent clustering algorithm for low power clock tree synthesis, in: Proc. ISPD, 2007, pp. 181–188.
[7] K. Blutman, H. Fatemi, A.B. Kahng, A. Kapoor, J. Li, J.P. de Gyvez, Floorplan and placement methodology for improved energy reduction in stacked power-domain design, in: Proc. ASP-DAC, 2017, pp. 444–449.
[8] K. Jeong, A. B. Kahng and H. Yao, RentCon: Rent Parameter Evaluation Using Different Methods. https://vlsicad.ucsd.edu/WLD/index.html.
[9] C.M. Fiduccia, R.M. Mattheyses, A linear-time heuristic for improving network partitions, Proc. DAC (1982) 175–181.
[10] G. Karypis, R. Aggarwal, V. Kumar, S. Shekhar, Multilevel hypergraph partitioning: applications in VLSI domain, in: Proc. DAC, 1997, pp. 526–529.
[11] A.E. Caldwell, A.B. Kahng, I.L. Markov, Improved algorithms for hypergraph bipartitioning, in: Proc. ASP-DAC, 2000, pp. 661–666.
[12] S. Fortunato, D. Hric, Community detection in networks: a user guide, Phys. Rep. 659 (2016) 1–44.
[13] M.E.J. Newman, M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69 (2004) 1–15.
[14] V.D. Blondel, J.L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, J. Stat. Mech. Theor. Exp. 10 (2008) 1–12.
[15] D.B. Mark, M. Overmars, O. Cheong, Computational Geometry: Algorithms and Applications, Springer, New York, 1997.
[16] D.L. Davies, D.W. Bouldin, A cluster separation measure, IEEE Trans. PAMI 1 (2) (1979) 224–227.
[17] T. Caliski, J. Harabasz, A dendrite method for cluster Analysis, J. Commun. Stat. 3 (1974) 1–27.
[18] P.J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster Analysis, J. Comput. Appl. Math. 20 (1987) 53–65.
[19] C.J. Alpert, A.B. Kahng, Recent directions in netlist partitioning: a survey, Integrat. VLSI J. 19 (12) (1995) 1–81.
[20] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Syst. Tech. J. 49 (1970) 291–307.
[21] K.M. Hall, An r-dimensional quadratic placement algorithm, Manag. Sci. 17 (1970) 219–229.
[22] E.R. Barnes, An algorithm for partitioning the nodes of a graph, in: Proc. CDC, 1981, pp. 303–304.
[23] S. Ou, M. Pedram, Timing-driven bipartitioning with replication using iterative quadratic programming, in: Proc. ASP-DAC, 1999, pp. 105–108.
[24] H. Yang, D.F. Wong, Efficient network flow based min-cut balanced partitioning, in: Proc. ICCAD, 1994, pp. 50–55.
[25] L. Hagen, A.B. Kahng, A new approach to effective circuit clustering, in: Proc. ICCAD, 1992, pp. 422–427.
[26] C.N. Sze, T.-C. Wang, Performance-driven multi-level clustering for combinational circuits, in: Proc. ASP-DAC, 2003, pp. 729–734.
[27] A.B. Kahng, X. Xu, Local unidirectional bias for smooth cutsize-delay tradeoff in performance-driven bipartitioning, in: Proc. ISPD, 2003, pp. 81–86.
[28] T. Jindal, C.J. Alpert, J. Hu, Z. Li, G.-J. Nam, C.B. Winn, Detecting tangled logic structures in VLSI netlists, in: Proc. DAC, 2010, pp. 603–608.
[29] M. Girvan, M.E.J. Newman, Community structure in social and biological networks, in: Proc. Natl. Acad. Sci., vol. 99, 2002, pp. 7821–7826.
[30] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, D. Parisi, Defining and identifying communities in networks, in: Proc. Natl. Acad. Sci., vol. 101, 2004, pp. 2658–2663. 9.
[31] P. Pons, M. Latapy, Computing communities in large networks using random walks, JGAA 10 (2) (2006) 191–218.
[32] A. Clauset, M.E.J. Newman, C. Moore, Finding community structure in very large networks, Phys. Rev. E 70 (2004) 66111–66116.
[33] M.E.J. Newman, Finding community structure in networks using the eigenvectors of matrices, Phys. Rev. E 74 (3) (2006) 36104–36122.
[34] F. Wu, B.A. Huberman, Finding communities in linear time: a physics approach, Eur. Phys. J. B. 38 (2) (2004) 331–338.
[35] K. Wakita, T. Tsurumi, Finding community structure in mega-scale social networks, in: Proc. WWW, 2007, pp. 1275–1276.
[36] Louvain. https://sites.google.com/site/findcommunities/.
[37] T. Heuer, S. Schlag, Improving coarsening schemes for hypergraph partitioning by exploiting community structure, in: Proc. SEA, 2017, pp. 21:121:19.
[38] N. Neubauer, K. Obermayer, Towards community detection in k-partite k-uniform hypergraphs, in: Proc. NIPS, 2009, pp. 1–9.
[39] N. Neubauer, K. Obermayer, Community detection in tagging-induced hypergraphs, in: Proc. WIN, 2010, pp. 24–25.
[40] T. Kumar, S. Vaidyanathan, H. Ananthapadmanabhan, S. Parthasarathy, B. Ravindran, Hypergraph Clustering: A Modularity Maximization Approach. arXiv https://arxiv.org/abs/1812.10869, 2018.
[41] T. Kumar, S. Vaidyanathan, H. Ananthapadmanabhan, S. Parthasarathy, B. Ravindran, A new measure of modularity in hypergraphs: theoretical insights and implications for effective clustering, in: Proc. Complex Networks, 2019, pp. 286–297.
[42] B. Kamiski, V. Poulin, P. Praat, P. Szufel, F. Thberge, Clustering via hypergraph modularity, PloS One 14 (11) (2019) 1–15.
[43] F. Chung, L. Lu, Connected components in random graphs with given expected degree sequences, Ann. Combinator. 6 (2) (2002) 125–145.
[44] Clustering via hypergraph modularity. https://gist.github.com/pszufe/.
[45] B. Chen, M. Marek-Sadowska, Timing-driven placement of pads and latches, in: Proc. IEEE ASIC Conf., 1992, pp. 30–33.
[46] S. Mantik, G. Posser, W. Chow, Y. Ding, W.H. Liu, ISPD 2018 initial detailed routing contest and benchmarks, in: Proc. ISPD, 2018, pp. 140–143.
[47] H. Edelsbrunner, D.G. Kirkpatrick, R. Seidel, On the shape of a set of points in the plane, IEEE Trans. Inf. Theor. 29 (4) (1983) 551–559.
[48] Alpha shape. https://en.wikipedia.org/wiki/Alpha_shape.
[49] E. Ihler, A.D. Wagner, F. Wagner, Modeling hypergraphs by graphs with the same mincut properties, Inf. Process. Lett. 45 (1993) 171–175.
[50] T. Lengauer, Combinatorial Algorithms for Integrated Circuit Layout, Wiley-Teubner, New York, 1990.
[51] D.J.H. Huang, A.B. Kahng, When clusters meet partitions: new density-based methods for circuit decomposition, in: Proc. European Design and Test Conf., 1995, pp. 60–64.
[52] R.-S. Tsay, E.S. Kuh, A unified approach to partitioning and placement, IEEE Trans. Circ. Syst. 38 (1991) 521–533.
[53] J. Frankle, R.M. Karp, Circuit placement and cost bounds by eigenvector decomposition, in: Proc. ICCAD, 1986, pp. 414–417.
[54] G. Flach, M. Fogaa, J. Monteiro, M. Johann, R. Reis, Rsyn an extensible physical synthesis framework, in: Proc. ISPD, 2017, pp. 33–40.
[55] Rsyn. https://github.com/RsynTeam/rsyn-x.
[56] OpenCores: open source IP-cores. http://www.opencores.org.
[57] C.-K. Cheng, A.B. Kahng, I. Kang, L. Wang, RePlAce: advancing solution quality and routability validation in global placement, IEEE Trans. CAD (2018), https://doi.org/10.1109/TCAD.2018.2859220.
[58] RePlAce version 1.1.1. https://github.com/The-OpenROAD-Project/RePlAce/tree/1.1.1.
[59] S. Do, M. Woo, S. Kang, Fence-region-aware mixed-height standard cell legalization, Proc. GLSVLSI (2019) 259–262.
[60] OpenDP version 0.1.0. https://github.com/The-OpenROAD-Project/OpenDP/tree/0.1.0.
[61] A.B. Kahng, J. Li, L. Wang, Improved flop tray-based design implementation for power reduction, in: Proc. ICCAD, 2016, pp. 20:1-20:8.
[62] scikit-learn. https://scikit-learn.org/stable/modules/clustering.html.
[63] M. Fogaa, A.B. Kahng, R. Reis, L. Wang, Finding placement-relevant clusters with fast modularity-based clustering, Proc. ASPDAC (2019) 569–576.