Grey-Box Optimization for the MaxCut problem

Alexandra Neagu
A.I.Neagu@student.tudelft.nl
Delft University of Technology
Delft, Zuid-Holland, Netherlands

Jan Warchocki
J.Z.Warchocki-1@student.tudelft.nl
Delft University of Technology
Delft, Zuid-Holland, Netherlands

Davis Kažemaks
D.Kazemaks@student.tudelft.nl
Delft University of Technology
Delft, Zuid-Holland, Netherlands

Artjom Pugatšov
A.Pugatsov@student.tudelft.nl
Delft University of Technology
Delft, Zuid-Holland, Netherlands

Tim Huisman
T.J.Huisman-1@student.tudelft.nl
Delft University of Technology
Delft, Zuid-Holland, Netherlands

ABSTRACT

The Max-Cut problem involves partitioning the vertices of a graph into two sets to maximize the weight of edges between them, which has applications in many areas, such as statistical physics [1]. Given its NP-complete nature, exact solutions are often impractical, leading to the exploration of approximate algorithms [11]. This paper evaluates and compares black-box and grey-box evolutionary algorithms for solving the Max-Cut problem. We focus on three research questions: the benefits of domain knowledge in grey-box optimization, improving the performance of a simple Genetic Algorithm (sGA) with problem-specific knowledge, and comparing the performance of sGA in black-box and grey-box settings. Our experiments demonstrate that grey-box methods generally outperform black-box methods, with the Qinghua algorithm [11] showing significant advantages for specific graph topologies. Incorporating local search universally enhances performance, often diminishing the differences between crossover methods. However, adaptive mutation did not yield significant improvements and introduced computational overhead. Future work should explore broader problem instances, optimize adaptive mutation, and investigate alternative metrics for evaluation. The source code of this project can be found at https://github.com/TimHuisman1703/simpleGA_python.

KEYWORDS

max-cut, grey box optimization, evolutionary algorithms

1 INTRODUCTION

In the Max-Cut problem, given is a graph $\mathcal{G}$ with a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. The goal is to then assign a colour $c_i \in \{0, 1\}$ to each vertex $v_i \in \mathcal{V}$, such that the sum of the weights of the edges between nodes of different colours is maximized. Precisely, the goal is to maximize the objective function $\mathcal{J}(c)$, defined as:

$$\mathcal{J}(c) = \sum_{\{i,j\} \in \mathcal{E}} w_{ij}(1 - \min\{c_i, c_j\}) \max\{c_i, c_j\} \tag{1}$$

where w_{ij} is the weight of the edge between vertices v_i and v_j. The Max-Cut problem has found practical applications in circuit layout design and statistical physics [1, 11].

There are two main classes of algorithms that attempt to solve this problem: exact [4, 10] and approximate [3, 5, 8, 11]. Since Max-Cut is known to be an NP-complete problem [11], and the exact solution is not always necessary, the approximate algorithms are often of higher interest. The approximate solutions can be further divided into *black-box* (no knowledge about $\mathcal{G}$), *grey-box* (partial knowledge of $\mathcal{G}$), and *white-box* (full knowledge of $\mathcal{G}$). Many modern approximate solutions assume the grey-box setting [3, 11].

In this work, we evaluate and compare different black-box and grey-box evolutionary algorithms for solving the Max-Cut problem. To this end, we aim to answer the following three research questions:

- **RQ1:** For which type(s) of MaxCut instances is it beneficial to use domain knowledge provided by the GBO setting?
- **RQ2:** How can the performance of the sGA be improved in a GBO setting through the use of problem-specific knowledge for these type(s) of instances?
- **RQ3:** How does the performance of the sGA for GBO compare to that of the sGA for BBO?

The contributions of this paper are, therefore, three-fold. First, we discuss and implement different black-box and grey-box algorithms for solving Max-Cut. Next, we evaluate these algorithms on various problem setups and report the performance. We then discuss on which problem instances it is (not) beneficial to use the domain knowledge from the grey-box setting.

2 BACKGROUND

To solve the Max-Cut problem it is common to define the genotype as the colours assigned to each vertex [3, 11]. Hence, we define the genotype x as $x = [c_0, c_1, \ldots, c_{n-1}]$. The fitness function is then defined as $f(x) = \mathcal{J}(x)$, where $\mathcal{J}(x)$ is as specified in Equation 1. That is, the fitness function is defined as the Max-Cut score of the genotype x. We then aim to maximize the fitness function.

2.1 Black-box optimization

In this type of optimization, no knowledge about the graph $\mathcal{G}$ is assumed. Following the provided template code, three different black-box variation operators are available. Those are one-point, two-point, and uniform crossovers. Given two parents p_i and p_j, the variation operators always return two new offspring o_1 and o_2. Parents are selected at random, making sure each parent is considered exactly once. To select the new population, we perform tournament selection of size 4 on the combined population and offspring $P + O$.

2.2 Grey-box optimization

Similarly to [3], in the grey-box optimization, we allow for partial evaluations of the fitness function. In particular, we allow the algorithm to calculate the updated fitness value given a change to a part of the genotype x. In other words, we can measure the cost of recolouring certain vertices in the graph without re-evaluating the entire fitness function. This gives the algorithm domain knowledge that could be exploited to find better solutions more efficiently than in the black-box setting.

To ensure fair comparison against black-box algorithms, we count these partial evaluations towards the total amount of evaluations. Thus, querying for a recolouring of a vertex v_i with n_i neighbours counts towards the evaluation cost as $n_i/|\mathcal{E}|$. In the following section, we describe the exact algorithms used for grey-box optimization.

3 METHODS

Baseline black-box variation operators do not utilize the knowledge of the graph topologies and can only perform full evaluations of genotypes. In a grey-box setting, we can use partial evaluations to enable new operators or searching methods that can converge faster to a good or optimal solution while decreasing computational costs.

Additionally, baseline implementations do not use any form of mutation. We believe that mutation could help algorithms escape local optima and find optimal solutions in cases where any of the operators converge prematurely.

3.1 Greedy crossover

Greedy is a general strategy that picks the most optimal choice given current information without considering the impact of this choice on future consequences. While greedy approaches cannot find the most optimal solution for all Max-Cut problems, it is a good heuristic to use since most of the greedy algorithm implementations can reach at least 0.5-approximate algorithm performance [6].

Based on this idea and the given grey-box setting, we created a greedy-crossover operator that uses currently available information to pick the most optimal choice. This will increase the average fitness of each new generation and will converge to a good or optimal value much quicker.

To create the next generation, two parents are selected, and both give their entire genotype to one of the offspring. If offspring share the same symbol at the same locus, it is preserved for both offspring. Otherwise, local optimum search is employed for each of the offspring to find the most optimal value to place within the corresponding gene. To reduce the risk of losing diversity too quickly, randomness is incurred when selecting the optimal symbol for the gene.

To assess the importance of random mutations in this algorithm, an additional operator was created that has a small chance of introducing non-adaptive random bitflips within the genotype. This will be referred to as "GreedyMut" in future sections.

Additional computations need to be performed to calculate fitness when looking for the local optimum. For Max-Cut in a grey-box setting, partial evaluations can be performed by changing the colour of a vertex and summing its edges. The colour with the highest local fitness will be considered as the optimal value. This partial evaluation is much cheaper than doing a full evaluation to derive the total fitness value.

Overall, this approach increases the time complexity of the algorithm by a factor relative to the degree of each vertex in the graph if compared to simple crossover algorithms such as uniform crossover. We believe that this complexity increase is mitigated by the fact that the greedy crossover can converge quicker to a good solution than a black-box crossover.

3.2 Approach from Wu et al. (Qinghua)

In [11], Wu et al. propose a new framework for optimizing grey-box Max-Cut problems using evolutionary algorithms. As opposed to black-box and greedy crossovers, the method only creates a single offspring at each generation. The offspring is then evaluated, and selection is performed. The selection simply selects the best N solutions where N is the population size. The authors additionally perform a tabu search on the offspring as an attempt to improve it. The proposed tabu search is expensive, and thus, we replace it with a custom local search operation, described in a later section.

The parent selection and crossover method require more attention. The parent selection calculates the Hamming distance $d(p_i, p_j)$ between the parents p_i and p_j, defined as the number of bits different in the parent genotypes. The average Hamming distance between all pairs of parents $\hat{d} = \frac{1}{N^2} \sum_{i,j} d(p_i, p_j)$ is then calculated. An offspring is then created from a random set of parents whose Hamming distance $d(p_i, p_j) \geq \hat{d}$. In this way, the authors claim [11], diversity is preserved in the population as only sufficiently varied parents are used in crossover.

During the crossover, the new offspring first inherits genes that both parents agree on. For the remaining bits a contribution value $VC_i^c(x)$ of assigning a bit i from genotype x to color c is used. The contribution value is given by $VC_i^c(x) = \sum_{x_j \in x, x_j = 1-c} w_{ij}$ and therefore measures the sum of the weights between the given node i and all other nodes whose colour is the opposite of c. To perform the crossover on the non-agreeing $k \leq n$ bits, the algorithm alternatively chooses one of the parents p_i and p_j, finds the bit with the highest contribution value if assigned with the colour of the parent, and places it in the genotype.

The paper by Wu et al. shows promising performance on the Max-Cut problem, being able to find new optima for several problems. In this work, we modify the tabu search to use a custom local search and compare the algorithm against the other discussed methods. In the rest of this work, we refer to this method as Qinghua from the first name of the first author.

3.3 Adaptive mutation

In evolutionary algorithms, mutation is a critical operator that introduces genetic diversity into the population, preventing premature convergence to local optima. Traditional mutation strategies apply a fixed mutation rate, which can be suboptimal: a high mutation rate can disrupt good solutions, while a low mutation rate can lead to premature convergence. Adaptive mutation dynamically adjusts the mutation rate based on the diversity of the population, aiming to balance exploration and exploitation more effectively [2].

The motivation behind adaptive mutation is rooted in the need for a dynamic approach that responds to the evolutionary state of the population. When diversity is low, the population risks stagnating and converging prematurely. Conversely, high diversity ensures that the population explores a broader search space, but excessive mutation can disrupt well-adapted individuals. Adaptive mutation seeks to maintain an optimal level of diversity, enhancing the algorithm's ability to find global optima.

In the case of the Max-Cut problem, an individual has a binary genotype of length equal to the number of vertices in the graph. Because of this discrete binary genotype representation, diversity can be measured using the average Hamming distance between all pairs of individuals in the population [11]. This measure captures the actual differences between individuals, providing a more accurate reflection of population diversity. The mutation rate is adjusted based on the calculated diversity. If diversity falls below a predefined threshold, the mutation rate is increased to introduce more variation. Conversely, if diversity is above the threshold, the mutation rate is decreased to stabilize the search process. A diversity threshold of 10% was chosen in order to ensure that the population retains enough variation to explore the search space effectively without being overly disrupted by mutation. This balance helps in sustaining an appropriate exploration-exploitation trade-off. The mutation operation modifies each bit of the genotype with a probability equal to the current mutation rate. This ensures that the mutation's impact is proportional to the current diversity of the population.

3.4 Local Search

Even though the weighted MaxCut problem is known to be NP-hard, there exists a simple local-search procedure that guarantees to produce solutions that are guaranteed to be at least half the size of the optimum [7]. The procedure continuously runs through the vertices and considers changing the assignment of each vertex. If this leads to an improvement, then the change is made, and the procedure repeats. If no improvement is possible then the procedure stops.

This local search approach can be applied to the offspring to increase their fitness and find the local minima. It does not put any additional restrictions on the rest of the evolutionary algorithm and can be combined with any crossover type. The danger of applying local search is the possibility that the population might quickly lose its diversity and stop exploring the problem space. To mitigate this problem, adaptive mutation can be employed.

This local search procedure has a high time complexity of $O(n^2)$ per individual, where n is the number of vertices. To account for this overhead, each computation of a vertex change computationally corresponds to a partial fitness function evaluation of that vertex. This way, the performance of algorithms with and without local search can be fairly compared to each other with respect to the number of fitness function evaluations.

4 EXPERIMENTS

To obtain the results, we evaluate 5 crossover methods: two-point, uniform, greedy, greedy with mutation, and Qinghua. Since we observed single-point crossover to perform almost identically to two-point, we decided to skip it for the final evaluation. The algorithms are evaluated with and without custom local search or adaptive mutation, and the number of evaluations needed to reach the optimum is measured (limited to 100k evaluations). Additionally, to ensure fair evaluation, we run with population sizes of 10, 50, 100, and 1000 and report the best-performing configuration. Qinghua and was not run with a population size of 1000 due to the increased runtime. All experiments are repeated 10 times and averaged. Standard deviations are reported alongside the mean. We perform Welch's t-test [9] to measure which algorithms obtain a mean number of evaluations that is not statistically different from that of the best algorithm. Additionally, the t-test is performed only for algorithms whose standard deviation is smaller than half of the mean ($\sigma < \mu/2$). We found this to be beneficial for the analysis as it allows us to focus on algorithms with an interpretable standard deviation.

We design the following procedure to select problem instances for evaluation. Given a set, we compute the distribution of the number of vertices of the contained problems. We then split the problems into small, medium, and large. Small instances are those with the least amount of vertices. Medium-sized problems are those with a number of vertices equal to the median of the found distribution. Finally, large problems are taken from the end of the distribution. We found small instances to lead to inconclusive or random results while running large instances was not feasible due to time constraints. As such, the models are evaluated on 3 medium-sized instances from each set. The random seed is fixed to ensure reproducibility.

4.1 Crossover comparison

The results of this comparison can be seen in Table 1. As we can observe, the Qinghua algorithm performs the best in 8 out of 15 instances. Notably, we observe that this algorithm greatly outperforms all others on set B, which suggests its suitability for 2D square-grid problems. Furthermore, the default Greedy method is the top-performing algorithm on 6 instances. Importantly, we can observe that grey-box methods outperform black-box methods on all instances. The consistency of grey-box methods is also better than black-box. As can be seen in the table, the standard deviations are typically larger for the black-box methods. Taking into account the statistical significance test shows that Qinghua and Greedy algorithms are the single best algorithms in all but two cases.

4.2 Local search impact

The performance of the algorithms with local search can be seen in Table 2. As we can observe, the number of evaluations needed to reach the optimum decreases for all models. Although, on average, Qinghua remains the best algorithm, it is so in only 5 out of the 15 instances. The algorithm retains its top performance on set B. Interestingly, we find that GreedyMut now outperforms Greedy, a result we discuss later. Finally, we can see that local search causes the black-box algorithms to outperform grey-box algorithms on certain problems.

The significance test further shows the similarity in performance between the algorithms. Taking the tests into account, we find that GreedyMut and Uniform are the best in 9 cases, while Qinghua

Table 1: The results for the crossover algorithms with local search turned off. The results are repeated 10 times, averaged and with standard deviation reported. Only the best-performing population size is reported. Bolded are the best-scoring algorithms. † indicates that the average is statistically the same as that of the best algorithm and that $\sigma < \mu/2$.

Set	Problem	Greedy	GreedyMut	Qinghua	TwoPoint	Uniform
A	n0000025i02.txt	**6753.1 ± 1598.7**	8861.2 ± 2303.6	10583.2 ± 31418.6	26543.2 ± 38807.2	10643.8 ± 2460.7
	n0000025i05.txt	**7534.6 ± 1210.6**	9059.4 ± 2277.7†	21223.1 ± 41526.3	46499.3 ± 46108.4	27476.6 ± 38285.1
	n0000025i00.txt	8137.2 ± 1550.9	8997.1 ± 2327.2	**809.1 ± 329.1**	29660.6 ± 37161.0	18838.5 ± 28569.2
B	n0000100i07.txt	49957.4 ± 43493.5	37100.7 ± 23026.8	**5570.0 ± 766.9**	92640.8 ± 23271.8	100000.0 ± 0.0
	n0000100i09.txt	48800.6 ± 44357.5	28741.4 ± 5544.6	**5448.0 ± 940.4**	93377.5 ± 20942.2	100000.0 ± 0.0
	n0000100i01.txt	32708.9 ± 35752.8	39123.9 ± 23739.3	**5646.2 ± 1233.3**	100000.0 ± 0.0	100000.0 ± 0.0
C	n0000050i03.txt	**12139.8 ± 1370.1**	20204.2 ± 2879.5	60739.6 ± 50687.9	92314.3 ± 24304.3	91658.3 ± 26378.8
	n0000050i05.txt	**57053.1 ± 45878.1**	75820.1 ± 40065.5	71193.2 ± 46388.6	100000.0 ± 0.0	84617.3 ± 32432.9†
	n0000050i08.txt	20028.8 ± 28280.8	19763.9 ± 3098.8	**1384.6 ± 256.6**	100000.0 ± 0.0	83986.2 ± 33760.4
D	n0000040i07.txt	10078.9 ± 1707.8	28381.3 ± 22633.8	**3006.3 ± 903.5**	12143.1 ± 1690.2	68606.5 ± 40547.1
	n0000040i05.txt	**9291.6 ± 1685.7**	24363.0 ± 26889.3	40746.0 ± 50999.6	20064.2 ± 28143.4	91744.0 ± 26107.8
	n0000040i08.txt	8054.7 ± 1152.4	13394.3 ± 3669.9	**3891.6 ± 1269.7**	11675.8 ± 1383.0	84130.4 ± 33494.7
E	n0000040i03.txt	**38394.7 ± 43134.6**	42533.2 ± 40242.7	71165.9 ± 46431.4	83169.5 ± 35566.1	100000.0 ± 0.0
	n0000040i02.txt	7653.0 ± 732.3	12531.1 ± 2363.9	**1116.3 ± 364.5**	83251.6 ± 35312.2	49380.6 ± 43639.6
	n0000040i09.txt	73663.3 ± 43528.0	**38152.4 ± 33886.0**	51600.7 ± 51022.5	100000.0 ± 0.0	83304.1 ± 35221.1

Table 2: The results for the crossover algorithms with local search turned on. The results are repeated 10 times, averaged and with standard deviation reported. Only the best-performing population size is reported. Bolded are the best-scoring algorithms. † indicates that the average is statistically the same as that of the best algorithm and that $\sigma < \mu/2$.

Set	Problem	Greedy	GreedyMut	Qinghua	TwoPoint	Uniform
A	n0000025i02.txt	452.1 ± 18.3	300.0 ± 431.0	177.8 ± 51.9	412.7 ± 8.7	**109.6 ± 38.3**
	n0000025i05.txt	914.9 ± 25.9	**286.5 ± 315.8**	381.1 ± 142.4†	468.1 ± 105.0†	465.7 ± 102.7†
	n0000025i00.txt	468.4 ± 21.0	**102.6 ± 18.2**	128.4 ± 50.1†	112.8 ± 61.8	121.9 ± 52.6†
B	n0000100i07.txt	14299.6 ± 3117.3	1997.8 ± 904.0†	**1344.8 ± 581.7**	1364.8 ± 312.5†	2115.9 ± 551.6
	n0000100i09.txt	18631.6 ± 6083.3	**1745.9 ± 381.2**	2214.0 ± 697.2†	12762.8 ± 30670.4	2237.1 ± 453.1
	n0000100i01.txt	17241.8 ± 4301.9	1774.3 ± 224.5†	**1513.0 ± 614.2**	1703.6 ± 523.6†	3216.2 ± 695.2
C	n0000050i03.txt	11312.8 ± 317.6	2447.6 ± 595.4†	3188.4 ± 4273.7	**1913.4 ± 1150.6**	2428.5 ± 1080.7†
	n0000050i05.txt	56067.4 ± 47727.1	12723.0 ± 30826.8	41273.2 ± 50556.8	12769.8 ± 4822.7	**2657.7 ± 1750.9**
	n0000050i08.txt	10981.5 ± 104.5	1106.6 ± 368.1†	1052.7 ± 692.4	**769.2 ± 339.6**	945.0 ± 374.2†
D	n0000040i07.txt	11180.9 ± 2876.9	2358.8 ± 693.3	**837.7 ± 459.3**	968.8 ± 255.7†	2732.5 ± 586.2
	n0000040i05.txt	8217.3 ± 278.4	1636.6 ± 749.6†	1716.1 ± 977.6	**1547.7 ± 515.1**	1950.0 ± 916.6†
	n0000040i08.txt	8277.2 ± 196.9	**386.5 ± 255.0**	758.1 ± 220.6	743.9 ± 210.0	1699.3 ± 442.5
E	n0000040i03.txt	11182.2 ± 3932.8	2233.3 ± 1452.4	**1921.7 ± 1033.2**	10225.7 ± 3711.8	12065.9 ± 30954.2
	n0000040i02.txt	8491.8 ± 145.7	529.3 ± 695.7	**528.2 ± 263.8**	791.8 ± 291.5†	747.8 ± 369.6†
	n0000040i09.txt	37819.9 ± 44537.4	2402.6 ± 1698.8	2579.1 ± 935.8	11956.0 ± 30974.9	**1420.8 ± 577.4**

and TwoPoint are the best in 8 instances. This indicates that local search is a technique strong enough to almost nullify the differences between the crossover methods. As we can observe, even with significance tests considered, Greedy is not the best algorithm in any case.

4.3 Adaptive Mutation

To measure the impact of adaptive mutation we compare the performances of Greedy, TwoPoint, and Uniform crossovers with and without adaptive mutation. We choose not to evaluate GreedyMut as it already contains its own mutation operator. Similarly, Qinghua is not evaluated as its mechanism already maintains diversity using the Hamming distance. Due to limited time constraints and a substantially increased runtime, we do not evaluate a population size of 1000 and only run the models on one instance per set. The algorithms are evaluated with local search. The remaining experimental details are identical to the previous evaluation.

4.3.1 Greedy Crossover experiment. To assess the impact adaptive mutation makes, we conducted an experiment comparing the performance of *Greedy Crossover* against *Greedy Crossover combined with Adaptive Mutation* on all sets. The goal was to determine if adaptive mutation could enhance the performance of the algorithm by maintaining diversity and preventing premature convergence.

Table 3: The results with local search and with and without adaptive mutation. The results are repeated 10 times, averaged and with standard deviation reported. Only the best-performing population size is reported. Bolded are the best-scoring algorithms. † indicates that the average is statistically the same as that of the best algorithm and that $\sigma < \mu/2$. The exact results might differ from Table 2 due to random chance.

Set	Problem	Greedy	GreedyAdapt	TwoPoint	TwoPointAdapt	Uniform	UniformAdapt
A	n0000025i02.txt	485.1 ± 83.6	442.4 ± 13.5	142.1 ± 57.6†	**100.6 ± 34.7**	110.3 ± 35.1†	416.6 ± 18.7
B	n0000100i07.txt	14807.4 ± 3880.0	15648.2 ± 4794.8	2673.2 ± 540.9	2247.7 ± 563.8	1648.2 ± 439.3†	**1620.0 ± 471.6**
C	n0000050i08.txt	8542.1 ± 17585.7	10982.8 ± 147.3	1013.7 ± 431.1†	959.7 ± 274.5†	1247.0 ± 557.3†	**861.0 ± 365.0**
D	n0000040i07.txt	9500.4 ± 2295.2	9294.8 ± 1918.0	924.6 ± 280.6†	**884.6 ± 202.9**	2565.5 ± 1006.8	2246.0 ± 1117.4
E	n0000040i06.txt	69551.3 ± 35809.4	80346.0 ± 41804.4	4951.4 ± 5350.6	11670.9 ± 31045.3	22837.1 ± 40822.7	**3073.3 ± 848.6**

The results, shown in Table 3, indicate that adaptive mutation does not provide a significant improvement over greedy crossover alone. In fact, adaptive mutation often causes undesirably large delays in convergence. Since Greedy Crossover is inherently exploitative, focusing on combining the best features of parent solutions to produce high-quality offspring quickly, it is well-suited for problems where rapid convergence to a good solution is beneficial. Adaptive Mutation introduces additional exploration by maintaining diversity. While this can be advantageous in avoiding local optima, it can also prevent the algorithm from quickly converging to an optimal or near-optimal solution in smaller or less complex problem instances, where the search space is relatively limited, and high-quality solutions can be found quickly through exploitation alone.

Conversely, in cases where the population already exhibits sufficient diversity or where the problem landscape does not necessitate high diversity for effective search, this mechanism can be redundant or even detrimental, leading to unnecessary delays in convergence. When running the experiment on a population size of 1000 individuals, the adaptive mutation was heavily detrimental, delaying the convergence of a single run by several hours.

4.3.2 Baseline experiments. In addition to the experiment with greedy crossover, we conducted similar comparative studies on baseline sGAs with and without adaptive mutation across different crossover operators: *two-point crossover*, and *uniform crossover*. Despite the introduction of adaptive mutation, the differences, as can be seen in Table 3, are statistically insignificant. Furthermore, convergence was observed to be delayed when adaptive mutation was employed alongside each crossover operator. This phenomenon suggests that the adaptive mutation mechanism may not effectively complement these specific crossover operators.

Two-point crossover selects two crossover points and swaps the genetic material between them to generate offspring. Adaptive mutation introduces perturbations to these offspring, aiming to enhance diversity and facilitate exploration. Similar to one-point crossover, the local nature of this operator may hinder the efficacy of adaptive mutation. Uniform crossover randomly selects genetic material from both parents to create offspring. While this operator promotes diversity by incorporating genetic material from both parents uniformly, the lack of a clear boundary between parent chromosomes may reduce the impact of adaptive mutation, leading to marginal improvement.

Furthermore, the delay observed in the experiments can be attributed to the overhead introduced by calculating the population diversity. This calculation involves determining the Hamming distance between every pair of individuals in the population to measure the overall diversity. Since this process is quadratic in complexity ($O(N^2)$, where N is the population size) and is calculated from scratch every time an offspring is made, it adds significant computational overhead.

5 DISCUSSION

5.1 Black-box versus gray-box

In the results, it has been observed that implementations that utilize partial evaluations almost always outperform the baseline variation operators. This confirms our hypothesis that utilizing partial evaluations leads to faster convergence toward optimal or high-quality solutions.

GreedyMut without local search underperforms compared to other grey-box operators, which implies that naive random mutation may only slow down convergence instead of helping to escape local minima. Interestingly, the opposite phenomenon is observed when introducing local search, as is discussed in the next section.

Qinghua is orders of magnitude faster than other implementations when graphs are formed as 2D square grids (set B). This suggests that topologies have an effect on the performance of algorithms, hence information about the graph's topology can be used to select the most appropriate algorithm.

Overall, if the topology of the graph is not known in advance, then the crossover type that performs generally well is Qinghua augmented with Local Search. But in a grey-box optimisation setting where the structure of the graph is known and the nodes that are highly connected to each other appear close in the gene encoding like in the sets C and D, then TwoPoint crossover augmented by local search shows marginally better performance. We propose that the reason for this is that the local search is best at optimizing smaller highly connected clusters of nodes and TwoPoint crossover is more likely to preserve these locally optimized groups. Thus this allows for efficient mixing of locally optimal clusters and leads to better performance.

5.2 Local search versus no local search

Local search with any variation operator outperforms both grey-box and black-box variation operators that do not use it. Additionally,

black-box variation operators perform as well or better than grey-box ones, meaning that the performance added by local search undermines any improvements observed earlier.

Surprisingly, GreedyMut performed significantly better than its counterpart. This can be caused by the fact that Greedy crossover has very little randomness, and combining it with local search caused a loss of diversity.

5.3 Limitations

Firstly, the experiments were conducted on a limited set of problem instances, which may not fully represent the diversity of possible graph structures. This could affect the generalizability of our findings to other types of graphs. Secondly, the computational cost and time constraints limited the evaluation of certain algorithms, particularly those with higher time complexities, such as Qinghua. Additionally, our implementation of adaptive mutation did not yield significant improvements and introduced overhead, suggesting that further tuning and optimization are needed. Lastly, while we ensured a fair comparison by using the same evaluation metrics, the differences in implementation details and computational resources could introduce biases.

5.4 Future improvements

5.4.1 Symmetry breaking. Since each colour value itself has no contribution to the fitness function, solutions such as "111000" and "000111" are equivalent since the same edges will be considered. Because of this, the sGA algorithm is simultaneously looking for 2 optimal values that have the opposite genotypes. This may cause the sGA to fluctuate between these two approaches and increase the time to find one of the optimal solutions. Symmetry breaking could be introduced in the algorithm, to ensure that only one of the optimal genotypes is being pursued.

5.4.2 Adaptive mutation. A potential improvement for adaptive mutation involves optimizing the diversity calculation by maintaining a cache of Hamming distances for each pair of individuals in the population. Instead of recalculating the diversity from scratch every time an offspring is produced, which is $O(N^2)$ in complexity, we can update the cache incrementally. By only updating the diversity values for pairs affected by the new offspring, the computational overhead becomes linear, $O(N)$. This approach significantly reduces the time complexity, making adaptive mutation more efficient and potentially mitigating the delay in convergence.

5.4.3 Fitness Sharing. Future improvements to our genetic algorithm could include implementing fitness sharing to enhance population diversity and maintain a broader search space. Fitness sharing modifies the fitness values based on the proximity of individuals in the genotype space, effectively penalizing solutions that are too similar to others. This encourages the population to spread out and explore various regions of the solution space, potentially avoiding premature convergence to suboptimal solutions. By promoting diversity, fitness sharing can lead to a more robust search process, improving the algorithm's ability to find high-quality solutions to complex instances.

5.4.4 Simulated Annealing. This optimization technique mimics the cooling process of metals to escape local optima by allowing occasional acceptance of worse solutions [12]. By gradually decreasing the acceptance probability of worse solutions over time, the algorithm balances exploration and exploitation effectively. Integrating simulated annealing with the current approach can help in exploring the solution space more thoroughly, avoiding premature convergence. This hybrid method can potentially yield better results by combining the strengths of evolutionary algorithms and simulated annealing.

6 CONCLUSION

This work aimed to evaluate and compare different black-box and grey-box evolutionary algorithms for solving the Max-Cut problem. We focused on answering three research questions related to the benefits of domain knowledge in the GBO setting, the improvement of sGA performance with problem-specific knowledge, and the performance comparison between sGA for GBO and BBO. Our main results showed that grey-box methods consistently outperformed black-box methods, particularly in scenarios where partial evaluations could exploit domain knowledge. The Qinghua algorithm demonstrated exceptional performance on 2D square-grid problems, emphasizing the impact of graph topology on algorithm efficiency. Additionally, incorporating local search significantly improved the performance of all algorithms, with black-box methods occasionally surpassing grey-box methods when local search was applied.

For future research, several areas hold promise. Introducing symmetry breaking could enhance algorithm efficiency by focusing the search on a single optimal genotype. Optimizing the adaptive mutation mechanism by caching Hamming distances can reduce computational overhead, potentially improving convergence times. Implementing fitness sharing and integrating simulated annealing are also potential avenues for improving population diversity and avoiding premature convergence. These enhancements could further advance the performance and applicability of evolutionary algorithms for the Max-Cut problem and similar combinatorial optimization challenges.

REFERENCES

[1] Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. 1988. An Application of Combinatorial Optimization to Statistical Physics and Circuit Layout Design. *Operations Research* 36, 3 (June 1988), 493–513. https://doi.org/10.1287/opre.36.3.493

[2] Stephan Blum, Romanas Puisa, Jörg Riedel, and Marc Wintermantel. 2001. Adaptive mutation strategies for evolutionary algorithms. In *The Annual Conference: EVEN at Weimarer Optimierungsund Stochastiktage*, Vol. 2.

[3] Anton Bouter and Peter A. N. Bosman. 2022. GPU-accelerated parallel gene-pool optimal mixing in a gray-box optimization setting. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, Boston Massachusetts, 675–683. https://doi.org/10.1145/3512290.3528797

[4] F. Della Croce, M.J. Kaminski, and V.Th. Paschos. 2007. An exact algorithm for MAX-CUT in sparse graphs. *Operations Research Letters* 35, 3 (May 2007), 403–408. https://doi.org/10.1016/j.orl.2006.04.001

[5] P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. 2002. Randomized heuristics for the Max-Cut problem. *Optimization Methods and Software* 17, 6 (Jan. 2002), 1033–1058. https://doi.org/10.1080/1055678021000090033

[6] Michel X. Goemans and David P. Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM* 42 (1995), 1115–1145. https://api.semanticscholar.org/CorpusID:15794408

[7] Michel X. Goemans and David P. Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM* 42, 6 (nov 1995), 1115–1145. https://doi.org/10.1145/227683.227684

[8] Gary A. Kochenberger, Jin-Kao Hao, Zhipeng Lü, Haibo Wang, and Fred Glover. 2013. Solving large scale Max Cut problems via tabu search. *Journal of Heuristics* 19, 4 (Aug. 2013), 565–571. https://doi.org/10.1007/s10732-011-9189-8

[9] B. L. Welch. 1947. The Generalization of 'Student's' Problem when Several Different Population Variances are Involved. *Biometrika* 34, 1/2 (Jan. 1947), 28. https://doi.org/10.2307/2332510

[10] Ryan Williams. 2005. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science* 348, 2–3 (Dec. 2005), 357–365. https://doi.org/10.1016/j.tcs.2005.09.023

[11] Qinghua Wu, Yang Wang, and Zhipeng Lü. 2015. A tabu search based hybrid evolutionary algorithm for the max-cut problem. *Applied Soft Computing* 34 (Sept. 2015), 827–837. https://doi.org/10.1016/j.asoc.2015.04.033

[12] Habib Youssef, Sadiq M Sait, and Hakim Adiche. 2001. Evolutionary algorithms, simulated annealing and tabu search: a comparative study. *Engineering Applications of Artificial Intelligence* 14, 2 (2001), 167–181.