Hypergraph Clustering with Path-Length Awareness

Julien Rodriguez, Francois Galea, François Pellegrini and Lilia Zaourar

May 13, 2024

Hypergraph Clustering With Path-Length Awareness

Julien Rodriguez[1,2][0000−0003−3583−0859], François Galea[3][0000−0002−1594−152X], François Pellegrini[2][0000−0003−3983−6289], and Lilia Zaourar[3][0000−0002−6660−4347]

[1] University of Montpellier, LIRMM, CNRS, Montpellier, France
`name.surname@lirmm.fr`
[2] Université de Bordeaux & INRIA, F-33405, Talence, France
`francois.pellegrini@u-bordeaux.fr`
and Université Paris-Saclay, CEA List, F-91120, Palaiseau, France
`name.surname@cea.fr`

Abstract. Electronic design automation toolchains require solving various circuit manipulation problems, such as floor planning, placement and routing. These circuits may be implemented using either Very Large-Scale Integration (VLSI) or Field Programmable Gate Arrays (FPGAs). However, with the ever-increasing size of circuits, now up to billions of gates, straightforward approaches to these problems do not scale well. A possible approach to reduce circuit complexity is to cluster circuits, to reduce their apparent size for critical processing operations, while preserving their topological properties (e.g., connection locality). Several models have been proposed to tackle the clustering problem [9]. In this work, we consider the problem of clustering combinatorial circuits, without cell replication. Our main objective is to minimize the overall delay, which conditions the circuit operating frequency. We propose a dedicated clustering algorithm based on binary search and study and improve the existing parameterized approximation ratio from M^2+M [9] (with M being the maximum size of each cluster) to M under specific hypothesis. We present an extension of the weighting schemes introduced in [23] to model path length more accurately. This weighting scheme is combined with clustering methods based on a recursive matching algorithm. We evaluate and compare our approximation algorithm and recursive matching on several circuit instances and we obtain better results for a large number of instances with our algorithm than recursive matching.

Keywords: Clustering · Hypergraph · Digital electronic circuit.

1 Introduction

Our research interest concerns circuit prototyping on multi-FPGA platforms, to map efficiently circuits that are too big to fit into a single FPGA. In this case, it is necessary to partition the circuit into several parts that have to be placed on the different components (*i.e.*, FPGAs) of the platform. Traditional partitioning tools use a classic multilevel scheme consisting of three phases: *coarsening*, *initial partitioning*, and *refinement* [14]. The coarsening phase uses recursively a clustering method to transform the circuit model, a hypergraph, into a smaller one. During the second phase, an initial partitioning is computed on the smallest coarsened hypergraph. Finally, in the third phase, for each coarsening level, the solution for the coarser level is extended to the finer level, and then refined using a local refinement algorithm. The clustering algorithms presented in this paper concern the first step of the multilevel framework described above.

In addition, the increasing size of high-end circuits makes them more challenging to handle. Combinatorial circuit clustering helps reduce the size of circuits, making them easier to manage. Several such clustering algorithms already exist in the literature; Z. Donovan [7] defines two classes to categorize them: CA and CN. CA algorithms aim at finding clustering of circuits that minimize signal propagation delay, while allowing logic replication (see e.g. [18–20,22]). On the opposite, CN algorithms compute clustering that minimize circuit delay without cell replication (see e.g. [9,13]).

An optimal solution for the CA problem can be computed in polynomial time. However, unbounded replication can yield very large circuits [18, 22]. In the context of circuit placement on multi-FPGA platforms, the number of resources is limited. Thus, it is necessary to perform either a disjoint clustering, or to bound the number of replications. In this work, we focus on the CN problem.

The remainder of the paper is organized as follows. Section 2 presents the notations, definitions, and previous works on CN. We introduce our weighting scheme for clustering in Sections 3. In Section 4, we introduce our clustering algorithms and complexity results. Our experiments are outlined in Section 5. We conclude and give perspectives in Section 6.

2 Preliminaries

Combinatorial circuits are often modeled as directed hypergraphs, *i.e.*, a generalization of directed graphs in which the notion of arc is extended to that of hyperarc. A hyperarc can connect one or more source vertices to one or more sink vertices. In a combinatorial circuit, a net (or wire) can connect more than two gates, and there is usually a single signal source per net. Hence, we consider only hyperarcs that comprise a single source vertex. Several works rely on a graph model to represent combinatorial circuits, such as: [9, 10, 13, 19, 20, 22]. While this model is relevant to represent dependencies between outputs and inputs of gates, it is not adequate to evaluate the number of cut wires and does not model critical path [23]. Consequently, in this work, we will represent circuits using a Directed Acyclic Hypergraph (DAH) model [23] to measure and control the size of the cut, while also relying on an underlying graph model to compute clustering scores between gates and solve the CN problem without considering the number of cut hyperarcs.

2.1 Notations and definitions

Let $H \overset{\text{def}}{=} (V, A, W_V, W_a)$ be a directed hypergraph, defined by a set of vertices V and a set of hyperarcs A, with a vertex weight function $W_V : V \to \mathbb{R}^+$ and a hyperarc weight function $W_a : A \to \mathbb{R}^+$. Every hyperarc $a \in A$ is a subset of vertex set V: $a \subseteq V$. Let $s^-(a)$ be the source vertex set of hyperarc a, and $s^+(a)$ its sink (destination) vertex set. We consider each hyperarc has a single source, so $\forall a, |s^-(a)| = 1$. As hyperarcs connect vertices, let $\Gamma(v)$ be the set of neighbor vertices of vertex v, and $\Gamma^-(v) \subseteq \Gamma(v)$ and $\Gamma^+(v) \subseteq \Gamma(v)$ the sets of its inbound and outbound neighbors, respectively.

In the model we propose, hypergraphs that model circuits will be represented as sets of interconnected DAHs, according to a red-black vertex coloring scheme. Red vertices correspond to I/O (Inputs/Outputs) ports and registers, and black vertices to combinatorial circuit components. Let $V^R \subset V$ and $V^B \subset V$ be the red and black vertex subsets of V, such that $V^R \cap V^B = \emptyset$ and $V^R \cup V^B = V$. A hypergraph or sub-hypergraph H is a DAH iff its red vertices $v_R \in V^R$ are either only sources or sinks (*i.e.*, $\Gamma^-(v_R) = \emptyset$ or $\Gamma^+(v_R) = \emptyset$), and no cycle path connects a vertex to itself.

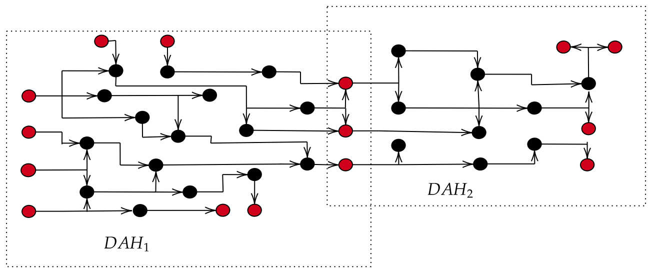

Fig. 1: Hypergraph composed of two 2 DAHs.

Using this definition, we can represent circuit hypergraphs as *red-black hypergraphs*, *i.e.*, sets of DAHs that share some of their red vertices, as illustrated in Figure 1. Let $\mathbf{H}(\mathbf{V}, \mathbf{A}) \overset{\text{def}}{=} \{H_i, i \in \{1 \ldots n\}\}$ be a red-black hypergraph, such that every H_i is a DAH and an edge-induced sub-hypergraph of $\mathbf{H}$. Consequently, $\mathbf{V} = \bigcup_i V_i$, $\mathbf{A} = \bigcup_i A_i$, $\mathbf{V}^R = \bigcup_i V_i^R$, and $\mathbf{V}^B = \bigcup_i V_i^B$. Moreover, $\forall i, j$ with $i \neq j$, if $V_{i,j} = V_i \cap V_j \neq \emptyset$, then H_i and H_j share source and/or sink vertices, *i.e.*, $V_{i,j} \subset \mathbf{V}^R$.

In this model, the paths in $\mathbf{H}$ to consider when addressing the objective of minimizing path-cost degradation during partitioning are only the paths interconnecting red vertices, as these red-red paths represent register-to-register paths in digital electronic circuits. Since only red vertices are shared between DAHs in $\mathbf{H}$, red-red paths only exist within a single DAH and can never span across several DAHs.

Let us define $\mathbf{P}$ as the set of red-red paths in $\mathbf{H}$, such that $\mathbf{P} \overset{\text{def}}{=} \{p | p \text{ is a path in } H \in \mathbf{H}\}$. From these paths and a function $d_{\max}(u, v)$ which computes the maximum distance between vertices u and v of some DAH H, we can define the longest path distance for H as: $d_{\max}(H) \overset{\text{def}}{=} \max(d_{\max}(u, v) | u, v \in H)$ and, by extension, for $\mathbf{H}$, as: $d_{\max}(\mathbf{H}) \overset{\text{def}}{=} \max(d_{\max}(H) | H \in \mathbf{H})$.

A clustering $\mathcal{C}$ of $\mathbf{H}$ is a splitting of $\mathbf{V}$ into vertex subsets $\mathcal{C}_i$, called clusters, such that:

(i) all clusters $\mathcal{C}_i$, given a capacity bound M, respect the capacity constraint: $\sum_{v \in \mathcal{C}_i} W_V(v) \leq M$;
(ii) all clusters are pairwise disjoint: $\forall i \neq j, \mathcal{C}_i \cap \mathcal{C}_j = \emptyset$; and
(iii) the union of all clusters is equal to $\mathbf{V}$: $\bigcup_i \mathcal{C}_i = \mathbf{V}$.

For a given clustering $\mathcal{C}$ of $\mathbf{H}$, the connectivity $\lambda_{\mathcal{C}}(a)$ of some hyperarc $a \in A$ is the number of clusters connected by a. If $\lambda_{\mathcal{C}}(a) > 1$, then a is said to be cut; otherwise, it is entirely contained within a single cluster and is not cut. The cut of clustering $\mathcal{C}$ is the set $\omega(\mathcal{C})$ of cut hyperarcs, *i.e.*, $\omega(\mathcal{C}) \overset{\text{def}}{=} \{a \in A, \lambda_{\mathcal{C}}(a) > 1\}$. The cut size is defined as $f_c \overset{\text{def}}{=} \sum_{a \in \omega(\mathcal{C})} W_a(a)$. If all hyperarcs have the same weight (equal to 1), the cut size equals to $|\omega(\mathcal{C})|$. Another cut metric used by some partitioning tools to measure the quality of clustering is called *connectivity-minus-one* [3]. The connectivity-minus-one cost function f_λ of some clustered hypergraph $\mathbf{H}^{\mathcal{C}}$ is defined as: $f_\lambda = \sum_{a \in \mathbf{A}} (\lambda_{\mathcal{C}}(a) - 1) \times W_a(a)$.

2.2 Related works

Clustering algorithms are essential for partitioning large graphs and hypergraphs. Since its inception, the multilevel scheme has been the most efficient and widely used method for clustering large hypergraphs. Indeed, all modern partitioning tools implement methods based on the multi-level scheme. At their core, clustering algorithms reduce the size of the hypergraph by contracting vertices according to a matching function. Several functions have been developed to evaluate the quality of vertex merging.

Hypergraph clustering B. Hendrickson and R. Leland [11] propose a *randomized matching* algorithm. The algorithm randomly traverses the vertices, and, if the visited vertex is not matched, a neighbor is randomly selected. The random aspect of the algorithm allows it to compute a solution quickly, making it practical for large instances. Another clustering algorithm used for the coarsening phase is called a *heavy edge matching.* Heavy edge matching is an algorithm that randomly visits the graph vertices. If the visited vertex is not already coupled, the algorithm selects a not yet selected neighbor connected by an edge of heaviest weight. G. Karypis and V. Kumar [16] compare the randomized matching and heavy edge matching during algorithms coarsening and conclude that the heavy edge matching algorithm provides partitions with better quality and reduces computation time during the refinement stage. Ü. Çatalyürek and Ç. Aykanat [2] proposed an algorithm based on heavy-connectivity matching, that favors merging vertices with highest connectivity. The *connectivity* metric used is also known as the *inner product.* The *inner product* between two vertices is defined as the number of hyperedges shared by these two vertices. T. Heuer and S. Schlag [12] propose a framework for hypergraph coarsening based on the exploitation of community structures in graphs. Their experimental results show that their coarsening method improves the initial partitioning cutsize as well as the final cutsize. Readers can consult the recent survey published by Ü. Çatalyürek *et al.* [3] for more information on coarsening/clustering methods applied to graphs and hypergraphs.

Circuit clustering when replication is not allowed A. A. Diwan *et al.* [6] address a similar problem, consisting in placing nodes of a memory access structure on disk pages such that a path through several nodes traverses as few disks as possible. Their data structure is a DAG, and their objective is to cluster the DAG such that the number of shared edges per cluster along a path is minimized. The problem is similar to the unweighted case of the CN problem. The authors also present a polynomial-time algorithm for trees, and show that the problem is NP-hard for unweighted DAGs.

More recently, Z. Donovan *et al.* [8–10] have studied the combinatorial circuit clustering problem, with and without vertex replication. They propose several algorithms to solve this problem. The authors present NP-hardness proofs for the DAG circuit clustering problem with minimization of critical path degradation during the clustering step, e.g., minimization of the number of cut penalties along the most critical paths. They propose exact exponential algorithms and approximation algorithms parameterized by cluster size. Further details of this work can be found in Z. Donovan's thesis [7]. Other work on combinatorial circuit clustering to minimize critical path degradation by placing neighboring vertex pairs in different clusters are availables [4, 20].

3 Model and weighting schemes

Criticality is a metric used in [1, 3, 23], to classify the cells of a circuit according to the cost of the combinatorial path they traverse. The *criticality* of a vertex v is equal to the length of the longest path traversing v, $d_{\max}(v)$. In this section, we present the various state-of-the-art weighting schemes used to measure vertex criticality. We propose a new weighting scheme that models more finely the criticality per vertex pair which we use to cluster red-black hypergraphs.

3.1 Model

In the CN problem, an additional constant cost D is added between two neighboring vertices placed in different clusters. Consequently, in our model, the distance between any two vertices u and v (*i.e.*, path cost) may increase during clustering, due to the additional cost that paths have to incur across clusters. Let us recall that the distance function between two vertices in a red-black hypergraph is defined by $d_{\max}(u, v)$ which is equal to the longest path between u and v. Let D be the penalty associated with the distance between two vertices u and v placed in different clusters; let us recall that the distance function for some clustering $\mathcal{C}$, is thus:

$$d^{\mathcal{C}}_{\max}(u, v) \geq d_{\max}(u, v) + D \ . \tag{1}$$

The objective function f_p is defined as the minimization of the longest path of H subject to clustering $\mathcal{C}$: $f_p = \min d_{\max}(H^{\mathcal{C}})$. We extend the definition of the $CN{<}w, M, \Delta{>}$ problem defined by Z. Donovan *et al.* [7] to red-black hypergraphs as follows:

Given a red-black Hypergraph $H = (V, A)$, with a vertex-weight function $w : V \to \mathbb{R}^+$, delay function $d : V \to \mathbb{R}^+$, maximum degree Δ, constant D, and a cluster capacity M, the goal is to partition V into clusters such that: (i) the weight of each cluster is bounded by M; and (ii) the maximum delay-length of any red-red path of H is minimized.

To be consistent with previous definitions of the CN problem, we will keep the Δ parameter, even though we will not use it in the following.

3.2 Weighting schemes

As we exposed in the previous section, the criticality of a vertex v measures the value of the longest path through v. Consequently, criticality seems to be an interesting weighting scheme for measuring the attractiveness between two connected vertices. In this subsection, we present three weighting schemes used to guide a clustering algorithm in the context of circuit partitioning with path cost minimization. First, we present the state of the art in weighting schemes and show their limitations. Then, we present our weighting scheme based on vertex criticality.

Delay propagation Several previous works have proposed metrics for clustering, with the objective of path minimization [1, 3]. For example, C. Ababei *et al.* [1] presented a weighting scheme based on delay propagation to drive *min-cut* tools, *i.e.*, the weight between two vertices u and v is equal to the longest path from a red source vertex to vertices u and v. This method calculates local weights along subpaths from red source vertices to any vertex. Thus, within each DAH, $H = (V, A)$ of H:

$$l(u) = \begin{cases} d(u) & \text{if } \Gamma^-(u) = \emptyset \ , \\ d(u) + \max\limits_{v \in \Gamma^-(u)} l(v) & \text{otherwise} \ . \end{cases} \tag{2}$$

For any vertex $u \in V$, the value $l(u)$ corresponds to the maximum path cost from any source vertex to u. Therefore, the maximum path cost within some DAH will be found at the level of its sink vertices. A calculation on the subpath does not indicate whether their subpath is on the critical path. Cutting anywhere along a path has the same detrimental effect as adding a penalty to the total path cost. It is to alleviate these issues that the next metric have been made.

Delay retro-propagation As critical vertices must be labeled with the same weight, the delay propagation scheme is not adequate. Hence, we have first devised a new weighting scheme based on the back-propagation of path cost:

$$r(u) = \begin{cases} l(u) & \text{if } \Gamma^+(u) = \emptyset \ , \\ \max\limits_{v \in \Gamma^+(u)} r(v) & \text{otherwise} \ . \end{cases} \tag{3}$$

For any $u \in V$, the value $r(u)$ represents an upper bound for the path cost of the longest red-red path traversing u. If u belongs to a path of maximum path cost, then $r(u)$ is equal to that path cost.

This weighting scheme accounts better for the overall impact of the cut along a path because, unlike the previous method, the information is back-propagated to all predecessors. However, it may include heavy vertices that do not belong to a longest red-red path, as shown in Figure 2. To overcome this problem, we need to define the value of the local critical path through each pair of vertices. For this reason, we have proposed a second weighting system in the next subsection.

Refined delay retro-propagation In this subsection, we present a weighting scheme based on the cost of the local critical path. This scheme retro-propagates critical information throughout the red-black hypergraph and avoids non-critical heavy vertices. The l, r, and r^* metrics are used as weighting schemes, as represented in Figure 2.

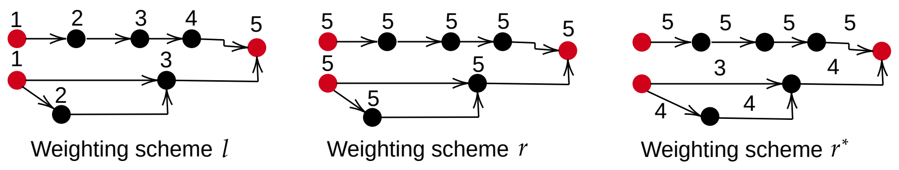

Fig. 2: An example of the three weighting schemes: l [1], r, and r^*. We consider a unit delay for each vertex and a delay equal to zero for each arc. In this example, we can clearly see that scheme l does not effectively weight critical vertices. Scheme r weights critical vertices correctly, but considers non-critical vertices. Scheme r^* is more relevant in its weighting, with respect to our objective.

Let $r^*(u, v)$ be the criticality value between connected vertices u and v, defined as follows:

$$r^*(u, v) = \begin{cases} l(u) & \text{if } u = v \ , \\ r(v) - \left(\max\limits_{u' \in \Gamma^-(v)} l(u') - l(u) \right) & \text{otherwise} \ . \end{cases} \tag{4}$$

$\max_{u' \in \Gamma^-(v)} l(u')$ represents the value of the arcs along the local critical path, which is the longest red-red path traversing v such that, for every other $l(u) < \max_{\forall u' \in \Gamma^-(v)} l(u')$, arcs (u, v) are not in the local critical path. It is a more accurate metric for improving the behavior of clustering algorithms because, in the context of circuit clustering, the aim is to group critical vertices together. If the relationships between vertices reflect correctly criticality, then the clustering algorithm can take advantage of this. An example of the computation of r^* is represented in Figure 3.

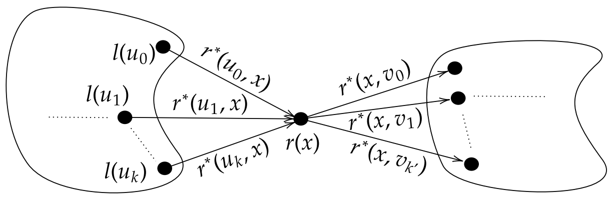

Fig. 3: This figure exhibits an example of schemes and how r^* is computed. $r^*(u_i, x)$ and $r^*(x, v_i)$ are the values of the local critical path between pairs of vertices (u_i, x) and (v_i, x) in this subgraph. There is a maximum value for each $l(u_i)$, $\overline{w}$. For each u_i, $\overline{w} - l(u_i)$ represents the contribution of u_i to the local critical path value $r(x) = \max_{v_i \in \Gamma^+(x)} r^*(x, v_i)$.

For each combinatorial sub-circuit modeled with a DAH, the r^* vertex-vertex criticality relation defines a graph G such that G is a DAG and is simple. Every hyperarc in the DAH defines a group of arcs in the DAG, in which each arc connecting the source vertex to a sink vertex. The cut weight of arcs correspond to the r^* value between source and sink in arcs. Hence, the cut weight of this hyperarc is the maximum of the r^* values between its source and sinks. We will use the criticality relation graph structure G in the next section to do proofs.

4 A parameterized M-approximation algorithm for red-black hypergraph clustering

Since the clustering problem is NP-hard and there is no approximation algorithm with a constant factor in the general case, approximation algorithms have been proposed to provide acceptable solutions in reasonable time, such as the parameterized $M^2 + M$ approximation algorithm presented by Z. Donovan *et al.* [10]. We propose an improved approximation ratio under delay hypothesis and a direct clustering algorithm based on binary search.

4.1 Binary Search Clustering (BSC)

Let $H = (V, A)$ be a DAH, and $p_{\max}$ its critical path. Let ϕ be a feasible minimum cost, $\phi \in [|p_{\max}| \times d, |A| \times D]$, with D the inter-cluster delay and d the intra-cluster delay. Given a fixed value ϕ, we can define a cut capacity for each pair of vertices (u, v) as:

$$\text{cut\_cap}(u, v) = \frac{\max(0, r^*(u, v) - \phi)}{D} . \qquad (5)$$

Suppose the cut capacity between two vertices u and v equals zero. Then, u and v should be placed in the same cluster. As the size of the cluster is constrained by the parameter M, it is possible to know whether some ϕ is unfeasible, by exceeding some cluster size.

Algorithm 1 Binary Search Clustering

```
Require: $H, M, D, d$
Ensure: $\mathcal{C}$ a clustering of $H$
 1: $\overline{\phi} \leftarrow |A| \times D$
 2: $\underline{\phi} \leftarrow |p_{\max}| \times d,\ p_{\max} \in H$
 3: while $\overline{\phi} > \underline{\phi}$ do
 4:     $\phi_{target} \leftarrow \frac{\underline{\phi}+\overline{\phi}}{2}$
 5:         ▷ Compute the cut capacity for every pair $(uv) = \frac{\max(0, r^*(u,v) - \phi_{\text{target}})}{D}$ and for all pair with cut
    capacity equal to zero, place $u$ and $v$ into the same cluster.
 6:     $\mathcal{C} \leftarrow$fusion_cut_cap($H, \phi_{\text{target}}$, max_size)
 7:     if $\max_{c \in \mathcal{C}} |c| \leq M$ then
 8:         $\overline{\phi} \leftarrow \phi_{\text{target}}$
 9:     else
10:         $\underline{\phi} \leftarrow \phi_{\text{target}}$
11:     end if
12: end while
13: Try to cluster yet unclustered vertices by looping over hyperarcs.
14: return $\mathcal{C}$
```

Lemma 1. *The binary search clustering runs in $O(m \cdot log(m))$, with m being the number of arcs ($r^*(u,v)$ relations).*

Proof. Algorithm 1 contains a `while` loop that will perform at most $\log(m)$ iterations. Lines 1 and 2 of the algorithm define the lower and upper bounds of the binary search. Even if the hypergraph is a path, *i.e.*, if the lower bound is equal to m and the upper bound is equal to m^2, the number of iterations of the `while` loop will be in $O(\log(m^2))$, which does not change the order of complexity.

Line 6 calls a procedure that works in $O(m)$. Indeed, the procedure computes the cut capacity of every arc and merges every pair of vertices with a cut capacity equal to zero. In line 10, to cluster the remaining unclustered vertices connected by arcs with non-negative cutting capacity, BSC calls a $O(m)$ procedure which loops over hyperarcs and works as follow: for each hyperarc, try to cluster yet unclustered vertices with other vertices in the hyperarc. Hence, the complexity of this algorithm is in $O(m \cdot \log(m))$.

The algorithm presented by Z. Donovan [7] has a complexity in $O(2^{\Delta \cdot M} + |V|^{O(1)})$ time. For a sufficiently large M, this algorithm can become impractical. The BSC algorithm has a complexity in $O(m \cdot \log_2(m))$, which is more attractive in practice. Also, circuit instances are relatively sparse, that is, m is not higher than the number of vertices.

Theorem 1. *The binary search clustering is an M-approximation algorithm for $CN{<}[w], M, \Delta{>}$ when $|p_{\max}| \times d > D$, $D \gg d$ and $\frac{D}{d} \leq M$, with $p_{\max}$ the critical path, d an intra-cluster delay, D an inter-cluster delay and M the maximum size of clusters.*

Proof. Let $H = (V, A)$ be a DAH, and $G = (V, A)$ be its corresponding r^*-weighted DAG. Let $|p_{\max}|$ be the longest path in H. As each vertex have a weight w, we will consider that $w = 1$. Let $\text{Sol}^*(H)$ be the optimal solution for a vertex-set clustering of H, an intra-cluster delay d, and an inter-cluster delay D, such that $D \gg d$ and $\frac{D}{d} \leq M$.

$$\text{Sol}^*(H) \geq \left(\left\lceil \frac{|p_{\max}|}{M} \right\rceil - 1\right) \times D + \left(|p_{\max}| - 1 - \left(\left\lceil \frac{|p_{\max}|}{M} \right\rceil - 1\right)\right) \times d \ . \tag{6}$$

Let $p_{\max}$ the critical path; we suppose $|p_{\max}| \times d > D$. Hence, we obtain:

$$|p_{\max}| \times d - D > 0 \ . \tag{7}$$

In many cases, the propagation time of a circuit's critical path is longer than the time it takes to transfer a signal from one FPGA to another. However, there are circuits for which this is not true, although they are very few. Therefore, this proof applies only to circuits that satisfy the equation 7.

The BSC algorithm groups vertices using a direct approach based on cut capacity. This makes it more practical than a recursive coupling approach. Let $\text{Sol}_{\text{bsc}}(H)$ be the solution produced by our algorithm BSC, presented as Algorithm 1. It can be bounded by the worst solution. A worst-case solution is one in which each vertex forms a cluster. Hence, we have:

$$\text{Sol}_{\text{bsc}} \leq (|p_{\max}| - 1) \times D \ . \tag{8}$$

Then, the approximation ratio is defined by:

$$\frac{\text{Sol}_{\text{bsc}}(H)}{\text{Sol}^*(H)} \leq \frac{(|p_{\max}| - 1) \times D}{\left(\left\lceil \frac{|p_{\max}|}{M} \right\rceil - 1\right) \times D + \left(|p_{\max}| - 1 - \left(\left\lceil \frac{|p_{\max}|}{M} \right\rceil - 1\right)\right) \times d} \ . \tag{9}$$

Let us calculate the approximation ratio for $|p_{\max}| > M$ and $|p_{\max}| \leq M$.
In the case when $|p_{\max}| > M$:

$$\left\lceil \frac{|p_{\max}|}{M} \right\rceil = \frac{|p_{\max}| + (M + r)}{M} \ . \tag{10}$$

By applying equation 10, we obtain:

$$\frac{\text{Sol}_{\text{bsc}}(H)}{\text{Sol}^*(H)} \leq \frac{(|p_{\max}| - 1) \times D}{(|p_{\max}| - r) \times D + (M - 1) \times |p_{\max}| \times d - (M - r) \times d} \ .$$

By applying equation 7, we obtain:

$$\frac{\text{Sol}_{\text{bsc}}(H)}{\text{Sol}^*(H)} \leq M \frac{(|p_{\max}| - 1) \times D}{(|p_{\max}| - r + M - 1) \times D - (M - r) \times d} \ .$$

Let us study the positivity of the expression $DM - Dr - (M - r)d$, we obtain:

$$DM - Dr > (M - r)d = DM - Dr - (M - r)d > 0 \ . \tag{11}$$

By applying equation 11, we obtain:

$$\frac{\text{Sol}_{\text{bsc}}(H)}{\text{Sol}^*(H)} \leq M \ .$$

In the case when $|p_{\max}| \leq M$, we have :

$$\left\lceil \frac{|p_{\max}|}{M} \right\rceil = 1 \ . \tag{12}$$

By applying equation 12 to equation 9, we obtain:

$$\frac{(|p_{\max}| - 1) \times D}{(|p_{\max}| - 1) \times d} \ .$$

Since we have $\frac{D}{d} \leq M$, we obtain:

$$\frac{(|p_{\max}| - 1) \times D}{(|p_{\max}| - 1) \times d} = \frac{D}{d} \leq M \ .$$

Hence, parameterized approximation ratio is M for $CN{<}[w], M, \Delta{>}$ under the condition specified in the theorem 1. In the general case, the ratio remain $M^2 + M$.

4.2 Heavy-edge matching

The heavy-edge matching (HEM) approach for graph coarsening, presented by G. Karypis [15], is widely used in (hyper)graph partitioning tools [17, 21] and yields efficient results in many cases. The advantage of this algorithm is that, in the unconstrained case, it almost halves the size of the instance during each of the first stages of the multilevel framework, which makes its complexity more interesting than that of our Algorithm 1. However, we will show in this subsection that HEM and other algorithms dedicated to 2-matching introduced by Z. Donovan *et al.* [7,9], do not capture path topology adequately. An example is presented in Figure 4 for a clustering with $M > 2$. We will also show that HEM, applied to the DAG weighted with the r^* scheme, yields an approximation ratio of 2 for the $CN{<}[1], 2, \Delta{>}$ problem. This algorithm differs from the two algorithms presented by Z. Donovan *et al.* [7, 9]: one of them looks for a dominant matching, and otherwise returns an arbitrary clustering, while the other is based on a linear programming rounding algorithm.

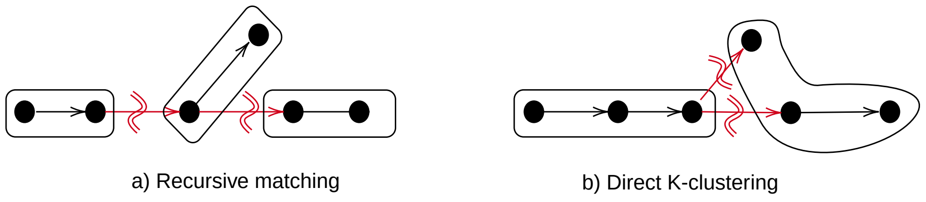

Fig. 4: This figure presents the effects of recursive matching vs. direct k-way clustering. On the left is a solution produced by a recursive matching algorithm for clustering with $M = 3$. On the right is the result of a direct clustering. As we can see, direct clustering produces less cut and clusters than recursive matching approach.

In the example shown in Figure 4, the recursive methods will match vertices only once and cannot match them at the next level, because new vertices have a weight equal to 2. A direct clustering algorithm like our Algorithm 1 will produce in this case a result as good as recursive matching methods. This suggests that a direct clustering algorithm will be more interesting than a recursive coupling algorithm when M is large.

Theorem 2. *Let H be a DAH and $G = (V, A)$ be its corresponding r^*-weighted DAG. The HEM algorithm applied to the DAG for $CN{<}[1], 2, \Delta{>}$ is a 2-approximation algorithm.*

Proof. Let d be the intra-cluster delay and D be the inter-cluster delay, such that $D \gg d$. Let $p_{\max}$ be the critical path, with $|p_{\max}| \times d > D$. Hence:

$$|p_{\max}| \times d - D > 0 \ . \tag{13}$$

Let $\mathrm{Sol}^*(H)$ be the optimal solution for a vertex set clustering of H. In the best case, for a cluster size bounded by 2, the critical path will be coupled $\frac{|p_{\max}|}{2}$ times, which will yield the following lower bound for $\mathrm{Sol}j(H)$:

$$\mathrm{Sol}^*(H) \geq \left(\left\lceil \frac{|p_{\max}|}{2} \right\rceil - 1\right) \times D + \left(|p_{\max}| - 1 - \left(\left\lceil \frac{|p_{\max}|}{2} \right\rceil - 1\right)\right) \times d \ . \tag{14}$$

Let $\mathrm{Sol}_{\mathrm{HEM}}(H)$ be the solution produced by the HEM scheme on our proposed DAG model. It can be bounded by the worst possible solution, in which every vertex forms a cluster. Hence:

$$\mathrm{Sol}_{\mathrm{HEM}}(H) \leq (|p_{\max}| - 1) \times D \ . \tag{15}$$

Then, the approximation ratio is defined by:

$$\frac{\mathrm{Sol}_{\mathrm{HEM}}(H)}{\mathrm{Sol}^*(H)} \leq \frac{(|p_{\max}| - 1) \times D}{\left(\left\lceil \frac{|p_{\max}|}{2} \right\rceil - 1\right) \times D + \left(|p_{\max}| - 1 - \left(\left\lceil \frac{|p_{\max}|}{2} \right\rceil - 1\right)\right) \times d} \ . \tag{16}$$

Let us calculate the approximation ratio for the even and odd cases of $|p_{\max}|$.

By performing the calculation similar to proof 4.1, we obtain:

$$\frac{\mathrm{Sol}_{\mathrm{HEM}}(H)}{\mathrm{Sol}^*(H)} = 2 \ , \tag{17}$$

for both cases, when $|p_{\max}|$ is even and odd.

5 Experimental Results

To validate our models and algorithms, we have performed experiments on benchmarks of 19 logic circuits (B01-14 and B17-22) presented in F. Corno *et al.* [5]. These circuits consist of acyclic combinatorial blocks, bounded by their input and output registers. Every combinatorial block can therefore be modeled as a DAH. Their computation time is conditioned by their critical path, defined as the longest path between two registers (*i.e.*, two red vertices). These circuits have a number of cells from 51 (B01), to 233685 (B19).

Remember that we want to minimize the number of cuts on the critical path. In fact, in our problem, a cut on a path means an additional delay in the path cost. Thus, the compared algorithms aim to group the red-black hypergraphs by minimizing the delay path-length, *i.e.*, the maximum path cost $p_{\max}$. Since the execution time and the number of clusters are important parameters, we measure and compare them. Recall that clustering minimizing the number of clusters refers to the bin-packing problem, which is known to be NP-hard.

To compare them, we measured the degradation of the critical path produced by algorithm $\mathcal{A}$ for each instance I, calculated by: $(\mathrm{Sol}_{\mathcal{A}}(I) - p^I_{\max})/p^I_{\max}$. BSC and HEM algorithm were run 10 times for each circuit and for each cluster size. The average of these 10 runs was used to calculate the path-cost averages for all instances per size of clusters.

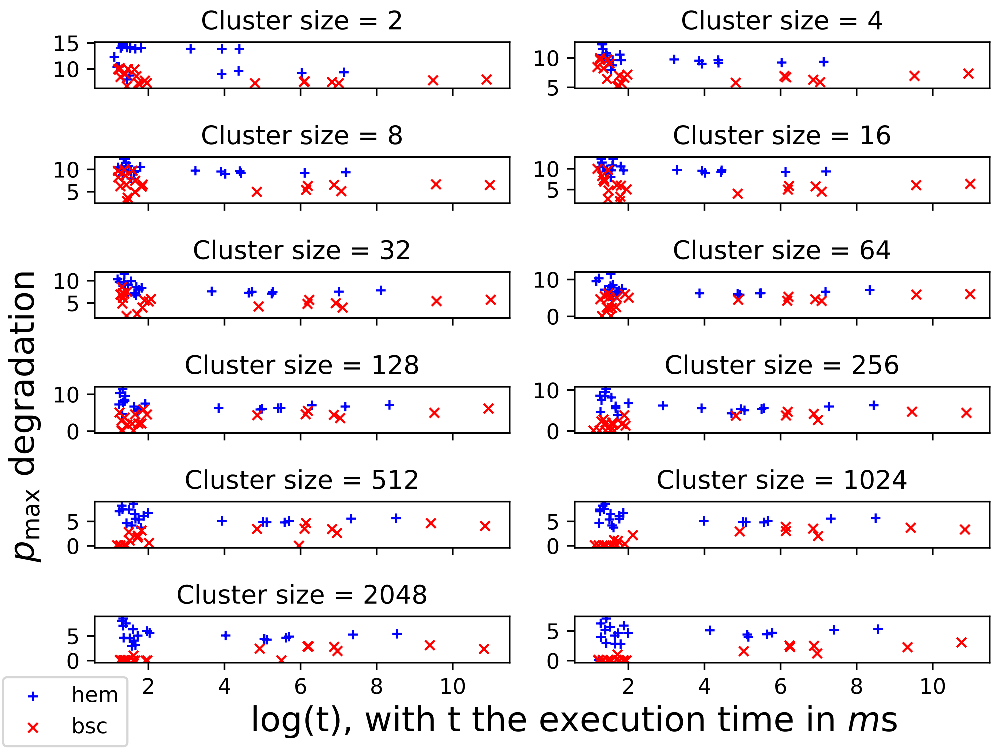

Fig. 5: Results of BSC and HEM on each circuit (point) for M values ranging from 2 to 4096. Each labeled point " $+$ " is a clustering result calculated by the HEM algorithm and each labeled point " $\times$ " is for BSC. Each point is defined by the degradation of the critical path (ordinate) as a function of its logarithmic execution time (abscissa). As shown in all sub-figures, the "$\times$" points are positioned below the "$+$" points, which are based on a lower critical path degradation for BSC. In addition, each point positioned to the left is based on a lower execution time. For two circuits, BSC take more execution time than HEM.

The results in Figure 5 show that our BSC clustering algorithm, applied to circuit hypergraph, outperforms the HEM algorithm for critical path degradation. It can be shown that HEM points are more on the left side than BSC points. This rely on the fact that HEM takes less execution time than BSC. However, the execution time of HEM increase in function of cluster size, that is, some HEM points moves from left to right. Indeed, as we increase the size of the clusters, we notice that HEM makes more recursive calls. Even if these recursive calls are executed on reduced hypergraphs, this increases the runtime. As a result, the complexity of HEM can be described by an additional factor of $\log_2(M)$, while the BSC algorithm admits a time complexity that depends only on the number of hyperedges. In practice, however, we find that the execution time of the BSC algorithm varies slightly as a function of M during the grouping phase, since this phase differs for each M. For BSC, however, these variations remain negligible, which explains why the points of BSC does not change on abscissa of 5 for each cluster size M.

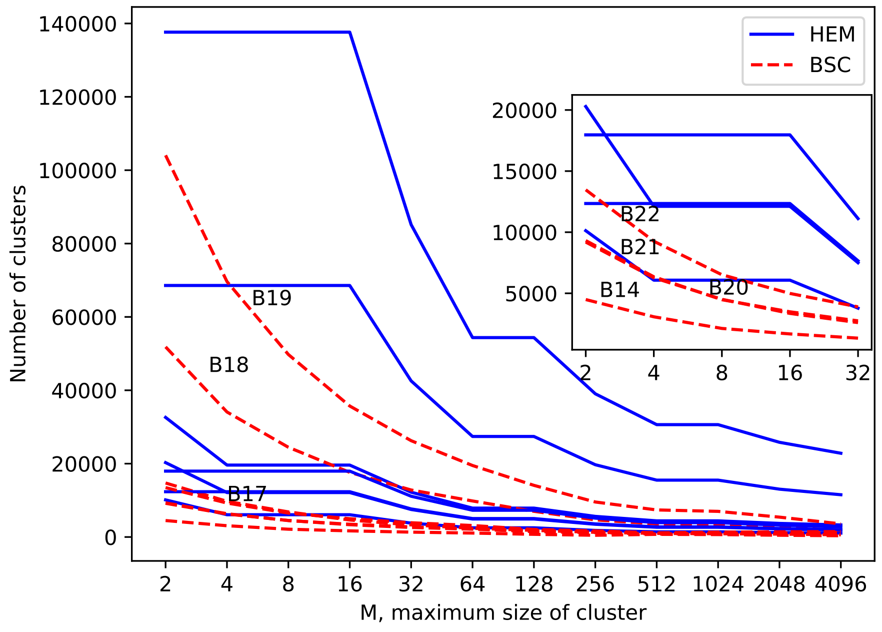

Fig. 6: Comparison between the number of clusters produced by BSC and HEM on a subset of highest circuits of vertices size for M values ranging from 2 to 4096. Each plain lines corresponds to HEM number of clusters and hatched lines to BSC number of clusters. Results show a less number of clusters for BSC than HEM. Subfigure shows us a zoom of B14, B20, B21, and B22 number of clusters for M values ranging from 2 to 32.

The results in Figure 6 show that each BSC curves are under HEM curves, that is, our BSC clustering algorithm produces less number of clusters compared to the HEM algorithm. This can be explained by the fact that BSC directly groups a set of related vertices and applies a second refinement step that tends to reduce the number of clusters. In contrast, the HEM algorithm recursively groups vertices in pairs, which can more easily lead to situations where there are several adjacent clusters of size $M/2 + 1$ that cannot be merged.

Nevertheless, in the context of a multilevel scheme, both algorithms can be used, but BSC clusters vertices directly and does not create clustering levels as HEM does. An adaptation is necessary for such use.

6 Conclusion

In this work, we studied the combinatorial circuit clustering problem for delay minimization (CN) and presented a brief state-of-the-art in Section 2.2.

The aims of clustering algorithms is to select vertices to merge. Hence, the key is to define an attractiveness between the vertices that models the objective. In Section 3, is presented existing weighting schemes l, r for attractiveness between pairs of vertices with our r^* weighting scheme. We shown that our r^* weighting scheme appears to be a better model to cluster critical vertices than the l and r.

In Section 4, we demonstrated that the approximation ratio parameterized by the cluster size is in M if $|p_{\max}| \times d > D$, $D \gg d$ and $\frac{D}{d} \leq M$. This result improves the existing $M^2 + M$-approximation ratio under conditions mentioned above. We also introduced in the same Section 4, our direct clustering BSC algorithm 1 which runs in $O(m \cdot \log(m))$ time, with m the number of hyperarcs, and an adaptation of Heavy Edge Matching with our r^* weighting scheme.

In Section 5, we presented experimental results about a comparison between HEM algorithm yet improved with our weighting scheme r^*, and our BSC algorithm using also r^*, on ITC [5] circuits. Experimental results show that BSC produces less critical path degradation results for a majority of circuit instances. Moreover, BSC produces less number of clusters than HEM. Future works will investigate the efficiency of the BSC algorithm in a multilevel scheme.

References

1. Ababei, C., Selvakkumaran, N., Bazargan, K., Karypis, G.: Multi-objective circuit partitioning for cutsize and path-based delay minimization. In: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design. pp. 181–185. ICCAD '02, Association for Computing Machinery, New York, NY, USA (Nov 2002). https://doi.org/10.1145/774572.774599, https://doi.org/10.1145/774572.774599
2. Çatalyürek, Ü.V., Aykanat, C.: Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. Journal of IEEE Transactions on Parallel and Distributed Systems **10**(7), 673–693 (Jul 1999). https://doi.org/10.1109/71.780863, conference Name: IEEE Transactions on Parallel and Distributed Systems
3. Çatalyürek, Ü.V., Devine, K.D., Faraj, M.F., Gottesbüren, L., Heuer, T., Meyerhenke, H., Sanders, P., Schlag, S., Schulz, C., Seemaier, D., Wagner, D.: More Recent Advances in (Hyper)Graph Partitioning. Tech. Rep. arXiv:2205.13202, arXiv (Jun 2022), http://arxiv.org/abs/2205.13202, arXiv:2205.13202 [cs]
4. Cong, J., Wu, C.: Global clustering-based performance-driven circuit partitioning. In: Proceedings of the 2002 International Symposium on Physical design. pp. 149–154. ISPD '02, Association for Computing Machinery, New York, NY, USA (Apr 2002). https://doi.org/10.1145/505388.505424, https://doi.org/10.1145/505388.505424
5. Corno, F., Reorda, M.S., Squillero, G.: RT-level ITC'99 benchmarks and first ATPG results. Journal of IEEE Design & Test of computers **17**(3), 44–53 (2000)
6. Diwan, A.A., Rane, S., Seshadri, S., Sudarshan, S.: Clustering techniques for minimizing external path length. In: Proceedings of the 22th International Conference on Very Large Data Bases. pp. 342–353 (1996)

7. Donovan, Z.N.: Algorithmic Issues in some Disjoint Clustering Problems in Combinatorial Circuits. Thesis dissertation, West Virginia University Libraries (Jan 2018). https://doi.org/10.33915/etd.3721, https://researchrepository.wvu.edu/etd/3721
8. Donovan, Z.N., Mkrtchyan, V., Subramani, K.: Complexity issues in some clustering problems in combinatorial circuits. arXiv:1412.4051 [cs] (Jan 2017), http://arxiv.org/abs/1412.4051, arXiv: 1412.4051 version: 2
9. Donovan, Z.N., Subramani, K., Mkrtchyan, V.: Disjoint Clustering in Combinatorial Circuits. In: Colbourn, C.J., Grossi, R., Pisanti, N. (eds.) Combinatorial Algorithms. pp. 201–213. Lecture Notes in Computer Science, Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-25005-8_17
10. Donovan, Z.N., Subramani, K., Mkrtchyan, V.: Analyzing Clustering and Partitioning Problems in Selected VLSI Models. Theory of Computing Systems **64**(7), 1242–1272 (Oct 2020). https://doi.org/10.1007/s00224-020-09989-2, https://doi.org/10.1007/s00224-020-09989-2
11. Hendrickson, B., Leland, R.: A multilevel algorithm for partitioning graphs. In: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing. p. 28–es. Supercomputing '95, Association for Computing Machinery, New York, NY, USA (1995). https://doi.org/10.1145/224170.224228, https://doi.org/10.1145/224170.224228
12. Heuer, T., Schlag, S.: Improving coarsening schemes for hypergraph partitioning by exploiting community structure. In: Proceedings of the 16th International Symposium on Experimental Algorithms, (SEA 2017). pp. 21:1–21:19 (2017)
13. Kagaris, D.: On minimum delay clustering without replication. Integration **36**(1), 27–39 (2003). https://doi.org/https://doi.org/10.1016/S0167-9260(03)00030-0, https://www.sciencedirect.com/science/article/pii/S0167926003000300
14. Karypis, G., Aggarwal, R., Kumar, V., Shekhar, S.: Multilevel hypergraph partitioning: applications in VLSI domain. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **7**(1), 69–79 (Mar 1999). https://doi.org/10.1109/92.748202, conference Name: IEEE Transactions on Very Large Scale Integration (VLSI) Systems
15. Karypis, G., Kumar, V.: Analysis of multilevel graph partitioning. In: Proceedings of the 1995 ACM/IEEE conference on Supercomputing. pp. 29–es (1995)
16. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on scientific Computing **20**(1), 359–392 (1998)
17. Karypis, G., Kumar, V.: Hmetis: a hypergraph partitioning package. ACM Transactions on Architecture and Code Optimization (1998)
18. Lawler, E.L., Levitt, K.N., Turner, J.: Module clustering to minimize delay in digital networks. IEEE Transactions on Computers **100**(1), 47–57 (1969)
19. Murgai, R., Brayton, R.K., Sangiovanni-Vincentelli, A.: On clustering for minimum delay/ara. pp. 6,7,8,9–6,7,8,9. IEEE Computer Society (Jan 1991). https://doi.org/10.1109/ICCAD.1991.185176, https://www.computer.org/csdl/proceedings-article/iccad/1991/00185176/12OmNASILTx
20. Pan, P., Karandikar, A.K., Liu, C.L.: Optimal clock period clustering for sequential circuits with retiming. IEEE transactions on computer-aided design of integrated circuits and systems **17**(6), 489–498 (1998)
21. Pellegrini, F.: Scotch and PT-Scotch Graph Partitioning Software: An Overview. In: Uwe Naumann, O.S. (ed.) Combinatorial Scientific Computing, pp. 373–406. Chapman and Hall/CRC (2012). https://doi.org/10.1201/b11644-15, https://inria.hal.science/hal-00770422
22. Rajaraman, R., Wong, M.D.F.: Optimal clustering for delay minimization. In: Proceedings of the 30th international Design Automation Conference. pp. 309–314. DAC '93, Association for Computing Machinery, New York, NY, USA (Jul 1993). https://doi.org/10.1145/157485.164907, https://doi.org/10.1145/157485.164907
23. Rodriguez, J., Galea, F., Pellegrini, F., Zaourar, L.: A hypergraph model and associated optimization strategies for path length-driven netlist partitioning. In: Proceedings of the 23rd International Conference on Computational Science (ICCS). pp. 652–660. Springer (2023)