Evolutionary Diversity Optimization With An Example Application For Vertex Covers

Evolutionary Computation

Sven Farwig and Daniel Zelenak

RWTH Aachen University, Germany
{sven.farwig, daniel.zelenak}@rwth-aachen.de

Abstract

The field of evolutionary computation orients itself on the real-world evolution of creatures. Bio-inspired algorithms that imitate nature are used to simulate evolution on a set of solutions. Like in a local search, the solutions can change to slightly different familiar versions of itself until they reach an optimum. Unfortunately, this can lead to local optima if the solution pool lacks diversity. Missing diversity is one of the reasons why in recent years, the field of diversity optimization has become more popular. Diversity optimization uses evolutionary algorithmic approaches to find many sufficiently good solutions but with the goal of them all being as different as possible from each other.

This paper first briefly introduces evolutionary algorithms and gives a broad overview of what has already happened in evolutionary diversity optimization. Subsequently, we present our own experiments with diversity optimization on the vertex cover problem to illustrate the procedure of creating an own diversity optimization. For that, we formulated a simple encoding with the sophisticated *MultiNodeSwap* mutation operator for vertex cover problems and a diversity function to find as many different vertex covers as possible on a graph.

Keywords

Evolutionary algorithms, diversity optimization, $\mu + 1$ EA, minimum vertex cover, diverse vertex cover, sophisticated vertex cover mutation operator

1 Introduction

We start with a short overview of the main idea of evolutionary algorithms before looking at diversity optimization. As in nature, we have a pool of individuals that we call population. An individual can be considered as a solution to a problem. Every individual is composed of a set of genes that can be considered as smaller building blocks of a solution. As a simple example, a gene could describe the color of a particular node in a graph coloring problem. So every gene influences the phenotype, like the individual's appearance in the graph coloring environment. The environment will rate the individuals with a fitness function. Individuals with a higher fitness get the chance for reproduction, as individuals with low fitness are not allowed to spread their genes in the population pool by producing offsprings. Starting from these basic ideas, we can outline the main cycle of any evolutionary algorithm.

0. **Initialization phase:** It is necessary to have a start population of individuals. This can be generated randomly or with a heuristic. The number of individuals in a population is μ.
1. **Parental selection:** This selection is usually based on fitness where high fitness goes hand in hand with high selection probability. In diversity optimization, this can be completely random.
2. **Generation of offspring:** The sexual-reproduction is inspired by nature. The first step is a recombination of gene information with the so-called crossover. The easiest crossover version takes half of the genetic information of each parent and combines them into a new individual. After that, the mutation is applied that randomly changes some genetic information.
3. **Survivor selection:** Assuming the current generation has λ offsprings but the envirnoment limits the population to μ individuals. Thus a mechanism needs to decide which individuals are sorted out. Like real natural selection, this is done by their fitness.

The algorithm continues then with step one. A termination condition could be after a fixed number of generations or after extinction - due to natural selection - went so far that only one individual survived. The extinction of unfit individuals clearly mirrors Darwin's theory of evolution, where the governing principle is "Survival of the fittest".
However, it is not guaranteed that the fittest surviving individual will also be the best possible individual in the environment. Genes of fit individuals dominate the gene pool. Thus, the whole population will lead to more and more similar results after some generations. Later generations will have a smaller diversity. Diversity is a measure of the number of different individuals in the population. Usually, the diversity measurement depends on the encoding of the solutions and if it should measure diversity in the genotype or phenotype [ES15]. Furthermore, a population where every individual is different from each other may have a lower diversity if they still share a high amount of similar values in the genes. Therefore a maximal diverse population would have used for every existing gene another value. As an example for maximal diversity: If we used a binary encoding and a population of $\mu = 2$ with n genes, the maximum diversity would be reached if they have a hamming distance of n.
If the diversity in a population drops too quickly, the effect of a premature convergence can cause a local optimum. Because of the low diversity, usual evolutionary operators can not generate new solutions as they are too far away in the local neighborhood. Especially for the crossover operator, different information in the individuals is crucial to find new solutions in later generations.
To resolve this problem, we could start with a diverse population at the beginning. This enables us to start from a diverse gene pool that is equal to a greater search space in local search. Obviously, a greater search space enables one to have the chance for fitter individuals in later generations. Also, it could be applied if the population's diversity hits a lower threshold and no new solutions have been generated for a while. Then a set containing lower fit or duplicate individuals could be replaced by a diverse population subset to have a fresh gene pool to escape a possible local optimum. This is basically the idea of a scatter search procedure [MLG06].
Generating such a diverse population can also be the task of an evolutionary algorithm. The field of evolutionary diversity optimization uses the previously described framework but with a diversity measurement as a fitness function. Therefore the fitness of each individual is dependent on others in the population. To illustrate the procedure in more detail, we decided to do our own experiments. We applied diversity optimization on minimum vertex covers mainly because of the following two reasons. Minimum vertex cover is a $\mathcal{NP}$-hard problem with practical relevance (e.g. applications in fields like scheduling or networking [OHY08]), and the research on this specific topic was rare when we implemented it. To contribute to the scientific community and ensure reproducibility, we published our results on GitHub [Dan23].
The paper is structured as follows. In the next section, we give an introduction to diversity optimization by presenting different theoretical results and practical applications of this technique. We continue with an overview of evolutionary algorithms and diversity optimization for vertex covers specifically. After a look at the related work, our own experimental setup for finding diverse vertex covers is introduced. We present a suitable encoding, mutation operator, a diversity fitness function, and how to initialize the population. This is followed by our investigations, where we will underline the advantages and weaknesses of our approach. The last section concludes with a summary and makes suggestions for improvements to our vertex cover diversification.

2 Background: Diversity Optimization

Evolutionary algorithms, particularly the $(\mu+1)$-EA, have been proposed to effectively compute a diverse population for different problems like MSTs, TSPs, or even art. The general framework for a $(\mu+1)$-EA is given by Algorithm 1. The basic idea is to do some mutation to a random individual and replace the worst individual in the population if the mutated one contributes to higher diversity [BN21]. This is usually done without the crossover operator, which is a typical operator in evolutionary computing. Otherwise, an EA without crossover simplifies the complexity, such it is easier to do some theoretical research on this construct. Also, it is better to use no crossover instead of using a crossover, which is not able to exchange genetic good information in a non-random way [Jon95].

Algorithm 1 Diversity Maximising $(\mu+1)$-Evolutionary Algorithm

1: Initialise a population P with μ solutions, satisfying certain conditions.
2: Select an element $T \in P$ uniformly at random and produce an offspring T' by applying a mutation.
3: If T' satisfies certain quality conditions, add T' to P.
4: If $|P| = \mu + 1$, remove exactly one individual T, determined by a specific criterion, from P.
5: Repeat steps 2 to 4 until a termination criterion is reached.
6: Return P.

In the following three subsections, we describe some applications for diversity optimization. They all use the $(\mu+1)$-EA, so we only concentrate on some main ideas, like measuring the diversity in this context.

2.1 Diversity Optimization in Minimum Spanning Trees (MST)

The MST problem tries to find a tree that spans all vertices in a graph and has minimal costs. In a cost setting, the edges can get weights (e.g. based on their Euclidian length). The MST problem can be solved by algorithms such as Prim's or Kruskal's. However, when we shift the problem to diversity optimization, the goal moves from finding a single optimal solution to discovering a population of spanning trees that is maximal with respect to a given diversity measure.
The diversity between two trees, T_1 and T_2, can be computed by the number of shared edges, denoted by $o(T_1, T_2)$. The trees are considered maximally diverse if $o(T_1, T_2) = 0$. This measure can be further generalized for the whole population of μ spanning trees. The diversity definition for the population would maximize the term $(n-1) - o(T_1, T_2)$ for each pair of individuals, where $(n-1)$ represents the highest achievable diversity. Thus, for a population $P = \{T_1, ..., T_\mu\}$ have the following diversity function:

$$D_\circ(P) := \mu(\mu-1)(n-1) - \sum_{i=1}^{\mu} \sum_{j=1, j\neq i}^{\mu} o(T_i, T_j)$$

This diversity measure works for the unconstrained and constrained quite well [BN21].

2.2 Diversity Optimization in Traveling Salesman Problem (TSP)

Diversity optimization in the context of the TSP is essential in generating diverse instances for algorithm selection and benchmarking. One application area is creating diverse instances that can challenge different algorithms, leading to more informative empirical runtime measurements. In general, we can consider three different objectives for TSP tours:

- **Performance Space:** The goal is to create different instances where the performance varies between different algorithms. This helps to understand the comparative strengths of the different solving strategies.
- **Feature Space:** The goal is optimizing the variety of different features.
- **Topology:** The structure of the instances themselves can also be diversified to reflect real-world structures and facilitate the creation of instances that are more representative of practical scenarios.

In this context, the paper *Evolving Diverse TSP Instances by Means of Novel and Creative Mutation Operators* introduced new operators that strongly affected the point coordinates. It was demonstrated that non-evolutionary processes are sufficient to produce diverse instances. However, it was also found that diversity in the instance space does not necessarily affect solver performance evenly and may require special attention to create instances that distinguish the different behavior of different solvers. This led to insights for algorithm selection and enables a more sophisticated understanding of how algorithms perform on different problems. This contributes to the development of more robust and adaptive solutions for TSP benchmarking. [Bos+19]

There are two ways to measure diversity in a TSP population. *Edge diversity* focuses on equalizing the representation of edges among tours in the population. One approach to measuring edge diversity can measure how often an edge occurs in the population and work to equalize this to ensure that each edge is evenly contained in the population. Another approach, *pairwise edge distance*, considers all pairs of tours in the population and calculates the edge overlap, with algorithms aimed at removing very close tours from each other by mutation. [Do+20]

2.3 Diversity Optimization in Evolutionary Art

Evolutionary art aims at the discovery of aesthetically pleasing images. It can produce diverse artistic expressions by using diversity optimization.

The diversity algorithm starts developing diverse images with an initial population of images (e.g., uniform grey squares or colored images of known objects). It applies mutation and validity checks to generate new generations of images. The feature diversity measure considers various aesthetic and general features such as Benford's law, global contrast factor, information theory, Ross bell curve, reflectance symmetry, mean hue, standard deviation of hue, and smoothness.

Experiments conducted at different resolutions and feature dimensions have shown that producing diverse artistic images within quality and time constraints is possible. This approach to diversity optimization offers opportunities to explore the artistic landscape and discover new visual compositions. [AKN17]

3 Related Work

Before demonstrating our own experiment on diversity optimization and vertex covers, we present two papers already published in this area. But first, we define vertex cover and the minimum vertex cover problem. Let $G = (E, V)$ be an undirected graph. A vertex cover is then a subset $V' \subseteq V$ such that every edge is incident with a node in V':

$$\forall (u, v) \in E : u \in V' \vee v \in V'$$

A trivial vertex cover would be $V' = V$, and for completely isolated graphs, a minimum vertex cover is $V' = \emptyset$. Now the minimal vertex cover problem demands a set V^* such that every other valid vertex cover has, at minimum, the same size:

$$V^* \textit{ is minimum vertex cover} \iff \forall V' \subseteq V : |V^*| \leq |V'|$$

In this setting, no node has any further value assigned. Nevertheless, this problem is already a $\mathcal{NP}$-hard problem as it can be reduced to the $CLIQUE$ decision problem [Kar72].

Nevertheless, solving the minimum vertex cover with population-based evolutionary algorithms is possible. The EA cycle could, for example, be a typical $(\mu + 1)$ with random mutations without paying attention to the vertex cover validity. A suitable fitness function has three parts: one for rewarding the covered edges, another for penalizing the still uncovered edges, and, of course, the number of used nodes. As a result, any cover solution will have better fitness than non-cover solutions. Oliveto et al. also propose a modified version of the $(\mu + 1)$-EA in their theoretical investigations, where the parents should be replaced if the offspring is better. This simple modification keeps for a longer time a more diverse population as other individuals also get the chance in later generations to reproduce. Otherwise, strong individuals at the beginning create already strong offsprings, which leads to a fast extinction of the weaker part. They showed that this simple diversity mechanism prevents premature convergence for bipartite graphs. [OHY08]

The previously mentioned paper has not aimed to get a very diverse final population. In contrast, in the paper *Runtime Analysis of Evolutionary Diversity Optimization and the Vertex Cover Problem*, the authors Neuman et al. focused on finding different vertex covers. They also concentrated their research on the class of bipartite graphs. The main objective was to analyze the runtime until the vertex covers meet a certain quality criterion. A binary encoding was used for the individuals, where each gene represents a node $v \in V$. If the gene bit is one, then the corresponding node is included in the vertex cover. To check how similar two vertex covers are, the authors suggest using the Hamming distance:

$$H(x, y) = \sum_{i=1}^{n} |x_i - y_i| \text{ with } x_i, y_i \in \{0, 1\}^n$$

To define the diversity of the whole population, the hamming distance of all pairs of individuals is computed. It is important to consider only the unique individuals as it could lead to local optima otherwise (an example is provided in the next section). So let be P the population and P^D the distinct population. Formally, the diversity of a population is defined as:

$$D(P) = \sum_{\{x,y\} \in P^D \times P^D} H(x, y)$$

They used this fitness function in a $(\mu + 1)$-EA, applying a random bitflip mutation on the individuals. Moreover, they mention a poorly defined quality creation that should check if the new individuum is a valid vertex cover.
With these definitions, the authors looked at how the initial population quality, population size, and bipartite graph size influence the runtime. They generally found a theoretical worst-case runtime of $\mathcal{O}(\mu^3 n^3)$ if the population fulfills some quality criteria. However, they do not present any empirical, experimental data where they also check the run time behavior on real instances. [GPN15]

4 Maximizing Diversity in Vertex Cover

The previous section has shown two things: there are methods for solving minimum vertex covers with evolutionary algorithms, and there are also ideas on how to define parts of a diversity optimization. However, the papers were limited to bipartite graph instances and only focused on the unconstrained case where each node is equally important. Therefore we decided to run experiments for different graph structures and, in addition, for constrained settings.
We add the cost function $c : V \to \mathbb{N}$ to our problem formulation for the constrained setting. It maps every vertice to an arbitrary value. Further, we define $c(C) := \sum_{v \in C} c(v)$, where C is a vertex cover. In this case, the minimum vertex cover formulation does not depend on the number of vertices in the set, but rather it considers the sum of the values of vertices in the cover. Therefore, the definition for the minimum vertex cover in this context is as follows:

$$V^* \textit{ is minimum cost vertex cover} \iff \forall V' \subseteq V(G) : c(V^*) \leq c(V')$$

This extension of the minimum vertex cover problem to a weighted variant is also motivated by its relevance in real-world applications. For instance, in telecommunication networks, nodes could have varying degrees of importance. To solve this posed problem formulation, the first step involves finding a suitable solution method.

4.1 Experimental Method

First, we used the commonly used $(\mu + 1)$-EA in the diversity optimization algorithm for our method as well. Moreover, our method is similar to the Neuman et al. paper ([GPN15]) described in the following.
We represent an individual as a binary string, where each bit represents if the corresponding node is part of the vertex cover. We use the Hamming distance as a fitness function to evaluate the difference between two individuals. We will see in the experimental investigations that the hamming distance even strives for some good-quality vertex covers. A new individual will be part of the next generation if it contributes to a higher diversity in the population. For a population P, the diversity function $D(P)$ is defined by first removing all duplicates in P and then calculating the diversity score for every pair of individuals in the distinct population. During some early experimental runs, we found that removing duplicates in the population is crucial. Otherwise, we can get stuck in local optima, as demonstrated by the following example on a graph with only two nodes and no edges:

- the population $P_1 = \{(0,0),(0,0),(1,1),(1,1)\}$ gets a diversity score of 16
- the population $P_2 = \{(0,0),(0,1),(1,0),(1,1)\}$ gets a diversity score of 16

However, while we consider P_1 not very diverse, P_2 is the optimal solution. Once we are at P_1, we cannot leave or move away from it. We cannot escape from this optimum because if we try to change just one number in one of the individuals, e.g., $\{(0,0),(0,0),(0,1),(1,1)\}$, the new score is 14. So the new individual does not make it into the population, although it is needed to reach the optimum.
Let us have a look at the distinct population method. After removing duplicates from P_1, we calculate the diversity of the shrunk population $P_1' = \{(0,0),(1,1)\}$. The diversity of P_1' is 8 and now significantly lower than that of the optimal diversity. For changing one number in P_1', we would get $\{(0,0),(0,1),(1,1)\}$, which has a diversity score of 9. This time the new individual $(0,1)$ would make it to the next generation.
After we looked at the similarities, we now present the differences between our method and the Neuman et al. approach. A significant change is the mutation operator. The mutation operator chooses a random node in the vertex cover and removes it from the cover. It then adds all adjacent nodes to the cover set to obtain a new correct solution for the vertex cover problem. This operator always produces a valid solution and can change many nodes inside the solution. Because of this property, we call this the *MultiNodeSwap* mutation. Neuman et al. used a pure random mutation operator. Random mutations in this context have the clear drawback of getting a lot of invalid solutions. Therefore a vertex cover verification check would be needed, which can be expensive for huge and dense graphs.
The downside of the *MultiNodeSwap* operator is that it sometimes needs to add more nodes to get to

a very different solution in further steps. On the one hand, in an unconstrained setting, this is no problem as there is no quality threshold. On the other hand, in a constrained setting, it can mean that we do not allow those interim solutions because they are more costly than an upper bound. To counter this problem, in some cases, we experimented with doing multiple mutations in a row. The Poisson distribution with $\lambda = 1$ determines the number of mutations.
Further to the related paper, we investigated the constrained setting, where we weighted the nodes. In the constrained setting, we used the approximation factor $\alpha \geq 0$ and only considered solutions with a cost less than $(\alpha + 1) \cdot OPT$, where OPT is the value of the optimal solution. We solved the optimal solution with an integer linear program (ILP) of the minimum (weighted) vertex cover problem to get the optimal solution. The ILP formulation is as follows:

$$\begin{aligned} \min_{x_v} \quad & \sum_{v \in V} x_v \cdot c(x_v) \\ \text{s.t.} \quad & \forall e_{ij} \in E : x_i + x_j \geq 1, \\ & \forall v \in V : x_v \in \{0, 1\} \end{aligned}$$

Like in the genetic algorithm, the problem is binary encoded. Each node v corresponds to a binary variable x_v. In the unconstrained setting, every node would have costs of one, and the objective would minimize the number of nodes in the vertex set. The used nodes are weighted by their value $c(x_v)$ in the constrained setting. To ensure that the ILP finds a vertex cover, the first constraint forces to put at least one node into the vertex set for each edge. For dense graphs, the solver can take a long time to find the optimal solution because of the exponential number of constraints in the number of nodes. So we included a timeout and took the best solution after this time. We highlighted the instances where the timeout occurred in the experimental investigation chapter.
The initial population is initialized differently for the constrained and the unconstrained setting. In the unconstrained setting, the starting population consists of μ individuals, where each solution includes all nodes. Including every node into the vertex cover initially has some advantages explained later. It has been proven for other population-based EAs that starting with a vertex cover, including all nodes or random covers, makes no significant difference in the performance for finding the optimal solution [OHY08]. Since we are not interested in an optimal solution, this should make even less difference for us.
In the constrained setting, we start with μ duplicates of a heuristically generated vertex cover C that fulfills $c(C) \leq (1 + \alpha) \cdot OPT$. To generate that population, we shuffle all edges and then add for every edge with no node in the vertex cover solution the node with the lower cost. On one hand, we would have even a 2-approximation for the unconstrained setting with this method [Vaz01]. On the other hand, we could exceed the upper bound for the constrained setting if we have an unlucky shuffle. In this case, we try it repeatedly until we have a solution that fits the quality threshold. However, even with different shuffled edges, this method does not guarantee to find a solution inside of $(1 + \alpha) \cdot OPT$ because it never adds the most costly node, although this node might be part of every solution inside the constraint. Thus, we implemented a timeout here. When reaching the timeout, we start with a population containing μ times the optimal solution the ILP solver gave.
We have described every component of our diversity optimizing evolutionary algorithm for vertex covers illustrated in Algorithm 2. A general termination criterion for the main cycle was a fixed number of generations or an early stop if the diversity of the population has stayed the same for a while. For the latter, we used an exponentially decayed number of generations depending on the size of the graph: $400 \cdot 0.99826^{|V|}$
Lastly, we look at randomly generated graph instances for our experiments. We motivate this by generalization because random graphs can be considered as a general form of graphs. So the structure is not skewed towards a specific type like bipartite graphs. However, we assumed that the density of the generated graphs could impact the number of generations and the diversity quality in the context of vertex covers. Thus, we used the density parameter Δ to generate the degree of expectation each node has. For $\Delta = 0$, we obtain isolated graphs, and for $\Delta = |V| - 1$, a complete graph. We will see in the investigation section that this assumption was correct.

Algorithm 2 Diversity Maximization for Vertex Cover using an $(\mu + 1)$-Evolutionary Algorithm

1: Begin by initializing a population set P which contains μ-times the same vertex covers (all included or 2-opt heuristic). Ensure that the costs $c(C)$ for this vertex covers C in P satisfies the inequality:

$$c(C) \leq (1 + \alpha) \cdot OPT$$

where α is an approximation factor, and OPT represents the value of the optimal solution found by an ILP.

2: Randomly select a vertex cover C from the set P. Create a new offspring vertex cover C' from C by applying the *MultiNodeSwap* mutation: Remove a random vertex node v from the vertex cover and add all adjacent nodes u to the vertex cover:

$$C' = (C \setminus v) \cup adj(v)$$

3: If the new vertex cover C' satisfies the quality condition:

$$c(C') \leq (1 + \alpha) \cdot OPT$$

then C' is added to the set P.

4: When $|P| = \mu + 1$, select an individual vertex cover C from P and remove it temporarily. Calculate the diversity $D(P')$ of the remaining population $P' = P \setminus C$. Afterwards, reinsert C back into P. Repeat this operation for each $C \in P$, and keep the population set P that maximizes $D(P')$.

5: Iterate through steps 2 to 4 until a predefined termination criterion is met.

6: Upon termination, return the final population set P.

4.2 Reproducibility

To guarantee reproducibility, we published the source code of the algorithm and our results in a GitHub repository[1]. For the implementation, we looked at different packages for evolutionary algorithms in Python, like DEAP or PyGAD. Both offer plenty of options to configure parameters and operators. However, it was difficult to configure the PyGAD library so that the evolutionary cycle behaved like the $(\mu + 1)$-EA. Usually, evolutionary algorithms use more than one parent for creating offspring, and the mutation operator typically works on each gene with some probability. [For+12][Gad21]

Because of this overhead of adjustments, we thought it was easier to implement the whole $(\mu + 1)$-EA on ourselves in Python. Further, we needed an ILP solver to find the best possible solutions, especially for the constrained case. We used for this the Gurobi solver, which offers a good Python API and academic licenses [Gur23].

For a fast execution, we paralyzed the most expensive operations like the computation of each individual's fitness. Nevertheless, the operator's runtime is exponential in the size of the population, such that it was necessary to use more advanced hardware. Thus we conducted all experiments on the HPC cluster of the RWTH Aachen.

[1] Source code and results can be found here: https://github.com/dunzel/eva_vc_diversity

5 Experimental Investigations

In this section, we present our executed experiments and describe the results and conclusions. In sum, we did 216 experiments that were performed about all cartesian combinations of the following parameter sets: $n = \{50, 100, 200\}$, $\mu = \{2, 16, 32, 64\}$, $\Delta = \{2, 4, 8\}$, mutation $= \{\text{single}, \text{poisson}\}$ and $\alpha = \{0.05, 0.5, \infty\}$. The value $\alpha = \infty$ is equivalent to no upper quality bound. After starting with the unconstrained setting, we also look at the impact of quality thresholds in the constrained setting.

5.1 Unconstrained Diversity Optimization

In this setting, we are interested in a population where every individual tries to differ from others as well as possible. The $(\mu + 1)$-EA will run as long as no significant diversity change occurs for some generations. Let us look at three basic properties first in the graph setting with 50 nodes, an expected node degree of $\Delta = 2$, and a population of $\mu = 64$ individuals. In Figure 1, we see three different line graphs going over the generations on the x-axis. The experiment was performed for a single mutation but also for a multiple mutation sequence, with the exact number for each generation drawn from the Poisson distribution.
The first graph examines the number of unique individuals in the current generation. The possible reachable maximum is μ when every individual is at least slightly different. It can be seen that the maximum number 64 is reached after about 64 generations. This is easily achievable because the search space in the unconstrained setting for sparse graphs is about $2^n \gg \mu$ individuals.
The second graph shows the fitness of the population (equal to the diversity) with the typical logarithmic shape. It is recognizable that the most significant fitness gain was done in the first tenth. Thus, the termination was not too early. Also seeable is that the Poisson distribution is gaining faster fitness. We come back to this later.
The last graph plots the vertex cover size of the best individual found in the current generation. The best individual has 50 nodes initially because we start in the unconstrained setting with all nodes in the vertex covers for all individuals. It can be clearly taken from the graph that the Poisson distribution is creating faster, better vertex covers. In the later generations, this is not true. The single mutation operator catches up. Sometimes the size of the best individual increases again, which is all right if the change contributes to more diversity. It can also be seen that in this particular run, the Poisson distribution approach still leads to changes in later generations so that it stops after about 1000 more generations than with simple mutation.

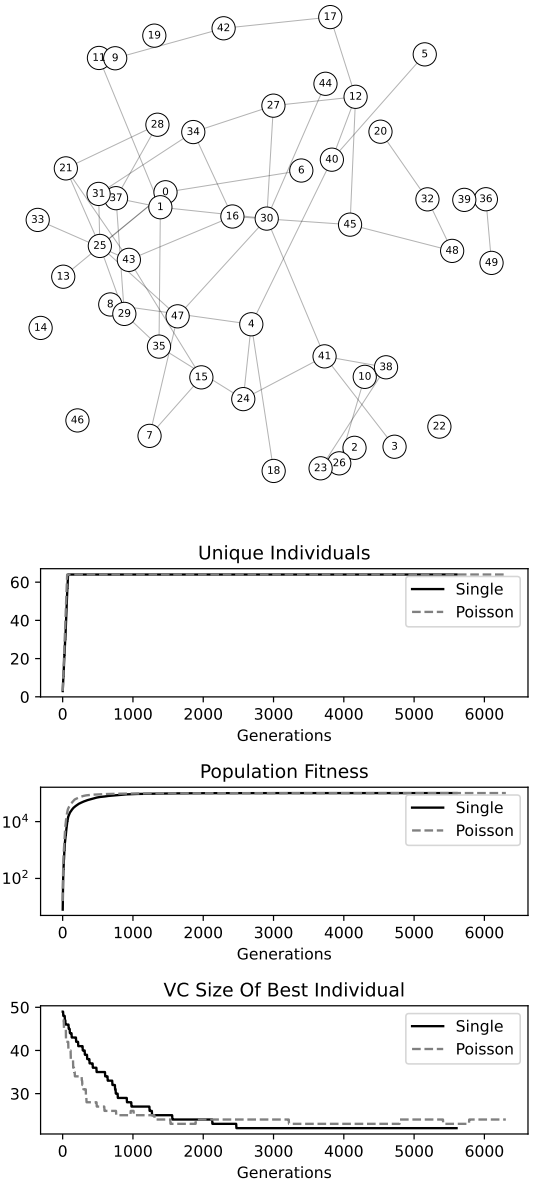


Figure 1: Three basic properties of the vertex cover diversity optimization over the generations. The underlying graph has $n = 50$ and $\Delta = 2$ and is rendered above. The EA was running with $\mu = 64$, single mutation, and a second run with Poisson mutation.

n	μ	$\Delta = 2$						$\Delta = 4$						$\Delta = 8$					
		Single			**Poisson**			**Single**			**Poisson**			**Single**			**Poisson**		
		Gen.	D_ρ	$Best_\rho$	Gen.	D_ρ	$Best_\rho$	Gen.	D_ρ	$Best_\rho$	Gen.	D_ρ	$Best_\rho$	Gen.	D_ρ	$Best_\rho$	Gen.	D_ρ	$Best_\rho$
50	2	939	**97.8**	20.0	1001	95.8	35.0	800	**80.3**	19.2	701	78.1	19.2	886	**73.5**	3.0	657	66.4	6.1
	16	2958	102.5	15.0	3901	102.5	15.0	2599	99.3	11.5	4424	**99.5**	7.7	3506	**94.8**	0.0	3581	93.7	3.0
	32	3874	102.8	10.0	4841	102.8	20.0	5499	**100.4**	7.7	6332	99.7	11.5	5854	94.7	3.0	9054	**95.3**	3.0
	64	5604	**102.6**	10.0	6305	102.8	20.0	7670	100.1	11.5	10301	**100.2**	3.8	9480	**96.1**	3.0	12250	95.7	3.0
100	2	1303	**95.9**	44.4	1432	87.7	30.6	1276	**81.2**	11.3	725	72.9	20.8	1351	**63.0**	6.1	896	54.3	12.1
	16	3434	**100.3**	36.1	6058	99.6	27.8	5590	**97.8**	9.4	8066	97.3	9.4	6941	**86.4**	3.0	5032	82.8	6.1
	32	7463	**100.4**	19.4	7321	100.1	25.0	9971	**98.2**	7.5	7723	97.1	7.5	7344	**86.1**	6.1	9836	84.7	6.1
	64	16870	**100.5**	27.8	18139	100.4	27.8	15155	**98.3**	3.8	17812	98.2	3.8	17605	87.2	3.0	27543	**87.5**	3.0
200	2	2466	**92.9**	24.7	1880	85.9	35.8	1578	78.1	12.1	2008	78.1	10.3	2312	61.4	5.3	2819	**64.0**	3.8
	16	6065	**99.3**	28.4	8951	99.1	25.9	8320	93.6	12.1	13224	**94.1**	12.1	11175	**83.0**	4.6	10767	80.1	9.2
	32	16605	**100.2**	22.2	15877	99.7	22.2	17022	**94.7**	8.4	18886	94.5	10.3	18637	**83.6**	5.3	14428	80.7	6.9
	64	22142	**100.1**	22.2	22229	99.8	25.9	22490	**95.2**	8.4	22278	94.3	10.3	23821	**83.5**	4.6	24121	82.1	6.1

Table 1: Comparison in terms of number of generations (Gen.), relative diversity (D_ρ), and deviation between der best individual in the population towards the optimal solution ($Best_\rho$). Both mutation strategies were compared by higher diversity, and the better one is always **highlighted**.

A comparison of all our experiments in the unconstrained setting is given by Table 1. The table is divided into three main columns considering the different classes of graph densities while they are divided again by the mutation strategy. Then we compare the amount generations until termination, the final diversity score, and the ratio between best found vs. best possible vertex cover. Usually, the diversity value is some absolute number higher for bigger graphs and populations with more individuals. To have the possibility to compare the diversity values within the different cells of the table, we need some relative measurements.
However, we have no theoretical formula that guarantees us a particular maximum diversity value for each setup. Therefore we decided to create some empirical bounds. The highest possible Hamming diversity for a population with valid covers is always achievable on a graph with no edges. For this kind of graph, we can take any node or no node into our vertex cover (2^n possibilities). Therefore we ran for every row a diversity optimization with the trivial graph $\Delta = 0$. To achieve the smallest possible Hamming diversity ($\underline{D}_{n,\mu}$), we need a graph that only allows a few valid vertex covers. This is the case for clique graphs ($\Delta = n - 1$). Clique graphs force us to use at least $n - 1$ of their nodes to cover all edges, which leads us to $n + 1$ different vertex covers. We can compute the maximum diversity ($\bar{D}_{n,\mu}$) for the clique graphs on our own. Every individual differs exactly in one gene from the other, and we add this up for every possible pair (including the symmetric opposite) in the population minus the self-pairs: $2 \cdot (\mu^2 - \mu)$. The given formula is only valid if $\mu \leq n$ as we would delete duplicates in the case $\mu > n$. For the second case, the maximum diversity would always be $2 \cdot n \cdot n$. We now have the maximum diversity values for both extreme cases of Δ, so we did a linear interpolation for our $\Delta \in \{2, 4, 8\}$ values. Eventually, our relative diversity is now defined as:

$$D_\rho(\Delta, n, \mu) = \underline{D}_{n,\mu} + \left(\frac{\Delta}{n-1}\right) \cdot (\bar{D}_{n,\mu} - \underline{D}_{n,\mu})$$

We can identify that this estimated upper diversity bound is close to a real one since the maximum relative value is slightly over a hundred in some cells. Exceeding the hundred percent mark indicates that there is no perfect linear behavior. Lastly, the formula for the deviation from the best vertex cover is:

$$Best_\rho = \frac{\text{Size of best found vertex cover in population}}{\text{Size of best possible vertex cover found by ILP}} - 1$$

The first thing we can observe from highlighting the D_ρ values is that the singular mutation strategy seems to perform better. An initial assumption was that the Poisson strategy would outperform the single strategy, which is clearly not the case. The explanation for these results could be related to the λ parameter. Maybe too many mutation steps lead back to already known individuals in the unconstrained setting instead of exploring first the local neighborhood. An idea would be to use a lower *lambda* such that we use single mutations more often and less multiple in a sequence.
The heatmap in Figure 2 is helpful to see more relations between the three properties and the parameters. We considered only the single mutation strategy during the aggregation of the values. A heatmap for the Poisson strategy would look similar.
The first row of three heatmaps looks at the influence on the generations. The highest impact on the generations has the size of the graph because more nodes allow more possible combinations of vertex covers such that we need more generations to observe them. The last argument is also valid for larger population sizes. The graph density has a minor influence on the runtime compared to the last two attributes.

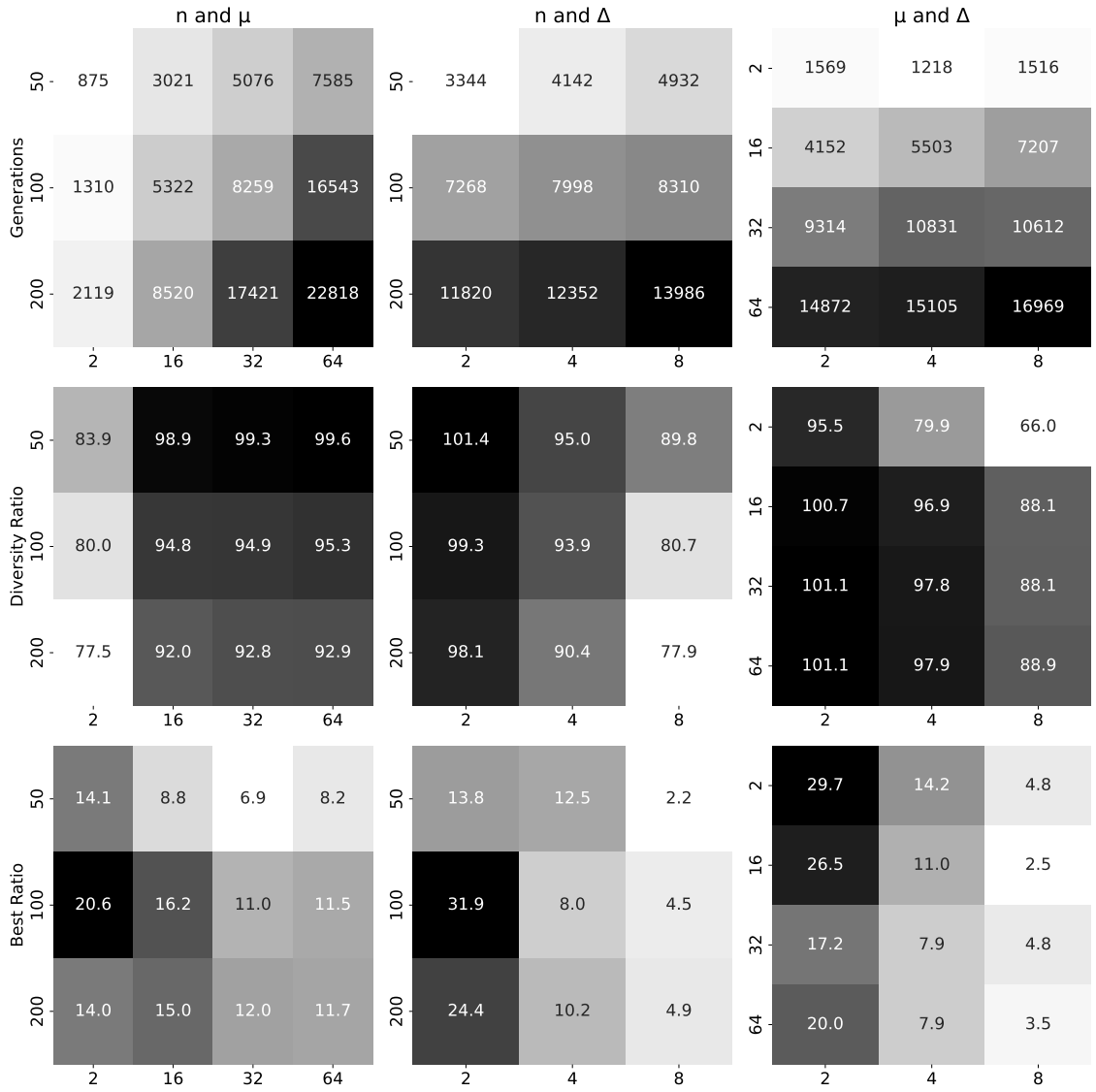


Figure 2: Heatmap of aggregated values in Table 1. Each column combines two of the properties in $\{n, \mu, \Delta\}$ (columns) and compares them with the generations, diversity ratio D_ρ, and best ratio $Best_\rho$ (rows). The values are only aggregated for the singular mutation strategy.

Let us now have a look at the diversity measurement. We can claim that in the unconstrained setting, our diversity optimization reaches good diversity ratios as they are, for most cases, above 90% on average. Starting with the left heatmap, we can see that the first column for the population size of two is an outlier as the other population sizes reach higher diversity values again. There is a simple explanation for this behavior. There will rarely be a chance for a Hamming distance of n for two individuals in a graph with a $\Delta > 0$. The reason is that random graphs might contain subgraphs with cliques. For these subgraphs, it is impossible to invert the vertex cover such that even in a population of two, the instances can not reach the highest Hamming distance. With a greater population size, there is more scoop for reaching higher diversity scores again. In contrast, the size of the graph has a minor impact on the maximal diversity achieved by our algorithm, as can be seen on both left heatmaps. Like population size, density plays a crucial role in achieving higher diversities. For graphs with higher diversity, the probability of cliques increases such that the same argumentations counts here. Finally, the best ratio reveals to us that the Hamming distance as a diversity measure also strives by itself to good quality graphs. This is shown by the last row of heatmaps, where the deviation is mostly around 10%. Better individuals are usually found when the population increases because we have more time to explore the local neighborhood. Also, a higher graph density leads to better results, as they are closer to the start population where already every node is included in the vertex cover. As an interim summary, we can see different behaviors of our diversity optimization depending on the graph density but not size. Additionally, a greater population can help gain diversity and quality.

Until now, we only considered the Hamming diversity, which was definitely a helpful measure to evolve the population. However, we might be further interested in how often a node occurs in multiple vertex covers. Ultimately, the node overlap between two individuals can be considered as a negative Hamming distance. We measured the mean and standard deviation overlap of the individuals in populations. Moreover, in real-world applications, we are interested in the resulting population's structural properties. Eventually, it can play a decisive role in how many nodes are leaves or what degrees the nodes in the vertex cover have.

All four mentioned properties are depicted in Figure 3 for the graph provided in the first figure. Let us have a short look at all of the four graphs. The mean overlap decreases over the generations while the diversity increases. Similar to the diversity, the Poisson distribution is faster in the beginning. The node overlap, on average, is 50 at the beginning because of the initial all-one individuals.

The standard deviation overlap is more interesting, where we can see why the Poisson distribution is so powerful initially. In contrast to the single mutation strategy, the Poisson strategy allows a higher variance in the beginning, reaching more different individuals in the beginning. After reaching the 4000th generation, the Poisson distribution seems to lose this advantage. An adjusted strategy could use an adaptive λ that gets smaller over time.

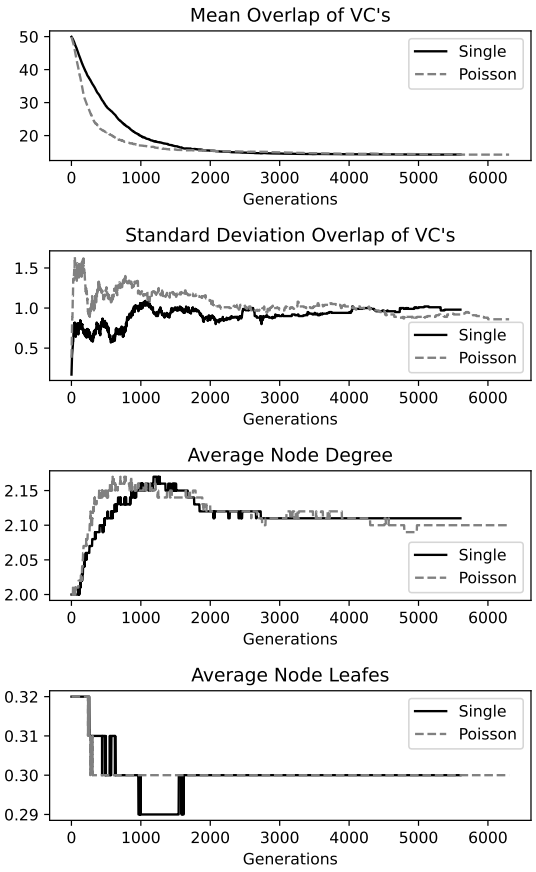


Figure 3: Four structural properties of the vertex cover diversity optimization over the generations. The underlying graph is again configured with $n = 50$ and $\Delta = 2$. The EA was running with $\mu = 64$, single mutation, and a second run with Poisson mutation.

The average node degree for this graph shows that it is a $\Delta = 2$ structured graph instance, as the average starts at 2. Then the average node degree begins to climb, which makes sense as we want to cover with fewer nodes more edges. Nevertheless, it stays close to 2 as we still accept bad individuals where we would include isolated vertex nodes in the cover for increasing diversity. In the constrained setting, such nodes are less included, making the node degree more away from the initial Δ. Lastly, we can again see the different behavior of the Poisson strategy in the beginning.
In contrast to the average node degree, the average amount of node leaves decreases. This makes sense as we included all node leaves in the beginning into the vertex cover and created more and more individuals trading them against other nodes. It can happen that we may resume node leaves again to gain diversity. This resume action happened, for example, around generations 1000 in the depicted run.
Table 2 displays the final population values of the four mentioned properties for almost all configurations in the unconstrained setting. For the mean overlap, it is recognizable that the values for all population sizes for a fixed n are similar, and $\mu = 2$ is an outlier for the $\Delta = 2$ graph instance. We can also see again that the degree average is always close to Δ as expected. For the leaves, we can observe that number decreases for a higher Δ, which makes sense as this graph structures naturally exist with fewer leaves.

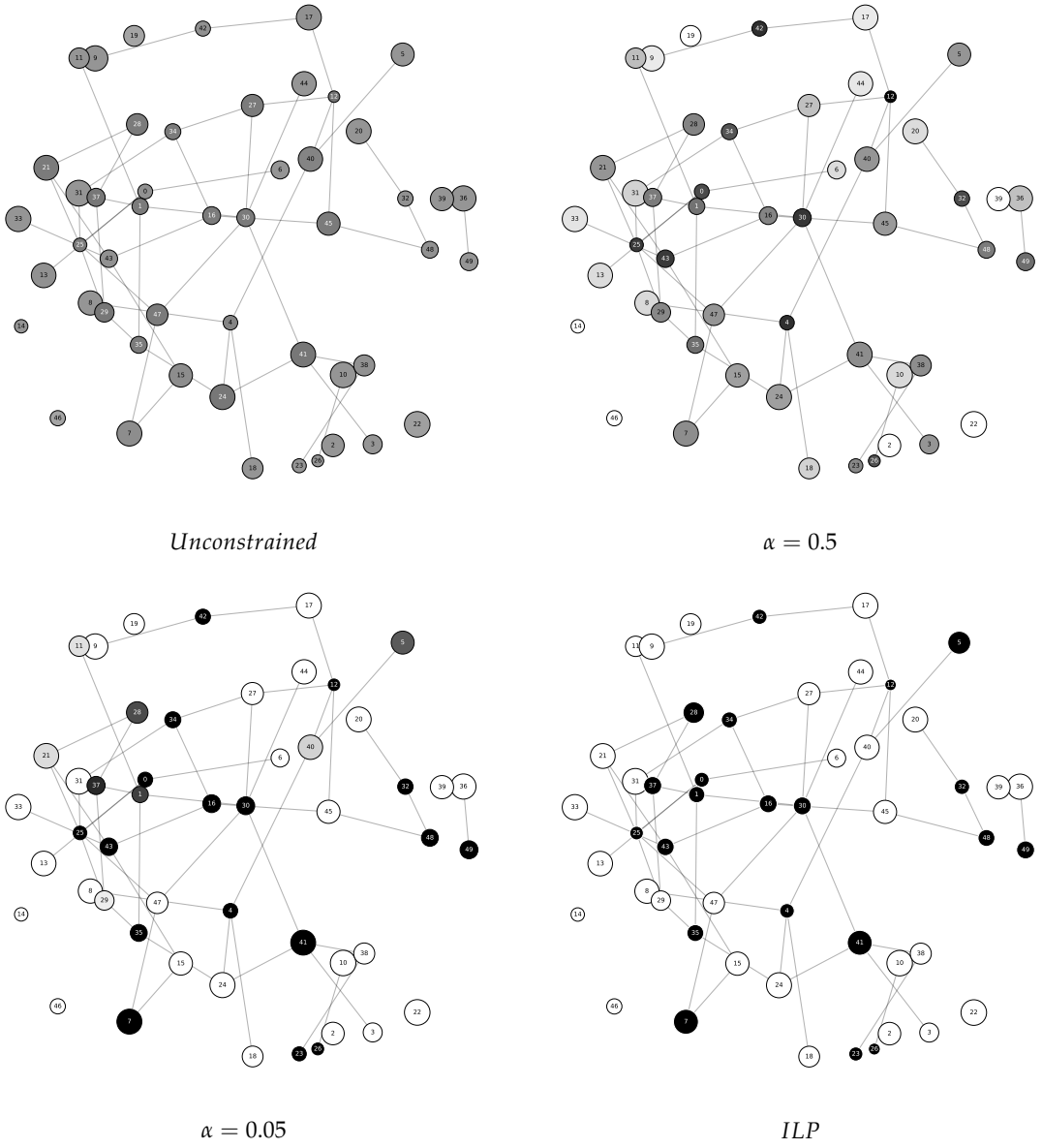
n	μ	$\Delta = 2$								$\Delta = 8$							
		Single				**Poisson**				**Single**				**Poisson**			
		$mean_{\Pi}$	std_{Π}	deg_v	$leaf_v$	$mean_{\Pi}$	std_{Π}	deg_v	$leaf_v$	$mean_{\Pi}$	std_{Π}	deg_v	$leaf_v$	$mean_{\Pi}$	std_{Π}	deg_v	$leaf_v$
50	2	3.00	0.00	2.07	0.30	4.00	0.00	2.11	0.30	19.00	0.00	8.47	0.00	22.00	0.00	8.36	0.00
	16	13.54	0.70	2.13	0.30	13.98	0.87	2.13	0.29	24.63	0.59	8.25	0.00	25.07	0.58	8.25	0.00
	32	14.19	0.93	2.11	0.30	14.06	0.60	2.10	0.30	24.94	0.46	8.25	0.00	24.78	0.46	8.22	0.00
	64	14.26	0.98	2.11	0.30	14.23	0.86	2.10	0.30	24.88	0.49	8.23	0.00	25.03	0.44	8.24	0.00
100	2	6.00	0.00	2.16	0.27	6.00	0.00	2.33	0.30	42.00	0.00	8.44	0.01	50.00	0.00	8.35	0.01
	16	28.54	1.98	2.17	0.27	27.58	1.35	2.20	0.27	50.40	1.00	8.22	0.01	53.16	0.80	8.24	0.01
	32	28.00	2.52	2.16	0.27	28.26	1.40	2.16	0.28	51.15	0.60	8.21	0.01	51.93	0.66	8.25	0.01
	64	28.12	1.77	2.14	0.27	28.19	1.86	2.15	0.27	50.33	0.73	8.24	0.01	50.13	0.83	8.23	0.01
200	2	16.00	0.00	2.09	0.27	26.00	0.00	2.15	0.27	82.00	0.00	8.44	0.00	77.00	0.00	8.50	0.00
	16	59.19	2.28	2.09	0.26	56.51	1.82	2.12	0.27	101.13	2.25	8.20	0.00	105.36	1.34	8.20	0.00
	32	56.88	2.20	2.09	0.27	57.03	1.79	2.11	0.27	101.44	1.37	8.22	0.00	105.82	1.46	8.22	0.00
	64	56.93	1.94	2.08	0.27	58.31	2.02	2.08	0.27	101.79	1.45	8.21	0.00	103.95	1.56	8.21	0.00

Table 2: A comparison of the final structure values in the unconstrained runs. Due to space constraints, $\Delta = 4$ was omitted because the results are very similar. $mean_{\Pi}$ is the average vertex cover overlap and std_{Π} is its standard deviation. deg_v is the average node degree, and $leaf_v$ is the average number of leaves contained in the final populations.

Let us conclude with an interim summary of the constrained section, summarizing the main results and findings. In the unconstrained diversity optimization, we find that, contrary to our initial expectations, the single mutation strategy consistently performs better than the Poisson distribution approach. The adaptive nature of the experiment, highlighted by the effects of population size and graph density on diversity and quality, shows that different graph structures can significantly affect the outcome. Finally, the examination of structural properties such as node overlap and its standard deviation reveals the underlying dynamics and points to potential improvements through adaptive strategies, especially in the early stages, providing opportunities for further optimization and applicability in practice.

5.2 Constrained Diversity Optimization

We now concentrate on constrained diversity optimization. In this setting, the vertexes get costs. The costs are considered during the calculation of the minimal vertex cover. We have run the diversity optimization for $\alpha \in \{0.05, 0.5\}$. Let us dive into the constrained setting by looking at Figure 4. Again we consider the graph with 50 nodes and an expected node degree of 2. The node size in this rendering can vary. A smaller node indicates lower costs for using this one, and a bigger node indicates higher costs. Additionally, the node color can go from white to black. The coloring depends on how often the node was taken into the vertex covered by the population. The coloring is normalized by the population size $\mu = 64$. Pure black would mean that every individual took this node, and pure white means no individual used it for its solution. For a better comparison, we also rendered the unconstrained solution and the ILP solution.



Unconstrained $\alpha = 0.5$

$\alpha = 0.05$ *ILP*

Figure 4: Visualized node usage on the graph $n = 50$ and $\Delta = 2$ instance. The nodes have different sizes and indicate the assigned weight. Also, the nodes have a grey color that gets darker if more individuals in a population use a certain node. All populations had a size of $\mu = 64$.

Looking at the top left graph, we see that every node has almost the same color. The reason for that is that in the unconstrained setting, the diversity optimization tries to distribute the usage of the nodes. Therefore it also includes completely isolated high-cost nodes that are not covering edges into the vertex cover.
We can see the result by adding a quality threshold in the top right graph. No solution is now allowed to be worse than $1.5 \cdot OPT$. The population does not include the isolated nodes anymore. It can also be seen that nodes with a higher grade and smaller size tend to be more black.
We achieve the graph bottom left if we only allow a 5% deviation from the optimal solution. Comparing it with the optimal solution on the bottom right, we can see that the population already uses the optimal nodes mainly. However, it sometimes swaps nodes that have equal costs. As a result, we obtain different high-quality solutions where a human decision-maker can pick, in the end, the most suitable.

n	μ	$\alpha = 0.05$						$\alpha = 0.5$						$\alpha = \infty$					
		Single			**Poisson**			**Single**			**Poisson**			**Single**			**Poisson**		
		Gen.	D_ρ	$Best_\rho$	Gen.	D_ρ	$Best_\rho$	Gen.	D_ρ	$Best_\rho$	Gen.	D_ρ	$Best_\rho$	Gen.	D_ρ	$Best_\rho$	Gen.	D_ρ	$Best_\rho$
50	2	669	**34.7**	3.6	658	23.9	0.0	843	**78.1**	0.0	948	71.6	10.7	800	**80.3**	19.2	701	78.1	19.2
	16	1026	17.2	0.0	3838	**50.7**	0.0	3603	**93.3**	-3.6	3044	93.1	3.6	2599	99.3	11.5	4424	**99.5**	7.7
	32	1330	4.1	0.0	1788	**6.9**	0.0	5128	93.3	0.0	8036	**93.8**	7.1	5499	**100.4**	7.7	6332	99.7	11.5
	64	1274	**1.1**	0.0	500	0.2	0.0	5254	93.6	-3.6	9567	**93.7**	0.0	7670	100.1	11.5	10301	**100.2**	3.8
100	2	405	16.7	7.3	410	**17.7**	5.5	1429	79.1	9.1	1597	79.1	0.0	1276	**81.2**	11.3	725	72.9	20.8
	16	4073	**51.6**	1.8	2876	37.7	1.8	5193	**94.1**	0.0	5309	93.1	7.3	5590	**97.8**	9.4	8066	97.3	9.4
	32	9614	**54.9**	-1.8	9184	52.0	1.8	8088	**94.8**	5.5	12135	94.7	3.6	9971	**98.2**	7.5	7723	97.1	7.5
	64	11460	51.5	1.8	9737	**51.9**	1.8	17177	94.9	3.6	16671	**95.3**	3.6	15155	**98.3**	3.8	17812	98.2	3.8
200	2	1058	28.1	4.4	1144	**30.6**	1.8	1578	73.5	14.2	2979	**76.0**	8.0	1578	78.1	12.1	2008	78.1	10.3
	16	6136	**47.7**	1.8	5396	39.8	1.8	8111	93.5	9.7	12745	93.5	8.0	8320	93.6	12.1	13224	**94.1**	12.1
	32	5370	**44.8**	3.5	11437	43.7	0.9	13304	**94.0**	5.3	18376	93.9	5.3	17022	**94.7**	8.4	18886	94.5	10.3
	64	14975	**52.7**	0.9	16767	47.8	1.8	20921	**94.3**	8.8	21635	93.4	7.1	22490	**95.2**	8.4	22278	94.3	10.3

Table 3: Similar to Table 1 but for the constrained case. For simplicity reasons, we fixed $\Delta = 4$. Both mutation strategies were compared again by higher diversity, and the better one is always **highlighted**.

In Table 3, we gathered our results the same way as in the unconstrained setting. For a better overview, we fixed $\Delta = 4$. First of all, the number of generations is slightly less. The decreased number of generations is because of the early stopping mechanism, as it gets harder to have significant changes still in the valid range. Also, it can be seen that it gets more challenging to gain a higher diversity because many solutions are not in the valid quality threshold anymore. While for an $\alpha = 0.5$, it is still possible to get over 90% diversity, the diversity scores for $\alpha = 0.05$ are barely over 50%. In contrast to the unconstrained change, we can notice that the Poisson strategy might work better here. Because it can produce intermediate solutions outside the valid quality threshold and then come back inside the range by some luck. Looking at $Best_\rho$, we can see that it gets closer to the best one. However, the measurement here considers the number of nodes in the vertex cover, not the costs. Therefore, some best-found individuals use fewer nodes (but more expensive ones) than the best cost cover.

For the sake of completeness, the structure table is also given in Table 4. It is noticeable that the standard deviation is minor, the degree of the nodes is generally higher, and leaves are rarely used in the higher constrained setting. When leaves are used, it is mainly because of subgraphs with two nodes.

n	μ	$\alpha = 0.05$								$\alpha = 0.5$							
		Single				**Poisson**				**Single**				**Poisson**			
		$mean_\Pi$	std_Π	deg_v	$leaf_v$	$mean_\Pi$	std_Π	deg_v	$leaf_v$	$mean_\Pi$	std_Π	deg_v	$leaf_v$	$mean_\Pi$	std_Π	deg_v	$leaf_v$
50	2	20.00	0.00	4.79	0.00	23.00	0.00	4.74	0.00	11.00	0.00	4.42	0.07	14.00	0.00	4.51	0.07
	16	25.82	0.65	4.86	0.00	21.47	0.50	4.76	0.01	18.06	0.67	4.42	0.06	18.33	0.54	4.40	0.07
	32	26.45	0.66	4.80	0.00	26.28	0.66	4.83	0.00	18.41	0.62	4.41	0.07	18.39	0.54	4.38	0.07
	64	26.04	0.65	4.85	0.00	27.14	0.44	4.84	0.00	18.53	0.58	4.40	0.07	18.47	0.59	4.41	0.07
100	2	50.00	0.00	4.61	0.04	48.00	0.00	4.78	0.03	21.00	0.00	4.32	0.03	21.00	0.00	4.43	0.04
	16	43.08	0.52	4.67	0.03	48.25	0.58	4.69	0.03	36.18	1.42	4.32	0.03	37.38	1.02	4.31	0.03
	32	42.38	0.67	4.65	0.03	43.72	0.67	4.66	0.04	35.83	0.84	4.28	0.03	35.73	1.08	4.30	0.03
	64	44.31	0.72	4.65	0.04	43.98	0.56	4.64	0.04	35.92	1.12	4.28	0.03	35.41	0.67	4.27	0.03
200	2	88.00	0.00	4.63	0.04	86.00	0.00	4.57	0.06	55.00	0.00	4.32	0.09	50.00	0.00	4.35	0.08
	16	91.47	0.89	4.54	0.06	95.10	0.81	4.57	0.06	75.96	1.58	4.21	0.08	76.30	1.67	4.19	0.08
	32	93.39	0.62	4.58	0.06	93.94	0.71	4.57	0.05	77.24	1.36	4.19	0.08	77.21	1.43	4.20	0.09
	64	88.63	0.64	4.57	0.06	92.45	0.80	4.56	0.06	76.19	1.55	4.19	0.08	78.46	1.42	4.19	0.08

Table 4: A comparison of the final structure values in the constrained runs with a $\Delta = 4$. $mean_\Pi$ is the average vertex cover overlap and std_Π is its standard deviation. deg_v is the average node degree, and $leaf_v$ is the average number of leaves contained in the final populations.

5.3 Weaknesses Of The Mutation Operator

We conclude this chapter with weaknesses of the *MultiNodeSwap* operator, which we obtained during our experiments. One example would be a star graph. A star graph $G(V, E)$ has one central node that is connected to every other node and no other edges. The optimal vertex cover solution includes one node (the central) for star graphs. To get a diverse set of solutions, we need to add more nodes to the optimal vertex cover, but when we use our mutation operator on the optimal solution, we end with a vertex cover with $|V| - 1$ nodes. Afterward, the operator can, step by step, remove all of the center node surrounding nodes. But in a constrained setting there, the first step will move the solution outside of the range of alpha, which prevents us from finding any other solutions except for the optimal when we start at the optimal solution.
Another problem are independent nodes without any edges. Those nodes can never be added to the solution because there is no adjacent node to remove from the vertex cover. On the one hand, we decided that this is not a big problem because, in a constrained setting, those nodes produce an unnecessary cost. On the other hand, in an unconstrained setting, we start with a population including these nodes so that if they are important for diversity, we just never remove them from the solution.

6 Conclusion

In this paper, we comprehensively investigated maximizing diversity in (weighted) vertex covers, basing our investigation on an experimental context.
After introducing the background and related work, we began our study with a detailed description of our experimental procedure. Through extensive experiments, we developed and tested methods for optimizing diversity in vertex cover problems with and without constraints. We used multiple parameters, such as α, to explore different scenarios. Moreover, we created the *MultiNodeSwap* operator, for which we empirically showed a good performance.
Our results showed fascinating behavior regarding how the diversity of vertex cover is affected by the constraints, the cost of the vertices, and the choice of specific strategies, such as the single mutation or Poisson strategy. Various figures and tables were presented to illustrate results in runtime, diversity, quality, and structural properties fully.
Looking ahead, we recognize that the principles and techniques presented in this work apply to other vertex cover problems and enrich areas such as Maximum Independent Set. The study has shown that it can be further refined with, for example, an adaptive λ in the Poisson strategy.
However, we only run the diversity optimization for each setup only once. Running the algorithm multiple times with different parameters can improve statistical reliability and reveal more sophisticated patterns. In addition, future work could formulate theoretical equations to calculate the maximum diversity for each Δ. This theoretical basis can complement and strengthen the empirical approach.

We hope that our study contributes to diversity optimization in vertex cover problems. Moreover, the identified weaknesses and opportunities to use our open-source code can be useful for future research.

References

[ES15] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*. Springer, 2015.

[MLG06] Rafael Martı, Manuel Laguna, and Fred Glover. "Principles of scatter search". In: *european Journal of operational Research* 169.2 (2006), pp. 359–372.

[OHY08] Pietro S. Oliveto, Jun He, and Xin Yao. "Analysis of population-based evolutionary algorithms for the vertex cover problem". In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. 2008, pp. 1563–1570. DOI: 10.1109/CEC.2008.4631000.

[Dan23] Sven Farwig Daniel Zelenak. *Vertex Cover Diversification Code*. https://github.com/dunzel/eva_vc_diversity. 2023.

[BN21] Jakob Bossek and Frank Neumann. "Evolutionary diversity optimization and the minimum spanning tree problem". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, June 2021. DOI: 10.1145/3449639.3459363. URL: https://doi.org/10.1145/3449639.3459363.

[Jon95] Terry Jones. "Crossover, Macromutation, and Population-based Search". In: (June 1995). URL: https://www.researchgate.net/publication/2611224_Crossover_Macromutation_and_Population-based_Search.

[Bos+19] Jakob Bossek et al. "Evolving diverse TSP instances by means of novel and creative mutation operators". In: *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*. ACM, Aug. 2019. DOI: 10.1145/3299904.3340307. URL: https://doi.org/10.1145/3299904.3340307.

[Do+20] Anh Viet Do et al. "Evolving diverse sets of tours for the travelling salesperson problem". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. ACM, June 2020. DOI: 10.1145/3377930.3389844. URL: https://doi.org/10.1145/3377930.3389844.

[AKN17] Brad Alexander, James Kortman, and Aneta Neumann. "Evolution of artistic image variants through feature based diversity optimisation". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, July 2017. DOI: 10.1145/3071178.3071342. URL: https://doi.org/10.1145/3071178.3071342.

[Kar72] Richard M. Karp. "Reducibility Among Combinatorial Problems". In: *Complexity of Computer Computations*. Ed. by R. E. Miller, J. W. Thatcher, and J.D. Bohlinger. New York: Plenum, 1972, pp. 85–103. ISBN: 978-1-4684-2003-6. DOI: 10.1007/978-1-4684-2001-2_9.

[GPN15] Wanru Gao, Mojgan Pourhassan, and Frank Neumann. "Runtime Analysis of Evolutionary Diversity Optimization and the Vertex Cover Problem". In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, July 2015. DOI: 10.1145/2739482.2764668. URL: https://doi.org/10.1145/2739482.2764668.

[Vaz01] Vijay V Vazirani. *Approximation algorithms*. Vol. 1. Springer, 2001.

[For+12] Félix-Antoine Fortin et al. "DEAP: Evolutionary Algorithms Made Easy". In: *Journal of Machine Learning Research* 13 (July 2012), pp. 2171–2175.

[Gad21] A. F. Gad. *PyGAD: Genetic Algorithm in Python*. https://github.com/ahmedfgad/GeneticAlgorithmPython/. Accessed: 2023-07-31. 2021.

[Gur23] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023. URL: https://www.gurobi.com.