AIMS Mathematics, 10(6): 13365–13392.
DOI: 10.3934/math.2025600
Received: 14 February 2025
Revised: 01 June 2025
Accepted: 04 June 2025
Published: 10 June 2025

https://www.aimspress.com/journal/Math

Research article

An adaptive greedy repair operator in a genetic algorithm for the minimum vertex cover problem

Seung-Hyun Moon[1] **and Yourim Yoon**[2,*]

[1] School of Software, Kwangwoon University, 20 Kwangwoon-ro, Nowon-gu, Seoul 01897, Korea

[2] Department of Computer Engineering, Gachon University, 1342 Seongnamdaero, Sujeong-gu, Seongnam-si, Gyeonggi-do 13120, Korea

* **Correspondence:** Email: yryoon@gachon.ac.kr; Tel: +82317505326.

Abstract: A genetic algorithm (GA) evolves the population of candidate solutions to a particular problem through crossover and mutation operators. Since a newly generated solution may not satisfy the constraints of the given problem, it is common to penalize infeasible solutions or to repair them to feasible solutions. For the minimum vertex cover (MVC) problem, we propose an adaptive greedy repair operator. Our repair operator first repairs an infeasible solution into a feasible one using a randomized greedy algorithm, and then removes unnecessary vertices from the feasible vertex cover, making it a minimal vertex cover. During the repair process, when adding or removing vertices, the degree of exploration and exploitation in the randomized greedy algorithm is adaptively adjusted based on the convergence level of the population. When the population lacks high-quality solutions, the operator strives to generate superior solutions greedily. However, when the solution set has enough high-quality solutions, it explores unexplored choices to break through the limitations of the existing solution set. We compared our GA with a deterministic greedy algorithm, a randomized greedy algorithm, and GAs using various repair operators. Experimental results on benchmark graphs from the Benchmarks with Hidden Optimum Solutions Library (BHOSLIB) demonstrated that the proposed repair operator improved the performance of the GA for the MVC problem.

Keywords: genetic algorithm; repair operator; greedy algorithm; approximation algorithm; vertex cover

Mathematics Subject Classification: 68T20

1. Introduction

In graph theory, a vertex cover of a graph is a set of vertices that contains at least one endpoint of every edge of the graph. In other words, each edge of the graph must be 'covered' by at least one vertex from the vertex cover. For example, in a connected graph $G = (V, E)$, the set of all vertices V would be a trivial vertex cover.

The vertex cover has many real-world applications including circuit design, telecommunications, and network flow [1]. For instance, it can be used to minimize the installation cost of a surveillance camera in transportation networks [2], and it can be used to effectively block the spread of worms in computer networks [3]. In biology, the problem was used to model the removal of repetitive DNA sequences for synthetic biology and metabolic engineering [4, 5].

The problem of finding the minimum vertex cover (MVC) consisting of the smallest number of vertices is NP-hard (where NP stands for Non-deterministic Polynomial time) [6], meaning that there is no known polynomial-time algorithm to solve it. Moreover, the MVC cannot be approximated arbitrarily well even in bounded degree graphs [7], and it was shown that the approximation factor of the MVC is at least $\sqrt{2} - \epsilon$ for any $\epsilon > 0$ [8]. According to the unique games conjecture [9], it may be impossible to achieve a constant approximation factor of less than 2.

However, there are simple algorithms that guarantee an approximation ratio of 2 for the MVC problem. The size of the vertex cover that consists of all endpoints of a maximal matching $M \subseteq E$ is at most two times the number of the optimum solution [10]. On the other hand, the vertex-based greedy algorithm starts with an empty set and adds one vertex that covers the most uncovered edges at each step. This algorithm has an approximation ratio of $\Theta(\log |V|)$, but randomizing this process by selecting a vertex with a probability proportional to the number of uncovered edges that it can cover gives an approximation ratio of 2 [11].

In addition to approximation algorithms, many attempts have been made to solve the MVC problem effectively. Shyu et al. [1] implemented ant colony optimization, where simulated ants mimic the behavior of real ants by exploring the graph of the problem they are solving. Bouamama et al. [12] used a population-based iterated greedy algorithm, which maintains a population of solutions and applies the basic steps of an iterated greedy algorithm to each member of the population. Cai et al. [13] developed a comprehensive algorithm which includes a construction heuristic, local search, and a preprocessing technique. Jovanovic et al. [14] introduced the fixed set search metaheuristic, which enhances solution quality by integrating a learning mechanism into a population-based framework.

Additionally, several studies have applied the genetic algorithm (GA) to tackle the MVC problem. Khuri and Bäck [15] showed that GA outperforms a traditional heuristic algorithm in solving MVC. Karci and Arslan [16] developed a GA that uses both randomly generated chromosomes and their inversion counterparts when creating the initial population. Singh and Gupta [17] introduced a hybrid GA that embeds a greedy local search heuristic, while Nagy and Szokol [18] proposed a GA with interval-valued fitness and a greedy repair operator.

The GA [19] is a metaheuristic inspired by the principles of natural selection and evolutionary

biology. They belong to the family of evolutionary algorithms and utilize mechanisms such as selection, crossover, and mutation to iteratively evolve a population of candidate solutions toward an optimal or near-optimal solution. By simulating the process of survival of the fittest, GAs are effective in exploring large, complex search spaces and solving problems where conventional optimization methods may struggle. Each individual in the population represents a potential solution, and the fitness function guides the evolutionary process by determining the quality of these solutions. GAs are widely applicable in diverse fields such as bioinformatics [20–22], financial engineering [23–25], and operational research [26, 27].

Throughout the evolutionary process of a GA, the stochastic nature of genetic operators such as crossover and mutation often generate infeasible solutions that violate problem constraints. To address this issue, the repair operator is introduced to transform infeasible solutions into feasible ones while preserving their genetic characteristics as much as possible. By incorporating repair operators, GAs can maintain a valid promising solution pool, enhance convergence rates, and effectively explore the search space without violating critical constraints [28–30].

To the best of our knowledge, previous studies that have used GA to solve the MVC problem have either penalized infeasible solutions or employed greedy algorithms to repair solutions [15–18]. In this paper, we introduce a novel adaptive greedy repair operator for the MVC problem, without the intention of surpassing the state-of-the-art results. For readers interested in state-of-the-art heuristics for the MVC problem, we refer them to [13, 31, 32], which provide comprehensive insights into recent advanced approaches.

We use a randomized greedy algorithm to cover all uncovered edges, and then construct a minimal vertex cover by removing unnecessary vertices from the feasible vertex set. The balance between exploration and exploitation in the repair operator is adaptively adjusted based on the convergence level of the population. Experiments using different repairing schemes were conducted on BHOSLIB (Benchmarks with Hidden Optimum Solutions Library) benchmark graphs [33], which are designed to test several NP-hard graph algorithms, including maximum clique, maximum independent set, vertex coloring, and MVC.

The rest of the paper is organized as follows: in Section 2 we provide the formal definition of the MVC problem and introduce vertex-based approximation algorithms. In Section 3 we provide genetic framework and propose an adaptive greedy repair operator utilizing the approximation algorithms. In Section 4, we present the experimental results and discussions. Finally, we draw conclusions in Section 5.

2. Preliminaries

A vertex cover of a graph $G = (V, E)$ is a subset of vertices $V' \subseteq V$ such that every edge $e \in E$ has at least one endpoint in V'. For a vertex $v \in V$ and a solution X to the MVC problem, $gain(v, X)$ represents the number of uncovered edges that would be covered by adding v to X, while $loss(v, X)$ denotes the number of edges that would become uncovered by removing v from X. The degree of a

vertex v is denoted by $deg(v)$.

A vertex-based greedy algorithm for the MVC problem is given in Algorithm 1. The greedy algorithm builds a vertex cover incrementally by repeatedly choosing vertices that cover the most uncovered edges. This deterministic algorithm achieves a $\Theta(\log |V|)$ approximation. Gao et al. [11] randomized the deterministic greedy algorithm and showed that their randomized version has an α-approximation in $2\ln(2) \cdot 2^{(2-\alpha)\cdot|\mathrm{OPT}|}$ repeated runs with probability at least 0.5, where $1 \leq \alpha \leq 2$ and OPT denotes the size of the optimal solution, i.e., the size of the minimum vertex cover. The randomized greedy algorithm for the MVC problem is given in Algorithm 2. This algorithm uses γ to determine how to select vertices. When γ is 0, vertices are selected uniformly. As γ increases, the selection becomes progressively biased toward vertices that cover a larger number of uncovered edges. That is, for $\gamma > 0$, vertices with higher coverage have a greater probability of being selected. For very large values of γ, the algorithm behaves similarly to a deterministic greedy algorithm. The authors reported achieving decent performance when using γ values between 20 and 1000.

Algorithm 1 Deterministic greedy algorithm for MVC.

Input: A graph $G = (V, E)$
Output: A vertex cover $V' \subseteq V$
$V' := \emptyset$
while V' is not a vertex cover of $G(V, E)$ **do**
    Select a vertex $u \in (V - V')$ that has the highest $gain(u, V')$, breaking ties in favor of the vertex with the smaller index
    $V' := V' \cup \{u\}$
end while

Algorithm 2 Randomized greedy algorithm for MVC

Input: A graph $G = (V, E)$
Output: A vertex cover $V' \subseteq V$
$V' := \emptyset$
while V' is not a vertex cover of $G(V, E)$ **do**
    Select a vertex $u \in (V - V')$ with the probability
        $p(u) = gain(u, V')^\gamma / \sum_{v\in(V-V')} gain(v, V')^\gamma$
    $V' := V' \cup \{u\}$
end while

We devise a novel repair operator for GA by developing the randomized algorithm of Gao et al. [11], and the GA with the repair operator is introduced in the next section.

3. Methods

We use a generational genetic algorithm whose pseudocode is given in Algorithm 3. The algorithm begins by generating a random initial population of candidate solutions, each representing a vertex

cover. A repair operator is immediately applied to every individual to ensure feasibility. In each subsequent generation, the entire population is replaced. For each individual, crossover is applied with probability χ using a selected mate from the population; otherwise, the individual is passed to the next generation unchanged. The resulting offspring undergoes mutation followed by a repair step to ensure that it remains a valid solution. The repaired offspring then replaces its predecessor in the population. Each individual is represented by a binary chromosome, where i-th gene in the chromosome indicates whether i-th vertex is included in the vertex cover (V') or not. Figure 1 illustrates the encoding scheme used in this study.

We begin by generating a random initial population. Each randomly created chromosome is then modified using a repair operator to ensure that it represents a feasible vertex cover. The resulting population of feasible solutions is evolved using a genetic algorithm framework. As the fitness function, we employ $|V - V'|$, which denotes the number of vertices not included in the vertex cover. This formulation encourages smaller vertex covers by assigning higher fitness values to solutions with fewer selected vertices.

Algorithm 3 The outline of our GA

```
Create initial population P
for i ← 1 to |P| do                                  ▷ repairs initial population
    Repair(P[i])                                     ▷ P[i] is the i-th chromosome
end for
for g ← 2 to n do                                    ▷ n is the last generation
    for i ← 1 to |P| do                              ▷ replaces the entire population
        x ~ Uniform(0, 1)
        if x < χ then                                ▷ χ is a crossover rate
            (parent1, parent2) := (P[i], Select(P))
            offspring := Crossover(parent1, parent2)
        else
            offspring := P[i]
        end if
        Mutation(offspring)
        Repair(offspring)
        Replace(P[i], offspring)
    end for
end for
```

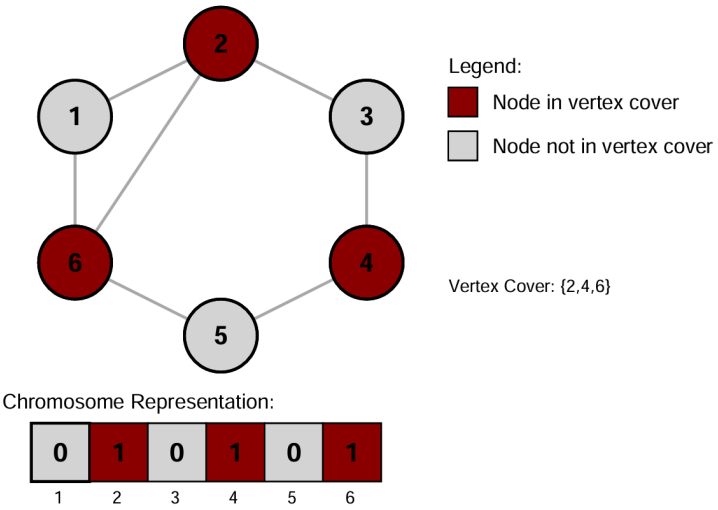


Figure 1. Visualization of a chromosome encoding for a graph with six vertices.

3.1. Repair operators

We devised an adaptive greedy algorithm that effectively balances exploration and exploitation. For comparative evaluation, we developed and tested three additional approaches: Uniform repair, deterministic greedy repair, and randomized greedy repair. Each of these algorithms is based on the greedy algorithms discussed in Section 2. Unlike greedy algorithms that may return non-minimal covers, our repair operators guarantee a *minimal* vertex cover, in the sense that no included vertex can be removed without violating feasibility.

The uniform repair operator constructs a vertex cover by uniformly selecting vertices with uncovered edges. This vertex addition process is equivalent to the randomized greedy algorithm with γ set to zero. Once all edges are covered, it identifies any redundant vertices in the vertex cover and uniformly removes them. The pseudocode for the uniform repair operator is presented in Algorithm 4.

Algorithm 4 Uniform repair for MVC

Input: A graph $G = (V, E)$ and a solution $V' \subseteq V$
Output: A minimal vertex cover $V' \subseteq V$

while V' is not a vertex cover of $G(V, E)$ **do** ▷ V' is a solution to be repaired
  Select a vertex $u \in (V - V')$ with the probability ▷ covers uncovered edges
    $p(u) = \mathbb{1}_{gain(u,V')>0} / \sum_{v\in(V-V')} \mathbb{1}_{gain(v,V')>0}$ ▷ $\mathbb{1}$ is an indicator function
  $V' := V' \cup \{u\}$
end while
while There is a $u \in V'$ with $loss(u, V') = 0$ **do**
  Select a vertex $u \in V'$ with the probability ▷ removes unnecessary vertices
    $p(u) = \mathbb{1}_{loss(u,V')=0} / \sum_{v\in V'} \mathbb{1}_{loss(v,V')=0}$
  $V' := V' \setminus \{u\}$
end while

The deterministic greedy repair adds vertices in the same way as the deterministic greedy algorithm, which always selects the vertex that covers the largest number of uncovered edges. After all edges are covered, it removes unnecessary vertices in order of lowest degree first. When adding or removing vertices, ties are broken in favor of the vertex with the smaller index. The pseudocode for the deterministic greedy repair operator is given in Algorithm 5.

Algorithm 5 Deterministic greedy repair for MVC

Input: A graph $G = (V, E)$ and a solution $V' \subseteq V$
Output: A minimal vertex cover $V' \subseteq V$
while V' is not a vertex cover of $G(V, E)$ **do**
    Select a vertex $u \in (V - V')$ that has the highest $gain(u, V')$
    $V' := V' \cup \{u\}$
end while
while There is a $u \in V'$ with $loss(u, V') = 0$ **do**
    Select a vertex $u \in V'$ that has the lowest $deg(u)$
    $V' := V' \setminus \{u\}$
end while

Randomized greedy repair adds vertices in the same manner as the randomized greedy algorithm, where vertices that cover a larger number of uncovered edges are more likely to be selected. Once all edges are covered, unnecessary vertices are removed, with lower-degree vertices being more likely to be removed. The greediness parameter γ is provided as a constant value. The pseudocode for the deterministic greedy repair operator is given in Algorithm 6.

Algorithm 6 Randomized greedy repair for MVC

Input: A graph $G = (V, E)$ and a solution $V' \subseteq V$
Output: A minimal vertex cover $V' \subseteq V$
while V' is not a vertex cover of $G(V, E)$ **do**
    Select a vertex $u \in (V - V')$ with the probability
        $p(u) = gain(u, V')^\gamma / \sum_{v \in (V - V')} gain(v, V')^\gamma$
    $V' := V' \cup \{u\}$
end while
$max\_deg := \max_{v \in V'} \mathbb{1}_{loss(v,V')=0} \cdot deg(v)$ ▷ the maximum degree among removable vertices
$min\_deg := \min_{v \in V'} \mathbb{1}_{loss(v,V')=0} \cdot deg(v)$ ▷ the minimum degree among removable vertices
for each $u \in V'$ **do** ▷ among removable vertices, the lower the degree, the higher its badness
    $bad(u) = \mathbb{1}_{loss(u,V')=0} \cdot (max\_deg - (deg(u) - min\_deg))$
end for
while There is a $u \in V'$ with $loss(u, V') = 0$ **do**
    Select a vertex $u \in V'$ with the probability
        $p(u) = bad(u)^\gamma / \sum_{v \in V} bad(v)^\gamma$
    $V' := V' \setminus \{u\}$
end while

Adaptive greedy repair is based on randomized greedy repair. Unlike randomized greedy algorithm which uses a fixed gamma parameter to control exploration and exploitation, adaptive greedy repair adjusts gamma according to the convergence level of the population. Let $best(P)$ denote the number of best solutions in population P, where best solutions are determined based on fitness value. After generating the initial population, adaptive greedy repair sets γ to $|P|/best(P)$. This approach increases exploration as the population converges toward the best solution and strengthens exploitation to increase the number of best solutions when $best(P)$ is low. Since all other aspects are identical to randomized greedy repair except for how gamma is set, we omit the pseudocode.

Claim 1. *Our repair operators always produce a minimal vertex cover.*

Proof. Let $G = (V, E)$ be an undirected graph and $V' \subset V$ be a solution of the vertex cover. Our repair operators proceed in two phases:

Phase 1 (Construction): While V' is not a vertex cover, select a vertex v with an uncovered edge $e = \{u, v\} \in E$ and add it to V'. Repeat this process until every edge in E is covered by V'; that is, for all $e \in E$, we have $e \cap V' \neq \emptyset$. At this stage, V' is a vertex cover.

Phase 2 (Reduction): For each vertex $v \in V'$, check whether $V' \setminus \{v\}$ is still a vertex cover. If so, remove v from V'. Continue this process until no further vertex can be removed without violating the vertex cover condition.

We now prove that the final set V' is a minimal vertex cover. By construction, it is a vertex cover. Suppose, for contradiction, that it is not minimal. Then, there exist some $v \in V'$ such that $W = V' \setminus \{v\}$ is also a vertex cover. However, such a vertex would have been removed during the reduction phase, contradicting the fact that the repair operation terminated with $v \in V'$.

Therefore, for all $v \in V'$, the set $V' \setminus \{v\}$ is not a vertex cover, and V' is minimal. □

3.2. Selection

In GA, selection serves as the source of exploitation [34], enabling the algorithm to progress toward desired improvements [35]. In this study, we employ stochastic tournament selection as the selection operator. Tournament selection enhances the search for optimal solutions by conducting a competition among s participants, where s represents the tournament size. Each participant is given a chance to be selected in descending order of fitness, with a probability p. Here, p represents the selection pressure. Participants with higher fitness values have a greater probability of survival and selection in the tournament, with winners being placed into the mating pool. As a result, this mating pool exhibits an average fitness value superior to that of the general population [36]. Our method implements random sampling with a tournament size of eight, meaning that each tournament selection round involves a competition among eight individuals from the population. The winner of each tournament is then selected as a parent chromosome. The pseudocode for this stochastic selection is presented in Algorithm 7.

Algorithm 7 Stochastic tournament selection

Input: Population P, tournament size s, selection probability p
Output: Selected parent chromosome
$T \leftarrow$ randomly select s individuals from P
Sort T in decreasing order of fitness: $T = \{t_1, t_2, \ldots, t_s\}$
for $i = 1$ to s **do**
    With probability $p \cdot (1-p)^{i-1}$, return t_i
end for
return t_s ▷ Return the least fit individual if none was selected probabilistically

3.3. Crossover and mutation

A crossover combines two parents to create new offspring. The crossover rate χ represents the probability of applying crossover to a target individual. We employed one-point crossover, where a random point is selected on both parent chromosomes, and bits to the right of this point are exchanged between the parents. This process generates two offspring, from which we randomly select one. While crossover is exploitative in that it recombines material from promising parents, it is also explorative as it generates new offspring [34].

After a crossover, the selected offspring undergoes mutation, during which binary genes are flipped from 0 to 1 or 1 to 0. Mutation serves both explorative and exploitative purposes: It introduces new offspring in an unbiased manner while preserving most of the original chromosome [34]. The mutation rate, denoted by μ, controls the probability of each gene being flipped and, thus, determines the extent of variation introduced during this process.

3.4. Replacement

The newly generated offspring typically replaces the first parent, $P[i]$ in Algorithm 3. However, if $P[i]$ has the best fitness and the offspring does not, elitism may be applied, discarding the offspring and preserving $P[i]$ for the next generation. When the number of best solutions is 10 or fewer, $P[i]$ is always selected. If it exceeds 10, $P[i]$ is selected with a probability of 10/(number of best solutions), which helps prevent elitism from preserving too many existing solutions.

3.5. Overview of the proposed GA

The overall procedure of our GA is outlined in Algorithm 3. This subsection provides a comprehensive explanation of each step, including initialization, selection, crossover, mutation, repair, and replacement.

Initialization: An initial population P of binary chromosomes is randomly generated. Each chromosome represents a candidate solution to the vertex cover problem, where each bit indicates whether the corresponding vertex is included in the cover. Since randomly generated chromosomes may not satisfy problem constraints, we apply a repair operator to each chromosome to ensure feasibility.

Selection: To select a parent for crossover, we use stochastic tournament selection with selection pressure p. This method probabilistically favors fitter individuals while preserving diversity in the population.

Crossover: With a crossover probability of χ, a pair of selected parents undergo crossover to produce an offspring. If crossover does not occur, the parent is copied directly to the next generation. In our implementation, we use one-point crossover.

Mutation: Each gene in the offspring has a probability μ of being flipped (i.e., from 0 to 1 or from 1 to 0). Mutation introduces genetic variation while preserving most of the original chromosome's structure.

Repair: Since crossover and mutation may produce infeasible solutions, a repair operator is applied to ensure that each offspring represents a valid vertex cover. We adopt adaptive greedy repair, which builds upon the randomized greedy repair strategy. Unlike the randomized version that uses a fixed exploration parameter γ, adaptive greedy repair dynamically adjusts γ based on the convergence level of the population. Specifically, γ is set to $|P|/best(P)$, where $best(P)$ denotes the number of best-performing individuals in the population. A larger γ results in greedier behavior, prioritizing vertices with higher gain more strongly, while a smaller γ allows for more exploration. This adaptive adjustment encourages exploration when the population is highly converged, and promotes exploitation when the diversity in the population is low.

Replacement: The newly generated offspring typically replaces the first parent. However, if the first parent belongs to the group of best-performing individuals and the offspring does not, elitism may be applied to preserve high-quality solutions. In such cases, the offspring is discarded and the parent is retained in the next generation. To prevent elitism from dominating the population, a probabilistic mechanism is introduced, in which the parent is preserved with decreasing probability as the number of elite individuals increases. This strategy strikes a balance between preserving excellent solutions and maintaining diversity within the population.

Termination: The algorithm runs for a predefined number of generations ($n = 300$ in our experiments), after which the best solution(s) found during the evolutionary process are reported.

4. Results

4.1. Data

We used the BHOSLIB dataset to compare the repair algorithms we developed. This dataset is utilized for various problems, including vertex cover, maximum clique, maximum independent set, and vertex coloring [33]. Table 1 shows the characteristics of the BHOSLIB graphs we used.

Table 1. Properties of the BHOSLIB dataset used in this study.

Instance	Maximum Clique	Nodes	Edges
frb45-21-1	45	945	59,186
frb45-21-2	45	945	58,624
frb45-21-3	45	945	58,245
frb45-21-4	45	945	58,549
frb45-21-5	45	945	58,579
frb50-23-1	50	1,150	80,072
frb50-23-2	50	1,150	80,851
frb50-23-3	50	1,150	81,068
frb50-23-4	50	1,150	80,258
frb50-23-5	50	1,150	80,035
frb53-24-1	53	1,272	94,227
frb53-24-2	53	1,272	94,289
frb53-24-3	53	1,272	94,127
frb53-24-4	53	1,272	94,308
frb53-24-5	53	1,272	94,226
frb56-25-1	56	1,400	109,676
frb56-25-2	56	1,400	109,401
frb56-25-3	56	1,400	109,379
frb56-25-4	56	1,400	110,038
frb56-25-5	56	1,400	109,601
frb59-26-1	59	1,534	126,555
frb59-26-2	59	1,534	126,163
frb59-26-3	59	1,534	126,082
frb59-26-4	59	1,534	127,011
frb59-26-5	59	1,534	125,982

4.2. Experimental setup

We compared the performance of uniform repair, deterministic greedy repair, randomized greedy repair, and adaptive greedy repair in generational GA. The population size was set to 100 with a maximum of 300 generations. The selection pressure p was fixed at 0.5, the crossover rate χ at 0.9, and the mutation rate μ at 0.03. All experiments were repeated 50 times using different seed numbers to measure both the best and average performance. We used the deterministic greedy algorithm and randomized greedy algorithm as baselines. The randomized greedy algorithm was executed 900 times to match the GA's computation time, and its best result was used for comparison. All code was written in C++ and executed on an Advanced Micro Devices (AMD) Ryzen Threadripper 2990WX processor.

4.3. Comparative analysis

We compared baseline approaches, the edge age strategy for vertex cover (EAVC) solver [31], and GAs using different repair operators for the MVC problem. The EAVC is one of the state-of-the-art solvers for the MVC problem, which is based on edge age strategy. Table 2 reports the best results from 50 runs of the GA for each repair operator, along with the best result from 50 runs of EAVC for reference. For a fair comparison, each single run of EAVC was executed with a time limit approximately equal to that of a GA run using the adaptive greedy repair operator. As expected, EAVC achieved the best performance on most instances. However, on 9 out of 25 instances, our adaptive greedy repair achieved the same best solution as EAVC. Among the baselines, the randomized greedy algorithm consistently outperformed the deterministic greedy algorithm, as it selected the best result from 900 runs. For the GA repair operators, uniform repair showed the weakest performance, yet still achieved better results than the baselines across all instances. Similar to the baselines, randomized greedy repair performed better than deterministic greedy repair. Lastly, our adaptive greedy repair achieved the best results among GAs for all instances except for three.

All methods except the deterministic greedy algorithm were executed multiple times. To calculate the expected performance of a single run, we computed the average of multiple executions for each method. Table 3 shows the average performance of each method. When comparing the best solutions, the randomized algorithm outperformed the deterministic algorithm in all instances. However, for single-run performance, it showed better results in 14 out of 25 instances. Although uniform repair showed the weakest performance among repair operators, it still achieved better results than the baselines across all instances. Randomized greedy repair generally demonstrated superior performance compared to deterministic greedy repair because it could generate diverse solutions, whereas deterministic repair always repairs solutions in the same way. Additionally, adaptive greedy repair, which can generate more varied solutions when the population experiences premature convergence, showed the best average performance among GAs across all instances. The performance of the adaptive greedy repair operator in the $(1+(\lambda,\lambda))$ GA can be found in Appendix A.

Table 2. Comparison of best solutions on BHOSLIB dataset. The best values among GAs are shown in bold type, and underlined values indicate that the results equal the best solution obtained by EAVC.

Instance	Baselines [11]		GAs				EAVC [31]
	D.G. Alg.*	R.G. Alg.†	U. Repair‡	D.G. Repair	R.G. Repair	A.G. Repair§	
frb45-21-1	913	909	907	**902**	903	**902**	901
frb45-21-2	913	909	906	**902**	**902**	**902**	900
frb45-21-3	915	910	907	903	903	**902**	901
frb45-21-4	912	908	907	**901**	902	**901**	901
frb45-21-5	913	909	907	**902**	**902**	**902**	900
frb50-23-1	1115	1109	1108	1103	**1102**	**1102**	1101
frb50-23-2	1116	1110	1107	1103	1104	**1102**	1101
frb50-23-3	1116	1109	1108	**1102**	1103	**1102**	1102
frb50-23-4	1114	1110	1108	1103	1103	**1102**	1100
frb50-23-5	1114	1110	1108	1103	**1102**	1103	1102
frb53-24-1	1237	1232	1227	**1222**	**1222**	**1222**	1221
frb53-24-2	1235	1231	1228	1223	**1222**	**1222**	1221
frb53-24-3	1232	1230	1228	**1222**	**1222**	**1222**	1221
frb53-24-4	1236	1230	1228	1223	**1222**	**1222**	1222
frb53-24-5	1238	1231	1228	**1222**	**1222**	**1222**	1221
frb56-25-1	1366	1357	1354	**1347**	**1347**	1348	1346
frb56-25-2	1361	1356	1353	1348	**1347**	1348	1347
frb56-25-3	1362	1357	1353	1348	1348	**1346**	1346
frb56-25-4	1360	1356	1354	1348	1348	**1347**	1346
frb56-25-5	1360	1356	1354	1348	1348	**1347**	1347
frb59-26-1	1492	1489	1485	1479	1479	**1478**	1478
frb59-26-2	1495	1487	1486	1479	1479	**1478**	1478
frb59-26-3	1496	1490	1486	**1479**	**1479**	**1479**	1478
frb59-26-4	1492	1488	1486	1479	**1478**	**1478**	1478
frb59-26-5	1494	1487	1486	1479	1479	**1478**	1478

* Deterministic Greedy Algorithm.

† Randomized Greedy Algorithm.

‡ Uniform Repair.

§ Adaptive Greedy Repair.

Table 3. Comparison of average solutions on BHOSLIB dataset. The best values among GAs are shown in bold type.

Instance	Baselines [11]		GAs				EAVC [31]
	D.G. Alg.*	R.G. Alg.†	U. Repair‡	D.G. Repair	R.G. Repair	A.G. Repair§	
frb45-21-1	913	912.72	908.08	904.06	904.16	**903.68**	901.92
frb45-21-2	913	913.05	907.9	904.16	903.96	**903.72**	901.92
frb45-21-3	915	914.31	908.04	904.34	904.22	**903.66**	901.96
frb45-21-4	912	912.01	907.84	904.04	903.8	**903.46**	901.86
frb45-21-5	913	914.09	908.14	904.36	904.16	**903.86**	901.92
frb50-23-1	1115	1114.68	1109.44	1105.04	1104.7	**1104.24**	1102.84
frb50-23-2	1116	1114.47	1109.44	1104.72	1104.74	**1104.22**	1102.76
frb50-23-3	1116	1114.97	1109.5	1104.9	1104.7	**1104.12**	1102.78
frb50-23-4	1114	1114.85	1109.34	1104.7	1104.64	**1104.06**	1102.8
frb50-23-5	1114	1115.05	1109.54	1104.72	1104.46	**1104.04**	1102.7
frb53-24-1	1237	1235.67	1229.46	1224.36	1224.12	**1223.84**	1222.48
frb53-24-2	1235	1235.72	1229.3	1224.1	1223.92	**1223.58**	1222.06
frb53-24-3	1232	1234.59	1229.3	1224.28	1224.02	**1223.72**	1222.26
frb53-24-4	1236	1235.189	1229.64	1224.58	1224.18	**1223.7**	1222.4
frb53-24-5	1238	1236.16	1229.7	1224.66	1224.14	**1223.88**	1222.42
frb56-25-1	1366	1362.28	1355.28	1349.5	1349.38	**1349.24**	1347.78
frb56-25-2	1361	1360.75	1354.92	1349.88	1349.88	**1349.28**	1347.94
frb56-25-3	1362	1361.57	1355.22	1349.52	1349.68	**1348.78**	1347.78
frb56-25-4	1360	1360.82	1355.06	1349.88	1349.4	**1349.14**	1347.72
frb56-25-5	1360	1361.32	1355.3	1349.7	1349.6	**1349.18**	1347.86
frb59-26-1	1492	1493.11	1486.84	1480.96	1480.86	**1480.22**	1479.3
frb59-26-2	1495	1493.16	1486.98	1481.2	1480.64	**1480.18**	1479.38
frb59-26-3	1496	1494.57	1487.16	1481.16	1480.9	**1480.34**	1479.54
frb59-26-4	1492	1492.79	1487.04	1481.16	1480.72	**1480.6**	1479.5
frb59-26-5	1494	1492.07	1487.28	1481.2	1480.96	**1480.5**	1479.5

* Deterministic Greedy Algorithm.

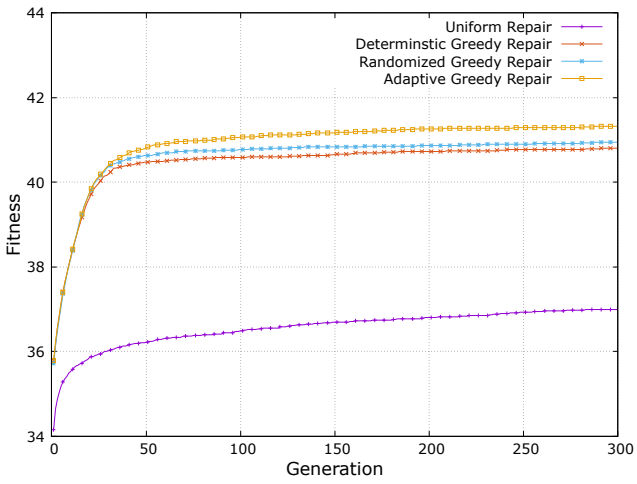
† Randomized Greedy Algorithm.

‡ Uniform Repair.

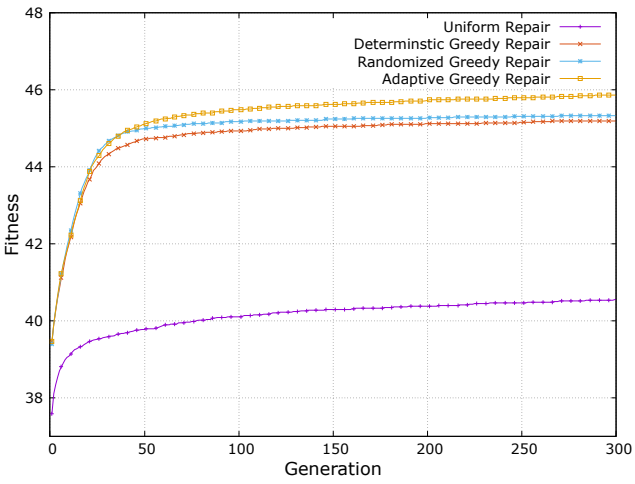
§ Adaptive Greedy Repair.

To verify whether the GAs converged appropriately, we plotted the fitness trends over generations for each repair operator in Figure 2. For datasets of the same size, the average fitness across all runs was plotted. Similar patterns were observed across all datasets: the uniform repair, which randomly selects vertices for repair, exhibited the poorest performance. Among the greedy repairs, the deterministic greedy repair showed the lowest performance, while the adaptive greedy repair demonstrated the best

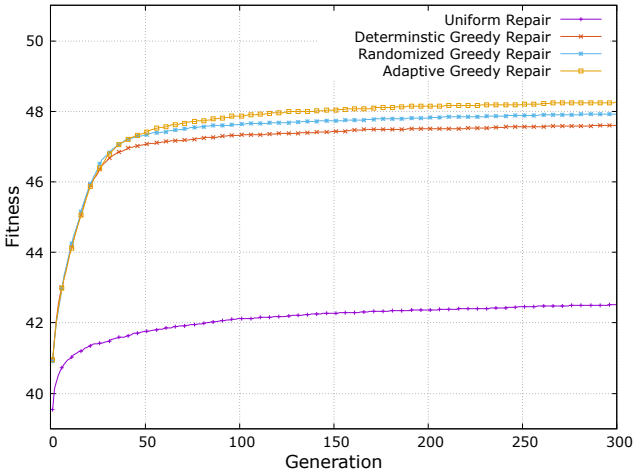
performance across all datasets. For all repair operators, the GAs appear to have converged sufficiently.



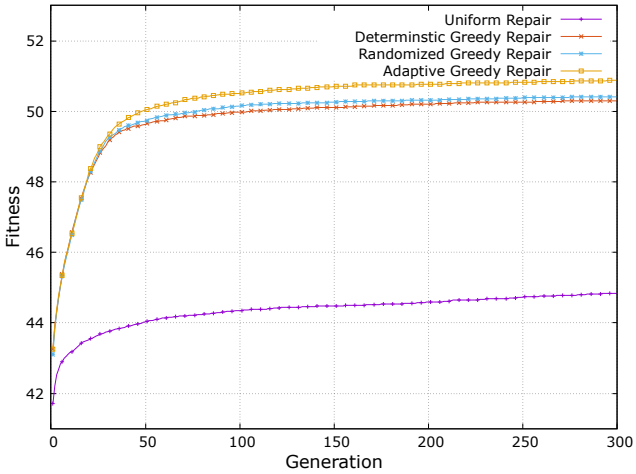
(a) frb45-21-*



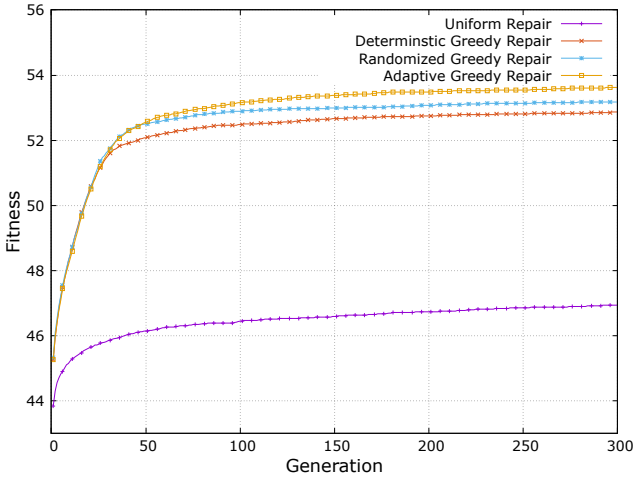
(b) frb50-23-*



(c) frb53-24-*



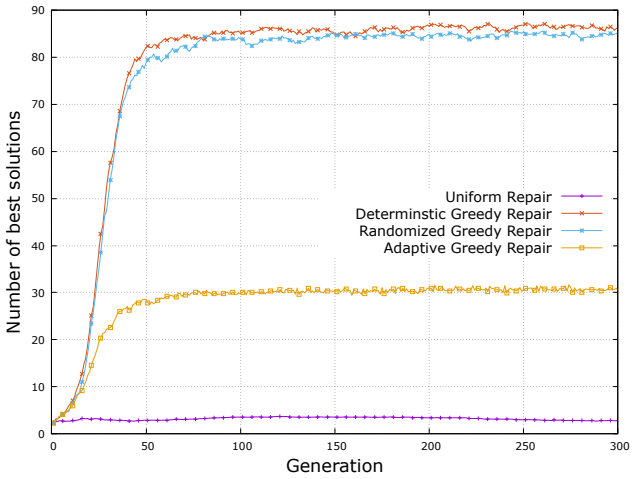
(d) frb56-25-*



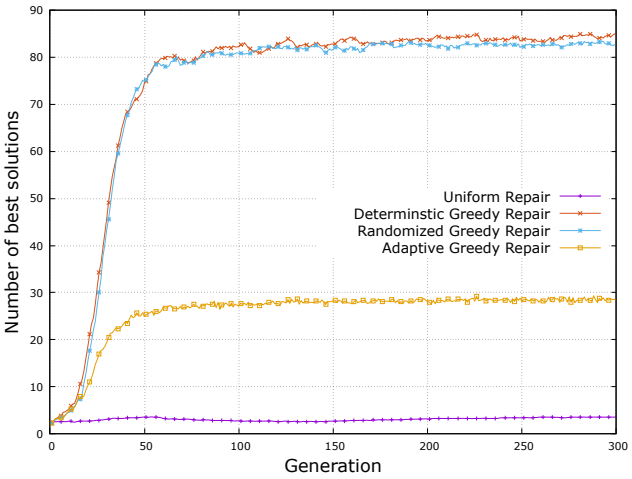
(e) frb59-26-*

Figure 2. Fitness comparison by generation.

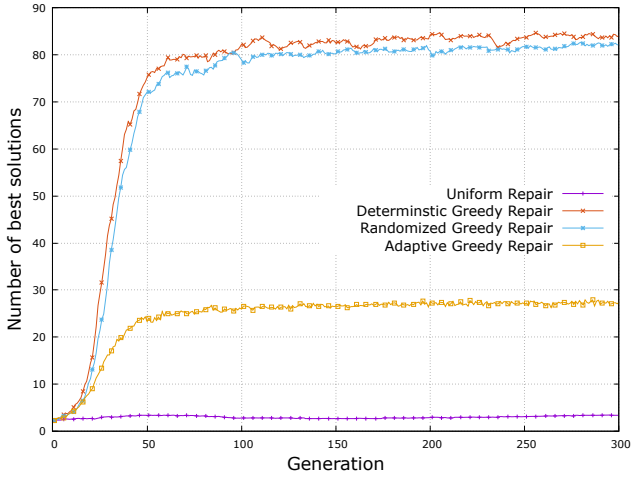
We counted the number of best solutions per generation and displayed them in Figure 3 to evaluate how well each repair operator balances exploitation and exploration. For the uniform repair method, which selects a uniformly random vertex during repair, the number of best solutions is very low because it focuses solely on exploration without any exploitation. In contrast, deterministic greedy and randomized greedy approaches show strong exploitation tendencies, which causes the population to converge around the current best solutions as generations progress. Lastly, in the case of adaptive greedy repair, as the number of best solutions in the population increases, the method reduces the greediness parameter γ to encourage exploration and generate diverse solutions. This process enables the generation of solutions that surpass the existing best ones, resulting in superior performance compared to other repair operators.



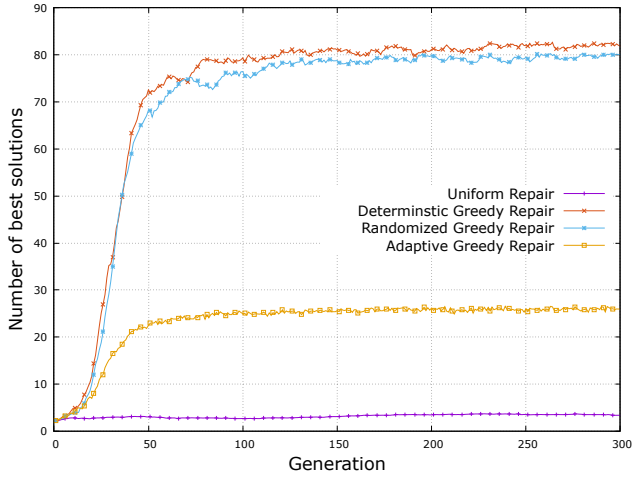
(a) frb45-21-*



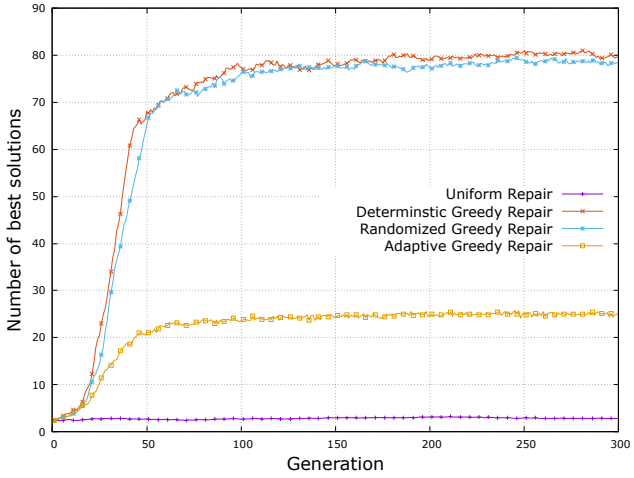
(b) frb50-23-*



(c) frb53-24-*



(d) frb56-25-*



(e) frb59-26-*

Figure 3. Number of best solutions by generation.

We also compared the average execution times of each technique in Table 4. The deterministic greedy algorithm, which generates only a single solution, was excluded from the comparison as it took less than 0.4 seconds for all instances. Among all instances, the execution time of the uniform repair was the longest. This was due to the poor quality of the solutions it generated, requiring significant time to calculate *gain* and $loss$. The randomized greedy algorithm consumed more time compared to the greedy-based repair operators but performed worse than them. There was little difference in execution time between the deterministic greedy repair and the randomized greedy repair. The adaptive greedy repair operator, which showed the best performance, used approximately 5 % more time than the randomized greedy repair.

Table 4. Comparison of average running time (sec) by method.

Instances	R.G. Algorithm[†]	U. Repair[‡]	D.G. Repair[*]	R.G. Repair	A.G. Repair[§]
frb45-21-*	170.36	210.06	131.11	133.01	139.82
frb50-23-*	234.77	320.37	199.92	201.40	210.83
frb53-24-*	283.27	368.87	225.70	228.04	239.11
frb56-25-*	333.79	388.57	280.17	282.51	295.88
frb59-26-*	385.43	413.59	327.46	330.02	345.52

[†] Randomized Greedy Algorithm.
[‡] Uniform Repair.
[*] Deterministic Greedy Repair.
[§] Adaptive Greedy Repair.

4.4. Statistical test

The Wilcoxon signed-rank test is a nonparametric statistical method used to compare paired observations from two different algorithms to determine if one systematically outperforms the other. For each pair of observations, the test calculates the difference, ranks these differences based on their absolute values, and then assigns signs (+ or –) back to the ranks based on whether the original difference was positive or negative. The sum of positive ranks (W+) and negative ranks (W–) are calculated, and the smaller of these sums is compared to critical values from a Wilcoxon table or used to compute a p-value. If the resulting p-value is less than the chosen significance level (typically 0.05), we can reject the null hypothesis that both algorithms perform equally and conclude that one algorithm consistently performs better than the other, without assuming the differences follow a normal distribution.

We conducted a Wilcoxon signed-rank test to assess if adaptive greedy repair significantly outperforms randomized greedy repair, based on 50 runs per instance using the same seed number. Table 5 presents the p-values of the Wilcoxon signed-rank test for each instance. Among the total 25 instances, adaptive greedy repair significantly outperformed randomized greedy repair in 21 instances. In the remaining four instances, the average performance of adaptive greedy repair was better than that of randomized greedy repair, but the difference was not statistically significant.

Table 5. The p-values from the Wilcoxon signed-rank test. Values below 0.05 are shown in bold type.

Instance	1	2	3	4	5
frb45-21-	**0.0030**	0.0746	**0.0003**	**0.0484**	**0.0288**
frb50-23-	**0.0068**	**0.0017**	**0.0002**	**4.8e-05**	**0.0085**
frb53-24-	**0.0430**	**0.0253**	**0.0197**	**0.0026**	**0.0438**
frb56-25-	0.2150	**0.0039**	**2.26e-06**	0.0567	**0.0373**
frb59-26-	**0.0004**	**0.0109**	**0.0009**	0.3393	**0.0023**

5. Conclusions

In this study, we proposed a novel repair operator based on a randomized greedy algorithm. Instead of using a fixed greediness parameter, it dynamically adjusts greediness according to the convergence level of the population. When the population contains few high-quality solutions, exploitation is emphasized to generate more superior solutions. Conversely, as the population converges, exploration is increased to create novel solutions, thereby enhancing the likelihood of discovering better-quality solutions. A comparative analysis showed that the adaptive greedy repair achieved the best performance on average across all instances and outperformed the other techniques with statistical significance in over 80 % of instances.

GAs are inherently prone to premature convergence and getting trapped in local optima because they are not effective at fine-tuning solutions near the optimum [37]. To address this, they are frequently combined with local search techniques to form memetic algorithms. Building on this insight, future research will focus on designing an effective local search method tailored specifically for the minimal vertex cover problem, aiming to further enhance the algorithm's efficiency and solution quality. Experimental results applying a simple local search to our method can be found in Appendix C.

Author contributions

Seung-Hyun Moon: Conceptualization, methodology, investigation, software, formal analysis, writing - original draft, validation; Yourim Yoon: Supervision, project administration, writing - review and editing, validation. All authors have read and approved the final version of the manuscript for publication.

Use of Generative-AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

The present research has been conducted by the Research Grant of Kwangwoon University in 2024. This work was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1F1A1066017).

Conflict of interest

The authors declare no conflicts of interest.

References

1. S. Shyu, P. Y. Yin, B. Lin, An ant colony optimization algorithm for the minimum weight vertex cover problem, *Ann. Oper. Res.*, **131** (2004), 283–304. https://doi.org/10.1023/B:ANOR.0000039523.95673.33
2. H. Tamura, H. Sugawara, M. Sengoku, S. Shinoda, Multiple cover problem on undirected flow networks, *Electron. Comm. JPN. 3*, **84** (2001), 67–74. https://doi.org/10.1002/1520-6440(200101)84:1<67::AID-ECJC7>3.0.CO;2-%23
3. V. V. Gusev, The vertex cover game: Application to transport networks, *Omega*, **97** (2020), 102102. https://doi.org/10.1016/j.omega.2019.08.009
4. A. Hossain, E. Lopez, S. M. Halper, D. P. Cetnar, A. C. Reis, D. Strickland, et al., Automated design of thousands of nonrepetitive parts for engineering stable genetic systems, *Nat. Biotechnol.*, **38** (2020), 1466–1475. https://doi.org/10.1038/s41587-020-0584-2
5. A. C. Reis, S. M. Halper, G. E. Vezeau, D. P. Cetnar, A. Hossain, P. R. Clauer, et al., Simultaneous repression of multiple bacterial genes using nonrepetitive extra-long sgrna arrays, *Nat. Biotechnol.*, **37** (2019), 1294–1301. https://doi.org/10.1038/s41587-019-0286-9
6. M. R. Garey, D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman & Co., USA, 1990.
7. C. H. Papadimitriou, M. Yannakakis, Optimization, approximation, and complexity classes, *J. Comput. Syst. Sci.*, **43** (1991), 425–440. https://doi.org/10.1016/0022-0000(91)90023-X
8. K. Subhash, D. Minzer, M. Safra, *Pseudorandom sets in Grassmann graph have near-perfect expansion*, in Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science, 2018, 592–601. https://doi.org/10.1109/FOCS.2018.00062
9. S. Khot, O. Regev, Vertex cover might be hard to approximate to within $2-\epsilon$, *J. Comput. Syst. Sci.*, **74** (2008), 335–349. https://doi.org/10.1016/j.jcss.2007.06.019
10. C. H. Papadimitriou, K. Steiglitz, *Combinatorial optimization: Algorithms and complexity*, Prentice-Hall, Inc., USA, 1982.

11. W. Gao, T. Friedrich, F. Neumann, C. Hercher, *Randomized greedy algorithms for covering problems*, in Proceedings of the Genetic and Evolutionary Computation Conference, 2018, 309–315. https://doi.org/10.1145/3205455.3205542
12. S. Bouamama, C. Blum, A. Boukerram, A population-based iterated greedy algorithm for the minimum weight vertex cover problem, *Appl. Soft Comput.*, **12** (2012), 1632–1639. https://doi.org/10.1016/j.asoc.2012.02.013
13. S. Cai, J. Lin, C. Luo, Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess, *J. Artif. Intell. Res.*, **59** (2017), 463–494. https://doi.org/10.1613/jair.5443
14. R. Jovanovic, A. P. Sanfilippo, S. Voß, Fixed set search applied to the multi-objective minimum weighted vertex cover problem, *J. Heuristics*, **28** (2022), 481–508. https://doi.org/10.1007/s10732-022-09499-z
15. S. Khuri, T. Bäck, *An evolutionary heuristic for the minimum vertex cover problem*, in Genetic Algorithms within the Framework of Evolutionary Computation–Proc. of the KI-94 Workshop, Saarbrücken, Germany, 1994, 86–90.
16. A. Karci, A. Arslan, Bidirectional evolutionary heuristic for the minimum vertex-cover problem, *Comput. Electr. Eng.*, **29** (2003), 111–120. https://doi.org/10.1016/S0045-7906(01)00018-0
17. A. Singh, A. K. Gupta, A hybrid heuristic for the minimum weight vertex cover problem, *Asia Pac. J. Oper. Res.*, **23** (2006), 273–285. https://doi.org/10.1142/S0217595906000905
18. B. Nagy, P. Szokol, A genetic algorithm for the minimum vertex cover problem with interval-valued fitness, *Acta Polytech. Hung.*, **18** (2021), 105–123. https://doi.org/10.12700/APH.18.4.2021.4.6
19. J. H. Holland, Genetic algorithms, *Sci. Am.*, **267** (1992), 66–73.
20. P. O. Lewis, A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data, *Mol. Biol. Evol.*, **15** (1998), 277–283. https://doi.org/10.1093/oxfordjournals.molbev.a025924.
21. S. L. K. Pond, D. Posada, M. B. Gravenor, C. H. Woelk, S. D. Frost, Automated phylogenetic detection of recombination using a genetic algorithm, *Mol. Biol. Evol.*, **23** (2006), 1891–1901. https://doi.org/10.1093/molbev/msl051
22. B. Chowdhury, G. Garai, A review on multiple sequence alignment from the perspective of genetic algorithm, *Genomics*, **109** (2017), 419–431. https://doi.org/10.1016/j.ygeno.2017.06.007
23. L. Bateni, F. Asghari, Bankruptcy prediction using logit and genetic algorithm models: A comparative analysis, *Comput. Econ.*, **55** (2020), 335–348. https://doi.org/10.1007/s10614-016-9590-3
24. C. B. Kalayci, O. Polat, M. A. Akbay, An efficient hybrid metaheuristic algorithm for cardinality constrained portfolio optimization, *Swarm Evol. Comput.*, **54** (2020), 100662. https://doi.org/10.1016/j.swevo.2020.100662
25. S.-H. Moon, Y. Yoon, Genetic mean reversion strategy for online portfolio selection with transaction costs, *Mathematics*, **10** (2022), 1073. https://doi.org/10.3390/math10071073

26. C. R. Reeves, A genetic algorithm for flowshop sequencing, *Comput. Oper. Res.*, **22** (1995), 5–13. https://doi.org/10.1016/0305-0548(93)E0014-K

27. Y. Yoon, Y. H. Kim, B. R. Moon, A theoretical and empirical investigation on the Lagrangian capacities of the 0-1 multidimensional knapsack problem, *Eur. J. Oper. Res.*, **218** (2012), 366–376. https://doi.org/10.1016/j.ejor.2011.11.011

28. D. Orvosh, L. Davis, *Using a genetic algorithm to optimize problems with feasibility constraints*, in Proceedings of the First IEEE Conference on Evolutionary Computation (CEC), vol. 2, 1994, 548–553. https://doi.org/10.1109/ICEC.1994.350001

29. S. Salcedo-Sanz, A survey of repair methods used as constraint handling techniques in evolutionary algorithms, *Comput. Sci. Rev.*, **3** (2009), 175–192. https://doi.org/10.1016/j.cosrev.2009.07.001

30. F. Samanipour, J. Jelovica, Adaptive repair method for constraint handling in multi-objective genetic algorithm based on relationship between constraints and variables, *Appl. Soft Comput.*, **90** (2020), 106143. https://doi.org/10.1016/j.asoc.2020.106143

31. C. Quan, P. Guo, A local search method based on edge age strategy for minimum vertex cover problem in massive graphs, *Expert Syst. Appl.*, **182** (2021), 115185. https://doi.org/10.1016/j.eswa.2021.115185

32. Y. Zhang, S. Wang, C. Liu, E. Zhu, TIVC: An efficient local search algorithm for minimum vertex cover in large graphs, *Sensors*, **23** (2023), 7831. https://doi.org/10.3390/s23187831

33. R. Rossi, N. Ahmed, *The network data repository with interactive graph analytics and visualization*, in Proceedings of the AAAI conference on artificial intelligence, **29** (2015). https://doi.org/10.1609/aaai.v29i1.9277

34. A. Eiben, C. Schippers, On evolutionary exploration and exploitation, *Fundam. Inform.*, **35** (1998), 35–50.

35. H. G. Beyer, *On the "explorative power" of ES/EP-like algorithms*, in Evolutionary Programming VII (eds. V. W. Porto, N. Saravanan, D. Waagen and A. E. Eiben), Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, 323–334. https://doi.org/10.1007/BFb0040785

36. B. L. Miller, D. E. Goldberg, Genetic algorithms, tournament selection, and the effects of noise, *Complex Syst.*, **9** (1995), 193–212.

37. T. Bui, B. Moon, *A new genetic approach for the traveling salesman problem*, in Proceedings of the First IEEE Conference on Evolutionary Computation. (CEC), **1** (1994), 7–12. https://doi.org/10.1109/ICEC.1994.350051

38. M. Buzdalov, *The (1+(λ,λ)) genetic algorithm on the vertex cover problem: Crossover helps leaving plateaus*, in Proceedings of the 2022 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2022, 1–10. https://doi.org/10.1109/CEC55065.2022.9870224

Appendix A. Adaptive greedy repair operator in the (1+(λ, λ)) GA

We evaluated the performance of our adaptive greedy repair operator within the (1+(λ, λ)) GA framework for MVC [38]. This experiment was conducted to assess whether the operator remains effective beyond the generational GA setting used in our main study. In the (1+(λ, λ)) GA, the parameter λ controls the search intensity: larger values of λ lead to broader exploration of the search space. In our experiments, we followed the self-adjusting strategy proposed by Buzdalov [38], where λ is dynamically adapted during the search process. In the adaptive repair operator, the greediness parameter γ was set to $0.01/\gamma$, meaning that as the search intensity λ increases, the value of γ decreases, thereby reducing the greediness of the repair process and promoting more exploratory behavior.

The approach proposed by Buzdalov does not use a repair operator but instead penalizes infeasible solutions in the fitness function. We adopted this method as a baseline in our experiments. To ensure a fair comparison in terms of computational effort, the baseline was run for 10,000 generations, while the versions using repair operators were executed for 2,000 generations. These settings were chosen so that the total running time of the experiments would be comparable to that of the main experiments in this paper.

Table A1 compares the best solutions found by each method across 50 independent runs, while Table A2 presents the average solutions obtained over the same runs. In the case without a repair operator, all vertices were selected at the end if a feasible solution could not be found. The results show that this approach failed to find feasible solutions for most instances. Unlike the experiments in the main text, the (1+(λ, λ)) GA framework has only a single parent, which limits its ability to generate diverse offspring. As a result, uniform repair, which promotes diversity, outperformed the more greedy methods such as deterministic greedy repair and randomized greedy repair. Our adaptive repair operator achieved the best performance by dynamically adjusting the γ value to balance exploration and exploitation. Specifically, it achieved the best results on 21 out of 25 instances in terms of best solution, and on 23 instances in terms of average solution. Overall, the (1+(λ, λ)) GA did not perform as well as the generational GA used in the main experiments.

Table A1. Comparison of best solutions using (1+(λ, λ)) GAs. The best values among (1+(λ, λ)) GAs are shown in bold type.

Instance	(1+(λ, λ)) GAs [38]					Generational GA
	W/O Repair*	U. Repair†	D.G. Repair‡	R.G. Repair§	A.G. Repair**	A.G. Repair**
frb45-21-1	913	904	908	908	**903**	902
frb45-21-2	912	904	906	907	**903**	902
frb45-21-3	917	**903**	907	906	904	902
frb45-21-4	945	**903**	907	905	**903**	901
frb45-21-5	945	904	907	906	**903**	902
frb50-23-1	1150	1105	1108	1108	**1104**	1102
frb50-23-2	1150	**1105**	1107	1107	**1105**	1102
frb50-23-3	1150	1105	1107	1107	**1104**	1102
frb50-23-4	1150	**1103**	1108	1108	1105	1102
frb50-23-5	1150	**1104**	1108	1107	**1104**	1103
frb53-24-1	1272	**1224**	1228	1227	**1224**	1222
frb53-24-2	1272	**1224**	1228	1226	**1224**	1222
frb53-24-3	1272	**1225**	1227	1226	**1225**	1222
frb53-24-4	1272	**1223**	1228	1227	1224	1222
frb53-24-5	1272	1225	1229	1227	**1224**	1222
frb56-25-1	1400	**1349**	1353	1354	**1349**	1348
frb56-25-2	1400	1350	1353	1353	**1349**	1348
frb56-25-3	1400	1349	1353	1353	**1347**	1346
frb56-25-4	1400	1350	1353	1351	**1349**	1347
frb56-25-5	1400	**1349**	1354	1352	1350	1347
frb59-26-1	1534	**1481**	1484	1485	**1481**	1478
frb59-26-2	1534	**1481**	1485	1485	**1481**	1478
frb59-26-3	1534	**1481**	1485	1485	**1481**	1479
frb59-26-4	1534	**1481**	1485	1484	**1481**	1478
frb59-26-5	1534	**1481**	1485	1484	**1481**	1478

* Without Repair Operator.

† Uniform Repair.

‡ Deterministic Greedy Repair.

§ Randomized Greedy Repair.

** Adaptive Greedy Repair.

Table A2. Comparison of average solutions using (1+(λ, λ)) GAs. The best values among (1+(λ, λ)) are shown in bold type.

Instance	(1+(λ, λ)) GAs [38]					Generational GA
	W/O Repair*	U. Repair†	D.G. Repair‡	R.G. Repair§	A.G. Repair**	A.G. Repair**
frb45-21-1	943.78	906.28	909.82	909.82	**906.18**	903.68
frb45-21-2	943.04	906.36	909.64	908.84	**905.86**	903.72
frb45-21-3	944.44	**906.28**	909.84	909.36	906.42	903.66
frb45-21-4	945.00	906.02	909.42	908.80	**905.52**	903.46
frb45-21-5	945.00	906.48	909.82	909.08	**906.04**	903.86
frb50-23-1	1150.00	1107.12	1110.98	1110.30	**1106.66**	1104.24
frb50-23-2	1150.00	1107.30	1110.74	1110.26	**1106.80**	1104.22
frb50-23-3	1150.00	1106.94	1110.74	1109.98	**1106.70**	1104.12
frb50-23-4	1150.00	1107.22	1110.86	1110.22	**1106.86**	1104.06
frb50-23-5	1150.00	1107.10	1111.00	1110.10	**1106.78**	1104.04
frb53-24-1	1272.00	1226.80	1230.88	1230.66	**1226.36**	1223.84
frb53-24-2	1272.00	1226.70	1230.50	1230.30	**1226.40**	1223.58
frb53-24-3	1272.00	1226.68	1230.54	1229.92	**1226.46**	1223.72
frb53-24-4	1272.00	1226.90	1230.42	1230.18	**1226.44**	1223.7
frb53-24-5	1272.00	**1226.74**	1231.16	1230.32	1226.90	1223.88
frb56-25-1	1400.00	1352.42	1357.30	1356.50	**1352.04**	1349.24
frb56-25-2	1400.00	1352.50	1356.36	1355.94	**1352.10**	1349.28
frb56-25-3	1400.00	1352.08	1355.86	1355.52	**1351.78**	1348.78
frb56-25-4	1400.00	1352.12	1356.52	1355.86	**1351.74**	1349.14
frb56-25-5	1400.00	1352.20	1356.32	1355.90	**1351.82**	1349.18
frb59-26-1	1534.00	1483.60	1488.10	1487.44	**1483.34**	1480.22
frb59-26-2	1534.00	1483.74	1488.20	1487.26	**1483.46**	1480.18
frb59-26-3	1534.00	1483.86	1488.90	1487.58	**1483.58**	1480.34
frb59-26-4	1534.00	1484.00	1488.28	1487.10	**1483.32**	1480.6
frb59-26-5	1534.00	1483.90	1488.30	1487.28	**1483.54**	1480.5

* Without Repair Operator.

† Uniform Repair.

‡ Deterministic Greedy Repair.

§ Randomized Greedy Repair.

** Adaptive Greedy Repair.

Appendix B. Experimental results on DIMACS dataset

We present additional experimental results evaluating our proposed algorithm on the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) benchmark dataset [33]. Our experiments were conducted under the same computational environment described in Section 4 of the main paper. For each instance, we performed 50 independent runs to ensure statistical significance of the results.

As shown in Table B1, the proposed adaptive greedy repair consistently achieved competitive or superior performance compared to the other repair strategies. Since the best solutions often resulted in ties across different methods, we focused our comparison primarily on the average solution quality. In this regard, the adaptive greedy repair yielded the best average results on the majority of instances.

Table B1. Comparison of best and average solutions on DIMACS dataset. The best average values are shown in bold type.

Instance	$\|V\|$	$\|E\|$	U. Repair*		D.G. Repair†		R.G. Repair‡		A.G. Repair§	
			best	avg.	best	avg.	best	avg.	best	avg.
brock800-1	800	207,505	790	790.14	790	790.58	790	790.48	790	**790.00**
brock800-2	800	208,166	790	790.16	790	790.66	790	790.62	790	**790.06**
brock800-3	800	207,333	789	**789.94**	789	790.30	790	790.22	789	**789.94**
brock800-4	800	207,643	790	790.10	790	790.54	790	790.44	790	**790.00**
C1000-9	1,000	450,079	994	994.00	994	994.54	994	994.32	994	**994.02**
C2000-5	2,000	999,836	1984	1985.62	1984	1985.28	1985	1985.38	1984	**1985.00**
DSJC500-5	500	62,624	487	487.48	487	487.72	487	487.60	487	**487.26**
DSJC1000-5	1,000	249,826	986	986.60	986	986.52	986	986.34	985	**986.00**
p-hat1000-1	700	122,253	630	633.42	625	626.16	625	625.36	625	**625.28**
p-hat1000-2	700	244,799	650	652.22	646	646.96	646	646.94	646	**646.92**
p-hat1000-3	700	371,746	689	689.98	689	689.92	689	689.88	689	**689.64**
p-hat1500-1	1,500	284,923	1424	1427.88	1413	1414.38	1414	1414.00	1413	**1413.86**
p-hat1500-2	1,500	568,960	1445	1448.10	1438	1438.13	1438	**1438.00**	1438	**1438.00**
p-hat1500-3	1,500	847,244	1488	1489.60	1488	1488.90	1488	**1488.80**	1488	1488.90

* Uniform Repair.

† Deterministic Greedy Repair.

‡ Randomized Greedy Repair.

§ Adpative Greedy Repair.

Appendix C. Experimental results with simple local search

We conducted an experiment in which our GA was augmented with a simple local search. This local search operates by attempting to enhance a minimal vertex cover obtained via repair: Specifically, it sequentially adds nonselected vertices to the current solution, and removes redundant ones. If the

inclusion of a new vertex allows for the elimination of multiple existing ones, a better solution is discovered. This process explores local improvements in a more structured neighborhood than our original GA.

To give the local search sufficient opportunity to improve solutions, we allowed the GA with local search (LS) to run for four times longer than the baseline GA without LS. The performance comparison over 50 runs is summarized in the Table C1 below. In general, the use of LS improved the average solution quality, though the improvement in best-found solutions was marginal for most instances.

Table C1. Comparison of Generational GA performance with and without LS when the adaptive greedy operator is applied (over 50 runs). The better values for each instance are shown in bold.

Instance	Best solutions		Average solutions	
	w/o LS	with LS	w/o LS	with LS
frb45-21-1	**902**	**902**	903.68	**903.36**
frb45-21-2	902	**901**	903.72	**903.34**
frb45-21-3	**902**	**902**	903.66	**903.54**
frb45-21-4	**901**	**901**	903.46	**903.14**
frb45-21-5	**902**	**902**	903.86	**903.50**
frb50-23-1	**1102**	1103	1104.24	**1104.00**
frb50-23-2	**1102**	**1102**	1104.22	**1103.88**
frb50-23-3	**1102**	1103	1104.12	**1103.80**
frb50-23-4	**1102**	**1102**	1104.06	**1103.84**
frb50-23-5	1103	**1102**	1104.04	**1103.84**
frb53-24-1	**1222**	**1222**	1223.84	**1223.36**
frb53-24-2	**1222**	**1222**	1223.58	**1223.18**
frb53-24-3	1222	**1221**	1223.72	**1223.36**
frb53-24-4	**1222**	1223	1223.70	**1223.48**
frb53-24-5	**1222**	**1222**	1223.88	**1223.56**
frb56-25-1	1348	**1347**	1349.24	**1348.84**
frb56-25-2	1348	**1347**	1349.28	**1349.08**
frb56-25-3	**1346**	1347	1348.78	**1348.66**
frb56-25-4	**1347**	**1347**	1349.14	**1348.88**
frb56-25-5	**1347**	**1347**	1349.18	**1348.76**
frb59-26-1	**1478**	**1478**	**1480.22**	1480.24
frb59-26-2	**1478**	**1478**	1480.18	**1480.14**
frb59-26-3	1479	**1478**	**1480.34**	1480.44
frb59-26-4	**1478**	**1478**	1480.60	**1480.18**
frb59-26-5	**1478**	**1478**	**1480.50**	**1480.50**

As a direction for future work, we aim to design a more effective local search algorithm that can synergize with the proposed adaptive repair operator, with the goal of developing a solution that is competitive with, or superior to, state-of-the-art approaches.

© 2025 the Author(s), licensee AIMS Press. This is an open access article distributed under the terms of the Creative Commons Attribution License (https://creativecommons.org/licenses/by/4.0)