UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

MATEUS PAIVA FOGAÇA

# Finding Placement-Relevant Clusters With Fast Modularity-Based Clustering

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Microelectronics

Advisor: Prof. Dr. Ricardo A. da L. Reis
Coadvisor: Prof. Dr. Andrew B. Kahng

Porto Alegre
Jun 2020

**CIP — CATALOGING-IN-PUBLICATION**

Fogaça, Mateus Paiva

Finding Placement-Relevant Clusters With Fast Modularity-Based Clustering / Mateus Paiva Fogaça. – Porto Alegre: PGMICRO da UFRGS, 2020.

88 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica, Porto Alegre, BR–RS, 2020. Advisor: Ricardo A. da L. Reis; Coadvisor: Andrew B. Kahng.

1. Microelectronics. 2. EDA. 3. Physical Design. 4. Floorplanning. 5. Placement. 6. Modularity-Based Clustering. I. Reis, Ricardo A. da L.. II. Kahng, Andrew B.. III. Título.

*"Men are rather reasoning than reasonable animals,*
*for the most part governed by the impulse of passion."*
— ALEXANDER HAMILTON

# ACKNOWLEDGMENTS

I want to thank the many colleagues and friends have that contributed to my academic life. Many thanks to Cristina Meinhardt, Guilherme Flach, Jucemar Monteiro, Ygor Aguiar, Eder Monteiro, Gracieli Posser, Cristal Villalba, Geancarlo Abich, Marcelo Johann, among many others! I acknowledge my advisor, Prof. Ricardo Reis, for his guidance and commitment to my professional growth.

The current thesis is not the result of an “one man army” – It is the result of a collaboration that started in 2018 when Prof. Andrew Kahng has accepted me as a visiting Ph.D. student in his research group. I want to thank him for this lasting and profitable collaboration. I also thank Lutong Wang (UCSD), Mingyu Woo (UCSD) and Eder Monteiro (UFRGS) for putting their hard work in the project that resulted in the current thesis. I have written the remainder of the present document using the plural form to honor the effort of everyone that has contributed to the project.

I send a special acknowledgment to Prof. Paulo Butzen, not only for helping in the elaboration of the present document but also for reminding me that I have friends willing to help without any expectation of return. Thanks for the enormous technical and emotional support, Paulo! :-)

Finally, I acknowledge CAPES for the financial support throughout most of my Ph.D. and DARPA/Precision Innovations, Inc. for the financial support in the final months of my Ph.D.

## ABSTRACT

In advanced technology nodes, IC implementation faces an 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 to avoid bottlenecks and loops in the physical implementation flow. Notably, with modern design methodologies it would be very valuable to better predict the 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. 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$\times$ 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 4.9M instances with less than 3% difference in routed wirelength.

**Keywords:** Microelectronics. EDA. Physical Design. Floorplanning. Placement. Modularity-Based Clustering.

**Encontrando Grupos Relevantes ao Posicionamento com Agrupamento Baseado em Modularidade**

## RESUMO

Em nodos tecnológicos avançados, a implementação de circuitos integrados deve lidar com o aumento da complexidade dos projetos e também com cronogramas mais restritos. Portanto, cria-se a necessidade de novas abordagens de *decomposição* que ajudem a reduzir a complexidade do problema e novas *metodologias preditivas* para evitar gargalos e iterações no fluxo de implementação. Em metodologias de projeto modernas, seria útil predizer o posicionamento do circuito em nível de portas lógicas. Essa habilidade tornaria possível avaliar com maior precisão a planta baixa de um circuito em termos de desempenho e congestionamento ainda nas etapas de projeto da planta baixa e planejamento da descrição do circuito em nível de transferências de registradores de sistemas em *chip*. Este trabalho apresenta um novo critério de avaliação do problema clássico de agrupamento do circuito em nível de portas lógicas: avaliar se as portas lógicas de um grupo "permanecem próximas" ao longo do fluxo de implementação. Métodos para a avaliação desse critério são propostos. Além disso, o trabalho utiliza uma classe de técnicas de agrupamento chamada de *agrupamento baseado em modularidade* para identificar "grupos naturais" em um grafo, dispensando a necessidade de ajustes de parâmetros do algoritmo ou restrições de balanceamento de tamanho dos grupos, tradicionalmente necessários em técnicas de particionamento utilizadas por ferramentas de CAD (do inglês, *computer-aided design*). Os experimentos realizados mostram que o mapeamento do circuito de hipergrafo para um grafo afeta significativamente a qualidade dos resultados. Também demonstra-se empiricamente que grupos obtidos com técnicas de agrupamento baseadas em modularidade possuem uma maior correlação com o posicionamento quando comparadas com técnicas de particionamento tradicionalmente empregadas por ferramentas de CAD (A técnica utilizada neste trabalho também é $2\times$ mais rápida que a ferramenta de particionamento tradicional hMetis nos maiores casos de teste). Por fim, é proposto um fluxo no qual se realiza posicionamento de grupos ("posicionamento de bolhas"). O "posicionamento de bolhas" é utilizado com ponto de partida ("semente") para uma ferramenta de posicionamento global. A ferramenta de posicionamento global utiliza a semente para realizar o posicionamento das portas lógicas do circuito. O fluxo proposto permite reduzir em 20% o tempo do posicionamento do circuito e a diferença nos resultados é menor que 3% em termos de comprimento de fios.

**Palavras-chave:** Microeletrônica. Automação do Projeto Eletrônico. Projeto Físico. Planta Baixa. Posicionamento. Agrupamento Baseado em Modularidade.

## LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| AS | Alpha shape |
| BP | “Blob placement” |
| CAD | Computer-aided design |
| CH | Convex hull |
| CL | Number of clusters |
| CNM | Clauset Newman Moore |
| CPU | Central processing unit |
| CTS | Clock tree synthesis |
| DARPA | Defense Advanced Research Projects Agency |
| DBi | Davies–Bouldin index |
| DBSCAN | Density-based spatial clustering of applications with noise |
| DRC | Design rule checking |
| DSP | Digital signal processing |
| DT | Delaunay triangulation |
| EDA | Electronic design automation |
| ERC | Electrical rule checking |
| GDSII | Graphic Design System II |
| HDL | Hardware description language |
| HLC | Hierarchical Louvain clustering |
| HPWL | Half-perimeter wirelength |
| I/O | Input/output |
| IC | Integrated circuit |
| IDEA | Intelligent Design of Electronic Assets |
| ISPD | International Symposium on Physical Design |

| | |
|---|---|
| LC | Louvain clustering |
| LIFO | Last in, first out |
| LVS | Layout versus schematic |
| NP | Nondeterministic polynomial |
| PDK | Process design kit |
| RAM | Random access memory |
| R&D | Research and development |
| RTL | Register-transfer level |
| SC | Silhouette coefficient |
| SCAN | Structural clustering algorithm for networks |
| SOC | System on chip |
| SP | “Seeded placement” |
| SPICE | Simulation program with integrated circuit emphasis |
| TT | Typical-typical corner |
| VHDL | Very high speed integrated circuit hardware description language |
| VILE | Very illegal |
| VLSI | Very large scale integration |
| VRC | Variance ratio coefficient |
| VT | Threshold voltage |
| WL | Wirelength |

# LIST OF FIGURES

## LIST OF TABLES

## CONTENTS

# 1 INTRODUCTION

Modern systems-on-chips (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 (ITRS, 2015). 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 (OLOFSSON, 2018) 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 (KAHNG, 2018). 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 (DARPA, 2018) 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 (OLOFSSON, 2018; DARPA, 2018). 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 mandatory to achieve the IDEA program goal.[1]

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 (ROY et al., 2006), clock tree synthesis (SHELAR, 2007) and, more recently, grouping of instances into different power domains (BLUTMAN et al., 2017). While many clustering methods for VLSI have been proposed, they have largely focused on *net cuts* (hyperedge min-cut, cluster perimeter, Rent parameter (RENTCON, 2008), etc.). Further, existing heuristics typically require design-dependent tuning and suboptimal heuristics. For instance, the well-known multilevel Fiduccia-Mattheyses (FIDUCCIA; MATTHEYSES, 1982) implementations hMetis (KARYPIS et al., 1997) and MLPart (CALDWELL; KAHNG; MARKOV, 2000) require *a priori* the target number of partitions as an input, and each aims to balance

[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.

the number of vertices or total vertex area across the partitions, which conflicts with the min-cut objective.

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*. Additionally, we propose a prototype flow that performs a fast placement of the netlist. 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.

Among the contributions of this work, we mention three 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 (FORTUNATO; HRIC, 2016). Community detection methods rely on metrics that help identify natural clusters inside graphs, notably, the *modularity* criterion (NEWMAN; GIRVAN, 2004). Our study centers on Louvain (BLONDEL et al., 2008), 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) (BERG et al., 1997). 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) (DAVIES; BOULDIN, 1979), Variance Ratio Criterion (VRC) (CALIńSKI; HARABASZ, 1974) and Silhouette Coefficient (SC) (ROUSSEEUW, 1987), 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.

The third aspect is the proposal of a prototype flow that performs fast placement of clusters to predict the flat placement. In doing so, we feed a state-of-the-art analytic placement tool with a *cluster netlist* that is significantly smaller than the gate-level netlist. Then, we use the cluster locations to generate an initial placement for the gate-level netlist. Finally, we finish the placement with a fast call of incremental placement. We compare predicted and actual gate-level netlist placement in terms of routed wirelength with a leading commercial tool router.

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 (DAVIES; BOULDIN, 1979), Variance Ratio Criterion (CALIńSKI; HARABASZ, 1974) and Silhouette Coefficient (ROUSSEEUW, 1987) 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 (BLONDEL et al., 2008) versus the traditional VLSI netlist clustering tool hMetis (KARYPIS et al., 1997), with 2$\times$ 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 1.2M

[2]The silhouette metric has not been widely used in the VLSI CAD clustering literature, with (KAHNG; LI; WANG, 2016) being the only example of which we are aware.

instances and 20% speed up for a testcase with 4.5M instances.[3]

The remainder of the current thesis is organized as follows. Chapter 2 introduces the basic concepts for the understanding of this thesis. Chapter 3 gives an overview of the existing literature on VLSI partitioning, modularity clustering and placement. Chapter 4 presents our comparison between traditional VLSI clustering methods and modularity based clustering while Chapter 5 proposes our prototype "blob placement flow". In Chapter 6, we present our final remarks.

[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. Additionally, our flow can be used to generate the actual placement in contexts where the quality of results can be sacrificed to reduce the placement runtime.

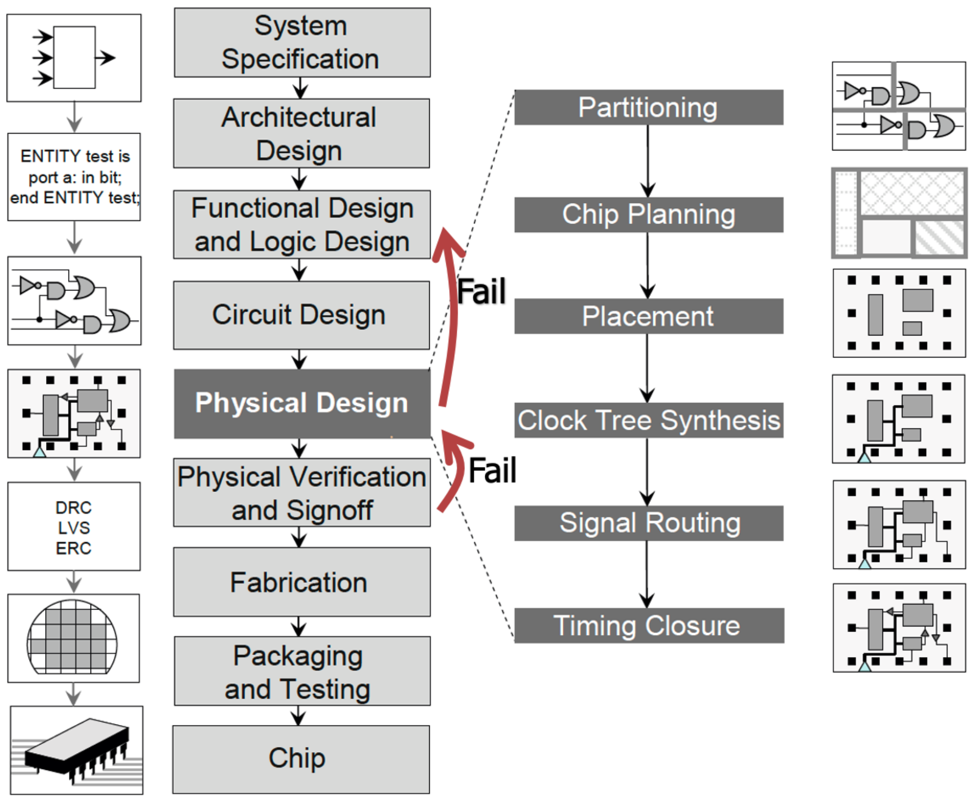
# 2 PRELIMINARIES

In this Chapter, our goal is to provide for the reader the fundamental background to understand the current thesis. We start reviewing the design flow of digital circuits in Section 2.1. In Section 2.2, we show how VLSI CAD algorithms represent the netlist hypergraph with graphs. Sections 2.3 and Section 2.4 define the concepts of partitioning and clustering. Finally, Section 2.5 presents wirelength estimation with half-perimeter wirelength. Readers with background on VLSI CAD literature may skip this Chapter.

## 2.1 Design Flow of Digital Circuits

The VLSI design flow is a set of steps that transform the functional description of a system into geometric masks that allow the system to be manufactured as an integrated circuit. Each step is performed by teams of engineers with expertise in their field. Due to the tight design schedule, engineers rely on hundreds of licenses of foundry-qualified EDA tools and massively parallel servers. Figure 2.1 depicts the design flow according to Kahng et al. (2011).

Figure 2.1: VLSI design flow.



Source: Kahng et al. (2011).

The first step of the flow is the **system specification**. This step consists in defining the overall goals and requirements of the system in terms of functionality, performance and area. The production teams also decide what is the target technology for the product. Next, the **architectural design** determines how the system is going to meet the system specification in terms of analog and mixed-signal blocks, memory configuration, number of cores, DSPs, I/Os, IP blocks, die packing interface, power requirements, technology process choice, layer stacks choice, etc.

In the **functional and logic design**, the functionality and connectivity of each module in the architecture are defined. The high-level behavior of the system is modeled using hardware description languages (HDLs), such as Verilog and VHDL. These descriptions are validated in terms of behavior and timing using thorough simulations. After validation, *logic synthesis* tools translate the high-level description into a circuit. This is done by specifying the system description and a technology library and results in a list of signal nets and logic gates mapped to the technology (e.g. logic gates mapped to standard cells.) The result of this step is called *gate-level netlist*. However, some critical elements, such as RAMs and I/Os, have to be validated by SPICE simulations using transistor-level descriptions. Such descriptions are generated in the **circuit design**.

The **physical design** produces a geometric description of the design. Kahng et al. (2011) divide the physical design into six steps:

- **Partitioning**. Splits the flat netlist into smaller modules so they can be designed and analyzed individually. Partitioning allows algorithms that do not scale well to be applied in the modern and complex system-on-chip designs. Partitions can also be assigned to different teams and designed in parallel.
- **Chip planning**. Commonly referred to as floorplanning. Determines the area and location of the design modules. Also determines the location of I/O ports and macro blocks. The chip planning also is responsible for the realization of power and ground networks.
- **Placement**. Finds a location for each element of the netlist while trying to optimize an objective function. Common placement objectives include wirelength, timing, routability and power distribution. A review of placement algorithms is presented in Section 3.3.
- **Clock tree synthesis (CTS)**. Performs the topology generation, buffering and routing of the clock network. The CTS usually tries to minimize the clock network power and latency given a skew target.

- **Routing**. Traces the signal paths using metal wires and vias.
- **Timing closure**. Optimizes the performance of the circuit using techniques such as sizing, cloning, buffering, $V_t$ swapping, incremental-detailed placement and routing. Such optimizations are traditionally called after every step of the physical design.

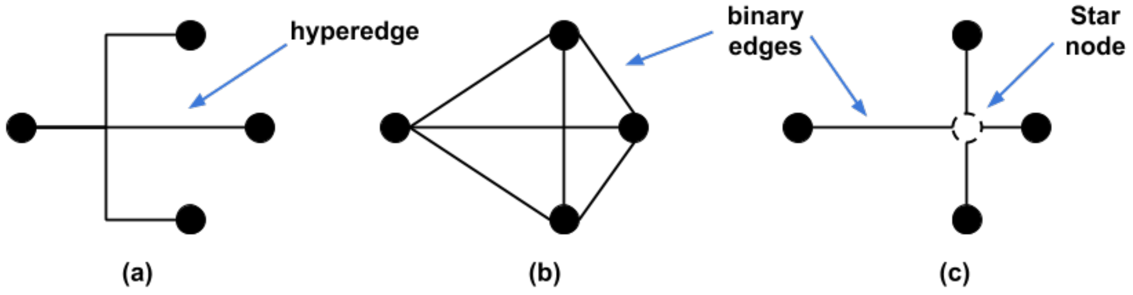
**Verification** is the step that assesses whether the final layout meets the design specification and respects the technology rules. The verification is performed using extraction and analysis tools. The *formal verification* tool checks if the netlist from a given stage of the flow ("target") has the same functionality as a known "golden" netlist (e.g., pre-route netlist (target) vs. RTL (golden).) The *design rule check* (DRC) verifies geometric constraints such as minimum metal area and distance. The *layout versus schematic* (LVS) compares the functionality of the layout with the netlist generated in the logic design. The *electrical rule checking* (ERC) verifies whether the design respects fanout, slew and capacitance constraints and whether the power and ground distribution is well formed. Extraction tools derive the electric parameters of the design. These parameters are used by timing and power analysis tools to assess if the design meets the timing and power budgets. If the design does not pass in any of these verifications, the designers need to do incremental changes. These changes are often performed manually by designers as even small perturbations in the layout may create new violations.

Once the layout passes the verifications it is sent to the foundry for **fabrication**. In the foundry, the design is patterned using a lithographic process upon silicon wafers. Finally, the chips in the silicon wafer are diced and placed in **packages** and **tested**. The chips that pass the testing are ready to be commercialized.

## 2.2 Hyperedge Decomposition

The circuit netlist is a hypergraph $H(V, X)$ in which the nodes of the netlist (instances, macro blocks, ports) compose the set of vertices $V$ and the signal nets compose the set of hyperedges $X$. However, many algorithms in the literature only work with *graphs*. Therefore, some models have been proposed to *decompose* the set of hyperedges into an equivalent set of edges $E$ to represent the netlist using a graph $G(V, E)$. The clique and star models depicted in Figure 2.2, are examples of decomposition models commonly used in placement algorithms (VISWANATHAN; CHU, 2004). In the clique model every hyperedge is replaced by a complete graph (i.e., all vertices are connected among themselves by a binary connection) and the star model introduces an additional vertex connected to every other vertex by a binary connection.

Figure 2.2: A hyperedge (a) decomposed using the clique (b) and the star (c) models.



Source: from author.

## 2.3 Partitioning

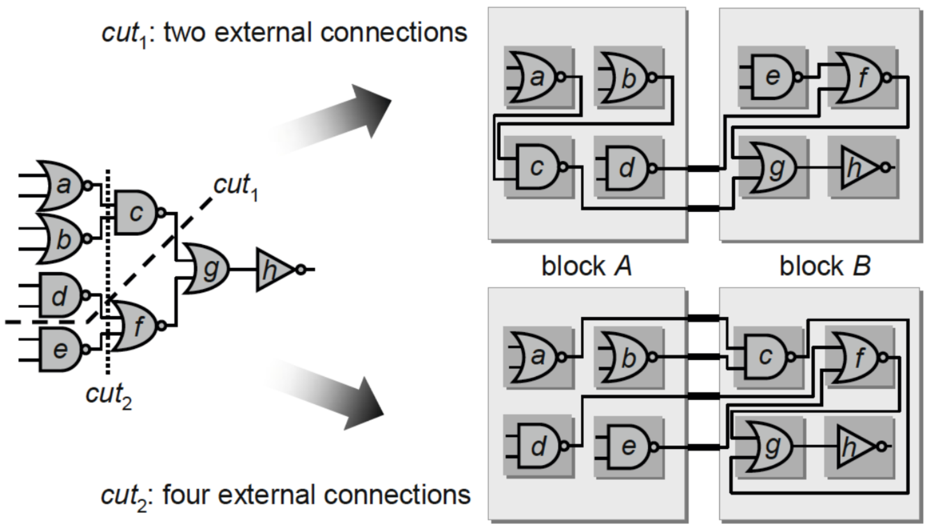
In partitioning, the graph vertices are divided into groups, called *partitions*. The most common goal is to minimize the number of connections between partitions. Partitioning algorithms belong to a category called *min-cut partitioning*, where *cut* is the sum of the edge weights crossing between partitions. Figure 2.3 presents a netlist graph and two possible cuts. The solution of $cut_1$ presents a better solution since it has only 2 connections between partitions while $cut_2$ has 4 connections between partitions.

## 2.4 Clustering and Community Detection

Clustering, or cluster analysis, is a field that aims to divide a set of objects into homogeneous groups, called clusters (WIERZCHON; KLOPOTEK, 2018). Two objects belonging to the same cluster should have more *similarity* than objects belonging to different clusters (WIERZCHON; KLOPOTEK, 2018). Clustering has been applied to a wide range of applications in VLSI, like placement (ALPERT et al., 2005), CTS (HAN; KAHNG; LI, in press) and flop tray design (KAHNG; LI; WANG, 2016).

Community detection is a type of clustering in which the objects being clustered are vertices of a graph usually originated from social or biological networks (NEWMAN; GIRVAN, 2004). The similarity is given by the connections between vertices – the number of edges inside the clusters, which are referred to as *communities*, should be higher

Figure 2.3: Alternative 2-way partitioning solutions of a circuit netlist (left). The first solution, obtained with $cut_1$ (upper), produces 2 partitions with 2 edges between them. The second solution, obtained with $cut_2$ (lower), produces 2 solutions with 4 edges between them.


$cut_1$: two external connections
$cut_1$
$cut_2$
block $A$
block $B$
$cut_2$: four external connections
a
b
c
d
e
f
g
h

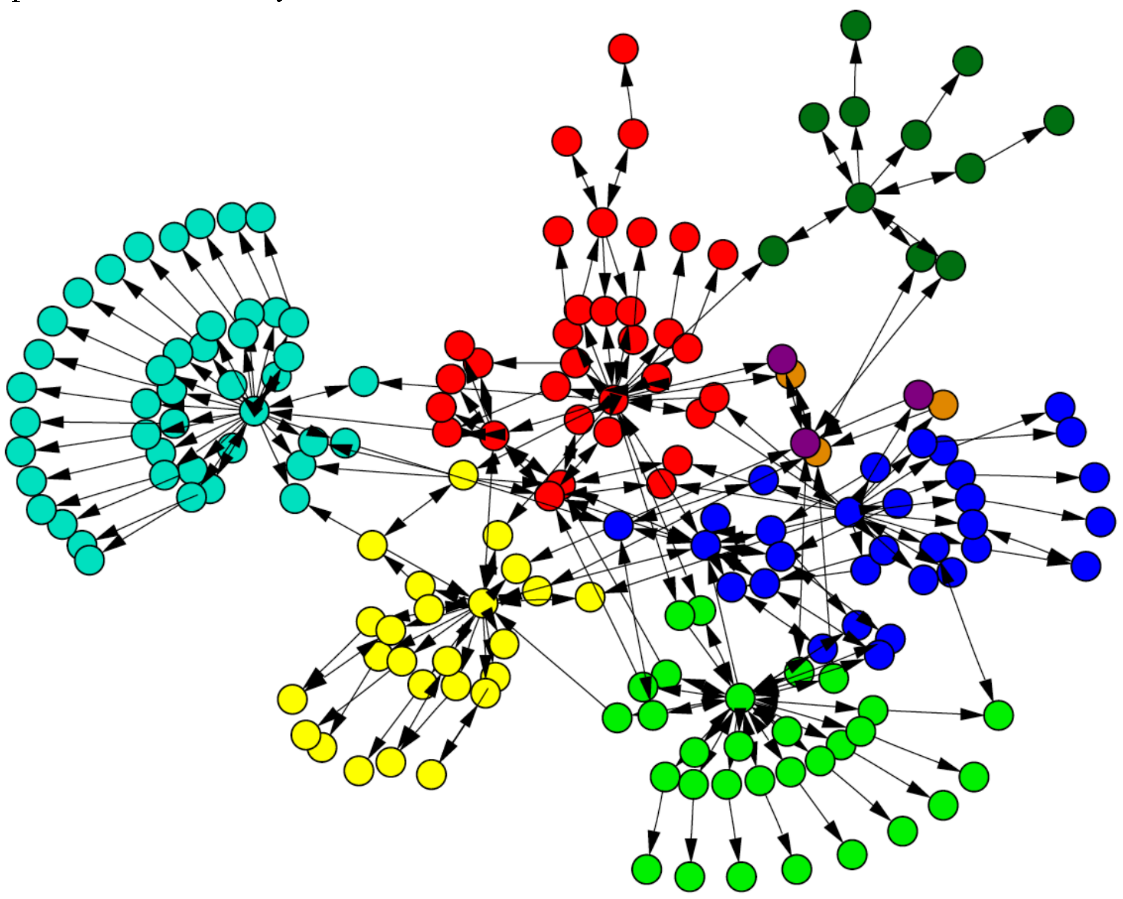
Source: Kahng et al. (2011).

than the number of edges spamming multiple clusters. Figure 2.4 depicts the outcome of the community detection algorithm of Newman and Girvan for a set of webpages (NEWMAN; GIRVAN, 2004). The color of each vertex denote the community the vertex has been assigned to and the edges represent hyperlinks between pages.

## 2.5 Half-Perimeter Wirelength

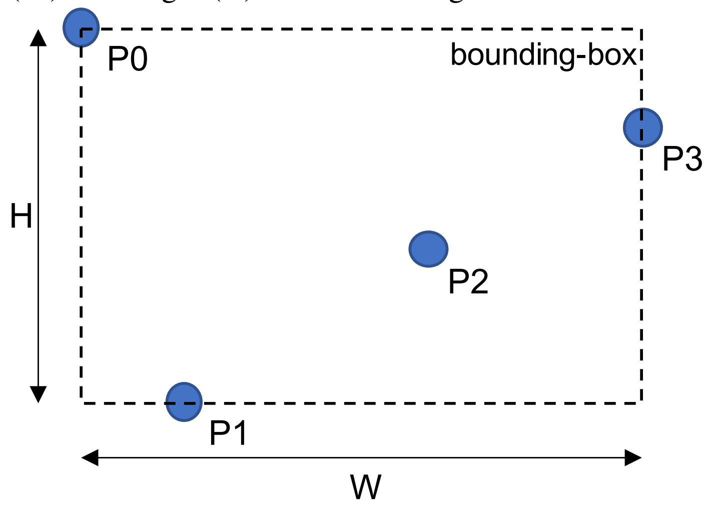
In the VLSI CAD tools, many optimization engines need to estimate wirelength efficiently. Many placement algorithms (and research) adopt the *half-perimeter wirelength* (HPWL) as an optimization goal and for solution quality measurement. HPWL is computed as follows. Consider the 4-pin net in Figure 2.5. The HPWL consists of the half-perimeter (summation of width and height) of the net pins' bounding-box. The half-perimeter can be computed in linear time, by just traversing all pins of the net once. Additionally, HPWL matches the minimum routed wirelength for nets with up to 3 pins.

Figure 2.4: Community structure for a set of webpages based on hyperlinks. Each color represents a community.



Source: Newman and Girvan (2004).

Figure 2.5: The bounding-box of 4-pin net. The half-perimeter wirelength is the summation of the width (W) and height (H) of the bounding-box.



Source: from author.

# 3 LITERATURE REVIEW

The scope of our present work spans three topics of interest: VLSI partitioning tools, community detection and placement. This Chapter presents a brief review of the literature and state-of-the-art on these topics.

## 3.1 VLSI Netlist Partitioning

Partitioning is extensively studied in VLSI research for many applications. For instance, one common strategy to cope with modern ICs complexity is to *decompose* the system into smaller logic and physical portions which can be implemented in parallel using partitioning. After implementation, these portions can be reassembled in a single die. Other applications include stacked voltage domain designs (BLUTMAN et al., 2017) and speed up of placement algorithms (ALPERT et al., 2005). In this Section, the literature on VLSI partitioning algorithms is studied following the taxonomy proposed by Alpert and Kahng (1995). Table 3.1 presents a summary of the partitioning algorithms studied in this work.

Table 3.1: Summary of the partitioning algorithms studied in this work and their category according to Alpert and Kahng (1995).

| Reference | Category |
|---|---|
| Kernighan-Lin algorithm (KERNIGHAN; LIN, 1970) | Move-based approaches |
| Fiduccia-Mattheyses algorithm (FIDUCCIA; MATTHEYSES, 1982) | |
| hMetis (GEORGE; VIPIN, 1998) | |
| MLPart (CALDWELL; KAHNG; MARKOV, 2000) | |
| Barnes (1981) | Geometric representation-based approaches |
| Yang and Wong (1994) | Combinatorial formulations |
| Blutman et al. (2017) | |
| Rajaraman and Wong (1995) | Clustering approaches |
| Alpert et al. (2005) | |

Source: from author.

### 3.1.1 Move-Based Approaches

Move-based approaches start from an initial arbitrary solution and try to improve it by iteratively swapping a single vertex from one partition to another or by swapping pairs of vertices belonging to different partitions. The core of these approaches is frequently inspired by the Kernighan-Lin (KERNIGHAN; LIN, 1970) and Fiduccia-Mattheyses algorithms (FIDUCCIA; MATTHEYSES, 1982).

The Kernighan-Lin algorithm has been introduced in 1970 to perform 2-way partitioning. The algorithm computes the cut improvement of swapping random pairs of vertices and stores the pair with best cut improvement. After being stored, the vertices of the pair are marked as *fixed*, i.e. cannot be selected again as candidates for swap. This process is repeated until all nodes become fixed. The algorithm then effectively swaps only the set of pairs that present the largest values of cut improvement. This is called a *pass*. After one pass, all vertices are unmarked as fixed. The algorithm stops when the gain seen after the pass is less than a threshold.

In 1982, Fiduccia and Mattheyses propose an extension to the Kernighan-Lin algorithm, called Fiduccia-Mattheyses, with multiple improvements: (i) the algorithm performs single-vertex swaps, allowing unbalanced unbalancing between partitions; (ii) support for hypergraphs; (iii) support for area constraints in partitions and (iv) faster selection of candidates to swap. While the overall flow is similar, the Fiduccia-Mattheyses cost function is modified. The gain of swapping a node from one partition to another is measured as:

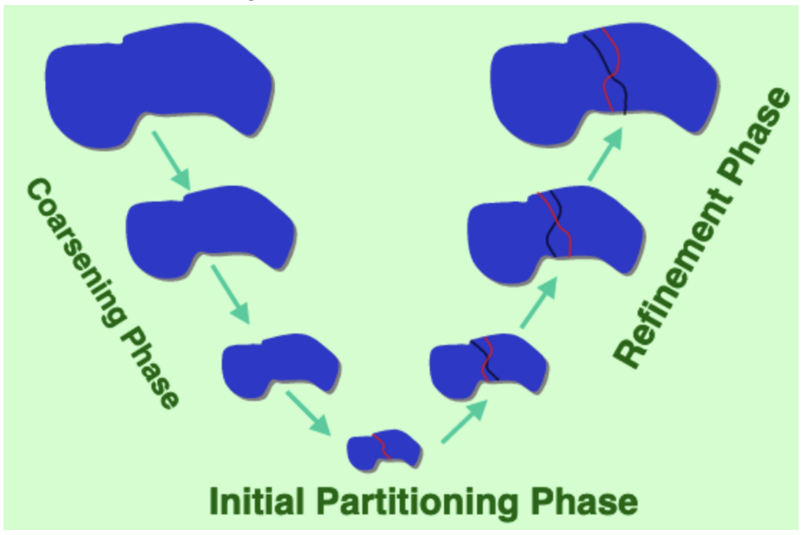
$$\Delta G = FS(v) - TE(v) \tag{3.1}$$

where $v$ is the vertex; $FS(v)$ is the number of nets or (hyper) edges connected to $v$ but not connected to any other vertex in the same partition, and $TE(v)$ is the number of nets or (hyper) edges connected to $v$ and not connected to any vertex from the other partition. Next, we discuss two Fiduccia-Mattheyses-based tools hMetis (KARYPIS; KUMAR, 1999) and MLPart (CALDWELL; KAHNG; MARKOV, 2000), widely used in academic and commercial flows.

#### *3.1.1.1 hMetis*

Karypis et al. propose a *multilevel* hypergraph partitioning algorithm called hMetis (GEORGE; VIPIN, 1998; KARYPIS; KUMAR, 1999). The flow of hMetis is composed of three phases: (i) coarsening phase, (ii) initial partitioning phase and (iii) refinement phase. In the *coarsening phase*, the size of the hypergraph is iteratively reduced by contracting vertices and edges. Karypis et al. study contracting schemes, called matching schemes and describe them in Karypis et al. (1997). Once the hypergraph is sufficiently coarse, the *initial partitioning* phase builds $N$ random 2-way partitions and uses the Fiduccia-Mattheyses to refine each solution. In the end, the solution with the best min-cut among all is chosen. In the *refinement phase*, the vertices and edges are iteratively uncontracted to obtain the original hypergraph. At each level, refinement algorithms are applied to improve the solution obtained in the initial partitioning phase. The hMetis flow is depicted in Figure 3.1.

Figure 3.1: The hMetis flow.



Source: George and Vipin (1998).

#### *3.1.1.2 MLPart*

Caldwell et al. propose MLPart, a multilevel partitioning tool similar to hMetis but with improvements in the 2-way partitioning and in the coarsening scheme (CALDWELL; KAHNG; MARKOV, 2000). The Fiduccia-Mattheyses implementation of MLPart starts by putting all vertices in a single partition, a strategy called VILE (“very illegal”) initial solution. The acceptance criterion for legal moves is relaxed, meaning that movements are accepted if they do not increase the balancing constraints. Their implementation and the gain of a movement is randomized in the first iterations. Caldwell et al. also implement a *LIFO Fiduccia-Mattheyses algorithm* (HAGEN; J.-H.; KAHNG, 1995) along with a "wiggling" strategy. For its multilevel partitioning, MLPart employs the strategy of edge coarsening from Alpert, Huang and Kahng (1997) and Karypis et al. (1997). Their edge coarsening scheme differs from previous works because it updates the graph continuously while clustering is performed. MLPart also adds balancing constraints when performing contraction of edges and vertices.

### 3.1.2 Geometric Representation-Based Approaches

This class is composed of partitioning algorithms that rely on the geometric embeddings of the netlist to achieve better partitioning results. Some algorithms represent the geometric embeddings with an adjacency matrix $C$, where each element $C_{ij}$ is the sum of the edge weights connecting vertices i and j. Other methods use the Laplacian matrix $L = C - M$, where $D$ is a $N \times N$ diagonal matrix where $N$ is the number of vertices in the netlist graph and each element $D_{ii}$ is the degree of vertex $i$. Hall et al. show that the second eigenvector ($\nu_2$) of the Laplacian matrix represents the 1D placement of the vertices with minimum squared wirelength (HALL, 1970). The graph can be divided into two partitions by sorting the vertices according to the entries of $\nu_2$ and then assigning the first half of the vertices to one partition and the second half to the other. Later, Barnes (1981) and Alpert, Kahng and Yao (1999) extend this strategy to perform $k$-way partitioning.

### 3.1.3 Combinatorial Formulations

Combinatorial formulations encompass partitioning methods based on formulations that can capture complex objective functions and constraints such as *network flow*. Yang and Wong (1994) apply network flow to perform 2-way min-cut partitioning. They propose a methodology to represent the netlist as a flow network and show how to perform balanced 2-way partitioning using a max-flow min-cut technique. More recently, Blutman et al. (2017) show how to extend this formulation to comprehend timing and layout information to perform partitioning on stacked voltage domain designs.

### 3.1.4 Clustering Approaches

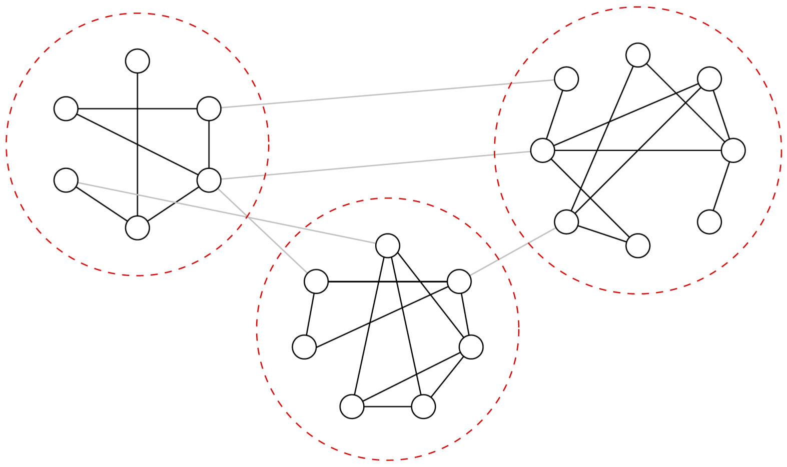
Clustering approaches are often taxonomized as being either bottom-up or top-down. Bottom-up methods start with each module being an individual cluster, with clusters being iteratively merged until a given condition is satisfied. Top-down methods start with a single cluster and iteratively split clusters into two or more (smaller) clusters. For instance, Rajaraman and Wong (1995) propose a bottom-up polynomial time algorithm for clustering networks aiming to minimize delay and subject to capacity constraints. Alternatively, Alpert et al. (2005) propose a semi-persistent clustering technique to speed up placement. Li, Behjat and Kennings (2007) propose a bottom-up clustering algorithm based on a score function that aims to reduce the number of nets in the clustered netlist and penalize large clusters. Yan, Chu and Mak (2010) use clustering to reduce the netlist size and speed up placement. A “safe condition” is devised and guarantees that clustering will not degrade wirelength. Rakai et al. (2012) devise a bottom-up clustering algorithm that relies on wirelength prediction of nets.

## 3.2 Community Detection

This Section presents a study on *community detection* methods. These methods are closely related to graph partitioning, as the ones covered in Section 3.1 and hierarchical clustering studied in sociology (NEWMAN; GIRVAN, 2004). *Communities* are defined as the division of the vertices of the graph into groups within the connections are denser and between and between which the connections are sparser (NEWMAN; GIRVAN, 2004). Figure 3.2 depicts a graph divided into 3 communities, highlighted by dashed red lines. Community detection differs from graph partitioning as follows.

- Community detection methods do not impose a target number of communities, in contrast with partitioning methods for which users define the target number of partitions.
- The communities do not have to be balanced in terms of area, number of edges or vertices.
- Community detection methods do not minimize cut, since it is natural for larger communities to have higher numbers of edges connected to them.

Figure 3.2: Example of communities in a graph. Bold black lines are edges connecting vertices that belong to the same community; Gray lines are edges connecting vertices that belong to different communities and dashed red lines outline the communities.



Source: Newman and Girvan (2004).

Community detection methods may be divided into three categories: *divisive methods* find communities by iteratively removing edges from the graph (GIRVAN; NEWMAN, 2002; NEWMAN; GIRVAN, 2004; RADICCHI et al., 2004); *agglomerative methods* iteratively merge vertices and communities (PONS; LATAPY, 2006; BLONDEL et al., 2008) and *optimization methods* maximize an objective function (CLAUSET; NEW-

MAN; MOORE, 2004; WU; HUBERMAN, 2004; NEWMAN, 2006) using heuristic methods (e.g. *simulated annealing (KIRKPATRICK; GELATT; VECCHI, 1983)*).

The remaining of this Section studies the evolution of the so-called *modularity-driven community detection* methods. Table 3.2 summarizes the method to be discussed from Sections 3.2.1 through 3.2.7. Section 3.2.1 starts showing how modularity has been first introduced by Newman and Girvan to evaluate the outcome of community detection methods. Then, Sections 3.2.2 through 3.2.4 show how modularity has become an objective function and how to effectively optimize it. Section 3.2.5 discusses the fast and effective Louvain algorithm, that will be used later in this work. Sections 3.2.6 and 3.2.7 discuss the limitations of modularity-driven techniques and present the alternative metrics *structural similarity* and *similarity-based modularity*. Section 3.2.8 concludes with a brief review of modularity-based community detection on hypergraphs.

Table 3.2: Summary of the community detection methods to be discussed in this Section.

| Reference | Objective function | Category |
|---|---|---|
| Newman and Girvan, 2004 | Edge betweenness | Divisive |
| Newman, 2004 | Modularity | Agglomerative |
| Clauset et al., 2004 | Modularity | Agglomerative |
| Wakita and Tsusumi, 2007 | Modularity + consolidation ratio | Agglomerative |
| Blondel et al., 2008 | Modularity | Agglomerative |
| Xu et al., 2007 | Structural similarity | Agglomerative |
| Feng et al., 2007 | Similarity-based modularity | Agglomerative |

Source: from author.

## 3.2.1 Finding and Evaluating Community Structure in Networks (NEWMAN; GIRVAN, 2004)

Newman and Girvan propose three *divisive methods* for community detection (NEWMAN; GIRVAN, 2004). In each iteration, the methods find the pair of connected vertices with least *similarity* to remove. The process is repeated iteratively until some stop condition is met. The end goal of this process is to divide the graph into smaller pieces. Each piece is an output community. Newman and Girvan propose three heuristics to quantify a pair of vertices similarity:

- **Shortest-path betweenness**. Compute the shortest path between any pair of nodes and find how many paths run along each edge.

- **Random-walk betweenness**. For each edge, compute the expected number of times a random walk between every two pairs of nodes will run along the current edge.
- **Current-flow betweenness**. Models every edge of the graph as one resistance with a constant value. Each pair of nodes is considered a source and sink of current. The heuristic computes the current in each edge for every pair of nodes. The values of the current for each edge are summed up.

Newman and Girvan compare their heuristics using artificial and real-world graphs whose ground-truth communities are known. However, they highlight that in most real-world problems, the ground-truth communities are not known. To tackle this problem, the authors propose a metric, called *modularity*, aiming to evaluate the quality of the division of a graph into communities. Modularity is defined as:

$$Q = \frac{1}{2m}\sum_{i,j}\Big[A_{ij} - \frac{k_i k_j}{2m}\Big]\delta(C(i), C(j)) \tag{3.2}$$

where $Q$ is the modularity value; $A_{ij}$ is the sum of the edge weights between communities $i$ and $j$; $k_i$ is the sum of the edges weights connected to $i$; $m$ is computed as $m = \frac{1}{2}\sum_{ij} A_{ij}$; $C(i)$ is the community of vertex $i$ and $\delta(C(i), C(j))$ assumes value 1 if $C(i)$ is equal to $C(j)$ and 0 otherwise.

The correlation between modularity and the ground-truth communities for a given graph has been assessed using artificial-generated test cases. Newman and Girvan have noticed peaks of modularity (E.g.: $Q > 0.7$) for solutions whose communities have correlated well with the ground-truth.

### 3.2.2 Fast Algorithm for Detecting Community Structure in Networks (NEWMAN, 2004)

Girman and Newman show that modularity is a good metric to assess the outcome of a community detection algorithm (NEWMAN; GIRVAN, 2004). Following this assumption, 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?"* (NEWMAN, 2004). Of course, it is infeasible to explore all the solution space for graphs with thousands or millions of vertices, but approximation algorithms could be applied.

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 results in the best increase in modularity. Since it is impossible to increase the value of modularity by merging pairs that do not have an edge between them, the algorithm only considers pairs of neighboring communities. The modularity delta of joining two communities is computed as

$$\Delta Q = \frac{1}{2m} \sum_{i,j} \left[ A_{ij} + A_{ji} - 2\frac{k_i k_j}{2m} \right] \tag{3.3}$$

where $\Delta Q$ is the variation of modularity value; $A_{ij}$ is the sum of the edge weights between communities $i$ and $j$; $k_i$ is the sum of the edges weights connected to to $i$ and $m$ is computed as $m = \frac{1}{2} \sum_{ij} A_{ij}$.

Each pass of this algorithm has a complexity of $O(|E| + |V|)$, where $|E|$ is the number of edges in the graph and $|V|$ is the number of vertices. There is a total of $|V| - 1$ passes to construct the dendrogram (EVERITT; SKRONDAL, 2002) of the graph. The entire algorithm runs in $O((|E|+|V|)|V|)$ and $O(|V|^2)$ in sparse graphs. Figure 3.3 represents the outcome of Newman's algorithm for the "Karate club" benchmark (ZACHARY, 1977) as a dendrogram. The dendrogram is a graphical representation of hierarchical clustering. In Figure 3.3, the shapes at the bottom level represent the known ground-truth. Girvan and Newman's only assigns vertex 10 to a "wrong" cluster.

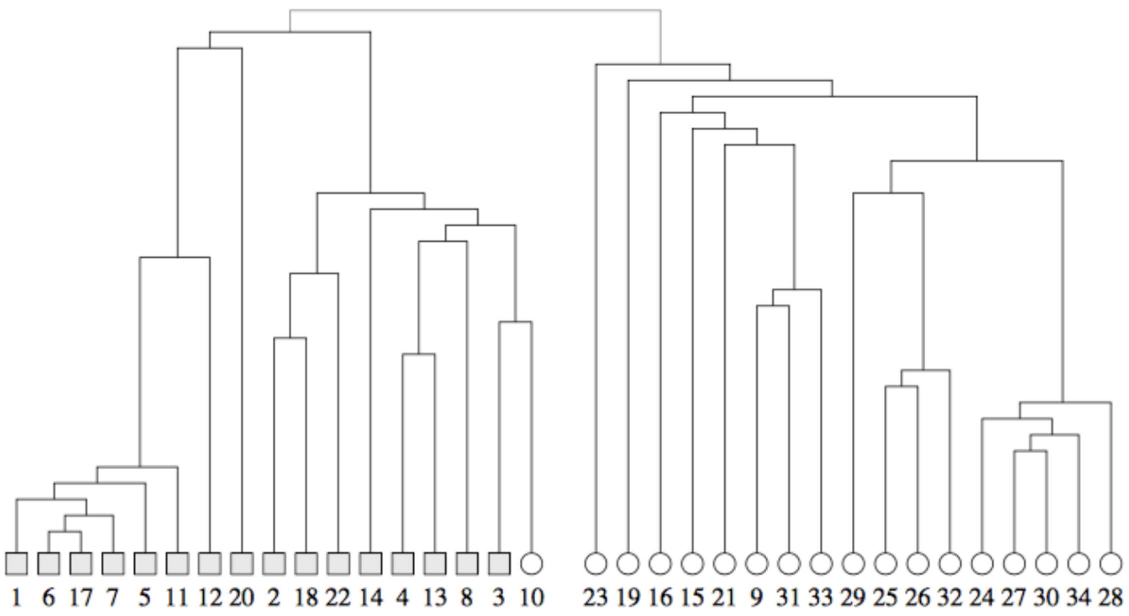
### 3.2.3 Finding Community Structure in Very Large Networks (CLAUSET; NEWMAN; MOORE, 2004)

Clauset et al. propose optimized data structures to speed up Newman's algorithm so it can be applied in larger and sparser graphs (CLAUSET; NEWMAN; MOORE, 2004). While Newman uses an adjacency matrix to represent the graph, Clauset et al. propose the use of a $\Delta Q$ matrix that stores the change in modularity by joining every pair of vertices $i$ and $j$. Since most real-life graphs are sparse, they propose to apply data structures optimized for sparse matrices. In addition, they propose to use fast data structures to keep track of the highest deltas of modularity.

In summary, the authors propose three data structures:

- A sparse matrix $\Delta Q_{ij}$ containing the delta of modularity for every pair of vertices.

Figure 3.3: Dendrogram of the clustering found by Newman for the "Karate club" benchmark. Numbers are the vertices indices and the shapes represent the ground-truth communities.



Source: Newman (2004).

The authors propose to implement each row of the matrix as a balanced binary tree, so elements can be efficiently inserted and found.

- A max heap containing the largest $\Delta Q$ for every row of the matrix along with labels identifying the indices of the vertices to join.
- A vector containing the values of the normalized degree of the vertices.

The algorithm, called CNM, may be outlined in three steps:

1. Initialize the values of $\Delta Q$, the vector and the max heap.
2. Find the largest $\Delta Q$, merge communities and update the matrix, the max heap and the vector.
3. Repeat (2) until only one community remains.

### 3.2.4 Finding Community Structure in Mega-Scale Social Networks (WAKITA; TSURUMI, 2007)

According to Wakita and Tsurumi (WAKITA; TSURUMI, 2007), the CNM algorithm only scales well for graphs with up to 500K vertices. They diagnose the cause of the inefficiency being the unbalanced nature of the process of merging vertices and communities. Their experiments show only a small number of communities growing very fast. Wakita and Tsurumi hence propose an extension to the CNM algorithm whose goal is to balance the size of communities during the merging process. In their extension, the candidate pairs of communities to be merged are ranked by the weighted delta in modularity, instead of delta of modularity solely. They propose weighting schemes called *consolidation ratio* that are expressed as:

$$consolidation\_ratio(c_i, c_j) = \min(h(c_i)/h(c_j), h(c_j)/h(c_i)) \tag{3.4}$$

where $c_i$ and $c_j$ are the communities and $h$ is a weighting scheme function. Wakita and Tsurumi propose three weighting schemes:

- **HE**. The value of $h$ for a given community is equal to the number of edges connecting the community to its neighbors;
- **HE'**. First, a set of pairs that produce the best delta modularity for each community is computed. The best pairs are then ranked as in HE.
- **HN**. The value of $h$ for a given community is equal to the number of vertices from the original graph that belongs to the given community.

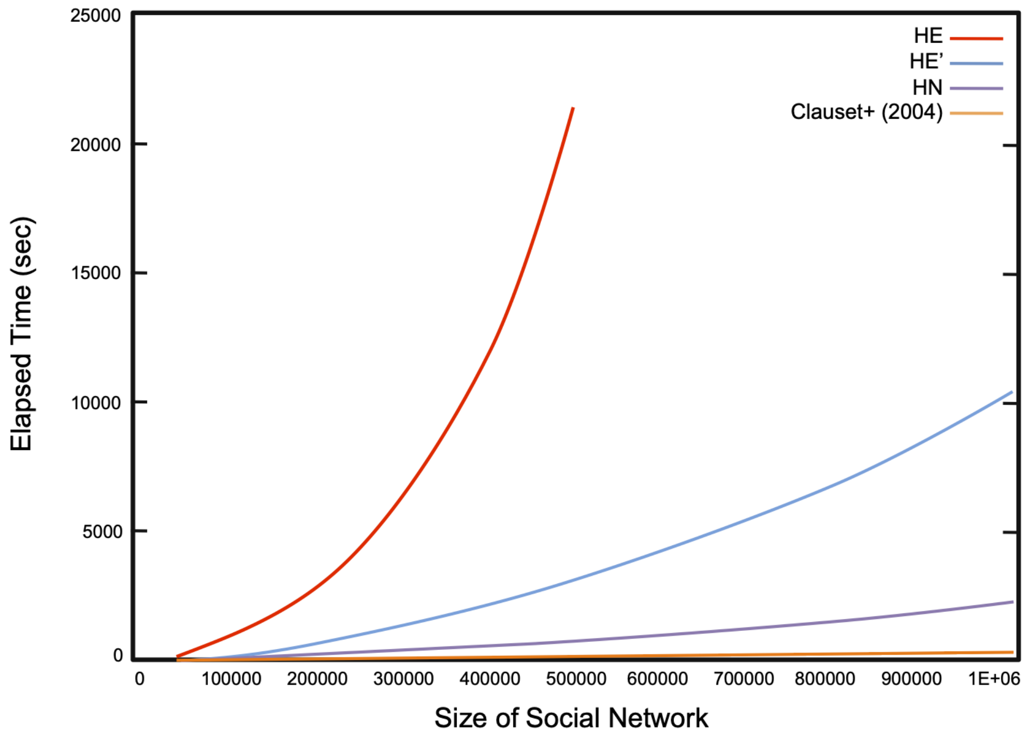
Figure 3.4 shows how the vanilla CNM algorithm and Wakita and Tsurumi extension scale for graphs with up to 1M vertices. Figure 3.5 shows the values of modularity vs. iterations ("progress of analysis") in a graph with 500K vertices.[4] The HE' heuristic is able to outperform CNM in terms of performance and modularity while HE and HN outperform CNM in performance but perform 21-28% worse than vanilla CNM.

## 3.2.5 Fast Unfolding of Communities in Large Networks (BLONDEL et al., 2008)

Blondel et al. propose an agglomerative modularity-driven method, called *Louvain algorithm* (BLONDEL et al., 2008). In the beginning, each vertex of the graph is considered a community. The *first phase* of the Louvain algorithm iterates through all vertices of the graph. For each vertex, Louvain computes the cost of moving the given vertex from its current community to the neighboring communities. The vertex is moved to the neighboring community that presents the higher cost. The vertex remains in the original community if the maximum cost is not positive. The cost is given by the modu-

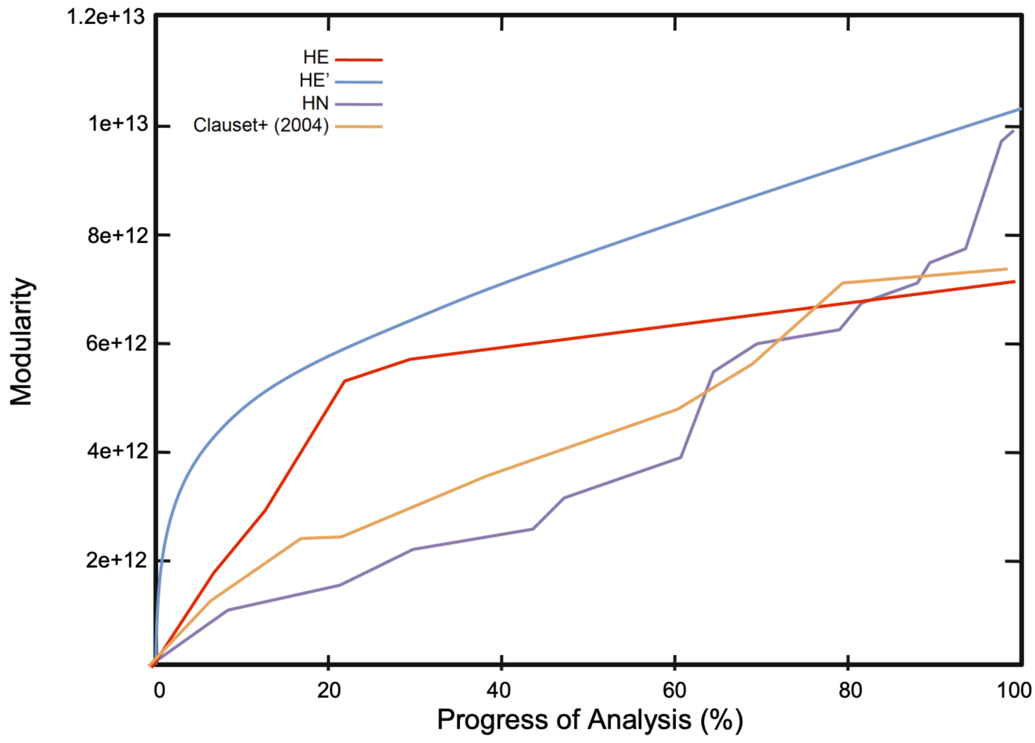
[4]In Figure 3.5, Wakita and Tsurumi multiply the value of modularity by the squared number of edges in the graph.

Figure 3.4: Runtime comparison between CNM and Wakita and Tsurumi.



Source: adapted from Newman (2004).

Figure 3.5: Values of modularity for CNM and Wakita and Tsurumi. For the latter, different weighting schemes are shown.



Source: adapted from Newman (2004).

larity delta, which is efficiently computed using equation 3.3 The first phase is repeated until modularity stops improving.[5]

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. Edges between nodes belonging to the same community are summed up and create a self-loop in the new graph. After the second phase is completed, the first phase is performed again, upon the new graph, and so forth. Each iteration between the first and second phases is called *a pass*. The algorithm stops when there is no gain in modularity after a pass.

Figure 3.6 depicts the steps of the Louvain algorithm. The input of the example is a graph with 16 vertices and 53 edges (all edges have weight $= 1$). In the first phase, the Louvain algorithm finds 4 communities, depicted in red, green, blue and gray. The second phase builds a new graph with 4 nodes corresponding to the communities found in the first phase and, 9 edges. The output of the second pass is a graph with 2 vertices and 3 edges.

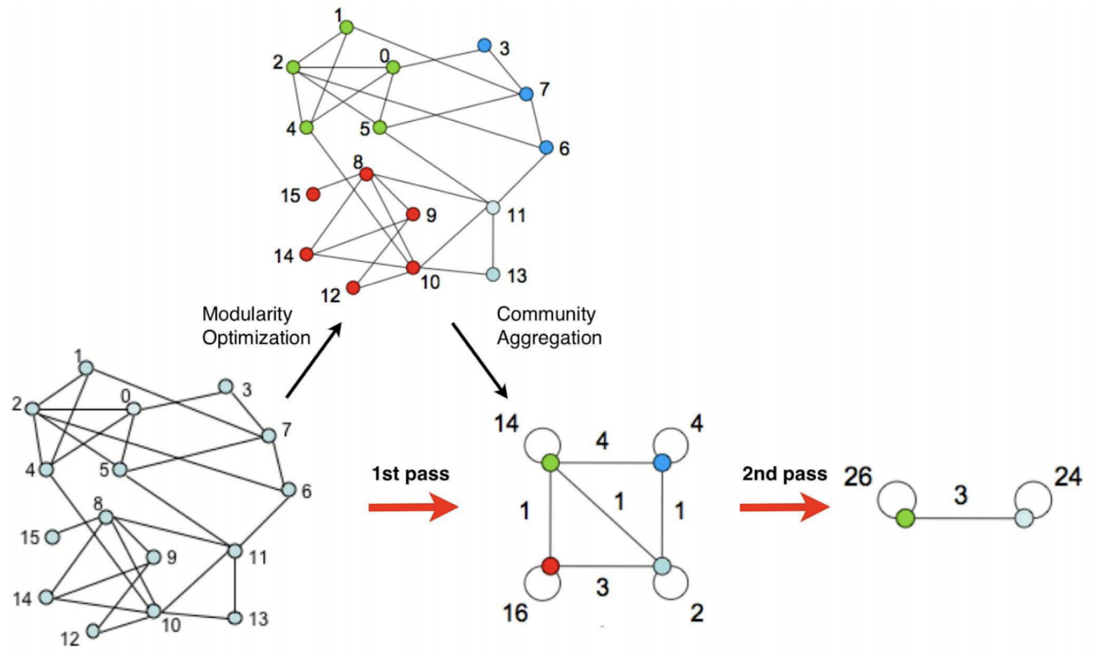
Unlike previous approaches, Louvain is fast and scalable. Experiments performed by Blondel et al. show linear runtime complexity with respect to the number of vertices in sparse graphs. For instance, Louvain is able to perform community detection in a graph with 118M vertices in 152 minutes. Since each pass reduces the size of the graph, most of the runtime is spent on the first iteration.

### 3.2.6 SCAN: A Structural Clustering Algorithm for Network (XU et al., 2007)

Modularity-driven methods do not detect and isolate two very common structures found in graphs: *hubs* and *outliers*. Consider the graph depicted in Figure 3.7. Vertices 0 through 5 and 7 through 12 clearly form two clusters. However, vertex 6 is equally connected with three other vertices from both clusters and, therefore, is considered a hub; Vertex 13 is connected only with vertex 9 and does not clearly make part of the any cluster, therefore it is called an outlier.

[5]One might note that the order in which nodes are iterated changes the output of the algorithm. However, Blondel et al. have performed experiments showing that the variation in modularity is not significant.

Figure 3.6: Outline of Louvain algorithm. The algorithm has two phases: *modularity optimization* and *community aggregation*. Each iteration of the two phases is called a *pass*.



Source: Blondel et al. (2008).

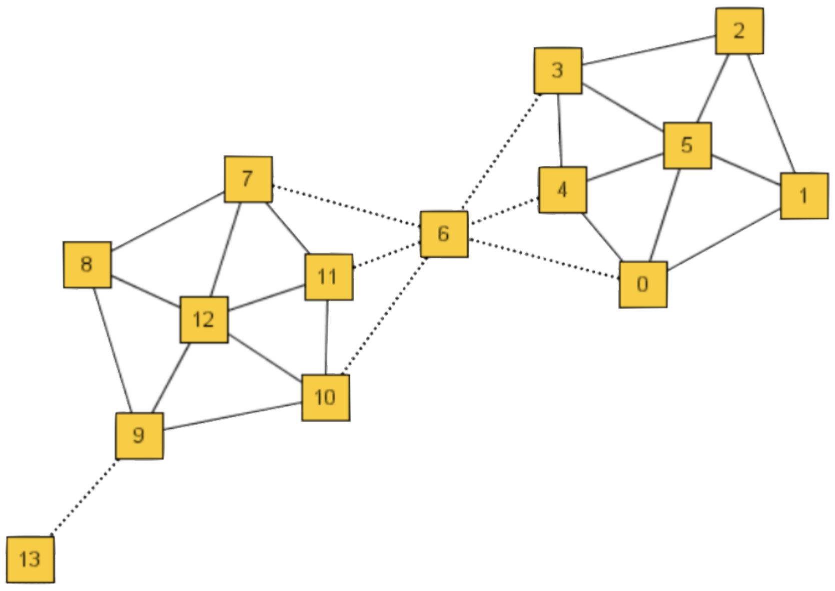
Xu et al. devise a metric, called *structural similarity*, that measures how strongly connected two vertices are (XU et al., 2007). The structural similarity metric is a real number ranging from 0 to 1, defined as:

$$\chi(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{\sqrt{|\Gamma(u)||\Gamma(v)|}} \tag{3.5}$$

where $\chi(u, v)$ is the value of structural similarity of vertices $u$ and $v$, and $\Gamma(u)$ is a set of vertices comprising $u$ and its topological neighbors. The more similar is the neighborhood of two adjacent vertices, higher is the value of their structural similarity and may be used to find clusters, hubs and outliers.

Xu et al. also propose an algorithm called SCAN, based on the traditional clustering algorithm DBSCAN. SCAN uses a heuristic to find vertices to be used as seeds for clusters and then apply structural similarity to expand the seeds and form clusters. The vertices not assigned to any clusters after the execution of SCAN are classified as hubs or outliers. The algorithm requires $O(|E|^2/|V|)$ runtime. Most of the runtime of the algorithm comes from the computation of structural similarity which takes $O(|E|/|V|)$ runtime.

Figure 3.7: A graph with two clusters, one hub and one outlier. The clusters are formed by vertices 0-5 and 7-12. Vertex 6 is a hub and vertex 13 is an outlier.



Source: Xu et al. (2007).

### 3.2.7 A Novel Similarity-Based Modularity Function for Graph Partitioning (FENG et al., 2007)

Inspired by the ideas of modularity and structural similarity, Feng et al. propose a new metric, called *similarity-based modularity* (FENG et al., 2007), defined as:

$$Q_s = \sum_{i=1}^{NC} \left( \frac{IS_i}{TS} - \frac{DS_i}{TS}^2 \right) \tag{3.6}$$

where $Q_s$ is the similarity-based modularity; $NC$ is the number of communities, $IS_i$ is the similarity of vertices within the community $i$, $DS_i$ is the structural similarity between vertices of the community $i$ and the remaining vertices of the graph and $TS$ is the total similarity between all pairs of vertices of the graph.

Feng et al. use a genetic algorithm to compare modularity and similarity-based modularity as alternative objective functions for community detection. Feng et al. find similarity-based modularity to be more accurate in artificial graphs. Several similarity-based modularity community detection algorithms have been proposed since then (SHIOKAWA; FUJIWARA; ONIZUKA, 2015; SHIOKAWA; ONIZUKA, 2017), but none scales as well as the Louvain Algorithm.

### 3.2.8 Modularity-Driven Clustering for Hypergraphs

In Section 3.2, 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 (HEUER; SCHLAG, 2017). 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 and Obermayer (2009) and Neubauer and Obermayer (2010) propose a modularity criterion and optimization method for $k$-partite $k$-uniform hypergraphs. Kumar et al. (2018) and Kumar et al. (2019) 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. (2019) adapt the modularity criterion for hypergraphs using the Chung-Lu model (CHUNG; LU, 2002). 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 (SZUFEL, 2020).

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 Chapter 4 using Szufel (2020). However, a design with 8K cells, which is much smaller than the netlists arising in our present work, takes an average of 25 minutes when we sweep the number of iterations of the algorithm from 500 to 10000 with step of 500. In contrast, Louvain can cluster a design with 1.4M instances in 7 minutes. On the small testcase *jpeg_encoder_14* with 44K instances, the scripts from (SZUFEL, 2020) crash due to stack overflow.

## 3.3 Placement

We now present an overview of the literature on instance placement. Placement, like partitioning, has been one of the most researched topics in EDA since the early 1970s. Placement tools aim to find a good position for each element of the netlist. Modern placement algorithms, especially in industry, model placement as a multiobjective problem. Traditional placement objectives include total wirelength, timing and congestion.

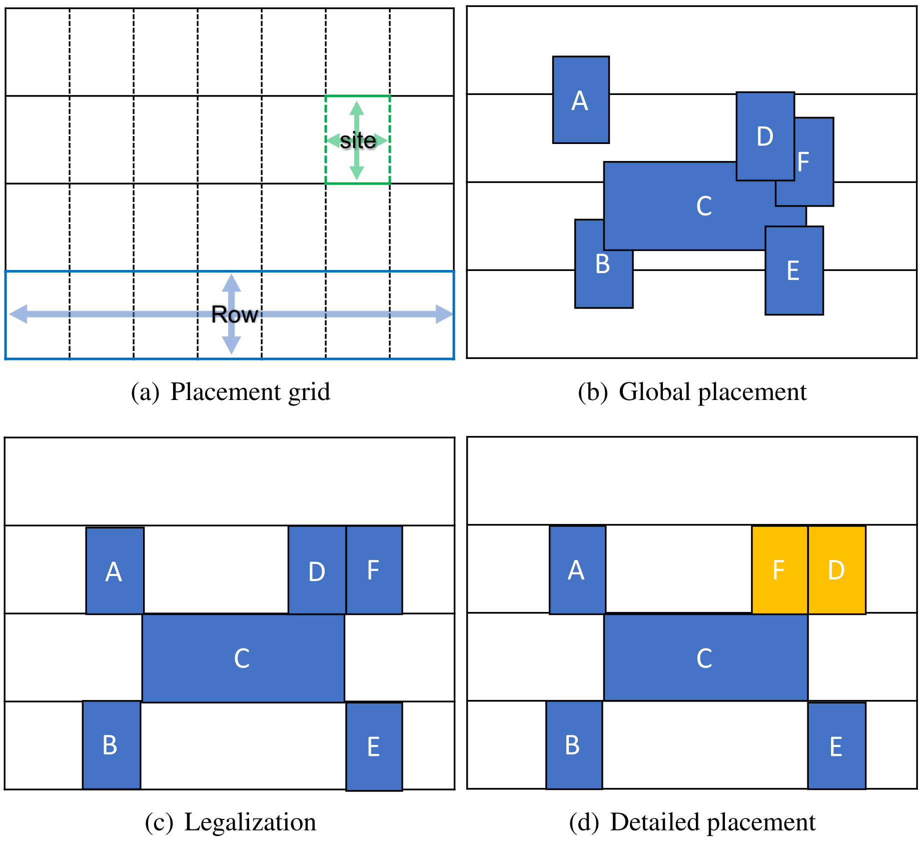
In cell-based design, placement tools have to assign instances to positions aligned to a *placement grid*. The placement grid is composed of *rows* and rows are divided into slices called *sites* (Figure 3.8(a)). Instances usually have one row-height and variable site-widths. Nevertheless, some standard cell libraries in modern technologies provide multi-height cells for complex functions (e.g., muxes, registers and latches) to achieve better intra-cell routing, save area and improve design for manufacturability (BAEK et al., 2008). When the placement tool has to place a netlist simultaneously composed of standard cells and *macro blocks*, the problem becomes much more complex and is called *mixed-size placement* (YAN; VISWANATHAN; CHU, 2009). Instance placement is a NP-hard combinatorial problem. Placement tools cope with the problem complexity by dividing placement into three steps: *global placement*, *legalization* and *detailed placement*.

**Global placement** algorithms aim to optimize wirelength (VISWANATHAN; CHU, 2004; KAHNG; WANG, 2006), timing (CHAN; CONG; RADKE, 2009; KAHNG; WANG, 2004) and routability (HSU et al., 2014; CHENG et al., 2019) while spreading the instances in the placement region.[6] In this step, the alignment to the placement grid constraint is relaxed to reduce the problem complexity. Figure 3.8(b) depicts a global placement solution. There is a broad literature on global placement and, therefore, we detail this literature separately in Section 3.3.1.

**Legalization** aligns instances to overlap-free sites in rows with less movement (displacement) as possible. Figure 3.8(c) shows the legalized solution of the global placement from Figure 3.8(b). Hill (2002) proposes a fast and greedy algorithm that traverses instances by ascending or descending $x$ coordinate and assigns each instance to the nearest available overlap-free on-grid position. In Hill (2002), instances are moved only once during legalization. Abacus (SPINDLER; SCHLICHTMANN; JOHANNES, 2008) works

[6]Academic global placement tools can outperform commercial tools on specific criteria such as wirelength and routing overflow but usually underperform commercial tools in highly constrained multiobjective optimization (MARKOV; HU; KIM, 2015).

Figure 3.8: (a) placement grid with a row outlined in blue lines and a site outlined in green; (b) global placement of a circuit with six instances; (b) legalization of the placement from (a) and (c) detailed placement where instances $D$ and $F$ are swapped.



(a) Placement grid (b) Global placement

(c) Legalization (d) Detailed placement

Source: from author.

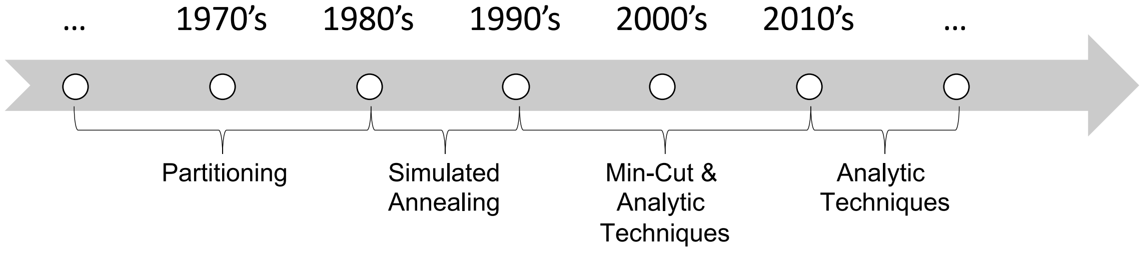
similarly to Hill (2002) but allows instances already legalized to move again. Whenever an instance is assigned to a grid position, a dynamic programming algorithm replaces the cells in the same row, aiming to minimize the total displacement. Jezz (PUGET et al., 2015) extends Abacus by providing an efficient cache system that stores available grid positions and supports placement obstacles. BonnPlace (BRENNER, 2013) proposes a network flow-based legalization algorithm that minimizes total and maximum displacement. Eh?Legalizer (DARAV et al., 2018) implements a network-based flow that effectively legalizes high-density areas by considering several candidate paths. Finally, Do, Woo and Kang (2019) propose a fence region and multi-height-aware legalization algorithm. Do, Woo and Kang (2019) perform legalization in three steps: *pre-legalization* legalize cells placed in invalid fence regions; *multi-deck standard cell legalization* assign cells to overlap-free on-grid locations and *quality refinement* applies simulated annealing to reduce the total displacement.

**Detailed placement** performs local movements aiming to improve placement quality or take into consideration advanced design rules. Flach et al. (2016) devise a set of drive strength-aware local movements to improve timing. Jung et al. (2018) apply the Bézier curve to smooth critical paths. Monteiro, Johann and Behjat (2019) implement a network flow to remove instances from high-density areas with minimal impact on timing. Heo et al. (2019a) present a dynamic programming formulation to maximize power staple insertion. Heo et al. (2019b) propose a detailed placement heuristic to minimize diffusion break effects.

### 3.3.1 Global Placement Tools

We adopt the survey of Markov, Hu and Kim (2015) as a basis for our studies on global placement. Figure 3.9 presents the evolution of global placement algorithms.[7] Partitioning approaches have been broadly employed in global placement in the early 1970s until the 1980s. Later, the stochastic algorithms based on the meta-heuristic simulated annealing have emerged as the most successful algorithms for global placement due to the quality of their results. Due to scalability issues, min-cut and analytic approaches overtake simulated annealing approaches between the 1990s and 2010s. Analytic techniques have been the most employed in modern global placement algorithms. In the remainder of this Section, we give an overview of works belonging to each category. Table 3.3 shows the global placement tools studied in our current work.

Figure 3.9: Evolution of global placement techniques.



Source: from author. Based on the studies of Markov, Hu and Kim (2015).

**Simulated-annealing based approaches** are best represented by the timing-driven placement tool TimberWolf (SWARTZ, 1993; SWARTZ; SECHEN, 1995). The algo-

[7]We note to the reader that Figure 3.9 estimates when each category/class of algorithm had most of the focus of attention in academia and industry. However, throughout this Section, we try to present the most recent publication of the placement tools.

Table 3.3: Global placement tools studied in our work and their taxonomy.

| Reference | Category |
|---|---|
| RAMP<br>(CHENG; KUH, 1984) | Partitioning approaches |
| PROUD<br>(TSAY; KUH; HSU, 1988) | |
| GORDIAN<br>(KLEINHANS et al., 1991) | |
| BonnPlace<br>(BRENNER; STRUZYNA; VYGEN, 2008) | |
| TimberWolf<br>(SWARTZ, 1993) | Simulated annealing based approaches |
| Dragon<br>(TAGHAVI; YANG; CHOI, 2005) | |
| CAPO<br>(CALDWELL; KAHNG; MARKOV, 1999) | Min-cut based approaches |
| FastPlace<br>(VISWANATHAN; CHU, 2004) | Analytic approaches |
| SimPL<br>(KIM; LEE; MARKOV, 2013) | |
| MAPLE<br>(KIM et al., 2012) | |
| Naylor, Donelly and Sha (2001) | |
| APlace<br>(KAHNG; WANG, 2006) | |
| NTUPlace<br>(CHEN et al., 2008) | |
| ePlace<br>(LU et al., 2015) | |
| RePlAce<br>(CHENG et al., 2019) | |

Source: from author.

rithm of Timberwolf optimizes timing by assigning higher weights to the $K$ worst timing-violating paths during the iterations of simulated annealing. Most of the late 1980s EDA flows have adopted the TimberWolf placement tool (MARKOV; HU; KIM, 2015). The commercial success of TimberWolf is so substantial that TimberWolf Systems, Inc. maintains the tool until today (TIMBERWOLF, 2014). However, VLSI designs are too big to perform simulated annealing upon the flat netlist. Simulated annealing-based tools combine other strategies to reduce the problem size. For example, TimberWolf performs clustering on the netlist. When the clustered netlist is sufficiently small, simulated annealing is applied upon a *a cluster netlist* to generate the initial solution. Then, TimberWolf starts flattening the netlist again between iterations of the simulated annealing.

Dragon (TAGHAVI; YANG; CHOI, 2005), on the other hand, uses hMetis to employ recursive bisection on the netlist and applies simulated annealing upon the intermediate partitions. A post-processing step performs local refinement on the flat netlist and removes overlaps.

**Partitioning and min-cut based approaches.** PROUD (TSAY; KUH; HSU, 1988) is an example of a *partitioning based* global placement algorithm. The main idea of PROUD is to place instances modeling a resistive network analogy solved using linear equations with the Gauss-Seidel method. After solving the linear equations, PROUD partitions the netlist and placement area into two parts using the center of mass of the instances' placement. After the partitioning, PROUD reruns the global placement on each partition individually. Many iterations of partitioning followed by global placement are applied until a stop condition is found. The algorithm proposed in PROUD has been later adapted for industrial designs by Cadence's QPlace in the early 1990s. *Min-cut based* placement tools work similarly, but use min-cut driven graph partitioners to break the netlist and the placement region into smaller pieces. For instance, CAPO (CALDWELL; KAHNG; MARKOV, 1999; ROY et al., 2006) employs the Fiduccia-Mattheysis algorithm in a top-down manner to determine the partitions. The placement region is split according to the total instance area of each partition. When partitions/regions are sufficiently small, CAPO uses a branch-and-bound formulation to determine instances location in each region. CAPO's code has been used by companies such as Synplicity and Achronix (MARKOV; HU; KIM, 2015).

**Analytic approaches** are commonly divided into *quadratic formulations* and *non-linear optimization*. FastPlace (VISWANATHAN; CHU, 2004) models global placement as a convex quadratic problem solved using the conjugate gradient method. The main drawback of the quadratic formulation is the presence of many overlaps among instances. To remove the overlaps, FastRoute proposes the use of a cell shifting heuristic, followed by the addition of spreading forces. SimPL (KIM; LEE; MARKOV, 2013) models half-perimeter wirelength as a quadratic objective function and more simple yet effective spreading forces based on a heuristic called *look-ahead legalization*. MAPLE (KIM et al., 2012), from IBM, extends SimPL by implementing multilevel placement based on netlist clustering.

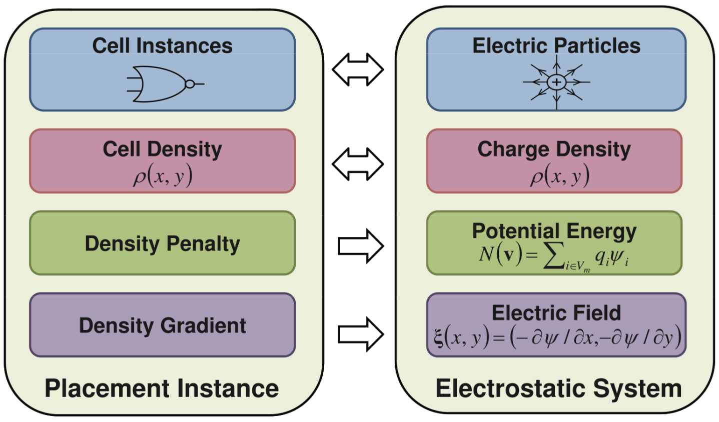
Non-linear optimization approaches optimize *log-sum-exp* functions. They are inspired in Synopsys (NAYLOR; DONELLY; SHA, 2001) modelling of half-perimeter wirelength using *log-sum-exp*. APlace (KAHNG; WANG, 2006) presents an implementation of Naylor, Donelly and Sha (2001) patent. The partitioning based NTUPlace (CHEN et al., 2008), commercialized by MediaTec, takes into consideration pre-placed blockages and density constraints. More recently, ePlace (LU et al., 2015) models global placement as an electrostatic system and RePlAce (CHENG et al., 2019) improves ePlace in terms of final half-perimeter wirelength, routability and scalability. In the next two Sections, we study ePlace and RePlAce.

#### *3.3.1.1 ePlace*

Lu et al. (2015) propose ePlace, a nonlinear placement algorithm. Unlike Naylor, Donelly and Sha (2001) and APlace (KAHNG; WANG, 2006), ePlace does not optimize the *log-sum-exp* approximation of the half-perimeter wirelength. Instead, ePlace optimizes the *weighted average* wirelength model proposed by (HSU; CHANG; BALABANOV, 2011) that presents a smaller error with respect to the actual half-perimeter wirelength. The main contribution of ePlace is a novel density function called *eDensity* which models the placement problem as a 2D electrostatic system. In doing so, every instance of the netlist is modeled as a positive particle whose quantity is equal to the area of the instance. The electric force that spreads instances is computed based on the Lorentz force law. Figure 3.10 shows the electrostatic modeling of the placement problem. If only particles originated from the netlist are taken into consideration, the system overspreads the instances, as shown in Figure 3.11(a). Consequently, the final wirelength is too large. Hence, *filler particles are inserted*. The filler particles are equally-sized rectangles, and the number of fillers inserted is the minimum required to achieve a user-specified *target density* for the placement. Figure 3.11(b) shows the filler particles in blue and Figure 3.11(c) shows the final placement with filler cells taken into consideration.

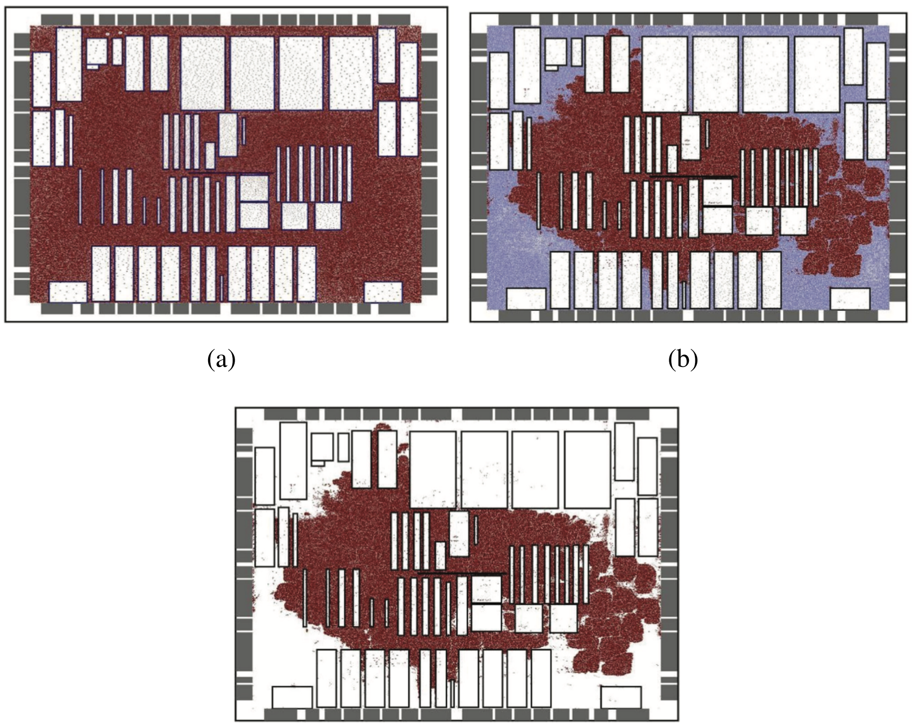
Additionally, ePlace solves the system of forces using the Nesterov Method, which presents faster convergence than the conjugate gradient used by other analytic placement tools. Finally, ePlace presents shorter wirelength and faster runtime than the leading academic tools for the ISPD2005 (NAM et al., 2005) and ISPD2006 (NAM, 2006) contest benchmarks.

Figure 3.10: ePlace modeling of the placement problem as an electrostatic system: instances are modeled as positive charges whose electric quantity is the instance area. Instance density is modeled as an electric force that spreads instances apart.


Cell Instances
Electric Particles
Cell Density $\rho(x,y)$
Charge Density $\rho(x,y)$
Density Penalty
Potential Energy $N(\mathbf{v}) = \sum_{i\in V_m} q_i\psi_i$
Density Gradient
Electric Field $\xi(x,y) = (-\partial\psi/\partial x, -\partial\psi/\partial y)$
Placement Instance
Electrostatic System

Source: Lu et al. (2015)

Figure 3.11: ePlace result (a) without fillers, (b) with fillers and (c) after fillers are removed. Instances are drawn in red and fillers are drawn in blue.



Source: Lu et al. (2015)

#### *3.3.1.2 RePlAce*

RePlAce (CHENG et al., 2019) improves many aspects of ePlace, targeting a more routing-friendly placement solution. The first contribution is the computation of two density penalty factors – The first one is computed per bin of a density grid and the second is computed per instance. With the addition of the density penalty factors, a better spreading of cells is obtained, resulting in a more routable solution. The second main contribution is an adaptive behavior for the density penalty factor considering the half-perimeter wirelength curve. The adaptive behavior aims to better allocate the optimization effort between wirelength and instance density. The last contribution is a layer-aware cell inflation technique based on global routing congestion information. RePlAce integrates the global router NCTU-GR (LIU et al., 2013) and performs global routing to estimate the routing congestion of the current placement solution. The information obtained with NCTU-GR is used to guide the cell inflation technique.

# 4 FINDING PLACEMENT-RELEVANT CLUSTERS WITH FAST MODULARITY-BASED CLUSTERING

In Chapter 3, we have provided a literature review on VLSI partitioning, community detection and placement. We now answer the question: "Can we predict instances that remain together in the physical implementation flow with clustering algorithms?" To answer this question, we use two well-known tools. The first one is the hypergraph partitioning tool hMetis (GEORGE; VIPIN, 1998), which has been successfully applied in the VLSI designs for many years. The second one is Louvain (BLONDEL et al., 2008), a fast and effective modularity-based community detection algorithm. Our goal is to assess whether we can find a better correlation between clustering and placement using modern techniques arisen from the artificial intelligence field, such as Louvain. We also show that Louvain does not need any user-input parameter in contrast with hMetis.

However, since netlists are hypergraphs and Louvain input is a graph, a netlist to graph mapping is needed. Therefore, we test three netlist to graph mapping schemes from the VLSI literature. Furthermore, we tune the graph model using five graph edge weighting alternatives and I/O proximity weights. Finally, we propose visual and numeric criteria to compare hMetis and Louvain in terms of the correlation between clustering and placement.

The remainder of this Chapter is organized as follows. We give the problem formulation in Section 3.1; we present details about the hypergraph to graph mapping and evaluation criteria in Section 3.2; we discuss the experimental setup and results in Section 3.3 and we make our conclusions in Section 3.4.

## 4.1 Problem Definition

In this work, we use the term *cluster* to refer to a group of *densely-connected* instances. Densely-connected means 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 is 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.

## 4.2 Methodology

We now present the methodology adopted in the present work. We start by showing the visualization techniques used to assess the correlation between clustering solutions and the actual placement. Then, we propose three numeric criteria to compare the quality of clustering results in terms of correlation with placement. We conclude by showing our graph modeling of the netlist.

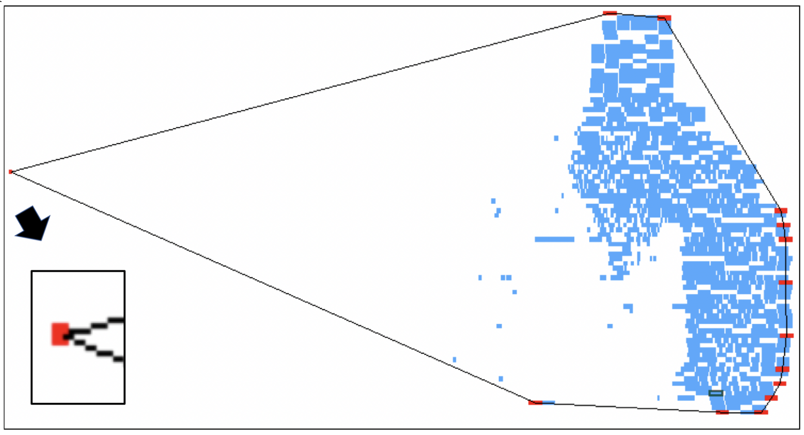
### 4.2.1 Clustering Visualization

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 (BERG et al., 1997), 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 Figure 4.1. Figure 4.2(a) depicts a "ground-truth" placement along with a cluster, with cells colored according to their clusters. Figure 4.2(b) draws the corresponding convex hulls. However, if we examine the highlighted blue cluster in Figure 4.2(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%.

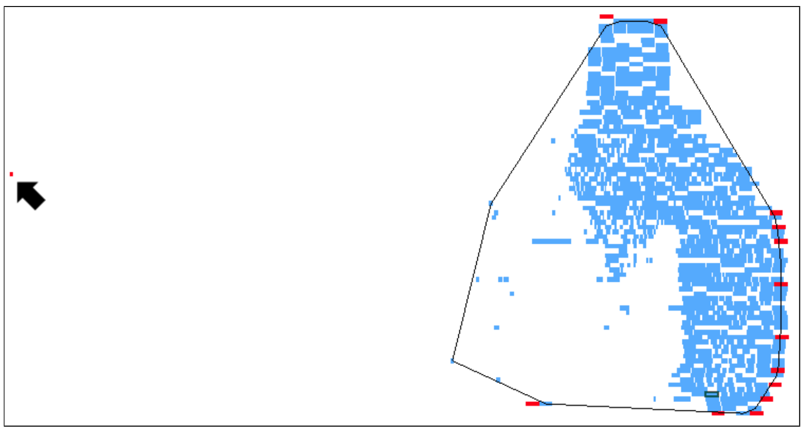
Alpha shapes (EDELSBRUNNER; KIRKPATRICK; SEIDEL, 1983), examples of which are shown in Figure 4.2(c), are a type of "shape formed by a pointset" wherein a parameter *alpha* defines the squared radius of a circle that is used to carve away space around the given points. The remaining space comprises the alpha shape of the pointset.[8] Alpha shapes are appealing in that – for appropriately chosen alpha – they provide more

[8]When alpha = $\infty$, 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.

Figure 4.1: 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 $40.1\%$.



(a)



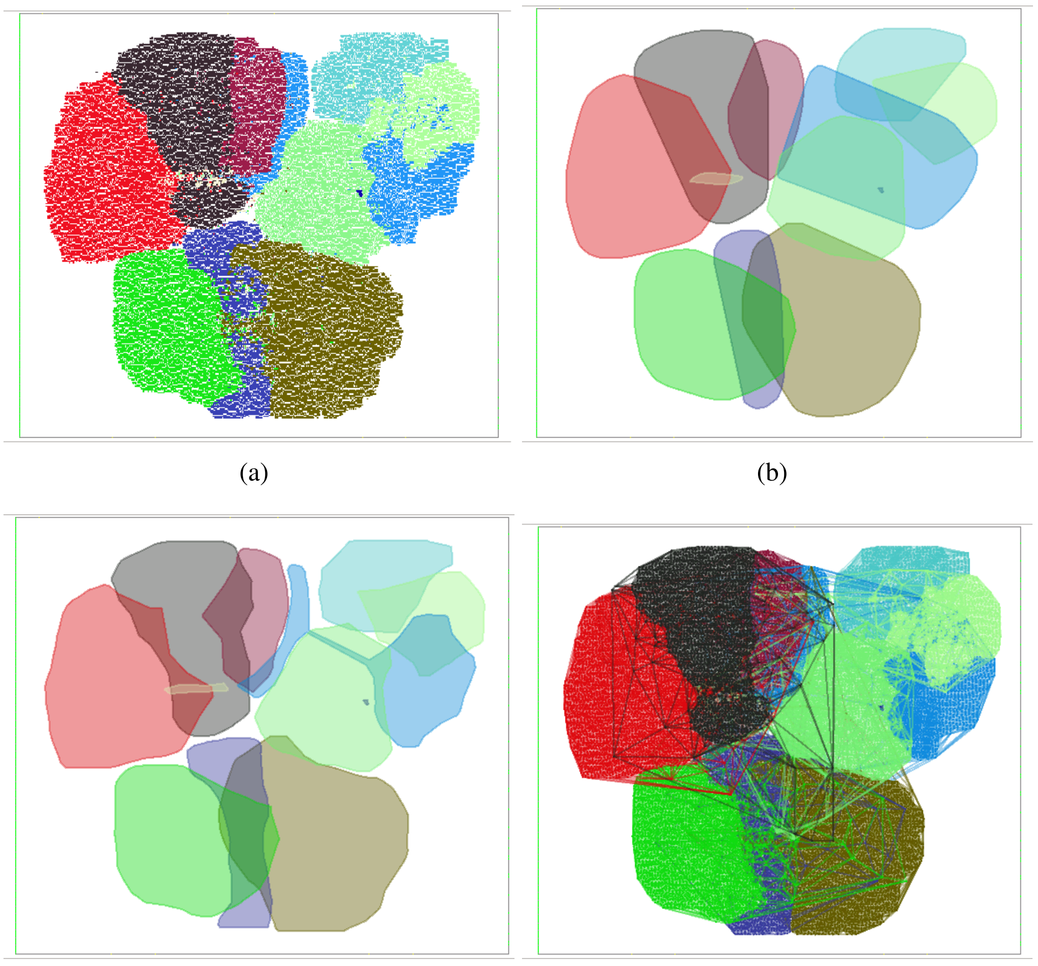
(b)

Source: from author.

accurate representations of pointsets than do convex hulls. In the following, for the test-cases 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 4.2(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 $\sim 66\%$.

Our last approach to retrieve cluster shapes is derived from the Delaunay triangulation (DT), depicted in Figure 4.2(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.

Figure 4.2: Different approaches to correlate clusters with the placement for the circuit ispd18_test2 (MANTIK et al., 2018): (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.



(a) (b)

(c) (d)

Source: from author.

### 4.2.2 Clustering 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) (DAVIES; BOULDIN, 1979), Variance Ratio Criterion (VRC) (CALIńSKI; HARABASZ, 1974) and Silhouette Coefficient (SC) (ROUSSEEUW, 1987), traditionally used for spatial clustering evaluation, as indicators of cluster quality.[9] The DBi is defined as:

$$DBi = \frac{1}{n}\sum_{i=1}^{n} max_{i \neq j}\Big(\frac{\sigma_i + \sigma_j}{l(\rho_i, \rho_j)}\Big) \tag{4.1}$$

where $n$ is the number of clusters, $\sigma_i$ is the the average distance from the cluster elements to the centroid of cluster $i$, $\rho_i$ is the centroid of cluster $i$ and $l(\rho_i, \rho_j)$ is the distance between centroids $\rho_i$ and $\rho_j$. 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:

$$VRC = \frac{Tr(B)}{Tr(W)} \times \frac{N - n}{n - 1} \tag{4.2}$$

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

$$B = \sum_{i}^{n} n_i(\rho_i - \rho)(\rho_i - \rho)^T \tag{4.3}$$

$$W = \sum_{i}^{n}\sum_{x \in c_i}(x - \rho_i)(x - \rho_i)^T \tag{4.4}$$

where $n_i$ is the number of elements in cluster $i$, $\rho$ 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:

$$SC = \sum_{i=1}^{n} \frac{b_i - a_i}{max(a_i, b_i)} \tag{4.5}$$

[9]To ensure a correct comparison, we implemented DBi, VRC and SC in the same way as in (SCIKIT LEARN, 2020).

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

### 4.2.3 Graph Model of the 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, Wagner and Wagner (1993) 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 4.1. 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.

Table 4.1: Description of net weighting alternatives.

| Name | Weight per edge | Rationale |
|---|---|---|
| Lengauer (LENGAUER, 1990) | $1/(p_h - 1)$ | Set the total weight of the net cut to be at least one. |
| Huang (HUANG; KAHNG, 1995) | $4/(p_h(p_h - 1))$ | Set the expected weight of a net cut to be one. |
| Tsay-Kuh (TSAY; KUH, 1991) | $2/p_h$ | Minimize the squared wirelength of the net. |
| Tsay-Kuh-2 (TSAY; KUH, 1991) | $(2/p_h)^3$ | Minimize the Manhattan wirelength of the net. |
| Frankle-Karp (FRANKLE; KARP, 1988) | $2/p_h^{1.5}$ | Minimize the worst deviation from the square of the spanning tree. |

Source: from author.

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.[10] 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:

$$w_{h,2} = min((d(I)), (d(O))) \tag{4.6}$$

$$w_h = w_{h,1} \frac{1}{(w_{h,2} + 1)} \tag{4.7}$$

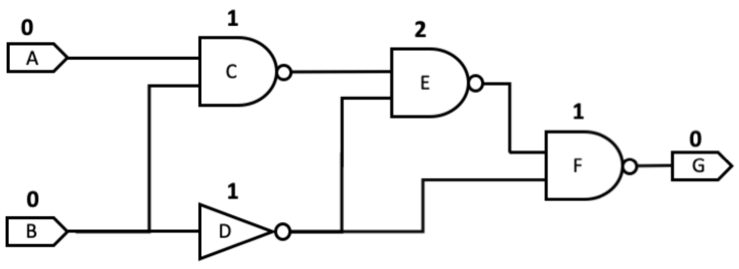
Figure 4.3(a) depicts a netlist with two input ports, four instances, and one output port. The number above each instance represents the topological distance to the closest I/O. Figure 4.3(b) shows the equivalent graph using the traditional clique decomposition, in which the number related to each edge represents its weight using the Lengauer weighting scheme. Figure 4.3(c) shows the equivalent graph using the star decomposition (virtual nodes are drawn as circles with a dashed outline). Finally, Figure 4.3(d) integrates the notion of I/O proximity according to Equation (4.7). In Subsection 4.3.1 we present experiments discussing the impact of adding netlist information. We note in Section 4.4 that incorporation of timing information (slack, etc.) in the graph modeling remains an open issue for future work.

## 4.3 Experimental Setup and Results

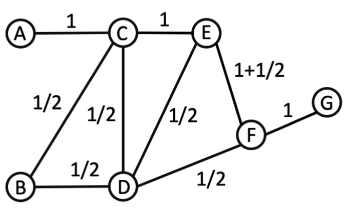
We implement our modularity-based clustering approach using Rsyn (FLACH et al., 2017; RSYN, 2016) and run all experiments on an Intel Xeon E5-2695 dual-CPU server at 2.1GHz with 256GB RAM. Our analyses are performed in a set of open design blocks (OPENCORES, 1999). We use a commercial synthesis tool 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: 14nm, 28nm and 65nm. 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 (CHENG et al., 2019; REPLACE, 2018) and the legalization tool OpenDP v0.1.0 (DO; WOO; KANG, 2019; OPENDP, 2020).

[10]In the experiments of Fogaça et al. (FOGAçA et al., 2019), the addition of I/O proximity weights improves the quality of results in terms of DBi by 28%, on average. In Section 4.3.1, we extend the experiments of (FOGAçA et al., 2019) and observe that I/O proximity weights only improve DBi in when combined with Tsay-Kuh-2 edge weights and the Star decomposition.

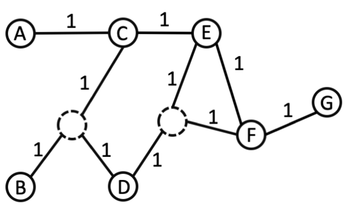
Figure 4.3: Netlist decomposition.



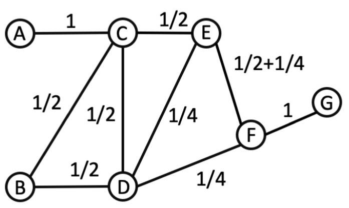
(a) Netlist



(b) Clique decomposition using Lengauer weighting scheme



(c) Star decomposition



(d) Clique decomposition with I/O proximity weights

Source: from author.

Table 4.2 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.

### 4.3.1 Evaluation of Different Graph Models

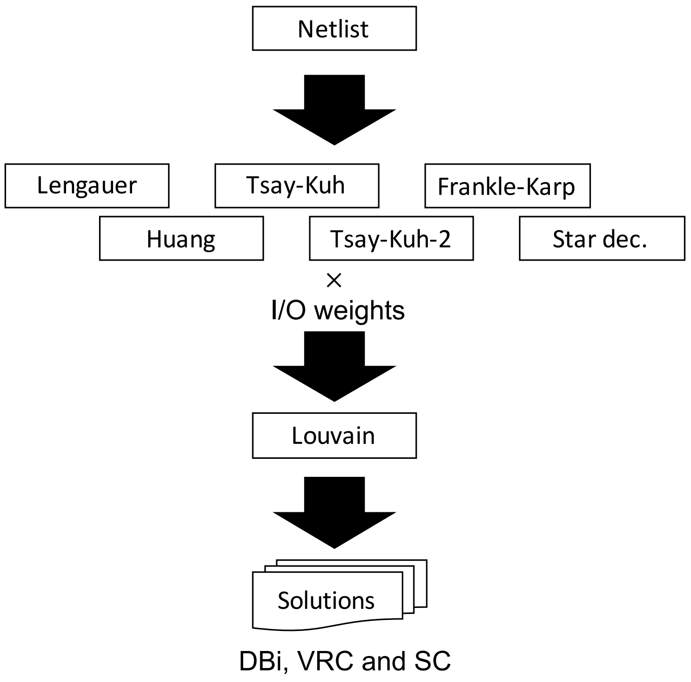
In our first experiment, we compare the clique decomposition using different weighting schemes and the star decomposition. We also evaluate the use of I/O proximity weights in our graphs. The flow of this experiment is shown in Figure 4.4.

Table 4.2: Benchmarks and attributes.

| Design | Insts | Nets | I/Os |
|---|---|---|---|
| jpeg_encoder_14 | 44083 | 45018 | 49 |
| ldpc_decoder_14 | 38559 | 40610 | 4100 |
| netcard_14 | 272865 | 274704 | 1849 |
| leon3mp_14 | 316537 | 316791 | 333 |
| MegaBoom_14 | 1249594 | 1254352 | 945 |
| jpeg_encoder_28 | 46962 | 47775 | 49 |
| ldpc_decoder_28 | 40402 | 42506 | 4100 |
| netcard_28 | 235277 | 237122 | 1849 |
| leon3mp_28 | 400836 | 401091 | 333 |
| MegaBoom_28 | 1419923 | 1425174 | 945 |
| netcard_65 | 239901 | 241740 | 1849 |
| leon3mp_65 | 325041 | 325295 | 333 |
| MegaBoom_65 | 1169564 | 1174669 | 945 |

Source: from author.

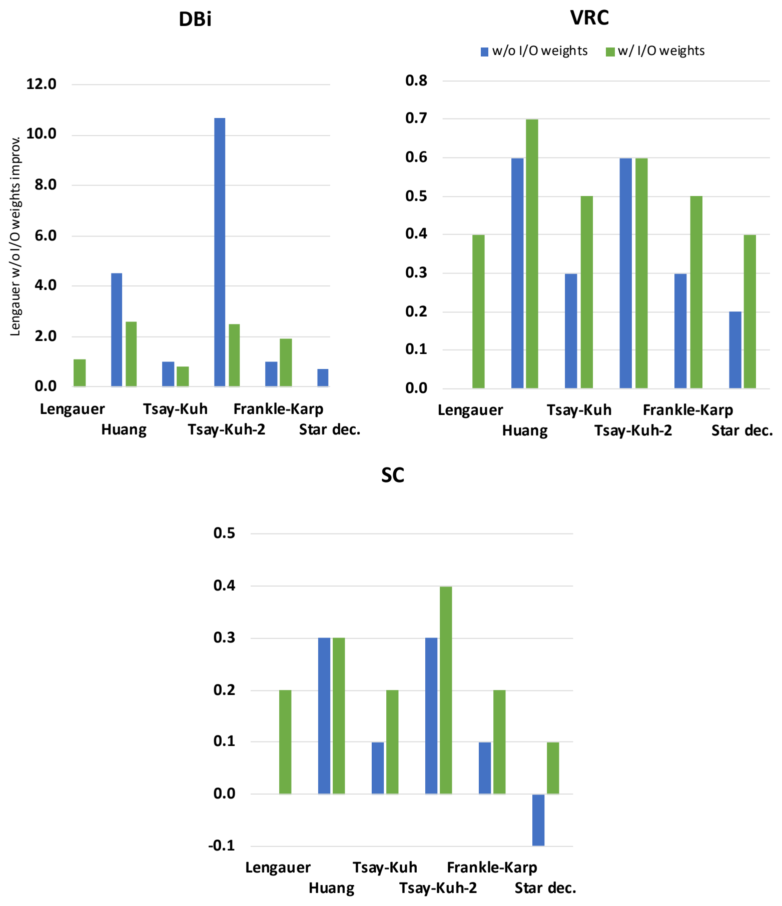
Figure 4.4: Experiment 1 flow: Each netlist is modeled as a graph using the clique and star decompositions. In using clique, we compare five edge-weighting approaches. Furthermore, we evaluate the use of I/O proximity weights in the graph edges. Next, Louvain clusters the graphs. Finally, the clustering solutions are evaluated in terms of DBi, VRC and SC.



Source: from author.

Table 4.3 shows the values of DBi, VRC and SC for each approach alone and with I/O proximity information of Equation (4.7). 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.[11] 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. The superiority of Lengauer without I/O proximity weights can also be observed in Figure 4.5, in which we present its average improvement over the other graph models. Therefore, all of our following experiments are performed using Lengauer without I/O proximity weights.

Figure 4.5: The average improvement of Lengauer w/o I/O proximity weights over the other graph models.



Source: from author.

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

Table 4.3: Netlist tuning.

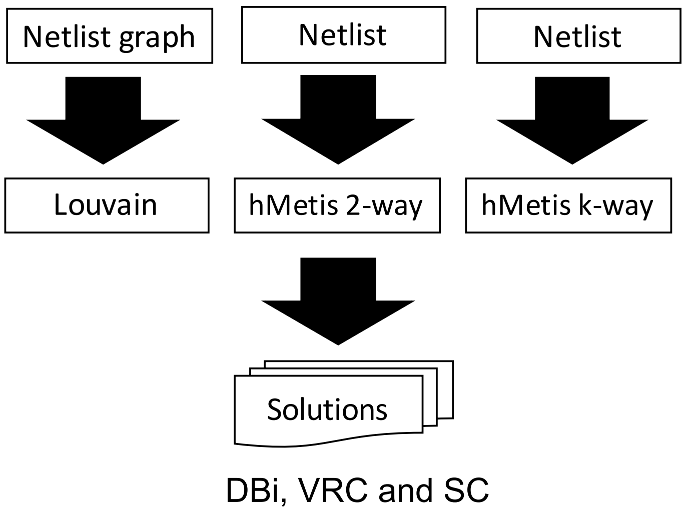
| | Design | Lengauer | | | Huang | | | Tsay-Kuh | | | Tsay-Kuh-2 | | | Frankle-Karp | | | Star decomposition | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | DBi | VRC | SC | DBi | VRC | SC | DBi | VRC | SC | DBi | VRC | SC | DBi | VRC | SC | DBi | VRC | 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 | 24745 | 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 | 24921 | -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 | 120483 | 0.074 | 8.0 | 31247 | -0.245 | 4.0 | 78663 | -0.029 | 53.0 | 9592 | -0.308 | 3.5 | 73884 | -0.054 | 3.9 | 73311 | -0.091 |
| | leon3mp_28_70 | 1.6 | 122899 | 0.066 | 11.6 | 29156 | -0.248 | 3.2 | 74776 | -0.027 | 55.2 | 8976 | -0.342 | 1.9 | 81224 | -0.025 | 2.3 | 76294 | -0.101 |
| | MegaBoom_28_55 | 2.1 | 375126 | 0.065 | 21.0 | 21945 | -0.352 | 2.2 | 404678 | 0.031 | 30.7 | 13496 | -0.358 | 2.1 | 370853 | -0.017 | 2.9 | 476911 | 0.118 |
| | MegaBoom_28_70 | 1.5 | 329231 | 0.043 | 21.6 | 11939 | -0.434 | 3.1 | 365299 | 0.056 | 33.2 | 10759 | -0.387 | 2.1 | 354274 | 0.016 | 3.1 | 210473 | 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 | 69923 | -0.079 | 5.4 | 20891 | -0.273 | 3.4 | 60302 | -0.109 | 8.9 | 9724 | -0.415 | 4.6 | 58482 | -0.087 | 2.0 | 97302 | -0.073 |
| | leon3mp_28_70 | 2.5 | 81649 | -0.039 | 5.0 | 21097 | -0.259 | 3.1 | 62439 | -0.093 | 10.4 | 9576 | -0.408 | 5.0 | 59640 | -0.085 | 2.6 | 94111 | -0.039 |
| | MegaBoom_28_55 | 1.7 | 307569 | -0.036 | 5.3 | 96128 | -0.224 | 2.0 | 290638 | -0.090 | 5.0 | 63940 | -0.237 | 2.7 | 198454 | -0.192 | 1.7 | 397838 | 0.005 |
| | MegaBoom_28_70 | 1.8 | 287494 | -0.001 | 14.2 | 73202 | -0.238 | 1.9 | 232260 | -0.093 | 8.2 | 45159 | -0.297 | 2.3 | 206758 | -0.178 | 2.9 | 171724 | -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 |

Source: from author.

### 4.3.2 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 discussed in Section 3.2.5, the modularity criterion guides Louvain toward high-quality clustering solutions without the need for input parameters. On the other hand, hMetis is a Fiduccia-Mattheyses-based partitioning tool that relies on a less robust criterion, called "cut", and needs input parameters from the user; this itself presents a challenge when we seek a fair comparison versus hMetis. To perform a fair comparison we explore two input parameters of hMetis: (i) the number of clusters and (ii) the unbalance factor.[12] 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.

Figure 4.6: Experiment 2 flow: Comparison of Louvain with hMetis 2-way and $k$-way.



Source: from author.

[12]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 4.4 and Figure 4.7 compare 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 4.3. 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.

Table 4.4: 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.

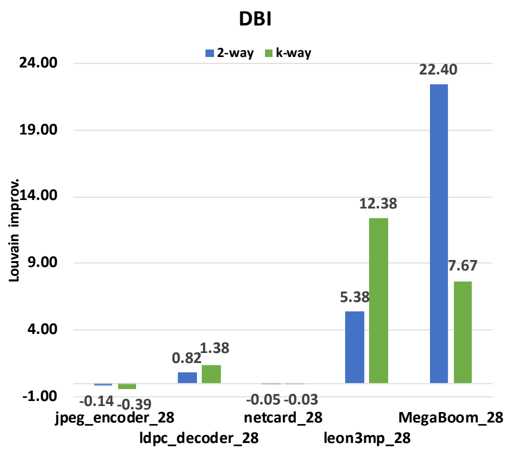
| Design | Louvain | | | | | hMetis 2-way | | | | | hMetis k-way | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #CL | DBi | VRC | SC | CPU(s) | #CL | DBi | VRC | SC | CPU(s) | #CL | DBi | VRC | SC | CPU(s) |
| jpeg_encoder_28 | 84 | 3.6 | 4987.89 | -0.096 | 2 | 64 | 3.1 | 10171 | 0.042 | 13 | 84 | **2.2** | **12987** | **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** | **67680** | **0.122** | 145 | 72 | 3.6 | 55675 | 0.101 | 138 |
| leon3mp_28 | 70 | **1.6** | **122899** | **0.066** | 75 | 64 | 10.2 | 33347 | -0.053 | 191 | 70 | 21.4 | 28753 | -0.088 | 179 |
| MegaBoom_28 | 40 | **1.5** | **329231** | **0.043** | 448 | 70 | 35.1 | 33897 | -0.119 | 826 | 40 | 13.0 | 28793 | -0.137 | 912 |
| Average | | **1** | 1 | 1 | 1 | | 6.7 | 1.6 | **1.2** | 4.5 | | 5.2 | **1.7** | 1.2 | 4.6 |

Source: from author.

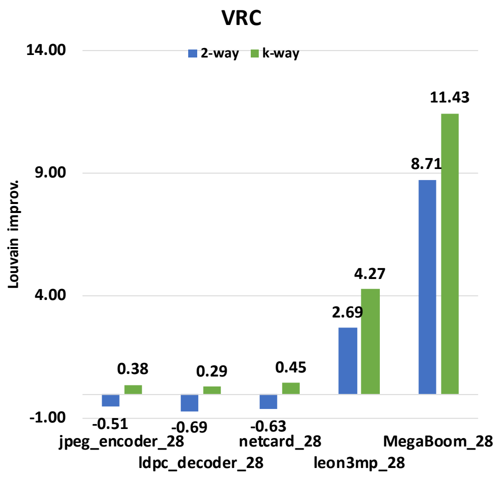
One of the key advantages of Louvain is its almost linear runtime in sparse graphs. Louvain is 6× faster than the fastest hMetis run for the smallest benchmark, ldpc_decoder (18s). In the largest benchmark, MegaBoom, Louvain is 1.8× faster than the fastest hMetis run (826s). On average, Louvain is 4.5× faster than hMetis.[13]

[13]We note that hMetis is a Fiduccia-Mattheyses-based partitioning tool. Every iteration of the Fiduccia-Mattheyses algorithm has the runtime complexity of $O(|V|)$, where $|V|$ is the number of vertices in the hypergraph. In contrast, Louvain has runtime complexity of $O(|V|log|V|)$. As noted in Section 3.2.5, every "pass" (iteration) of Louvain reduces the graph size based on the current clustering result. In our experiments, most of the Louvain runtime comes from the first "pass". Additionally, previous studies suggest that Louvain converges in less than 5 iterations for most graphs (BLONDEL et al., 2008). In practice, by reducing the graph size after every "pass", the Louvain algorithm converges faster than Fiduccia-Mattheyses-based heuristics.

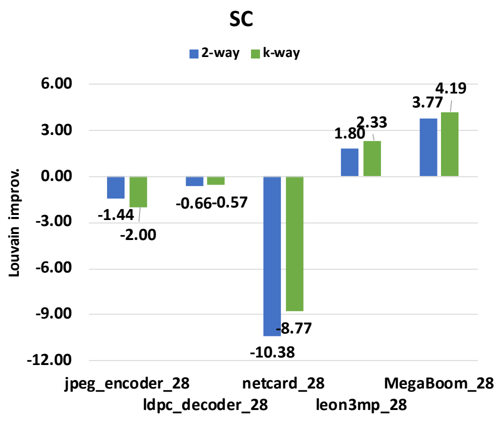
Figure 4.7: Comparison between Louvain and hMetis in terms of DBi, VRC and SC.



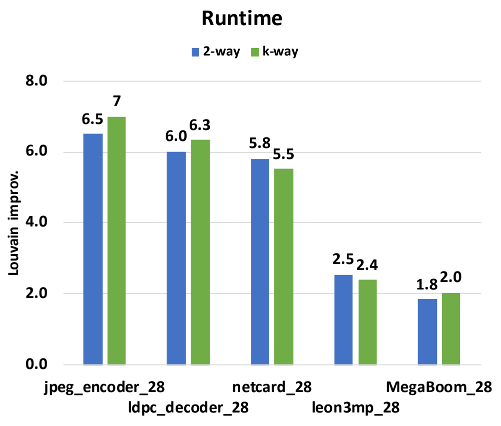
(a) 1:1



(b) 1:1



(c) 1:1



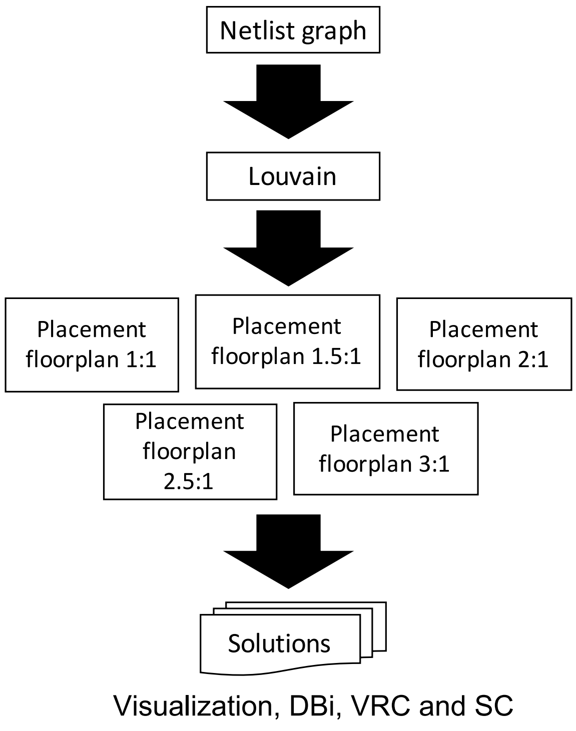
(d) 1:1

Source: from author.

### 4.3.3 Robustness With Respect to Design Floorplan

In this subsection, we show the robustness of Louvain using different floorplan configurations. Figure 4.8 shows our experiment flow. First, we cluster the netlist graphs using Louvain. Next, 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.

Figure 4.8: Experiment 3 flow: The netlist graphs are clustered with Louvain and the clustering results are evaluated using placement under five floorplan aspect ratios: 1:1, 1.5:1, 2:1, 2.5:1 and 3:1.



Source: from author

Table 4.5 and Figure 4.9 show the delta from the floorplan with 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 (4.8)-(4.10), so that 0 means no change with respect to aspect ratio 1:1, 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%.

$$\Delta DBi = 1 - \frac{DBi_{ar}}{DBi_{1:1}} \tag{4.8}$$

$$\Delta VRC = \frac{VRC_{ar}}{VRC_{1:1}} - 1 \tag{4.9}$$

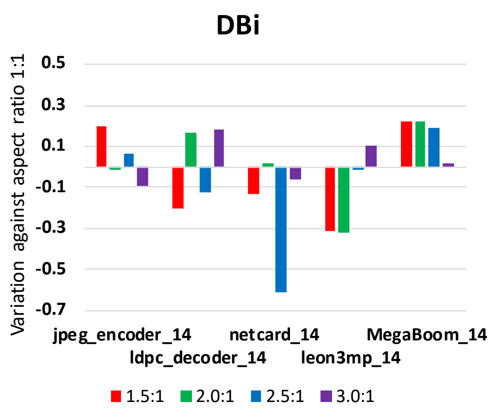
$$\Delta SC = \frac{SC_{ar} - SC_{1:1}}{|SC_{1:1}|} \tag{4.10}$$

Table 4.5: 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 (4.8)-(4.10).

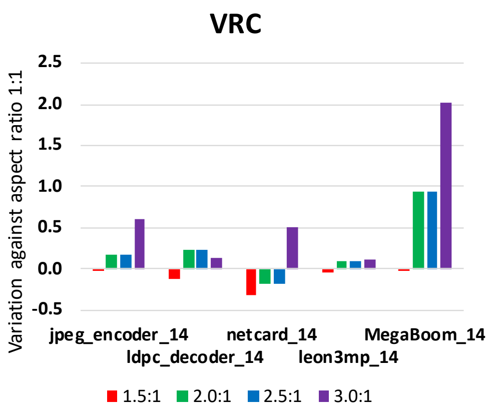
| Design | 1.5:1 | | | 2.0:1 | | | 2.5:1 | | | 3.0:1 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | DBi | VRC | SC | DBi | VRC | SC | DBi | VRC | SC | DBi | VRC | SC |
| jpeg_encoder_14 | 0.20 | -0.03 | -0.25 | -0.02 | 0.17 | -0.50 | 0.06 | 0.17 | -0.18 | -0.09 | 0.60 | 0.25 |
| ldpc_decoder_14 | -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 |

Source: from author.

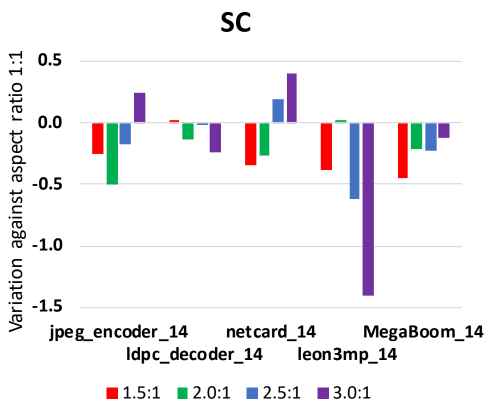
Figure 4.9: The deltas of DBI, VRC and SC from the floorplan with aspect ratio 1:1 to aspect ratios 1.5:1, 2:1,2.5:1 and 3:1. Values are normalized according to Equations (4.8)-(4.10).



(a)



(b)

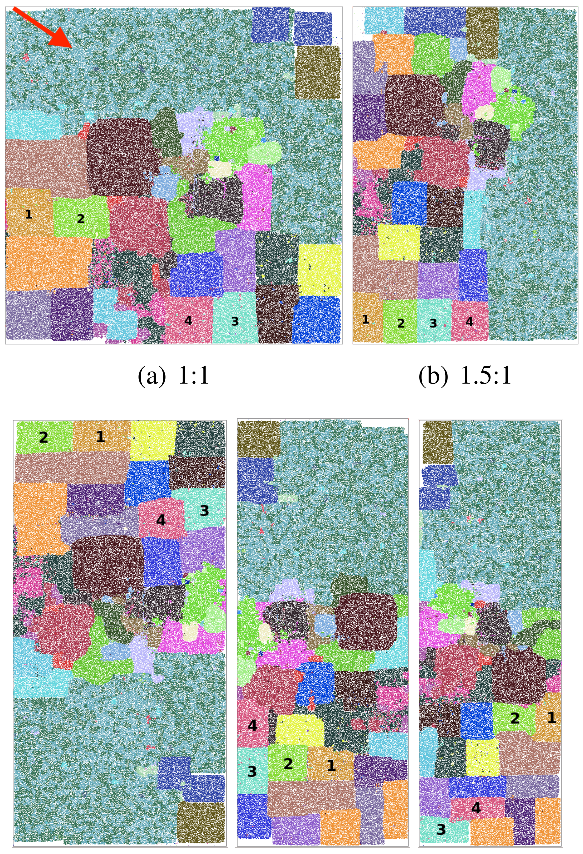


(c)

Source: from author.

The reason for this behavior can be seen in Figure 4.10. 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.

Figure 4.10: Visual comparison of MegaBoom_14 with different aspect ratios and same utilization. The images have been scaled for a better visualization. The red arrow highlights two blue clusters blending together.



(a) 1:1 (b) 1.5:1

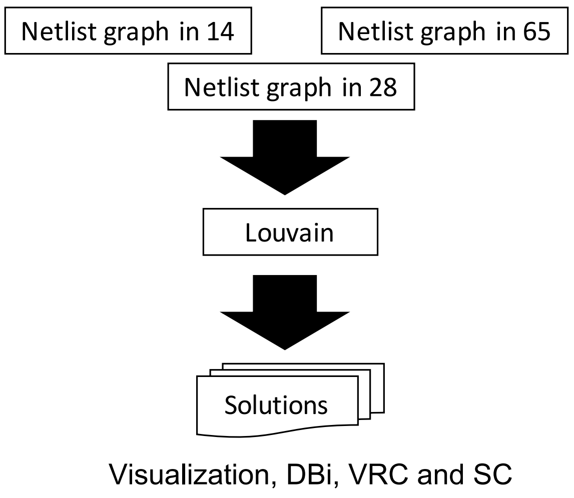
(c) 2:1 (d) 2.5:1 (e) 3:1

Source: from author.

### 4.3.4 Validation Across Technology Nodes

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 14nm enablement and the other in a 28nm 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: 14nm, 28nm and 65nm. Figure 4.11 shows our experiment flow.

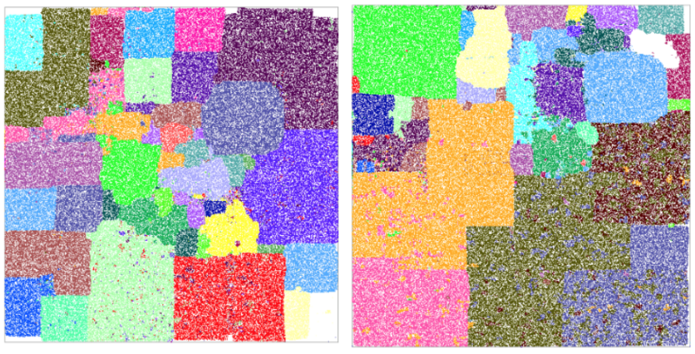
Figure 4.11: Experiment 4 flow: We synthesize the same design in 14nm, 28nm and 65nm. Next, we cluster the netlist graphs using Louvain. Finally, we compare the results visually and numerically using DBi, VRC and SC.



From Table 4.2, 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 4.6 and Figure 4.13. 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. Figure 4.12 shows the instances of MegaBoom_28 and MegaBoom_65, colored according to Louvain clustering. The visualization for MegaBoom_14 can be seen in Figure 4.10(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.

Figure 4.12: Clustering results for (a) MegaBoom_28 and (b) MegaBoom_65. Compare with MegaBoom_14 from Figure 4.10(a).



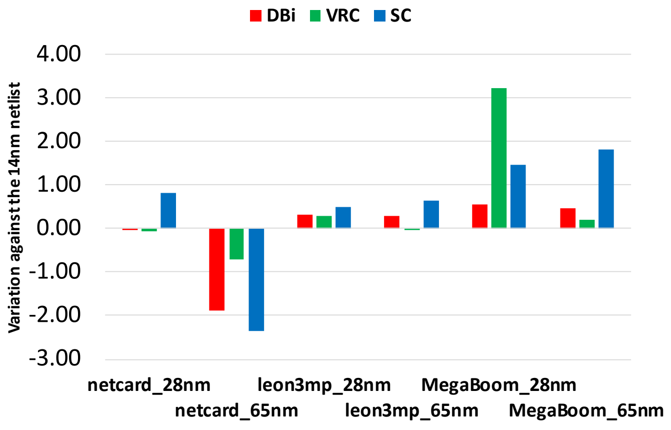
Source: from author.

Table 4.6: Variation of DBi, VRC and SC for netcard, leon3mp and MegaBoom when compared to their implementation in 14nm. Values are normalized according to Equations (4.8)-(4.10).

| 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 |

Source: from author.

Figure 4.13: Graphic representation of the Variation of DBi, VRC and SC for netcard, leon3mp and MegaBoom when compared to their implementation in 14nm. Values are normalized according to Equations (4.8)-(4.10).



Source: from author.

The numerical and visual results of Louvain for a given RTL synthesized in 14nm, 28nm and 65nm 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.

## 4.4 Conclusion

In this Chapter, we have tackled the problem of predicting instances that remain together throughout the physical implementation flow using netlist clustering. We proposed the use of convex hulls, alpha shapes and Delaunay triangulation as methods for visualization and manual debugging of the correlation between clustering and actual placement. We have shown in a practical example that the Delaunay triangulation is the best way for visualization of cluster shapes and outliers. To compare clustering solutions, we have proposed the use of three numerical criteria: Davies-Bouldin index, variance ratio coefficient and silhouette coefficient.

We have applied the fast modularity-based graph clustering algorithm Louvain in VLSI for the first time. In performing netlist to graph mapping, we have compared the clique and star decomposition methods and five edge weighting alternatives. Additionally, we have evaluated the use of I/O proximity weights. Our first experiment has shown that using clique decomposition with the Lengauer edge weighting scheme leads to clustering results with the best overall values of DBi, VRC and SC. We also show that works, the use of I/O proximity weights does not lead to better results for all consistently weighting schemes.

In our second experiment, we have assessed the superiority of Louvain clustering when compared to hMetis, a traditional VLSI clustering tool – Louvain presents the best results of DBi on average. Louvain also outperforms hMetis in all metrics for our two largest testcases. In terms of runtime, Louvain is, on average, 4.5× faster than hMetis. Furthermore, in contrast with hMetis, Louvain does not need user-input parameters.

Our third experiment has evaluated the robustness of Louvain clustering using floorplans with different aspect ratios. We have noticed a considerable variation in DBi, VRC and SC. However, we have demonstrated through visualization that this variation comes from the chaotic behavior of the placement tool. The visualization has also shown that the neighborhood of clusters remains the same. Therefore, we conclude that Louvain clustering *is* robust to different floorplan configurations.

The final experiment has compared Louvain clustering solutions for the same RTL synthesizes with different technologies. We conclude that the gate-level netlist changes considerably in terms of the number of instances depending on the technology. We observe a difference of up to 27% in the number of instances in our experiments. The differences in the gate-level netlists affect the results of Louvain clustering. In one testcase the number of clusters between MegaBoom_14 and MegaBoom_65 is around 37%. The values of DBi, VRC and SC also change significantly among technologies. From this experiment, we conclude that Louvain clustering *is not* robust across different technologies.

### 4.4.1 Directions for Future Works

The experiment on Section 4.3.1 has shown that the graph mapping of the netlist has a significant impact on the quality of the results. We list three directions for studies aiming to improve the present work:

- *Timing weights.* Improve cluster quality by modeling timing information as edge weights. E.g., increase edge weights based on slacks or critical paths.
- *Machine learning-based graph mapping.* We believe machine learning could identify patterns on the netlist graph and guide the graph mapping of the netlist.
- *Hierarchy weights.* Instances belonging to the same hierarchy in the RLT tend to be placed together. It is possible to identify RTL hierarchy in the mapped netlist in the instances names (e.g., "hier1/hier2/inst1"). Giving higher weights to edges between instances that belong to the same hierarchy can potentially lead to better clustering results.

# 5 FAST PLACEMENT OF INSTANCES WITH BLOB 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 propose two techniques. The first one, called *"blob placement"*, consists of placing a clustered netlist. In the clustered netlist, the cells from a given cluster are considered a single instance. Our clustered netlists have approximately 50–100$\times$ fewer instances than the flat netlists. Hence, "blob placement" is potentially much faster than the flat placement. Our second technique, called *"seeded placement"*, flattens the placed clustered-netlist and spread the instances with a global placement technique. Finally, we implement a prototype fast placement flow with "blob placement" and "seeded placement" techniques, which aims to significantly reduce the placement runtime with minimum impact in routed wirelength.

The remainder of this Chapter is organized as follows: in Section 5.1, we detail our prototype flow. In Section 5.2, we perform experiments to assess the quality of our prototype flow in terms of runtime and final routed wirelength. Finally, in Section 5.3 we make our conclusions.

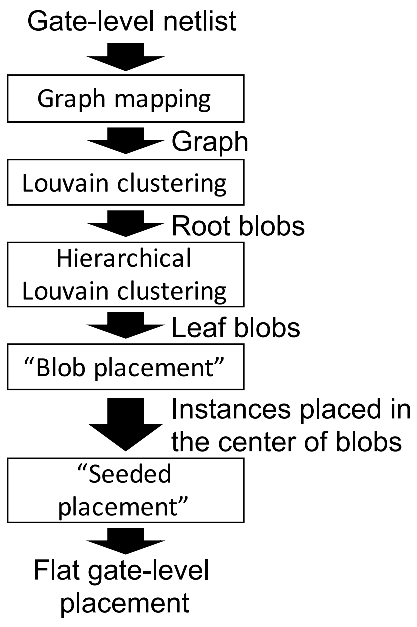
## 5.1 Prototype Blob and Seeded Placement Flow

We have developed a simple experimental flow to predict final placement using (i) modularity-based clustering without any user configuration 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 Figure 5.1.

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

Figure 5.1: Experimental fast placement flow.



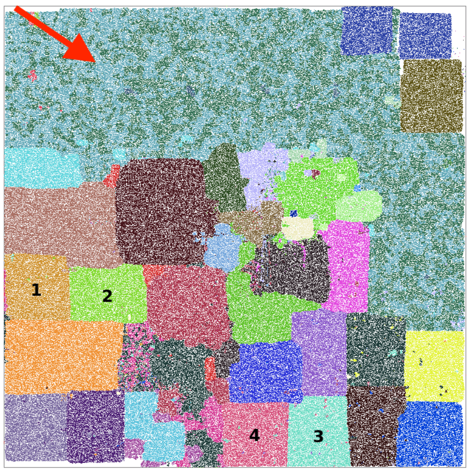
Source: from author.

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 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 21K, 12K and 19K clusters, and 72K, 59K and 74K inter-cluster nets, respectively. The same behavior is observed in other testcases, such as netcard_14 and leon3mp_14 that have 2.8K and 2.6K clusters, and 27K and 37K nets, respectively.

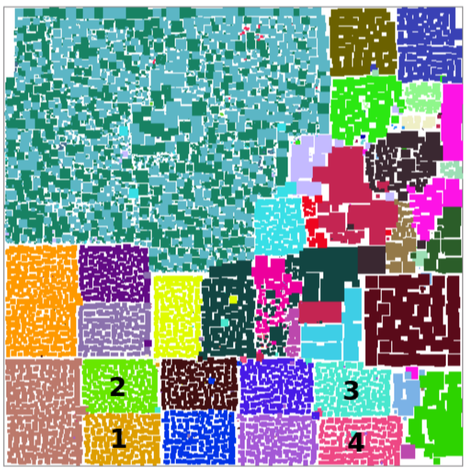
Figure 5.2(b) depicts the outcome of "blob placement" for the MegaBoom_14 that has 27 root blobs and 21K leaf blobs. The root blobs contain an average of 46K instances and leaf blobs contain an average of 59 instances. We adapt the open-source academic tool RePlAce to perform the blob placement. In doing 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 (1.2M instances), using a single thread of a 2.1GHz Xeon server.[14]

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. Figure 5.2(a) shows the original flat placement, Figure 5.2(b) shows the blob placement and Figure 5.2(c) shows the “seeded placement” for MegaBoom_14.

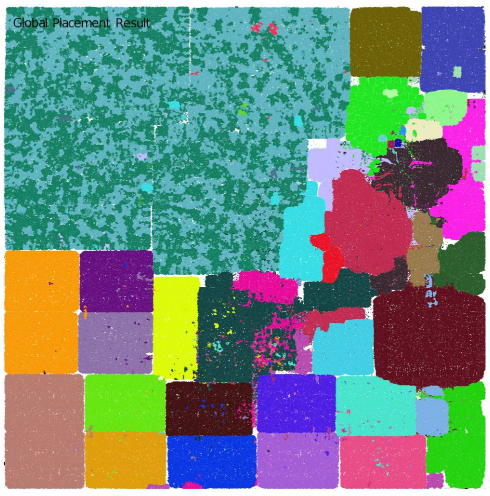
Figure 5.2: MegaBoom_14: (a) flat placement, (b) “blob placement” and (c) “seeded placement”.



(a)



(b)



(c)

Source: from author.

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

## 5.2 Experimental Setup and Results

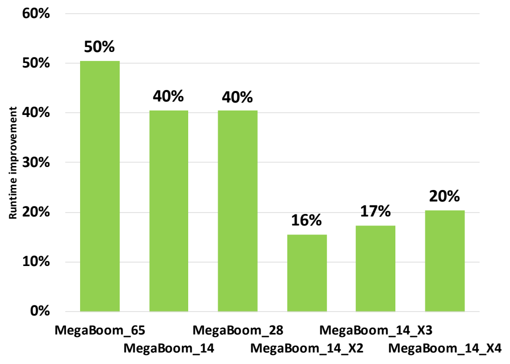
We now compare our prototype flow with the flat placement. Table 5.1 summarizes the testcase information, routed wirelength and runtime of the flat placement and fast placement to a set of six testcases. Our testcases are synthesized using 65nm, 28nm and 14nm industrial technologies following the same recipe from Chapter 4. We use a commercial tool to perform global routing and extract routed estimated wirelength information. As shown in Figure 5.3, our experimental flow presents runtime speed ups that range from 20% (MegaBoom_X4_14) to 50% (MegaBoom_14 and MegaBoom_65). In Figure 5.4, the total runtime of fast placement is broken into Louvain clustering, hierarchical Louvain clustering, "blob placement" and "seeded placement". 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×. Figure 5.5 shows that 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.

Table 5.1: Benchmark attributes and 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 4.2 for the instance complexities.

| Design | Inst ($\times 10^6$) | Nets ($\times 10^6$) | I/Os | Flat placement | | Fast "seeded placement" | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | WL(m) | CPU(s) | WL(m) | LC CPU(s) | HLC CPU(s) | BP CPU(s) | SP CPU(s) | Total CPU(s) |
| MegaBoom_14 | 1.2 | 1.2 | 945 | 21 | 1941 | 21.3 | 268 | 266 | 184 | 437 | 1156 |
| MegaBoom_28 | 1.4 | 1.4 | 945 | 36 | 1623 | 36.2 | 335 | 108 | 116 | 408 | 967 |
| MegaBoom_65 | 1.1 | 1.1 | 945 | 57.5 | 1613 | 57 | 164 | 100 | 104 | 432 | 800 |
| MegaBoom_14_X2 | 2.5 | 2.5 | 1888 | 41.6 | 3214 | 41.8 | 705 | 271 | 440 | 1299 | 2714 |
| MegaBoom_14_X3 | 3.7 | 3.7 | 2831 | 62.3 | 5211 | 64.1 | 1020 | 415 | 943 | 1933 | 4311 |
| MegaBoom_14_X4 | 4.9 | 4.9 | 3774 | 83.4 | 8642 | 85.6 | 1418 | 572 | 1400 | 3492 | 6882 |

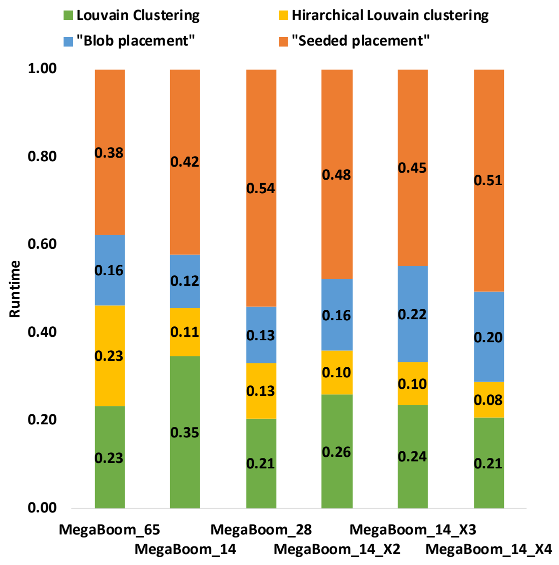
Source: from author.

Figure 5.3: Runtime improvement of “seeded placement” over flat placement.



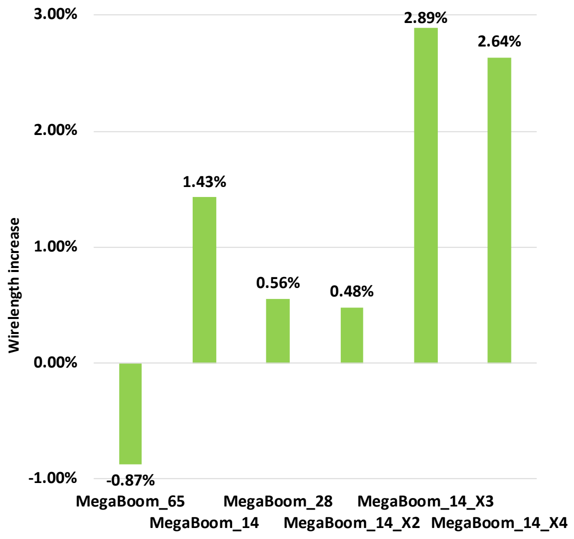
Source: from author.

Figure 5.4: Runtime breakdown of our fast placement flow.



Source: from author.

Figure 5.5: Post-route wirelength degradation of “seeded placement” over flat placement.



Source: from author.

## 5.3 Conclusions

In this Chapter, we have presented a prototype flow to enable fast placement of instances. First, we have proposed a methodology to create a cluster netlist from Louvain's clustering solution. Then, we have used a global placement tool to perform fast placement of clusters – a technique that we call "blob placement." We have visually shown that "blob placement" correlates well with the flat placement of the netlist. Finally, we have demonstrated how to flatten the "blob placement" from using a technique that we call "seeded placement."

Our experiments have shown that our prototype flow can achieve considerably less runtime when compared to the flat placement. To further assess the correlation of our prototype fast placement flow and flat placement, we have matched the post-route wirelength of our prototype fast placement flow against the post-route wirelength of the flat placement. In doing so, we have observed an impact of less than 3% on the quality of the results.

The presented results reveal the potential of our prototype fast placement flow to predict the actual flat placement and hence opens the door to early netlist and floorplan evaluation. Next, we give directions to enhance the runtime and the quality of the results of our prototype flow.

### 5.3.1 Directions for Future Works

We have presented a *prototype* fast placement flow to assess that we can efficiently predict flat placement using clustering. Although the results presented in the last present a substantial runtime speedup with similar quality of results, we now offer many directions in which our flow can be improved.

- *Parallelism.* As mentioned in Section 5.2, we performed hierarchical Louvain in a single core, while the problem is highly parallelizable. The runtime could be scaled down almost linearly with the number of cores available.
- *Cluster netlist.* Our methodology for the creation of the cluster netlist is purely based on empirical data. We believe that embedding timing weights (e.g., timing paths) and netlist information (e.g., tangled logic inside blobs, similar to (JINDAL et al., 2010)) could lead to better results. Please note that this is closely related to

the graph modeling of the netlist used in the clustering (Section 4.2.3).

- *Placement*. Our global placement has been designed to perform placement of *standard cells*. Standard cells have different widths but the area ratio among standard cells is usually of a few dozens. In our clustered netlist, the cluster area range from a few dozen instances to thousands of instances. In this case, a *mixed-size* placement tool could better handle the cluster netlist.

# 6 FINAL CONCLUSIONS

In the present work, 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 (BLONDEL et al., 2008) algorithm, is clearly superior to mincut- or Rent parameter-driven methods (KARYPIS et al., 1997) (CALDWELL; KAHNG; MARKOV, 2000) (RENTCON, 2008) 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 (LENGAUER, 1990) is effective in conjunction with Louvain. Comparisons with traditional hMetis-based clustering (KARYPIS et al., 1997) show that our Louvain-based approach achieves on average 50% better correlation to actual netlist placements, as well as $2\times$ faster runtimes for our largest testcases. Last, we demonstrate the potential of using modularity-based clustering with fast “blob placement” of clusters and “seeded placement” 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. 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.

## REFERENCES

ALPERT, C. et al. A Semi-persistent Clustering Technique for VLSI Circuit Placement. In: **Proc. International Symposium on Physical Design**. San Francisco: ACM, 2005. p. 200–207.

ALPERT, C. J.; HUANG, J.-H.; KAHNG, A. B. Multilevel Circuit Partitioning. In: **Proc. Design Automation Conference**. Anaheim: ACM, 1997. p. 530–533.

ALPERT, C. J.; KAHNG, A. B. Recent Directions in Netlist Partitioning: A Survey. **Integration, the VLSI Journal**, v. 19, p. 1–81, 1995.

ALPERT, C. J.; KAHNG, A. B.; YAO, S.-Z. Spectral Partitioning With Multiple Eigenvectors. **Elsevier Discrete Applied Mathematics**, v. 90, n. 1, p. 3–26, 1999.

BAEK, S.-H. et al. Ultra-High Density Standard Cell Library Using Multi-Height Cell Structure. **SPIE Smart Structures, Devices, and Systems**, p. 70–77, 2008.

BARNES, E. R. An algorithm for partitioning the nodes of a graph. In: **Proc. Conference on Decision and Control including the Symposium on Adaptive Processes**. San Diego: IEEE, 1981. p. 303–304.

BERG, M. d. et al. **Computational Geometry: Algorithms and Applications**. Santa Clara: Springer, 1997.

BLONDEL, V. D. et al. Fast Unfolding of Communities in Large Networks. **SISSA/IOP Journal of Statistical Mechanics: Theory and Experiment**, v. 10, p. 1–12, 2008.

BLUTMAN, K. et al. Floorplan and Placement Methodology for Improved Energy Reduction in Stacked Power-Domain Design. In: **Proc. Asia and South Pacific Design Automation Conference**. Chiba: IEEE, 2017. p. 444–449.

BRENNER, U. BonnPlace Legalization: Minimizing Movement by Iterative Augmentation. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 32, n. 8, p. 1215–1227, 2013.

BRENNER, U.; STRUZYNA, M.; VYGEN, J. BonnPlace: Placement of Leading-Edge Chips by Advanced Combinatorial Algorithms. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 27, n. 9, p. 1607–1620, 2008.

CALDWELL, A. E.; KAHNG, A. B.; MARKOV, I. L. Optimal Partitioners and End-Case Placers for Standard-Cell Layout. In: **Proc. International Symposium on Physical Design**. Monterey, California, USA: ACM, 1999. p. 90–96.

CALDWELL, A. E.; KAHNG, A. B.; MARKOV, I. L. Improved Algorithms for Hypergraph Bipartitioning. In: **Proc. Asia and South Pacific Design Automation Conference**. Yokohama: IEEE, 2000. p. 661–666.

CALIńSKI, T.; HARABASZ, J. A Dendrite Method for Cluster Analysis. **Journal of Communications in Statistics – Theory and Methods**, v. 3, n. 1, p. 1–27, 1974.

CHAN, T. F.; CONG, J.; RADKE, E. A rigorous framework for convergent net weighting schemes in timing-driven placement. In: **Proc. International Conference on Computer-Aided Design**. San Jose: IEEE, 2009. p. 288–294.

CHEN, T. et al. NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 27, n. 7, p. 1228–1240, 2008.

CHENG, C. et al. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 38, n. 9, p. 1717–1730, 2019.

CHENG, C. K.; KUH, E. S. Module Placement Based on Resistive Network Optimization. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 3, n. 3, p. 218–225, 1984.

CHUNG, F.; LU, L. Connected Components in Random Graphs with Given Expected Degree Sequences. **Springer Annals of Combinatorics**, v. 6, n. 2, p. 122–145, 2002.

CLAUSET, A.; NEWMAN, M. E. J.; MOORE, C. Finding Community Structure in Very Large Networks. **APS Physical Review E**, v. 70, n. 6, p. 66111–66116, 2004.

DARAV, N. K. et al. Eh?Legalizer: A High Performance Standard-Cell Legalizer Observing Technology Constraints. **ACM Transactions on Design Automation of Electronic Systems**, v. 23, n. 4, p. 43:1–43:25, 2018.

DARPA. **DARPA Rolls Out Electronics Resurgence Initiative**. 2018. Available from Internet: <https://www.darpa.mil/news-events/2017-09-13>. Accessed in: 11 Dec. 2018.

DAVIES, D. L.; BOULDIN, D. W. A Cluster Separation Measure. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 1, n. 2, p. 224–227, 1979.

DO, S.; WOO, M.; KANG, S. Fence-Region-Aware Mixed-Height Standard Cell Legalization. In: **Proc. Great Lakes Symposium on VLSI**. Tysons Corner: ACM, 2019. p. 259–262.

EDELSBRUNNER, H.; KIRKPATRICK, D.; SEIDEL, R. On the Shape of a Set of Points in the Plane. **IEEE Transactions on Information Theory**, v. 29, n. 4, p. 551–559, 1983.

EVERITT, B.; SKRONDAL, A. **The Cambridge dictionary of statistics**. [S.l.]: Cambridge University Press Cambridge, 2002.

FENG, Z. et al. A Novel Similarity-Based Modularity Function for Graph Partitioning. In: **Proc. International Conference on Data Warehousing and Knowledge Discovery**. Regensburg: Springer, 2007. p. 385–396.

FIDUCCIA, C. M.; MATTHEYSES, R. M. A Linear-Time Heuristic for Improving Network Partitions. In: **Proc. Design Automation Conference**. Las Vegas: IEEE, 1982. p. 175–181.

FLACH, G. et al. Drive Strength Aware Cell Movement Techniques for Timing Driven Placement. In: **Proceedings of the 2016 on International Symposium on Physical Design**. [S.l.]: Association for Computing Machinery, 2016. p. 73–80.

FLACH, G. et al. Rsyn: An Extensible Physical Synthesis Framework. In: **Proc. International Symposium on Physical Design**. Portland: ACM, 2017. p. 33–40.

FOGAçA, M. et al. Finding Placement-Relevant Clusters With Fast Modularity-Based Clustering. In: **Proc. Asia and South Pacific Design Automation Conference**. Tokyo: ACM, 2019. p. 569–576.

FORTUNATO, S.; HRIC, D. Community Detection in Networks: A User Guide. **Elsevier Physics Reports**, v. 659, p. 1–44, 2016.

FRANKLE, J.; KARP, R. M. Circuit Placements and Costs Bounds by Eigenvector Decomposition. In: **Proc. International Conference on Computer-Aided Design**. Santa Clara: IEEE, 1988. p. 414–417.

GEORGE, K.; VIPIN, K. **hMETIS – A Hypergraph Partitioning Package**. 1998. Available from Internet: <http://glaros.dtc.umn.edu/gkhome/fetch/sw/hmetis/manual.pdf>. Accessed in: 24 Mar. 2019.

GIRVAN, M.; NEWMAN, M. E. J. Community Structure in Social and Biological Networks. **Proc. National Academy of Sciences of the United States of America**, v. 99, p. 7821–7826, 2002.

HAGEN, L. W.; J.-H., H.; KAHNG, A. B. On Implementation Choices for Iterative Improvement Partitioning Algorithms. In: **Proc. European Design Automation Conference**. Brighton: IEEE, 1995. p. 144–149.

HALL, K. M. An r-Dimensional Quadratic Placement Algorithm. **Management Science**, v. 17, n. 3, p. 219–229, 1970.

HAN, K.; KAHNG, A. B.; LI, J. Optimal Generalized H-Tree Topology and Buffering for High-Performance and Low-Power Clock Distribution. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, in press.

HEO, S. i. et al. Detailed Placement for IR Drop Mitigation by Power Staple Insertion in Sub-10nm VLSI. In: **Proc. Design, Automation & Test in Europe & Conference**. Florence: IEEE, 2019. p. 830–835.

HEO, S. i. et al. Diffusion break-aware leakage power optimization and detailed placement in sub-10nm vlsi. In: **Proc. Asia and South Pacific Design Automation Conference**. Tokyo: ACM, 2019. p. 550–556.

HEUER, T.; SCHLAG, S. Improving Coarsening Schemes for Hypergraph Partitioning by Exploiting Community Structure. In: **Proc. International Symposium on Experimental Algorithms**. Kalamata: Springer, 2017. p. 21:1–21:19.

HILL, D. **Method and System for High Speed Detailed Placement of Cells Within an Integrated Circuit Design**. 2002. US Patent 6,370,673.

HSU, M.; CHANG, Y.; BALABANOV, V. TSV-Aware Analytical Placement for 3D IC Designs. In: **Proc. Design Automation Conference**. New York: IEEE, 2011. p. 664–669.

HSU, M. et al. NTUplace4h: A Novel Routability-Driven Placement Algorithm for Hierarchical Mixed-Size Circuit Designs. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 33, n. 12, p. 1914–1927, 2014.

HUANG, D. J.-H.; KAHNG, A. B. When Clusters Meet Partitions: New Density-Based Methods for circuit decomposition. In: **Proc. European Design and Test Conference**. Paris: IEEE, 1995. p. 60–64.

IHLER, E.; WAGNER, D.; WAGNER, F. Modeling Hypergraphs by Graphs with the Same Mincut Properties. **Elsevier Information Processing Letters**, v. 45, n. 4, 1993.

ITRS. **International Technology Roadmap for Semiconductors**. 2015. Available from Internet: <http://www.itrs2.net/>. Accessed in: 6 May 2020.

JINDAL, T. et al. Detecting Tangled Logic Structures in VLSI Netlists. In: **Proc. Design Automation Conference**. Anaheim: IEEE, 2010. p. 603–608.

JUNG, J. et al. OWARU: Free Space-Aware Timing-Driven Incremental Placement With Critical Path Smoothing. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 37, n. 9, p. 1825–1838, 2018.

KAHNG, A. B. INVITED: Reducing Time and Effort in IC Implementation: A Roadmap of Challenges and Solutions. In: **Proc. Design Automation Conference**. San Francisco: ACM, 2018. p. 1–6.

KAHNG, A. B.; LI, J.; WANG, L. Improved Flop Tray-based Design Implementation for Power Reduction. In: **Proc. International Conference on Computer-Aided Design**. Austin: IEEE/ACM, 2016. p. 20:1–20:8.

KAHNG, A. B. et al. **VLSI Physical Design: From Graph Partitioning to Timing Closure**. New York: Springer, 2011.

KAHNG, A. B.; WANG, Q. An Analytic Placer for Mixed-Size Placement and Timing-Driven Placement. In: **Proc. International Conference on Computer Aided Design**. San Jose: IEEE, 2004. p. 565–572.

KAHNG, A. B.; WANG, Q. A Faster Implementation of APlace. In: **Proc. International Symposium on Physical Design**. San Jose: ACM, 2006. p. 218–220.

KAMIńSKI, B. et al. Clustering Via Hypergraph Modularity. **Public Library of Science One**, v. 14, n. 11, 2019.

KARYPIS, G. et al. Multilevel Hypergraph Partitioning: Application in VLSI Domain. In: **Proc. Design Automation Conference**. Anaheim: ACM, 1997. p. 526–529.

KARYPIS, G.; KUMAR, V. Multilevel K-way Hypergraph Partitioning. In: **Proc. Design Automation Conference**. New Orleans: ACM, 1999. p. 343–348.

KERNIGHAN, B. W.; LIN, S. An Efficient Heuristic Procedure for Partitioning Graphs. **The Bell System Technical Journal**, v. 49, n. 2, p. 291–307, 1970.

KIM, M.-C.; LEE, D.-J.; MARKOV, I. L. SimPL: An Algorithm for Placing VLSI Circuits. **Communications of the ACM**, v. 56, n. 6, p. 105–113, 2013.

KIM, M.-C. et al. MAPLE: Multilevel Adaptive Placement for Mixed-Size Designs. In: **Proc. International Symposium on International Symposium on Physical Design**. Napa: ACM, 2012. p. 193–200.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by Simulated Annealing. **Science**, v. 220, n. 4598, p. 671–680, 1983.

KLEINHANS, J. M. et al. GORDIAN: VLSI Placement by Quadratic Programming and Slicing Otimization. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 10, n. 3, p. 356–365, 1991.

KUMAR, T. et al. Hypergraph Clustering: A Modularity Maximization Approach. **arXiv**, 2018.

KUMAR, T. et al. A New Measure of Modularity in Hypergraphs: Theoretical Insights and Implications for Effective Clustering. In: **Proc. International Conference on Complex Networks and Their Applications**. Lisbon: Springer, 2019. p. 286–297.

LENGAUER, T. **Combinatorial Algorithms for Integrated Circuit Layout**. New York: Wiley & Sons, Inc., 1990.

LI, J.; BEHJAT, L.; KENNINGS, A. Net Cluster: A Net-Reduction-Based Clustering Preprocessing Algorithm for Partitioning and Placement. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 26, n. 4, p. 669–679, 2007.

LIU, W. et al. NCTU-GR 2.0: Multithreaded Collision-Aware Global Routing With Bounded-Length Maze Routing. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 32, n. 5, p. 709–722, 2013.

LU, J. et al. ePlace: Electrostatics-Based Placement Using Fast Fourier Transform and Nesterov's Method. **ACM Transactions on Design Automation of Electronic Systems**, v. 20, n. 2, p. 17:1–17:34, 2015.

MANTIK, S. et al. ISPD 2018 Initial Detailed Routing Contest and Benchmarks. In: **Proc. International Symposium on Physical Design**. Monterey: ACM, 2018. p. 140–143.

MARKOV, I. L.; HU, J.; KIM, M. Progress and Challenges in VLSI Placement Research. **Proceedings of the IEEE**, v. 103, n. 11, p. 1985–2003, 2015.

MONTEIRO, J.; JOHANN, M.; BEHJAT, L. An optimized cost flow algorithm to spread cells in detailed placement. **ACM Transactions on Design Automation of Electronic Systems**, v. 24, n. 3, 2019.

NAM, G.-J. ISPD 2006 Placement Contest: Benchmark Suite and Results. In: **Proc. International Symposium on Physical Design**. San Jose: ACM, 2006. p. 167.

NAM, G.-J. et al. The ISPD2005 Placement Contest and Benchmark Suite. In: **Proc. International Symposium on Physical Design**. San Francisco: ACM, 2005. p. 216–220.

NAYLOR, W. C.; DONELLY, R.; SHA, L. **Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer**. 2001. US Patent 6,301,693.

NEUBAUER, N.; OBERMAYER, K. Towards Community Detection in k-partite k-uniform Hypergraphs. **Proc. Workshop on Analyzing Networks and Learning with Graphs**, Whistler, p. 1–9, 2009.

NEUBAUER, N.; OBERMAYER, K. Community Detection in Tagging-Induced Hypergraphs. **Proc. Workshop on Information in Networks**, New York, p. 24–25, 2010.

NEWMAN, M. E.; GIRVAN, M. Finding and Evaluating Community Structure in Networks. **APS Physical review E**, v. 69, p. 1–15, 2004.

NEWMAN, M. E. J. Fast Algorithm for Detecting Community Structure in Networks. **APS Physical Review E**, v. 69, n. 6, p. 66133–66137, 2004.

NEWMAN, M. E. J. Finding Community Structure in Networks Using the Eigenvectors of Matrices. **APS Physical Review E**, v. 74, n. 3, p. 36104–36122, 2006.

OLOFSSON, A. **Silicon Compilers - Version 2.0**. 2018. Available from Internet: <http://www.ispd.cc/slides/2018/k2.pdf>. Accessed in: 11 Dec. 2018.

OPENCORES. **OpenCores: Open Source IP-Cores**. 1999. Available from Internet: <http://www.opencores.org>. Accessed in: 11 Dec. 2018.

OPENDP. **Open Source Detailed Placement engine**. 2020. Available from Internet: <https://github.com/The-OpenROAD-Project/OpenDP/tree/0.1.0>. Accessed in: 12 Apr. 2020.

PONS, P.; LATAPY, M. Computing Communities in Large Networks Using Random Walks. **Journal of Graph Algorithms and Applications**, v. 10, n. 2, p. 191–218, 2006.

PUGET, J. C. et al. Jezz: An effective legalization algorithm for minimum displacement. In: **Proc. Symposium on Integrated Circuits and Systems Design**. Salvador: IEEE, 2015. p. 1–5.

RADICCHI, F. et al. Defining and Identifying Communities in Networks. **Proc. National Academy of Sciences of the United States**, v. 101, n. 9, p. 2658–2663, 2004.

RAJARAMAN, R.; WONG, D. F. Optimum clustering for delay minimization. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 14, n. 12, p. 1490–1495, 1995.

RAKAI, L. et al. A New Length-Based Algebraic Multigrid Clustering Algorithm. **Hindawi VLSI Design**, p. 395260, 2012.

RENTCON. **RentCon: Rent Parameter Evaluation Using Different Methods**. 2008. Available from Internet: <https://vlsicad.ucsd.edu/WLD/index.html>. Accessed in: 11 Dec. 2018.

REPLACE. **RePlAce: Advancing Solution Quality and Routability Validation in Global Placement**. 2018. Available from Internet: <https://github.com/The-OpenROAD-Project/RePlAce/tree/1.1.1>. Accessed in: 11 Dec. 2018.

ROUSSEEUW, P. J. Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis. **Journal of Computational and Applied Mathematics**, v. 20, p. 53–65, 1987.

ROY, J. A. et al. Min-Cut Floorplacement. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 25, n. 7, p. 1313–1326, 2006.

RSYN. **Rsyn: An Extensible Physical Synthesis Framework**. 2016. Available from Internet: <https://github.com/RsynTeam/rsyn-x>. Accessed in: 11 Dec. 2018.

SCIKIT LEARN. **scikit-learn**. 2020. Available from Internet: <https://scikit-learn.org/stable/modules/clustering.html>. Accessed in: 12 Apr. 2020.

SHELAR, R. S. An Efficent Clustering Algorithm for Low Power Clock Tree Synthesis. In: **Proc. International Symposium on Physical Design**. Austin: ACM, 2007. p. 181–188.

SHIOKAWA, H.; FUJIWARA, Y.; ONIZUKA, M. SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-scale Graphs. **Proc. VLDB Endowment**, v. 8, n. 11, p. 1178–1189, 2015.

SHIOKAWA, H.; ONIZUKA, M. Scalable Graph Clustering and Its Applications. In: ____. **Encyclopedia of Social Network Analysis and Mining**. New York: Springer, 2017.

SPINDLER, P.; SCHLICHTMANN, U.; JOHANNES, F. M. Abacus: Fast legalization of standard cell circuits with minimal movement. In: **Proc. International Symposium on Physical Design**. Portland: ACM, 2008. p. 47–53.

SWARTZ, W.; SECHEN, C. Timing Driven Placement for Large Standard Cell Circuits. In: **Proc. Design Automation Conference**. New York: IEEE/ACM, 1995. p. 211–215.

SWARTZ, W. P. **Automatic Layout of Analog and Digital Mixed Macro/Standard Cell Integrated Circuits**. Thesis (PhD) — Yale University, New Haven, CT, USA, 1993.

SZUFEL. **Clustering Via Hypergraph Modularity**. 2020. Available from Internet: <https://gist.github.com/pszufe/>. Accessed in: 1 May 2020.

TAGHAVI, T.; YANG, X.; CHOI, B.-K. Dragon2005: Large-Scale Mixed-Size Placement tool. In: **Proc. International Symposium on Physical Design**. San Francisco: ACM, 2005. p. 245–247.

TIMBERWOLF. **TimberWolf Systems, Inc.** 2014. Available from Internet: <http://www.twolf.com/>. Accessed in: 7 May 2020.

TSAY, R.-S.; KUH, E. S. A Unified Approach to Partitioning and Placement. **IEEE Transactions on Circuits and Systems**, v. 38, p. 521–533, 1991.

TSAY, R.-S.; KUH, E. S.; HSU, C.-P. PROUD: A Sea-of-Gates Placement Algorithm. **IEEE Design & Test of Computers**, v. 5, n. 6, p. 44–56, 1988.

VISWANATHAN, N.; CHU, C. C.-N. FastPlace: Efficient Analytical Placement Using Cell Shifting, Iterative Local Refinement and a Hybrid Net Model. In: **Proc. International Symposium on Physical Design**. Phoenix: ACM, 2004. p. 26–33.

WAKITA, K.; TSURUMI, T. Finding Community Structure in Mega-scale Social Networks. In: **Proc. International Conference on World Wide Web**. Banff: ACM, 2007. p. 1275–1276.

WIERZCHON, S.; KLOPOTEK, M. **Modern Algorithms of Cluster Analysis**. New York: Springer, 2018.

WU, F.; HUBERMAN, B. A. Finding Communities in Linear Time: a Physics Approach. **The European Physical Journal B**, v. 38, n. 2, p. 331–338, 2004.

XU, X. et al. SCAN: A Structural Clustering Algorithm for Networks. In: **Proc. International Conference on Knowledge Discovery and Data Mining**. San Jose: ACM, 2007. p. 824–833.

YAN, J. Z.; CHU, C.; MAK, W.-K. SafeChoice: A Novel Clustering Algorithm for Wirelength-Driven Placement. In: **Proc. International Symposium on Physical Design**. New York: ACM, 2010. p. 185—-192.

YAN, J. Z.; VISWANATHAN, N.; CHU, C. Handling Complexities in Modern Large-Scale Mixed-Size Placement. In: **Proc. Design Automation Conference**. San Francisco: ACM, 2009. p. 436—-441.

YANG, H.; WONG, D. F. Efficient Network Flow Based Min-cut Balanced Partitioning. In: **Proc. International Conference on Computer-Aided Design**. San Jose: ACM, 1994. p. 50–55.

ZACHARY, W. W. An Information Flow Model for Conflict and Fission in Small Groups. **Journal of Anthropological Research**, v. 33, n. 4, p. 452–473, 1977.

# APPENDIX A — PUBLICATIONS, AWARDS AND INTERNSHIPS

## A.1 Publications

**Journals:**

1. M. Danigno, M. Fogaça, E. Monteiro, J. Ferreira, A. Oliveira, R. Reis and P. Butzen, “Algorithms for Access Point Selection at Pre-Routing Stage”, *Journal of Integrated Circuits and Systems*, to appear late 2020.
2. M. Fogaça, A. B. Kahng, E. Monteiro, R. Reis, L. Wang and M. Woo, “On the Superiority of Modularity-Based Clustering for Determining Placement-Relevant Clusters”, *Integration: The VLSI Journal* 74 (2020), pp. 32–44.

**Conferences:**

1. V. Bandeira, M. Fogaça, E. M. Monteiro, I. Oliveira, M. Woo, R. Reis, “Fast and Scalable I/O Pin Assignment with Divide-and-Conquer and Hungarian Matching”, *Proc. International New Circuits and Systems Conference*, to appear June 2020.
2. M. Danigno, P. F. Butzen, J. Ferreira, A. Oliveira, E. Monteiro, M. Fogaça and R. A. da L. Reis, “Proposal and Evaluation of Pin Access Algorithms for Detailed Routing”, *Proc. International Conference on Electronics Circuits and Systems*, 2019, pp. 602-605.
3. T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem, G. Pradipta, S. Reda, M. Saligane, S. S. Sapatnekar, C. Sechen, M. Shalan, W. Swartz, L. Wang, Z. Wang, M. Woo and B. Xu, “Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project”, *Proc. Design Automation Conference*, 2019, pp. 76:1–76:4. (Invited Paper)
4. T. Ajayi, D. Blaauw, T.-B. Chan, C.-K. Cheng, V. A. Chhabria, D. K. Choo, M. Coltella, S. Dobre, R. Dreslinski, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Li, Z. Liang, U. Mallappa, P. Penzes, G. Pradipta, S. Reda, A. Rovinski, K. Samadi, S. S. Sapatnekar, L. Saul, C. Sechen, V. Srinivas, W. Swartz, D. Sylvester, D. Urquhart, L. Wang, M. Woo and B. Xu, “OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain”, *Proc. Government Microcircuit Applications and Critical Technology Conference*, 2019, pp. 1105–1110.

5. M. Fogaça, A. B. Kahng, R. Reis and L. Wang, "Finding Placement-Relevant Clusters With Fast Modularity-Based Clustering", *Proc. Asia and South Pacific Design Automation Conference*, 2019, pp. 569–576.
6. G. Flach, M. Fogaça, J. Monteiro, M. O. Johann and R. A. da L. Reis, "Rsyn: An Extensible Physical Synthesis Framework", *Proc. International Symposium on Physical Design*, 2017, pp. 33–40.
7. M. Fogaça, G. Flach, J. Monteiro, M. O. Johann, R. Reis, "Quadratic timing objectives for incremental timing-driven placement optimization", *Proc. International Conference on Electronics, Circuits and Systems*, 2016, pp. 620–625.
8. M. Fogaça, G. Flach, J. Monteiro, M. O. Johann, R. Reis, "Drive Strength Aware Cell Movement Techniques for Timing Driven Placement", *Proc. International Symposium on Physical Design*, 2016, pp. 73-80.
9. J. Monteiro, N. K. Darav, G. Flach, M. Fogaça, R. A. da L. Reis, A. A. Kennings, M. O. Johann, L. Behjat, "Routing-Aware Incremental Timing-Driven Placement", *Proc. Annual Symposium on VLSI*, 2016, pp. 290–295.
10. G. Flach, J. Monteiro, M. Fogaça, J. C. Puget, P. F. Butzen, M. O. Johann, R. A. da L. Reis, "An Incremental Timing-Driven flow using quadratic formulation for detailed placement", *Proc. International Conference on Very Large Scale Integration*, 2015, pp. 1–6.

## A.2 Awards

- ICCAD 2015 CAD Contest on Incremental Timing-driven Placement – 1st place award.
- ICCAD 2016 CAD Contest on Incremental Timing-driven Placement – 2nd place award.
- ISPD 2018 Contest on Initial Detailed Routing – 4th place award.

## A.3 Internships

- The author has joined the Synopsys, Inc. (Mountain View, CA) Design Compiler team for a six months internship as a Technical Intern, from August 2017 to February 2018. In the course of his internship, the author has studied algorithms for timing and area optimization during physically-aware synthesis.
- The author has spent six months as a Visiting PhD Student in Prof. Andrew Kahng's research group at the University of California, San Diego (La Jolla, CA). Prof. Kahng's lab has one of the most traditional and influential VLSI CAD research groups. During his visit, the author has had the opportunity to get in contact with cutting-edge research and to know some industry and academic experts.
- The author has had a six-month internship as a Software Engineer Intern at Cadence Design Systems, Inc. (Austin, TX) from February 2019 to August 2019. During his internship, the author has joined the Early Global Route team and has studied and developed algorithms for global routing and track assignment.