Cluster Editing: an evolutionary approach

Filippo Momesso, *University of Trento*

***Abstract*—This study investigates the application of evolutionary algorithms to CLUSTER EDITING, a notable combinatorial optimization challenge with real-world relevance. While traditional approaches have predominantly focused on classical optimization techniques, the potential of evolutionary strategies remained relatively uncharted. The study introduces specialized genomic representations and evolutionary operators tailored to the problem. However, experimental results reveal that even when evolutionary algorithms are augmented with local search strategies, drifting into the domain of memetic algorithms, they often underperform when compared to a heuristic local search. The findings underscore the challenges associated with navigating the expansive search space of high-dimensionality problems using evolutionary techniques, suggesting the need for further refinement in their application to CLUSTER EDITING. Code available at `evolutionary-cluster-editing` Github repository.**

I. INTRODUCTION

The CLUSTER EDITING problem is a significant combinatorial optimization challenge with broad applicability in real-world scenarios. It involves transforming an input graph into a collection of disjoint cliques, or clusters, through the addition or removal of a minimum number of edges. CLUSTER EDITING finds applications in diverse domains such as bioinformatics, social network analysis, and image segmentation. Researchers have proposed various algorithmic approaches to tackle this NP-hard problem, aiming to strike a balance between computational complexity and solution quality.

Evolutionary computation represents a thriving field of research, offering effective solutions to numerous NP-hard problems, including the Traveling Salesman Problem. In this context, this work seeks to explore the potential advantages of employing evolutionary algorithms in the domain of cluster editing.

The conducted experiments, however, demonstrate that a $\mu + \lambda$ genetic algorithm, equipped with customized mutation and crossover operators, falls short in delivering competitive results when compared to a relatively straightforward local search approach. The integration of local search into the evolution process, transitioning to the domain of memetic algorithms, yields slightly improved results at the cost of increased computation time. This approach enhances the algorithm's performance in addressing the cluster-editing problem, making it a promising avenue for further exploration.

A. Problem definition

Let $G(V, E)$ be an undirected graph, where V represents the set of vertices and E represents the set of edges. CLUSTER EDITING aims to transform G into a collection of disjoint cliques, or clusters, through the introduction or removal of a minimum number of edges. Figure 1 shows an example of the problem and one possible solution.

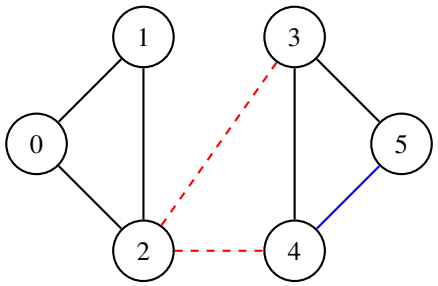

Fig. 1. Example of a graph and its solution to CLUSTER EDITING. Removed edges are in red while added edges are in blue.

The problem has been proved NP-hard several times [1]–[3], while it can be solved in polynomial time for particular type graphs such as unit interval graphs (1-dimensional vicinity graphs) since clusters in an optimal solution are consecutive sets [4], and graphs of maximum degree two [5].

The parametrized version of the problem expresses the number of modifications as a parameter k, a cost limit that is given in advance. Through this formulation, cluster editing is recast to a decision problem in which the algorithm has to answer if a solution with cost $\leq k$ exists. This work, however, focuses on solving the non-parametrized version.

Furthermore, CLUSTER EDITING can be seen as a constrained optimization task, where the primary constraint demands that each connected component in the modified graph forms a clique. Assessing the feasibility of a solution can be executed in $O(|V| + |E|)$ time, which encompasses the complexity of the DFS visit to find the connected components and an iteration on all the vertices to check that $degree(v_{i,j}) = |c_i| : \forall v_{i,j} \in c_i, \forall c_i \in C$ where C is the set of connected components.

B. Data

To facilitate the experimental analysis, a diverse set of test cases has been meticulously generated[1], comprising graphs with varying levels of complexity for the Cluster Editing Problem. These test cases encompass graphs of different sizes, considering both the number of vertices ($|V|$) and edges ($|E|$), as well as exhibiting a broad spectrum of graph characteristics and shapes. The generated graphs belong to the following categories:

- Cliques connected by a single edge (ring, line, tree of cliques).
- Cactus graphs where each cycle is transformed in a clique.
- Quasi-cliques connected by a limited number of edges.

[1]I am grateful for the support by the Data Structures and Algorithms tutors team, to which I belong, in providing me with a portion of the test cases.

- Partition-graphs.
- Stochastic block model graphs.

Each class of test case exhibits specific characteristics that demand different strategies to achieve a solution. For instance, in the first two categories, the optimal solution involves merely removing edges that connect different cliques, while the others require both edge additions and deletions. In the last three cases, the algorithm must identify communities, separate them, and add the necessary edges to transform each community into a clique. This diversity of graph structures challenges the algorithmic approaches, ensuring a comprehensive assessment of their performance across various scenarios. A summary of the test cases used can be found in Table I

For each test case, lower and upper bounds have been computed. The upper bound is computed as the number of edge removals such that each node of the modified graph has at most degree two. The lower bound is computed as described in KaPoCE [6], which uses subgraph packing techniques.

TABLE I
TEST CASES

Graph	Nodes	Edges	L. Bound	U. Bound
cliques_ring	500	2400	50	2239
cliques_binary_tree	2000	6216	229	5587
cliques_quasicliques	5000	123 523	9883	122 205
cliques_tree_small	1000	9549	49	9263
cliques_tree_big	5000	249 731	49	248 414
cactus_small	1321	15 242	868	15 103
cactus_big	2945	33 664	2012	33 306
partition_small	543	46 017	33 794	45 862
partition_big	1077	177 208	132 126	176 942
sbm_A	1180	330 343	219 341	330 079
sbm_B	1195	143 408	113 469	143 206
sbm_C	1100	97 277	75 203	97 058
sbm_D	1040	124 759	96 510	124 521

II. METHODOLOGIES

The current section describes the algorithms investigated in this study. It starts by introducing the genomic representation strategy, followed by the heuristic local search algorithm, which serves as the baseline for comparison against the genetic and memetic algorithms explained in the subsequent sections.

A. Genomic representation

Each individual or solution is represented as an array of $|V|$ integers, which represent the cluster assignment for each node in the graph. This representation offers several advantages, allowing for: *a*) compact and efficient representation of each individual, *b*) efficiently compute the modified graph selectively, *c*) design genetic operators with ease, and *d*) transform the problem from constrained to unconstrained.

The latter advantage is particularly important since it allows to avoid the overhead of constrained optimization algorithms. With such representation, the modified graph is, by construction, always a feasible solution. The modified graph can be computed in $O(|V|^2)$ time, by the addition or removal of edges based on the cluster assignment defined by the genomic representation. With such representation the solution shown in Figure 1 is represented by the genome $[\,0,\,0,\,0,\,1,\,1,\,1\,]$.

B. Objective function

As discussed so far, in CLUSTER EDITING the objective function is the number of edge modifications. Evaluating a solution based on either the modified graph or the cluster assignment vector, as outlined in Section II-A, demands a time complexity of $O(|V|^2)$. Therefore, due to this quadratic complexity, it is crucial to minimize the number of evaluations performed when the algorithms are executed.

Through the use of auxiliary data structures, it is possible to efficiently update the fitness of a solution each time a modification to the individual is performed. Details are reported in Appendix A.

C. Heuristic local search

The local search algorithm used as a baseline to compare the genetic and memetic algorithms proposed is based on the strategies employed in KaPoCE heuristic solver [6]. Starting from an initial solution in which every node is assigned to its own cluster, heuristic refinement techniques are applied iteratively. Each iteration is composed of three steps: *a*) perturbation of the current solution, by means of mutation operators defined in Section II-D2, *b*) a round of label propagation heuristic, and *c*) a round of clique removal heuristic.

1) Label propagation: The label propagation algorithm is a well-known local search heuristic, with various application which was successfully used for cluster editing [7]. It works by visiting the vertices in random order and, for each vertex, moving it to a cluster that minimizes the objective function, i.e. the number of edit operations. Furthermore, the algorithm takes into account also isolation from all existing clusters.

2) Clique removal: The clique removal heuristic [7] iterates the clusters in random order and, for each cluster, tries to remove it by assigning each of its nodes to a cluster that minimizes the objective function (node isolation is also considered).

D. Genetic algorithm

The algorithm employed is a $\mu + \lambda$ genetic algorithm, which, starting from an initial population, evolves individuals represented as described in Section II-A, minimizing the fitness function defined in Section II-B. At each generation, λ offsprings are generated through crossover, performed with probability $p_{crossover}$, and mutation, performed with probability $p_{mutation}$. The population and the newly generated offspring are concatenated and a new population of μ individuals is selected through tournament selection. The $\mu + \lambda$ strategy, combined with tournament selection, allows a good trade-off between elitism and exploration.

1) Population initialization: An initial population of size μ is generated randomly. Given the high dimensionality search space, characterized by each individual's representation involving $|V|$ genes, the initial cluster assignments are constrained to selections from a pool of $max\_clusters = 0.1 \cdot |V|$ clusters. Furthermore, heuristic-generated individuals are added to the initial population. The heuristics used include: *a*) cluster assignments matching the connected components of the graph, *b*) each node belonging to the same cluster.

2) Mutation operator: Three types of mutations have been implemented. For each mutation step, only one mutation is chosen according to its dedicated probability.

a) `node_isolation`: a subset of nodes is randomly chosen and isolated from their cluster. The rationale is that the introduction of new clusters might be beneficial for exploration. However, this mutation has the downside of rapidly increasing the number of clusters in a solution, leading to an explosion of the search space. Therefore this mutation has a low probability and is applied to a limited number of nodes.

b) `node_mover`: a subset of nodes is randomly chosen and each node is moved to a random existing cluster. This strategy contributes to solution diversification by encouraging nodes to interact with different clusters, potentially uncovering previously unexplored configurations.

c) `clique_removal`: a subset of clusters is randomly chosen; each cluster is removed and their nodes are assigned to random clusters. The aim of this strategy is to promote the exploration of alternative clustering configurations by systematically disassembling clusters and redistributing their elements.

3) Crossover operator: Conventional integer genome crossover operators, such as one-point, two-point, or uniform crossover, lack the capacity to effectively exploit existing solutions, as they disregard the problem's intrinsic structure.

Therefore, a specialized crossover operator has been integrated. This *common cluster crossover* identifies shared clusters between two parent individuals and generates offspring by retaining node assignments related to common clusters, while assignments of nodes in non-shared clusters are selected from either parent with equal probability. If no common clusters are identified, the algorithm defaults to uniform crossover. An example is shown in Figure 2.

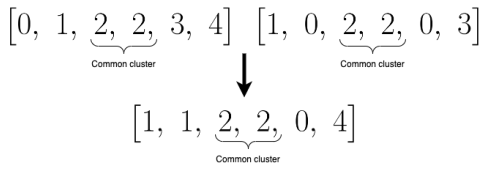

Fig. 2. Example of the Common Cluster Crossover operator.

E. Memetic algorithm

As shown in Section III, despite the dedicated operators the genetic algorithm is not able to obtain competitive results compared to the heuristic local search algorithm. To overcome its limitations, local search strategies have been integrated into the genetic algorithm, transitioning to the domain of memetic algorithms. In this setting, following mutation and crossover, a round of label propagation and clique removal heuristic is performed on each newly generated offspring with probability p_{local_search}.

Experiments of the effectiveness of applying these search algorithms on a randomized subset of nodes for each new individual have thus been conducted. Executing such search

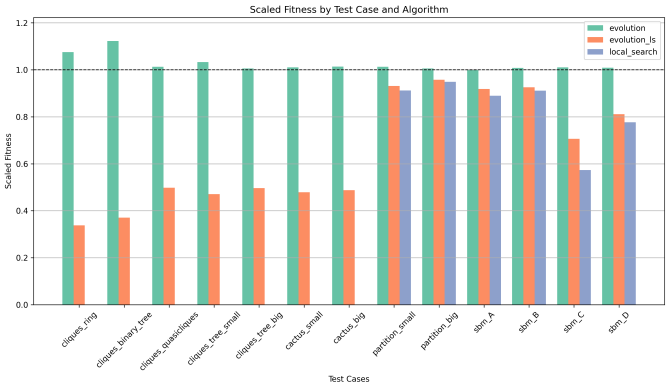

Fig. 3. Comparison of minimum fitness solutions for each test case between genetic algorithm, local search only, and memetic algorithm.

algorithms on all nodes presents significant computational demands, particularly when performed across numerous individuals. Therefore, it becomes crucial to discern whether constraining the search space might yield improved results, while effectively managing the computational overhead posed by both evolution and local search processes.

III. Experimental results

Comprehensive experiments were conducted on the graphs outlined in Section I-B utilizing the DEAP framework for evolutionary computation. These tests were executed on an Intel Core i7-7800X CPU clocked at 3.50GHz. Notably, no multiprocessing techniques were employed during the computations.

Due to the high variance in the graphs, in order to compare the performances of the different algorithms, the fitness has been scaled using the lower and upper bound values for each test case as $fitness\_norm = \frac{fitness - lb}{ub - lb}$. The evolutionary-based experiments discussed in this section are performed on a population of 100 random individuals, with mutation and crossover probability of 0.5. Custom mutations probabilities are 0.45, 0.45, and 0.1 for `node_mover`, `clique_removal` and `node_isolation` respectively. The memetic algorithm performs local search with $p_{local\_search} = 0.5$, on a 10% random subset of each individual's nodes.

Figure 3 showcases a comparison between the best fitness reached by the three algorithms after 25 generations. It can be noticed that the evolutionary algorithm, in almost every case, is not able to surpass the naive upper bound described in Section I-B. In several instances moreover, the heuristic local search algorithm described in Section II-C manages to reach the optimal solution (i.e. the lower bound), while both the genetic and the memetic algorithms fail. In partition graphs and stochastic block models graphs, however, the genetic algorithm combined with local search is able to reach similar fitness, though slightly higher.

A deeper examination of the time taken to discover the minimal fitness solution indicates a superior performance by the heuristic local search. Furthermore, it emerges an interesting correlation between the execution time of the genetic

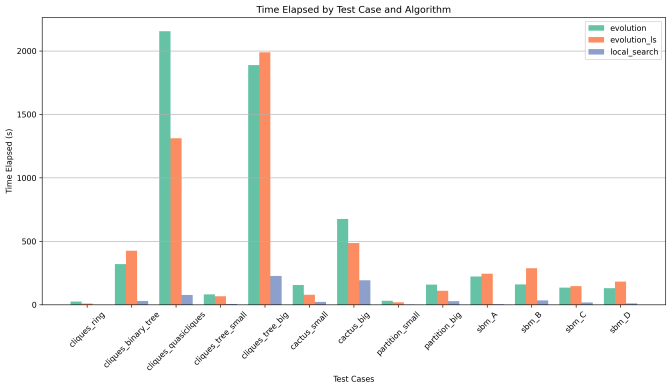

Fig. 4. Comparison of the time elapsed to find the minimum fitness solution for each test case between genetic algorithm, local search only, and memetic algorithm.

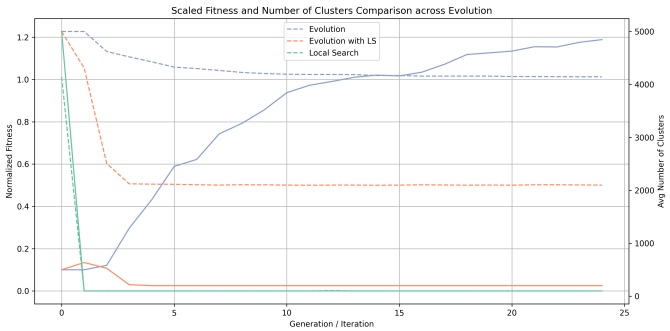

Fig. 5. Fitness and average number of clusters during the evolution/optimization process for the three algorithms on `cliques_quasicliques` test case.

and memetic algorithms and the seemingly easy test cases, which are graphs where the solution can be found by just removing edges that connect cliques, or by removing or adding a limited number of edges. Inspecting the average number of clusters during the evolution, as shown in Figure 5 for one of such test cases, it appears that the genetic algorithm focuses on isolating nodes in order to minimize the fitness, due to the `node_isolation` mutation operator. Indeed the upper bound is a graph where each node has maximum degree 2, and the optimization is guided towards this *easy* solution while the true optimum would require identification and consolidation of cliques or near-cliques. It is likely that the search space is too large to be explored efficiently by the evolution process, although specialized mutation and crossover operators have been implemented.

A. Ablation study

Given the observations in Section III, it is crucial to understand how impactful are the various components of the genetic algorithm. This ablation study is performed on test cases `cliques_quasicliques` and `sbm_C`, whose results can be found in Table II.

Results show that `clique_removal` and `node_mover` mutation operators have no significant role in the optimization process, which is powered mainly by the `node_isolation` operator. However, as suspected in the previous section, the latter leads the solution toward the isolation of the nodes, which is a suboptimal but valid solution. Due to the high dimensionality of the problem, there exist too many possible configurations to be searched by uninformed evolution. The addition of local search in the evolution process allows for better solutions, yet results show that evolution plays a marginal role in the optimization.

TABLE II
ABLATION STUDY ON EVOLUTIONARY OPERATORS.

move	remove	isolate	LS	crossover	`cliques_quasicliques`	
					Fitness ↓	# Clusters
✓	✗	✗	✗	✓	147 075	500
✓	✓	✗	✗	✓	147 299	500
✓	✓	✓	✗	✓	123 627	4096
✗	✗	✗	✓	✓	65 379	200
✓	✓	✓	✓	✓	65 833	200
move	remove	isolate	LS	crossover	`sbm_C`	
					Fitness ↓	# Clusters
✓	✗	✗	✗	✓	100 593	110
✓	✓	✗	✗	✓	100 505	110
✓	✓	✓	✗	✓	97 284	1081
✗	✗	✗	✓	✓	90 679	48
✓	✓	✓	✓	✓	90 643	49

IV. Difficulties

The project's development encountered multiple challenges. Firstly, while the literature has thoroughly explored the problem, the emphasis has been on classical optimization techniques such as heuristic local search, branch and bound, integer linear programming, and parametrized algorithms, and no relevant research on evolutionary approaches has been found. The second challenge was discerning an effective representation of the problem, determining the appropriate operators, and implementing them within the DEAP framework. Lastly, there was difficulty in ensuring the evolutionary algorithm's competitiveness with the basic local search technique, a challenge that remained unresolved.

V. Conclusion

This study investigated the application of evolutionary computation in the domain of Cluster Editing, a significant combinatorial optimization challenge. While the literature predominantly emphasizes classical optimization techniques, the potential of evolutionary algorithms in addressing this problem remained underexplored. Despite the rigorous attempts to harness the strengths of evolutionary algorithms, including the design of specialized genomic representations and dedicated operators, results demonstrated that the evolutionary approach, even when enhanced with local search strategies (memetic algorithms), failed to consistently outperform the straightforward heuristic local search. Particularly, the evolutionary strategy often gravitated towards suboptimal solutions, highlighting the challenge posed by the search space, especially for large graphs. These findings suggest that while evolutionary methods are promising in many domains, their application to Cluster Editing requires further refinement and innovation to achieve competitive results.

REFERENCES

[1] N. Bansal, A. Blum, and S. Chawla, "Correlation clustering," vol. 56, 02 2002, pp. 238 – 247.

[2] S. Delvaux and L. Horsten, "On best transitive approximations to simple graphs," *Acta Inf.*, vol. 40, pp. 637–655, 09 2004.

[3] R. Shamir, R. Sharan, and D. Tsur, "Cluster graph modification problems," *Discrete Applied Mathematics*, vol. 144, no. 1, pp. 173–182, 2004, discrete Mathematics and Data Mining. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166218X04001957

[4] B. Mannaa, "Cluster editing problem for points on the real line: A polynomial time algorithm," *Inf. Process. Lett.*, vol. 110, pp. 961–965, 10 2010.

[5] S. Böcker, "A golden ratio parameterized algorithm for cluster editing," *Journal of Discrete Algorithms*, vol. 16, pp. 79–89, 2012, selected papers from the 22nd International Workshop on Combinatorial Algorithms (IWOCA 2011). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570866712000597

[6] T. Bläsius, P. Fischbeck, L. Gottesbüren, M. Hamann, T. Heuer, J. Spinner, C. Weyand, and M. Wilhelm, "KaPoCE - An Exact and Heuristic Solver for the Cluster Editing Problem," Jun. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4892524

[7] L. Bastos, L. Ochi, F. Protti, A. Subramanian, I. Martins, and R. Pinheiro, "Efficient algorithms for cluster editing," *Journal of Combinatorial Optimization*, vol. 31, 05 2014.

APPENDIX A
EFFICIENT FITNESS UPDATE

Through the use of auxiliary data structures such as 1) the *graph adjacency list* (implemented as a dictionary of sets), and 2) the *cluster mapping dictionary*, which returns the set of nodes assigned to a cluster given its index, given a valid solution, it is possible to efficiently compute the fitness of the solution obtained by moving a node to a given cluster, without the need of performing a complete evaluation of the individual after such modification.

The *cost update algorithm* is quite simple and works in two phases:

1) Compute the fitness of the solution after *virtually* isolating the node.

$$\begin{aligned} cost_{isolation} = cost_{current} \\ + |old\_cluster\_neighbors| \\ - |old\_cluster\_non\_neighbors| \end{aligned} \tag{1}$$

2) Compute the fitness of the solution after *virtually* adding the node to the new cluster.

$$\begin{aligned} cost_{new} = cost_{isolation} \\ + |new\_cluster\_non\_neighbors| \\ - |new\_cluster\_neighbors| \end{aligned} \tag{2}$$