IMPart: Integration of Memetic Operations into Multi-Level Framework for Large-k-Way Hypergraph Partitioning

Yugao Zhu, Zhicheng Guo, Shang Liu, Mengming Li, Jing Wang, Zhiyao Xie*
Hong Kong University of Science and Technology
{yzhuel, zguobx, ywu092, mengming.li, jwangjw}@connect.ust.hk, eezhiyao@ust.hk

Abstract

The problem of k-way hypergraph partitioning is fundamental with significant applications in various fields, including VLSI design and scientific computing. State-of-the-art hypergraph partitioners commonly employ a multi-level framework encompassing coarsening, initial partitioning, uncoarsening, and refinement phases. However, many existing methods do not scale well to problems requiring a large number of partitions (i.e., large k). In pursuit of exceptionally high solution quality, existing memetic approaches often execute their two key operations, recombination and mutation, by invoking separate, standalone multi-level partitioners. This design choice, however, renders them significantly more time-consuming than standard multi-level partitioners. To make such memetic approaches more practical, we propose an advanced memetic framework, IMPart, which introduces novel recombination and mutation operators and integrates them directly into the uncoarsening phase of a single multi-level framework. This transforms the local searches of different granularities in the traditional multi-level framework into a sophisticated, collaborative search. Experimental results on multiple standard benchmarks demonstrate our framework more effectively escapes local optima and explores the global solution space for higher-quality solutions, substantially outperforming all existing hypergraph partitioners for large-k-way hypergraph partitioning. Our framework highlights a new paradigm for the development of advanced hypergraph partitioners.

ACM Reference Format:
Yugao Zhu, Zhicheng Guo, Shang Liu, Mengming Li, Jing Wang, Zhiyao Xie. 2026. IMPart: Integration of Memetic Operations into Multi-Level Framework for Large-k-Way Hypergraph Partitioning. In *63rd ACM/IEEE Design Automation Conference (DAC '26), July 26–29, 2026, Long Beach, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770743.3804129

1 Introduction

Hypergraph partitioning is an important problem with widespread applications across various domains, including VLSI design and scientific computing [23]. k-way hypergraph partitioning is an NP-hard problem [17] that aims to divide the nodes of a hypergraph into k disjoint partitions while minimizing the number of hyperedges cut (cut-size). Over the past few decades, a rich array of heuristic algorithms has emerged, including *spectral methods* [6, 7, 27], *flow methods* [13, 15], *local search methods* (e.g., Kernighan-Lin (KL) [20] and Fiduccia-Mattheyses (FM) [12]), *memetic (genetic) methods* [3, 4], deep learning-based methods [10, 21], and the highly successful *multi-level framework* [8, 19, 24].

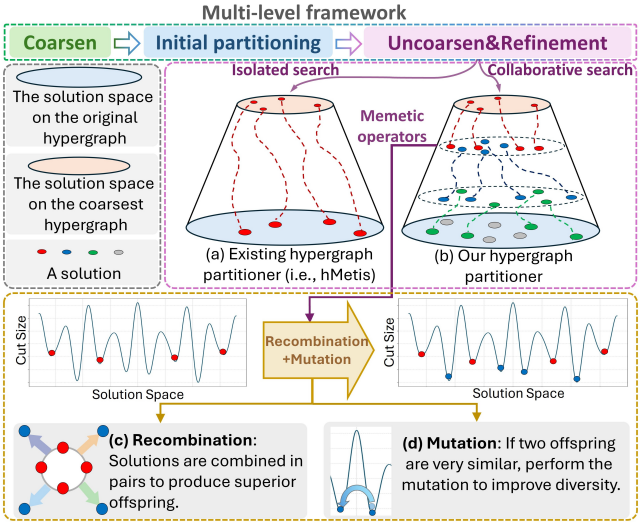


Figure 1: Framework comparison between the standard multi-level framework and ours.

The *multi-level framework* is widely regarded as the most successful approach to hypergraph partitioning [1, 14, 16, 26]. It forms the basis for leading partitioners like hMETIS [19] and KaHyPar [24]. This framework operates in three stages: (1) **Coarsening:** The hypergraph is iteratively coarsened by merging tightly connected nodes. (2) **Initial Partitioning:** A balance-constrained partition is computed on the coarsest hypergraph. (3) **Uncoarsening and Refinement:** The partition is projected back through intermediate levels to the original hypergraph, with local search (e.g., FM) applied at each level to refine the solution and reduce the cut size.

Most existing literature [6, 13, 27] has focused on the simpler case of bipartitioning ($k = 2$). Even for general k-way partitioning, many works [4, 5, 7] only focus on relatively small cases, typically $k = 2, 3, 4$. The general k-way partitioning problem, however, presents substantial challenges: (1) Increasing the number of partitions k leads to a combinatorial explosion of the solution space, as the number of possible ways to partition n vertices is fundamentally governed by a k^n term. This vast search landscape makes it increasingly difficult for heuristic algorithms to find a globally effective solution. (2) Refinement becomes substantially more complex and local search heuristics such as FM are more prone to stalling in poor local optima. Consequently, research into effective and scalable algorithms specifically tailored to the difficult k-way partitioning problem remains insufficient.

To address this gap, this paper *introduces a new, open-source*[1] *framework for the challenging large-k-way partitioning problem,*

*corresponding author

DAC '26, Long Beach, CA, USA

ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804129

[1] https://anonymous.4open.science/r/IMPart-2B9D

stemming from a fundamental overhaul of the conventional *memetic algorithm* in hypergraph partitioning.

Memetic algorithm is an evolutionary method that evolves a population of solutions using operators such as recombination and mutation, and the individuals are improved via a local search. Although corresponding methods [4, 9, 25] were applied to the hypergraph partitioning problem early, most of these initial approaches were rudimentary and non-multilevel. Consequently, such methods [4, 9, 25] are not competitive with state-of-the-art multilevel hypergraph partitioners [1, 11]. KaHyPar-E [3] was the first to combine a memetic method with the multi-level partitioner, achieving results competitive with state-of-the-art partitioners and particularly demonstrating leading performance on large-k-way hypergraph partitioning problems. However, its approach was highly time-consuming, as **each recombination and mutation** operation relied on a **complete** multilevel partitioner to generate new solutions from existing ones.

In contrast, we propose **IMPart**, which thoroughly redesigns this memetic approach by integrating all recombination and mutation operations directly within a **single** multi-level partitioning process. This eliminates the need to repeatedly invoke complete external partitioners. This design is more efficient than the KaHyPar-E paradigm. Furthermore, IMPart introduces novel operators for recombination and mutation adapted to this integrated structure, establishing a new memetic paradigm.

Figure 1 contrasts the traditional multi-level architecture (a) with our unified memetic framework (b). A fundamental architectural challenge exists in the traditional design: During uncoarsening, the local refinement is based on the partition structure inherited from the coarser level. Consequently, the multi-level framework is not flexible enough to adjust its partitioning scheme as the uncoarsening proceeds. The initial partition can exert an unpredictable influence on the final solution quality. Some methods, like hMETIS, attempt to improve solution quality by employing multiple initial partitions, as different initial solutions can explore different parts within the vast and unbounded solution space—as illustrated in Fig.1 (a). However, these independent searches operate in isolation, failing to leverage collective insights. Each run can still become trapped in its own local optimum, unable to learn from the partitioning schemes of others.

This challenge motivates our core insight: multiple solutions should explore the solution space collaboratively, providing mutual global guidance, as illustrated in Fig. 1(b). This population-based approach also lends itself to straightforward parallelization. To leverage the collaborative guidance, IMPart integrates novel memetic operators directly within the multi-level partitioning process:

❶ **Our recombination:** During uncoarsening, we periodically implement this collaborative learning via a recombination operator that merges the superior structures of different parent solutions. This approach is particularly effective for navigating the complex landscape of large-k-way hypergraph partitioning, allowing our framework to discover superior solutions. To manage these interactions and maintain a stable population, solutions are periodically reorganized into a ring topology (Fig. 1(c)), where each solution recombines with its neighbors. As these offspring are uncoarsened and refined, they develop new local structures, providing diverse material for future recombinations.

❷ **Our mutation:** A drawback of the above recombination technique is that it may generate offspring that are too similar to each other. Since adjacent offspring in the ring-based structure share a common parent, they often inherit highly correlated features. Consequently, as the uncoarsening proceeds, offspring that have become overly similar tend to converge towards nearly identical partitioning schemes. This lack of diversity stifles the exploration of varied local structures from which solutions could mutually learn. To address this issue, we employ a diversity enhancement mechanism, as illustrated in Fig. 1 (d). This mechanism identifies lower-quality solutions (i.e., those with a higher cut size) among similar ones, and replaces them with new offspring generated via a targeted mutation. This approach actively injects diversity into the population, guiding the search towards unexplored regions of the solution space.

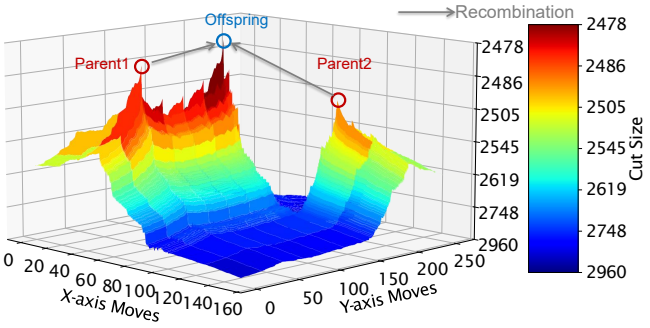


Figure 2: The jump mechanism in dataset `sparcT1_core`: An improved global solution (offspring) is discovered through our recombination of parents.

These operators endow IMPart with a strong, periodic *jumping* competence, as illustrated in Fig. 2: parent solutions leap beyond local optima to produce a globally higher-quality offspring.

Evaluated on standard benchmarks (Titan23 [22], ISPD98 [2]), our algorithm consistently produced superior solutions compared to state-of-the-art (KaHyPar [24]), spectral (K-SpecPart [7]), foundational (hMETIS [19]), and memetic (KaHyPar-E [3]) methods. Our main contributions are:

- **A memetics-integrated multi-level framework.** We propose IMPart, a novel memetic method embedded within a single multi-level partitioning process. This paradigm differs from existing memetic methods that repeatedly invoke the complete multi-level partitioner on the original hypergraph to perform each recombination and mutation. We designed novel recombination and mutation operators to support this structure. Our recombination during uncoarsening transforms the local searches of different granularities in the traditional multi-level framework into a sophisticated, collaborative search, enabling the discovery of globally higher-quality solutions, and inherently provides a parallelization opportunity to boost execution efficiency.
- **Extensive experimental validation.** We provide both qualitative and quantitative evidence of our framework's effectiveness. We first conduct comprehensive comparisons against state-of-the-art partitioners on standard benchmarks, demonstrating that IMPart consistently achieves superior solution quality. Complementing these results, we visually demonstrate the distinct *jumping* behavior that enables our method to effectively escape local optima, and finally, we verify the robust scalability of our algorithm in large-k partitioning settings.

2 Problem Formulation

Given a hypergraph $H(V, E)$ where V is the set of vertices and E is the set of hyperedges, the weights of all vertices and hyperedges are positive, associated with w_v and w_e respectively. The hypergraph partitioning problem[2] is to find a partition scheme S that divides the nodes into k disjoint blocks $V_1, V_2, ..., V_k$ which $V_i \cap V_j = \emptyset$ and $\bigcup_{i=0}^{k-1} V_i = V$, the node weight W_{V_i} of each block i should satisfy:

$$W_{V_i} \leq (1+\epsilon) \cdot \left\lceil \frac{W_V}{k} \right\rceil$$

where $W_V = \sum_{v \in V} w_v$, $W_{V_i} = \sum_{v \in V_i} w_v$ and ϵ is the imbalance factor. Under this constraint, the objective is to find the smallest $\text{cutsize}_H(S)$, where

$$\text{cutsize}_H(S) = \sum_{e \not\subseteq V_i \text{ for all } V_i \in S} w_e$$

3 Methodology

Figure 3 illustrates how our method, IMPart, enhances the standard multi-level paradigm by integrating two key evolutionary components: recombination and mutation. Recombination allows for mutual learning between different solutions, while mutation is employed to maintain population diversity and avoid premature convergence. Unlike the existing memetic method KaHyPar-E that repeatedly invokes the multi-level partitioner on the original hypergraph, IMPart integrates recombination and mutation *internally*, making the framework more lightweight and efficient. It also elevates the local searches at different granularities during uncoarsening and refinement into a sophisticated, collaborative search.

This section details the design and integration of these operators within a single multi-level partitioning process. We will first detail our recombination operator in Section 3.1, and then describe the mutation operator in Section 3.2. While our implementation builds upon the state-of-the-art KaHyPar framework, the proposed methods are generalizable to any multi-level partitioning approach.

3.1 Recombination

Recombination is a technique for finding improved solutions from multiple existing partition schemes $S_1, S_2, \ldots, S_l$ for a hypergraph. We will first provide an overview of our recombination operator in Section 3.1.1, followed by its specific operational details in Section 3.1.2.

3.1.1 Overview of Our Recombination. Our recombination is *embedded directly within the uncoarsening and refinement phase*. It is invoked periodically at different coarsening levels, allowing solutions to learn from one another and reposition their search to more globally advantageous regions of the solution space.

Specifically, our method first generates α diverse solutions (offspring). As uncoarsening progresses, these α solutions explore diverse directions in the solution space at each level, uncovering distinct partitioning schemes. Throughout the uncoarsening phase, we perform β rounds of recombination at key moments ($\alpha = \beta = 7$ in all our experiments). This process pairwise merges the current solutions to form α new, fused solutions in a circular manner[3]. This allows each solution to break free from its original search trajectory and incorporate high-quality local structures from others. Then, as illustrated in Fig. 1, the resulting offspring are superior solutions produced by recombination, which are positioned in more globally advantageous regions of the solution space. From there, they continue the search through subsequent uncoarsening and refinement. This architectural adjustment marks a significant departure from conventional methods. In existing multi-level frameworks, solutions in the uncoarsening phase are often restricted to minor improvements within the confines of the current local structure. Our proposed architecture, however, empowers these solutions to escape local optima, significantly enhancing their capacity to discover globally superior solutions.

The timing of these β rounds is specifically tailored to the architecture of KaHyPar's n-level framework, where n is the node number of the original hypergraph. In this paradigm, each coarsening step merges only a single pair of nodes, and refinement during uncoarsening is localized to the neighborhood of that node pair. We leverage its n-level framework to trigger recombination whenever the number of nodes being uncoarsened reaches a predefined threshold. Specifically, the set of thresholds is geometrically spaced over the uncoarsening trajectory as follows:

$$\left\{ n_c^{1-1/\beta} n^{1/\beta},\ n_c^{1-2/\beta} n^{2/\beta},\ \ldots,\ n_c^{1/\beta} n^{1-1/\beta},\ n \right\},$$

where n_c is the node number of the coarsest hypergraph.

We will now describe the detailed operation of recombination, explaining how to generate a globally higher-quality solution from two given solutions at a specific coarsening level.

3.1.2 The Specific Operations of Our Recombination. We aim to produce superior solutions by leveraging a recombination process that integrates beneficial features from two parent solutions. The methodology involves first applying recombination to hypergraphs at various levels to yield a clustered hypergraph as illustrated in Fig. 3. Subsequently, we employ a hybrid strategy to process the clustered hypergraph based on its complexity. For instances below a predefined threshold, we apply an integer linear programming (ILP) formulation [6, 18]. While ILP guarantees an optimal solution, its significant time complexity is prohibitive for larger cases. Therefore, for hypergraphs exceeding this threshold, we resort to the V-cycle in KaHyPar, a technique applied to improve the current solution.

The constraints in ILP are defined as below. We introduce integer $\{0, 1\}$ variables: $x_{v,i}$ for each vertex v and for each block V_i, and $y_{e,i}$ for each hyperedge e and for each block V_i. Specifically, $x_{v,i} = 1$ signifies that vertex v is in block V_i. The variable $y_{e,i} = 1$ specifically means all vertices of hyperedge e are entirely contained within block V_i. Then we can define the hypergraph partitioning problem as below:

- $\sum_{j=0}^{K-1} x_{v,j} = 1$, for all $v \in V$
- $y_{e,i} \leq x_{v,i}$ for all $e \in E$, and $v \in e$
- $\sum_{v \in V_i} w_v x_{v,i} \leq (1+\epsilon) \cdot \left\lceil \frac{W_V}{k} \right\rceil$

The objective is to maximize the total weight of the hyperedges that are not cut, i.e.,

$$\text{maximize} \sum_{e \in E} \sum_{i=0}^{K-1} w_e y_{e,i}.$$

Specifically, we use C++ Gurobi to tackle the aforementioned ILP problem. Since the hypergraph partitioning problem is NP-hard, its

[2] Our hypergraph partitioning formulation aligns with that of established tools (e.g., KaHyPar, PaToH, KaHyPar-E) and the recent hMETIS release (v2.0-pre1).

[3] i.e., pairing offspring 1 and 2, 2 and 3, ..., $\alpha - 1$ and α, and finally α and 1.

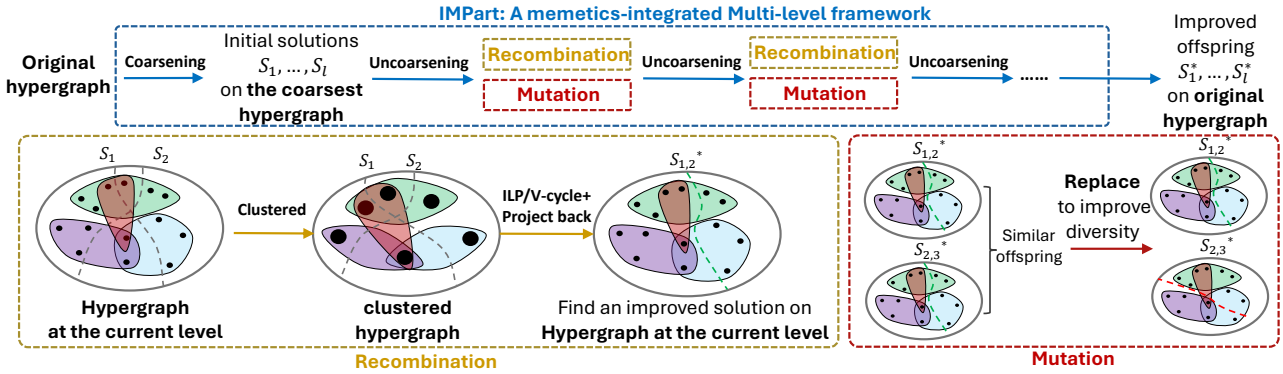


Figure 3: IMPart. Our memetics-integrated multi-level framework with recombination and mutation operators.

solution space is vast. Therefore, we employ several strategies to accelerate the ILP solver:

(1) **Symmetry Breaking.** To eliminate redundant symmetric solutions (i.e., block ID permutations), we restrict vertex v to be assigned only to partitions i such that $i \leq v$.
(2) **Warm Start.** To accelerate convergence, we initialize the ILP solver with the better of the two parent solutions (the one with the smaller cut size).

We let n' be the number of nodes in the clustered hypergraph and use the metric $n' \times k$ to determine the solution method. For metrics below 600, we use the ILP solver to find a provably optimal solution. For metrics in the range $[600, 1000)$, we configure the ILP for an approximate solution with an optimality gap of at most 1%. Finally, for metrics ≥ 1000, we resort to the V-cycle method.

3.2 Mutation

While the technique discussed in the previous subsection is effective, a common challenge arises as uncoarsening progresses: due to mutual learning, offspring often converge to similar or even identical structures. To counteract this loss of diversity and enable a broader exploration of the solution space, we employ a mutation operator to ensure differentiation among offspring.

The mutation operators. After executing the recombination operator, we sort all the obtained new offspring by their cut size in ascending order and process them sequentially. For each selected offspring S_i, we calculate its similarity to every subsequent offspring $S_j (j > i)$ using an edge-based distance $d_e(S_i, S_j)$. If this metric is less than a predefined threshold t ($t = 20$ in our experiments), offspring S_i is added to the set $M(S_j)$. If an offspring S_j has an empty set $M(S_j)$ (meaning it's sufficiently distinct from all preceding offspring), it does not undergo mutation. Otherwise, it is mutated based on the offspring within its $M(S_j)$ set.

The guiding principle is to mutate S_j in a way that not only seeks a small cut-size but also increases its distance from the solutions in $M(S_j)$. To achieve this, we construct a new hypergraph where the sets of vertices and hyperedges are identical to the current-level hypergraph, and vertex weights remain unchanged. However, the weights of the hyperedges are updated as $w'_e = w_e(1 + 0.1 \cdot C_{M(S_j)}(e))$, where $C_{M(S_j)}(e)$ represents the count of offspring in $M(S_j)$ that cut hyperedge e. We then process this new hypergraph using the partitioner KaHyPar, and the resulting partition scheme replaces the original solution S_j. The more times a hyperedge is cut in $M(S_j)$, the greater its weight is increased, and the lower the likelihood that the hypergraph partitioner will cut it. Consequently,

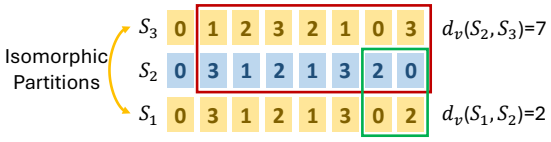


Figure 4: An example of the partition isomorphism problem in node-based metric.

the updated solution S_j is guided to be structurally dissimilar to the offspring in $M(S_j)$, which helps maintain population diversity at this level.

The measurement of similarity. To measure the similarity between two partitions, S_i and S_j, an intuitive **node-based metric** is the Hamming distance, which counts the number of mismatched node assignments:

$$d_v(S_i, S_j) = \sum_{k=1}^{|V|} \delta_k, \quad \text{where } \delta_k = \begin{cases} 1 & \text{if } S_i(k) \neq S_j(k) \\ 0 & \text{if } S_i(k) = S_j(k) \end{cases} \quad (1)$$

where $S_i(k)$ denotes the partition ID of node k in partition S_i. However, this metric is susceptible to the **partition isomorphism problem**, where identical partitions can be measured as different simply due to a relabeling of the partition groups (an example is shown in Fig. 4).

Therefore, we adopt a more robust, label-invariant **edge-based metric**. This metric is the L_1 distance between the connectivity vectors of the two partitions:

$$d_e(S_i, S_j) = \sum_{e \in E} |\text{connectivity}_{S_i}(e) - \text{connectivity}_{S_j}(e)| \quad (2)$$

where $\text{connectivity}_{S_i}(e)$ is the number of distinct partitions spanned by hyperedge e in partition S_i. This metric is therefore well-aligned with the primary goal of hypergraph partitioning, as it directly quantifies differences in the cut structure.

4 experiments

In this section, we first introduce the experimental setup in Section 4.1. Then, in Section 4.2, we conduct comprehensive performance comparisons on standard benchmarks including Titan23 [22] and ISPD98 [2].

4.1 Experimental Setup

Our experimental evaluation comprises three main components:

Table 1: Experimental results on Titan23 benchmark.

Design	$k = 4$, imbalance* = 2%					$k = 4$, imbalance = 5%				$k = 10$, imbalance = 2%				$k = 10$, imbalance = 5%			
	hMETIS	**K-Spec**	**KaHy**	**KaHy-E**	**Ours**	**hMETIS**	**KaHy**	**KaHy-E**	**Ours**	**hMETIS**	**KaHy**	**KaHy-E**	**Ours**	**hMETIS**	**KaHy**	**KaHy-E**	**Ours**
sparcT1_core	2633	2492	2646	2485	**2229**	2573	2467	**2313**	2315	3790	3721	3682	**3596**	3884	3452	3524	**3291**
neuron	580	431	422	424	**405**	505	415	410	**406**	1032	724	728	**684**	1058	659	652	**647**
stereo_vision	455	475	394	**367**	**367**	424	**315**	317	**315**	773	732	**724**	728	730	681	**652**	678
des90	700	747	699	**685**	**685**	768	625	**623**	628	1494	1301	1309	**1252**	1543	1158	1128	**1036**
SLAM_spheric	3423	3241	3217	3201	**3191**	3354	3114	**3056**	3103	5029	4456	4361	**4305**	4931	4400	4205	**4102**
cholesky_mc	983	984	**975**	**975**	**975**	984	**965**	**965**	**965**	2229	1836	**1773**	1777	2121	1700	1673	**1667**
segmentation	557	490	478	**477**	**477**	531	**443**	**443**	**443**	1536	1238	1139	**1115**	1498	883	**879**	**879**
bitonic_mesh	1113	1311	1100	1094	**1091**	1115	1088	1086	**1084**	2388	2149	2212	**2102**	2340	1962	1929	**1894**
dart	1449	1401	1302	1253	**1242**	1475	1027	**1025**	**1025**	1663	1529	1528	**1527**	1693	1630	1528	**1527**
openCV	574	522	670	519	**514**	597	575	531	**497**	970	727	713	**681**	1052	752	695	**645**
stap_qrd	712	674	612	**611**	614	710	612	547	**539**	1472	972	972	**967**	1410	881	878	**875**
minres	411	407	411	**405**	411	413	347	347	**341**	866	799	765	**764**	882	699	**662**	667
cholesky_bdti	1901	1865	1913	1863	**1861**	1891	1793	**1791**	**1791**	3785	3154	**3152**	**3152**	3897	3172	3140	**3126**
denoise	953	1001	944	872	**861**	1166	870	865	**814**	1906	1501	1496	**1487**	1889	1237	**1116**	**1116**
sparcT2_core	3165	3558	2682	2696	**2680**	2506	2415	2374	**2350**	5220	5125	4945	**4887**	5438	3826	3817	**3779**
gsm_switch	5354	4404	2883	2901	**2841**	5289	2692	2692	**2681**	6056	6139	6139	**5820**	5923	5078	5377	**4950**
mes_noc	1400	1346	1332	**1307**	**1307**	1406	1262	1262	**1255**	3251	2616	2616	**2581**	3156	2493	2353	**2296**
LU230	**5794**	6310	5821	6001	5905	5755	5889	5938	**5591**	**8997**	9568	9568	9290	9046	9185	9048	**8924**
LU_Network	1509	1417	1247	1252	**1246**	1466	**1044**	**1044**	**1044**	3988	3446	**3442**	**3442**	3918	2086	2086	**2084**
sparcT1_chip2	1759	1601	1431	1602	**1399**	1773	**1216**	**1218**	**1216**	3900	**2688**	**2688**	2778	3410	2236	2467	**2159**
directrf	1070	1092	1069	1209	**972**	1065	955	959	**933**	2594	2534	2528	**2522**	2746	**2081**	2104	2104
bitcoin_miner	2601	2737	1862	1896	**1612**	2269	1349	1382	**1294**	5496	3142	2984	**2926**	5108	1701	**1699**	1700
Norm Avg.	1.113	1.086	1	0.987	**0.954**	1.213	1	0.988	**0.973**	1.186	1	0.988	**0.971**	1.365	1	0.983	**0.961**

* The imbalance is set as a percentage (p) of the total number of vertices. It relates to the imbalance factor ϵ via $\epsilon = k \cdot p$.

(1) Sections 4.2.1 and 4.2.2 present overall comparisons between our method and leading hypergraph partitioners, including hMETIS, K-SpecPart (K-Spec), KaHyPar (KaHy), and KaHyPar-E (KaHy-E), on the Titan23 and ISPD98 benchmarks, respectively. We focus on the multi-way partitioning problem, with experiments conducted for $k = 4$ and $k = 10$ partitions. We establish two imbalance constraints by setting the maximum allowable size to the average size ($|V|/k$) plus an additional 2% or 5% of the total node count ($|V|$).[4] We provide a normalized average (Norm. Avg.) for each method, calculated as the geometric mean relative to KaHyPar. To ensure a fair comparison, our method (IMPart), hMETIS, and KaHyPar were tested against **the same total execution time**, and the best result obtained within that time was reported. Due to library compatibility issues (missing `triton_part_refine` command) preventing the execution of K-SpecPart, we can only report the results for K-SpecPart ($k = 4$, imbalance= 2%) directly from its original publication. Furthermore, owing to its inherent complexity, KaHyPar-E was allocated double this execution time.

(2) In Section 4.2.3, we visualize the optimization process to provide evidence of our framework's ability to escape local optima. We contrast our method's trajectory, characterized by abrupt, significant improvements, with baseline approaches that tend to show only gradual refinement.

(3) In Section 4.2.4, we extend our analysis to larger k values ($k = 16$ and 32), presenting an overall comparison of cut size performance against the baselines to demonstrate the scalability of our algorithm.

We have open-sourced all our code and released all partitioning results for reproducibility. Our implementation is based on KaHyPar, with all parameters consistent with it[5]. All experiments were conducted on a `single core` of an `Intel(R) Xeon(R) Gold 6438Y+ CPU @ 2.00GHz`, running `Ubuntu 22.04.5 LTS` with `512GB RAM`. Our implementation was compiled from source using `g++ 11.4.0` with the `C++17` standard.

4.2 Comprehensive Performance Comparison

4.2.1 Comparison on the Titan23 Benchmarks. Table 1 shows the comparison of our results with the baselines on the Titan23 benchmarks. In the scenarios with $k = 4$ and imbalance=2% and 5%, IMPart achieves 4.6% and 2.7% improvement over the base partitioner KaHyPar, and 3.3% and 1.5% improvement over KaHyPar-E, respectively. In the scenarios with $k = 10$ and imbalance=2% and 5%, our method IMPart achieves 2.9% and 3.9% improvement over the base partitioner KaHyPar, and 1.6% and 2.3% improvement over KaHyPar-E, respectively.

4.2.2 Comparison on the ISPD98 Benchmarks. Table 2 shows the comparison of our results with the baselines on the ISPD98 benchmarks. In the scenarios with $k = 4$ and imbalance=2% and 5%, IMPart achieves 1.6% and 1.2% improvement over the base partitioner KaHyPar, and 1.1% and 0.6% improvement over KaHyPar-E, respectively. In the scenarios with $k = 10$ and imbalance=2% and 5%, our method IMPart achieves 2.5% and 2.5% improvement over the base partitioner KaHyPar, and 0.7% and 0.7% improvement over KaHyPar-E, respectively.

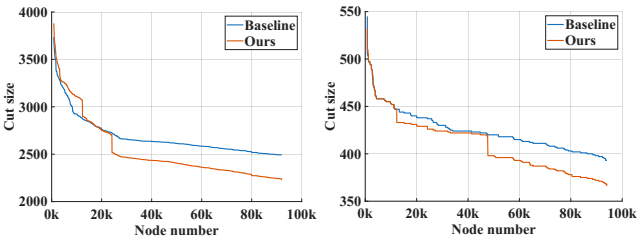
4.2.3 Efficacy of the Jumping Mechanism. In this section, we demonstrate the ability to escape local optima through recombination in our framework, i.e., the *jumping* mechanism. This capability is closely associated with its eventual success in achieving significantly smaller cut-sizes. Building on the KaHyPar framework, our process starts with seven distinct initial solutions (seeds -1 to 5),

[4]For $k = 4$ partitions, these constraints translate directly to the standard imbalance factors (ϵ) of 0.08 and 0.20. For $k = 10$ partitions, they correspond to ϵ values of 0.20 and 0.50, respectively.

[5]The configuration document is *cut_kKaHyPar_sea20.ini*.

Table 2: Experimental results on ISPD98 benchmark.

Design	$k = 4$, imbalance = 2%					$k = 4$, imbalance = 5%				$k = 10$, imbalance = 2%				$k = 10$, imbalance = 5%			
	hMETIS	**K-Spec**	**KaHy**	**KaHy-E**	**Ours**	**hMETIS**	**KaHy**	**KaHy-E**	**Ours**	**hMETIS**	**KaHy**	**KaHy-E**	**Ours**	**hMETIS**	**KaHy**	**KaHy-E**	**Ours**
ibm01	476	522	462	458	**450**	458	416	416	**415**	880	808	807	**796**	887	699	697	**695**
ibm02	596	706	**584**	618	607	596	553	550	**528**	1978	1896	1893	**1888**	1832	1602	1595	**1589**
ibm03	1672	1690	1659	1647	**1625**	1638	1647	**1646**	**1646**	2659	2617	2560	**2510**	2643	2518	2428	**2391**
ibm04	1620	1626	**1590**	**1590**	**1590**	1636	1524	1517	**1512**	3203	3007	2921	**2827**	3103	2615	2539	**2531**
ibm05	2946	2946	2960	**2897**	2945	2913	2901	2878	**2855**	4656	4205	4117	**4106**	4451	4005	3827	**3820**
ibm06	1481	1476	1467	**1465**	**1465**	1486	1466	1461	**1455**	2662	2531	2454	**2417**	2569	2261	**2187**	2232
ibm07	2176	2154	2068	2049	**2045**	2148	1978	**1954**	**1954**	3758	3465	3436	**3392**	3730	3140	3081	**3033**
ibm08	2331	2328	2292	**2278**	2286	2313	2185	**2178**	2188	3774	3521	3441	**3419**	3727	3299	**3178**	3212
ibm09	1700	1676	1672	1672	**1651**	1711	1594	**1583**	1586	3187	2768	2672	**2665**	3223	2557	2563	**2462**
ibm10	2433	2400	2190	**2188**	**2188**	2396	**2155**	**2155**	2156	4599	4395	4128	**4122**	4537	3669	3555	**3522**
ibm11	2476	2452	2346	2341	**2327**	2459	2035	2033	**2032**	4005	3591	3580	**3509**	4102	3394	3235	**3172**
ibm12	3936	3844	3940	**3699**	**3699**	3775	3527	3357	**3270**	6546	**6075**	6093	6086	6609	5616	**5433**	5585
ibm13	1818	1904	1754	1751	**1715**	1817	1565	1563	**1562**	2873	2660	**2653**	**2653**	2927	2631	2610	**2491**
ibm14	3346	3475	3240	3189	**3116**	3187	2878	2871	**2868**	5612	4916	4923	**4902**	5506	4428	4456	**4413**
ibm15	4859	4720	4763	4769	**4582**	4916	4411	4335	**4326**	7379	6947	**6547**	6677	7237	6048	6010	**5963**
ibm16	4068	4060	3758	**3746**	**3746**	4131	3593	**3592**	**3592**	7657	6636	**6544**	6556	7604	5531	**5515**	5550
ibm17	5632	5583	5475	5440	**5145**	5663	4734	4716	**4649**	10177	9337	9294	**9271**	10213	8492	8561	**8399**
ibm18	3183	2918	2925	2922	**2917**	3091	2748	**2745**	**2745**	7003	5959	5849	**5825**	6970	4569	**4499**	4537
Norm Avg.	1.034	1.043	1	0.995	**0.984**	1.094	1	0.993	**0.988**	1.089	1	0.982	**0.975**	1.204	1	0.981	**0.975**



(a) sparcT1_core (b) stereo_vision

Figure 5: The jumping mechanism in our framework: As uncoarsening proceeds and the number of nodes increases, IMPart periodically exhibits a sharp decrease in cut size.

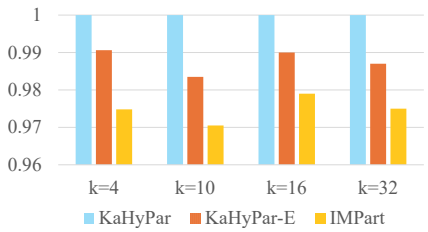


Figure 6: Normalized cut size comparison.

which then undergo continuous recombination and mutation during the uncoarsening and refinement stages. To establish a fair comparison, we configured a baseline where KaHyPar also generates seven initial solutions using the same seeds; however, each solution is processed independently through uncoarsening and refinement. For this baseline, the solution quality achieved using any of these seeds is roughly equivalent to that of a standard, single execution of KaHyPar. In Fig. 5, we present a comparison on two datasets from the Titan23 benchmark suite (`sparcT1_core` and `stereo_vision`). It visualizes the optimization process in uncoarsening, providing clear evidence of our framework's ability to escape local optima. The optimization trajectory of our method is characterized by distinct, sharp drops in cut size, in stark contrast to the baseline, which primarily shows gradual, steady refinement. These abrupt improvements demonstrate that our recombination operator successfully identifies globally superior solutions, effectively escaping local optima that trap conventional local search methods.

4.2.4 Scalability to Large-k Partitioning. In this section, we compare the normalized cut size of solutions produced by KaHyPar, KaHyPar-E, and our proposed method, IMPart, across various k-way partitioning scenarios. While the results for $k = 4$ and $k = 10$ are summarized from Tables 1 and 2, we extend this analysis to larger scenarios, specifically $k = 16$ and $k = 32$. The results presented in Figure 6 demonstrate that our method consistently achieves superior solutions compared to both KaHyPar and the memetic algorithm KaHyPar-E. This highlights the potential and robustness of our novel memetics-integrated multi-level framework for large-k-way hypergraph partitioning.

5 Conclusion

In this paper, we propose IMPart, a memetics-integrated multi-level framework. Unlike previous memetic algorithms that employ a complete hypergraph partitioner for each recombination and mutation, we designed novel recombination and mutation operators embedded directly into the uncoarsening phase of a single multi-level partitioning process. This design makes our approach significantly more lightweight and efficient. IMPart transforms the local searches of different granularities in the traditional multi-level framework into a sophisticated, collaborative search, thereby effectively escaping local search regions. Comprehensive experimental results demonstrate that our framework consistently yields significant improvements in solution quality over state-of-the-art methods for large-k-way hypergraph partitioning problems.

6 Acknowledgement

This work is supported by Hong Kong Research Grants Council (RGC) CRF-YCRG C6003-24Y, National Natural Science Foundation of China (NSFC) 92364102, and RGC T46-415/25-R. It was partially conducted by ACCESS – AI Chip Center for Emerging Smart Systems, supported by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government.

References

[1] Utku Umur Acikalin and Bugra Caskurlu. 2022. Multilevel memetic hypergraph partitioning with greedy recombination. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 168–171.

[2] Charles J Alpert. 1998. The ISPD98 circuit benchmark suite. In *Proceedings of the 1998 international symposium on Physical design*. 80–85.

[3] Robin Andre, Sebastian Schlag, and Christian Schulz. 2018. Memetic multilevel hypergraph partitioning. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 347–354.

[4] Shawki Areibi and Zhen Yang. 2004. Effective memetic algorithms for VLSI design= genetic algorithms+ local search+ multi-level clustering. *Evolutionary Computation* 12, 3 (2004), 327–353.

[5] Ismail Bustany, Grigor Gasparyan, Andrew B Kahng, Ioannis Koutis, Bodhisatta Pramanik, and Zhiang Wang. 2023. An open-source constraints-driven general partitioning multi-tool for VLSI physical design. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[6] Ismail Bustany, Andrew B Kahng, Ioannis Koutis, Bodhisatta Pramanik, and Zhiang Wang. 2022. SpecPart: A supervised spectral framework for hypergraph partitioning solution improvement. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 1–9.

[7] Ismail Bustany, Andrew B Kahng, Ioannis Koutis, Bodhisatta Pramanik, and Zhiang Wang. 2023. K-SpecPart: Supervised embedding algorithms and cut overlay for improved hypergraph partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 4 (2023), 1232–1245.

[8] Ümit V Çatalyürek and Cevdet Aykanat. 2011. PaToH (Partitioning Tool for Hypergraphs).

[9] Heming Chan and Pinaki Mazumder. 1993. A systolic architecture for high speed hypergraph partitioning using a genetic algorithm. In *Workshop on Evolutionary Computation*. Springer, 109–126.

[10] Magi Chen and Ting-Chi Wang. 2024. A Hypergraph Partitioner Utilizing a Novel Graph Generative Model. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*. 1–9.

[11] James Cohoon, John Kairo, and Jens Lienig. 2003. Evolutionary algorithms for the physical design of VLSI circuits. In *Advances in evolutionary computing: theory and applications*. Springer, 683–711.

[12] Charles M Fiduccia and Robert M Mattheyses. 1988. A linear-time heuristic for improving network partitions. In *Papers on Twenty-five years of electronic design automation*. 241–247.

[13] Lars Gottesbüren, Michael Hamann, and Dorothea Wagner. 2019. Evaluation of a flow-based hypergraph bipartitioning algorithm. *arXiv preprint arXiv:1907.02053* (2019).

[14] Lars Gottesbüren, Tobias Heuer, Nikolai Maas, Peter Sanders, and Sebastian Schlag. 2024. Scalable high-quality hypergraph partitioning. *ACM Transactions on Algorithms* 20, 1 (2024), 1–54.

[15] Lars Gottesbüren, Tobias Heuer, and Peter Sanders. 2022. Parallel flow-based hypergraph partitioning. *arXiv preprint arXiv:2201.01556* (2022).

[16] Lars Gottesbüren, Tobias Heuer, Peter Sanders, and Sebastian Schlag. 2021. Scalable Shared-Memory Hypergraph Partitioning. In *2021 Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 16–30.

[17] Juris Hartmanis. 1982. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review* 24, 1 (1982), 90.

[18] Alexandra Henzinger, Alexander Noe, and Christian Schulz. 2020. ILP-based local search for graph partitioning. *Journal of Experimental Algorithmics (JEA)* 25 (2020), 1–26.

[19] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. 1997. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proceedings of the 34th annual Design Automation Conference*. 526–529.

[20] Brian W Kernighan and Shen Lin. 1970. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal* 49, 2 (1970), 291–307.

[21] Rongjian Liang, Anthony Agnesina, and Haoxing Ren. 2024. Medpart: A multilevel evolutionary differentiable hypergraph partitioner. In *Proceedings of the 2024 International Symposium on Physical Design*. 3–11.

[22] Kevin E Murray, Scott Whitty, Suya Liu, Jason Luu, and Vaughn Betz. 2013. Titan: Enabling large and complex benchmarks in academic cad. In *2013 23rd International Conference on Field programmable Logic and Applications*. IEEE, 1–8.

[23] David A Papa and Igor L Markov. 2007. Hypergraph Partitioning and Clustering. *Handbook of Approximation Algorithms and Metaheuristics* 20073547 (2007), 61–1.

[24] Sebastian Schlag, Tobias Heuer, Lars Gottesbüren, Yaroslav Akhremtsev, Christian Schulz, and Peter Sanders. 2023. High-quality hypergraph partitioning. *ACM Journal of Experimental Algorithmics* 27 (2023), 1–39.

[25] Josef Schwarz and Jiří Očenášek. 1999. Experimental study: Hypergraph partitioning based on the simple and advanced genetic algorithm BMDA and BOA. In *Proceedings of the Mendel*, Vol. 99. 124–130.

[26] Justin Sybrandt, Ruslan Shaydulin, and Ilya Safro. 2020. Hypergraph partitioning with embeddings. *IEEE Transactions on Knowledge and Data Engineering* 34, 6 (2020), 2771–2782.

[27] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. 2006. Learning with hypergraphs: Clustering, classification, and embedding. *Advances in neural information processing systems* 19 (2006).