Dynamic Mutation Rate Control for the Genetic Algorithm for Global Geometry Optimization

Jacek Kulik[1]

Supervisors: Peter A.N. Bosman[1,2]**, Anton Bouter** [2]**, Vanessa Volz** [2]

[1]**EEMCS, Delft University of Technology, The Netherlands**
[2]**Centrum Wiskunde Informatica, Amsterdam, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Jacek Kulik
Final project course: CSE3000 Research Project
Thesis committee: Peter A.N. Bosman, Anton Bouter, Vanessa Volz, Thomas Abeel

Abstract

Global Geometry (or Cluster) optimization is the process of finding the most stable formations of a cluster of some atoms. A genetic algorithm was developed to find the global minimum of a cluster using the Lennard-Jones atom interaction model efficiently. Determining the optimal hyper-parameters for the algorithm is a computationally intractable task without proper search strategies. Furthermore, during different stages of the algorithms it may be beneficial to change the importance of exploration vs exploitation of the search space. To solve these problems, nine strategies of choosing the algorithm's mutation rate are presented and benchmarked on the time to find the global optimum. Experimental results show that such strategies can reduce the mean time required to find a global minimum for a given cluster when compared to a constant low mutation rate. However, the results are not statistically significant for all clusters tested. The variance in the results might be caused by the understudied factor of the local optimization the algorithm utilizes. A majority of the algorithm's runtime is taken up by it, leading to the GA components showing less impact on the results.

1 Introduction

Material science is a field that has grown in popularity in recent decades thanks to significant advances in technology. Discovering new materials has always been important for creating new advanced technologies. Nowadays, the qualities demanded from new materials are becoming more difficult to attain. To this end, a field of optimization has developed which aims in helping researchers find potential candidates for new material technology. Global Geometry Optimization (GGO), also called Structure or Cluster Optimization, is the process of finding structures of those atoms with the lowest energy possible. Such configurations are called (global) minima. An example optimization of a cluster of 13 carbon atoms can be seen in Figure 1. It shows a cluster of randomly positioned atoms, and their globally optimal structure. The search space, called the Potential Energy Surface (PES), is complex, with many (possibly deep) local minima, hindering the process of finding a globally optimal solution of a cluster[1]. Atoms exhibit several interactions in between themselves such as electrical repulsive forces and strong bonding forces. A useful relaxation of the problem is the use of Lennard-Jones (LJ) potential [2]. It consists of treating the atoms as infinitely small points, with the energy between them being a simple function of their distance. The most widely used potential is Lennard-Jones 12-6 and its formulation is as follows [3]:

$$V_{LJ} = 4\epsilon([\frac{\sigma}{r}]^{12} - [\frac{\sigma}{r}]^{6}) \quad (1)$$

where ϵ is the depth of the attractive well, r the distance between the atoms, and σ the distance at which the potential changes signs. While some LJ global optima have been shown to not be the real-world global minima [4], most of them hold up to scrutiny and are used by researchers in the real world to experiment on materials. GGO has many uses for researchers and manufacturers in the material science field. The process of finding new materials with desired properties is difficult but has many uses. Some examples of materials discovered using GGO include: photonic coating [5] and space radiation shields [6]. Even when there is uncertainty as to if the found solution is a global minimum, producing several possible candidates is crucial for researchers conducting physical experiments. Using these estimations instead of guessing materials to test, they now have a curated list of experiments to perform that are more likely to produce suitable materials.

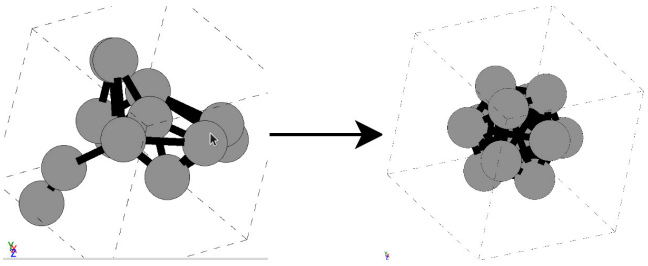

Figure 1: Optimization of a cluster of 13 carbon atoms. On the left is a random arrangement of the atoms, and on the right is the optimized version of the cluster. Such organization leads to the cluster having the lowest energy between the atoms. This cluster shows an example of icosahedral symmetry.

Much research has been done in the field of GGO. It has been shown to be an NP-hard problem [7]. Due to this, it is impossible to make a polynomial time algorithm that could find optimal solutions quickly, so traditional methods of solving NP-hard problems have to be utilised, like the genetic algorithm in this case. One of the most important early contributions is the Cambridge Energy Landscape Database [2]. It contains the clusters currently believed to be the global minima for LJ clusters of size 3 through 150. It is important to note that these solutions are not guaranteed to be the true global minima, as there is no method of proving this attribute. They are however very likely to be the global minima, as no better solutions have been found so far.

The first algorithms for GO were brute-force algorithms. They searched through the space of possible positions of atoms exhaustively to find optimal solutions. This approach did not scale well to more atoms, as the number of atom interactions grows quadratically with their number, so the solutions become increasingly more complex. Early sophisticated solutions involved Minima Hopping [8], consisting of employing short duration temperature-based molecular dynamics to shift the cluster into a new configuration, followed by local optimization. Another approach was Basin Hopping [2], which involves randomly perturbing the atoms in some way and applying local optimisation on them to find the nearest local minima. Soon thereafter, Genetic Algorithms (GA) started to be used for GO [9]. They involve the use of processes akin to natural selection to balance the exploration and exploitation of candidate solutions. A more thorough expla-

nation of genetic algorithms is presented in section 3.

The tuning of parameters for a GA is a task for which no perfect solution has been found. Different formulas and general practices have been created by various researchers. For example considering the mutation rate, [10] suggests a rate between 0.001 to 0.01 depending on the problem, while [11] suggests using a mutation rate as high as 0.05 to 0.2 (with a low population size). These rates have a large variance, however they are intended for different problems. This highlights the problem of finding good parameters for a specific problem or a specific problem instance. Most algorithms are currently tuned by hand, where a researcher tries several values for each parameter in isolation and selects the ones giving the best results. In addition to it being hard to select the correct parameters for a general algorithm, the optimal parameters vary between different problems. Researching the selection of these parameters dynamically is the field of dynamic parameter control. It focuses on assigning parameters during the runtime of the algorithm, such as mutation rate (p_m), crossover rate (p_c), and population size (N) [12][13]. Aside from the need of assigning hyper-parameter values for a problem, there is also a need to vary the values of the parameters during execution [14]. At the start of the runtime of the GA it is likely favourable to increase the amount of exploration done, and during the latter stages of the algorithm, exploitation of the search space is more important. This heuristic is the assumption underpinning of simulated annealing search, which has shown great results in certain fields [15]. This suggests that such an approach can be beneficial for many problems.

There has been little work done in dynamic parameter control literature of applying self-adaptability to solve specific problems. Most papers focus on the theoretical basis of self-adaptability and showcase its performance on sample problems that are meant to highlight the aspects of the algorithm [13]. General guidelines about how to apply parameter control in practice are still missing. Thus trying to apply this knowledge in real-world contexts, and trying to specifically tune the self-adaptability strategy to a specific problem is an understudied field. There has been little work done on self-adaptability in the GGO field. This paper aims to fill that void by examining the efficacy of methods proposed in the literature to the GGO problem. It describes the common approaches used, benchmarks their results in comparison to a baseline GA algorithm and discusses the results and future recommendations.

2 Adaptation Strategies

A. Eiben, et al. [16] proposes a categorization of adaptation strategies by the following two criteria:

1. What is changed?
2. How is the change made?

Under this classification, this paper chooses to investigate the second question. Both questions are equally interesting, however trying to answer both at once would incorporate a lot of complexity and possible combinations. This would make an analysis needlessly long and difficult to interpret. A decision was made to only consider the adaptation of the mutation rate. In the context of GO, mutation is important because it allows the algorithm to escape local minima. The energy surface of atomic clusters is filled with many local minima, some of which may be hard to get out of[17]. Dynamically managing the mutation rate is a topic studied in literature already [13][11], thus making any results comparable to previous findings, and allowing for the use of verified methods to apply to the GO problem.

Again by [16], the how aspect can be further clarified into the following methods:

- Deterministic Parameter Control. This involves creating some heuristic or algorithm to set the parameter values **without** any input from the algorithm. Examples of attributes this can take as input are: execution time, generation number, random number. These methods can be simpler to develop, as they can follow a natural human understanding of the problem. For example, it could make sense to have a high mutation rate at the start of the search in order to explore more, and then decrease the mutation rate over time in order to allow the algorithm to explore the space more.
- Adaptive Parameter Control. Such algorithms can take as input any data from the algorithm, and from external sources. Examples of inputs may include: average fitness value, convergence metric, generation number, best fitness value so far. This control strategy allows for a lot more complex control of the parameters, however it also makes developing such a strategy much more difficult. The designer has to decide which inputs to take, how to combine them, and what to output. It is plausible to not foresee something and make some mistake that could cripple the performance of the GA.
- Self-adaptive Parameter Control. Self-adaptation is the process of utilising biological evolutionary processes to iteratively improve some aspect of an individual. It can be compared to utilising a GA to optimize the selected parameter. In the context of GAs, self-adaptation is achieved by appending parameter information to the end of an individual's genome [18]. This is a natural and simple approach to parameter control. Since the algorithm already executes evolutionary processes, why not implement those to the control of the parameters as well? The downside of this approach is that it extends the length of the genome, meaning that there are more possible individuals, increasing the generation count required to start seeing benefits from a better parameter value. Implementation of it is however simpler than the other methods, but it does require choosing the behaviour of the parameter value in mutation and crossover events.

A parameter can be applied at multiple possible levels, also called scopes. The possible scopes for mutation specifically are:

- Population - mutation rate applies to the entire population of individuals during a given generation.
- Individual - mutation rate applies to a single individual during a generation.

- Mutation - mutation rate applies only to a single type of mutation.
- Gene - mutation rate applies to a single or a set of genes. For example in the case of GO, this could mean different atoms being subject to different mutation rates.

The mutation rates considered for this research come from a variety of literature and knowledge-based sources. They were selected based on their prevalence, ease of implementation, or the potential efficiency of their results. The strategies used are as follows:

1. Constant 0 - a constant 0% mutation rate. This was included to test the algorithm's performance without any mutation
2. Constant 10 - a low constant mutation rate of 10%.
3. Constant 30 - a high constant mutation rate of 30%.
4. Time Deterministic. Extrapolates linearly between a mutation rate of 20% down to 1% over the generations [19].
$$p_m(t) = 0.1 + 0.19 \times \frac{1-t}{T} \tag{2}$$
where t is the current generation, and T is the maximum generation number.
5. Average Fitness Heuristic (AFH) from [20]. Takes as input the mean fitness of the current generation, and the fitness of the considered individual. This allows for the worse individuals to try to explore the space more to find better solutions, while not interfering significantly with high-fitness individuals.
If $f_i < \bar{f}$
$$p_m = k_4 \tag{3}$$
else
$$p_m = k_2 \times \frac{f_{max} - f_i}{f_{max} - \bar{f}} \tag{4}$$
where f_i is the fitness of the individual, $\bar{f}$ is the mean fitness, f_{max} is the best fitness found so far, and k_2 and k_4 are constants, set here respectively to 1 and 0.6.
6. Self-adaptive (SA), as suggested by [18]. The mutation rate for an individual is encoded in their genome. The range of possible rates is $p_m \in (0, 0.5)$. Crossover of the mutation rate consists of taking the average of the mutation rates of the parents. Mutation of the mutation rate happens with a probability equal to the mutation rate and is defined as follows [21]:
$$p_m = p_{m_{old}} + \sigma_{sa} \times X \sim \mathcal{N}(0, 1) \tag{5}$$
The value p_m is then clipped between the values $(0, 0.5)$. σ is the self-adaptation speed parameter and is set to 0.3.
7. Fuzzy Logic Controller (FLC) from [13]. Utilises a fuzzy logic representation of the convergence of the algorithm and uses human-understandable rules to produce a sensible mutation rate. Refer to the paper for a specification of its workings. It follows understandable rules, however those rules may not function correctly if the relationship between the mutation rate and the resultant fitness values is complex. In such cases, the rules made by human knowledge may fail to produce satisfactory results.
8. Mutation Success Condition (MSC) [21]. This strategy tries to set a mutation rate such that the success rate of the mutations will be equal to some constant. A mutation is defined as a success if after the mutation the fitness of the individual has increased. The mutation rate is updated every 3 generations and the update procedure is as follows:
$$p_m = p_{m_{old}} + \sigma_{msc}(x - s) \tag{6}$$
where σ is the aggressiveness factor equal to 0.5, x is the success rate target equal to 0.2, and s is the success rate of the mutations over the last 3 generations.
9. Distance To Estimation (DTE). This strategy utilises an estimation of the global minimum in order to set the mutation rate for a generation based on how close it got to it. This follows the intuition that the mutation rate should be high at the start when exploring the space and low at the end when trying to exploit it.
$$p_m = 0.05 + 0.25 \times (1 - e^{-|q_i - q_e|/10}) \tag{7}$$
where q_i is the energy of the individual, and q_e is the estimation of the global minimum. The estimations here were created by taking the best result for a cluster of one fewer atoms than the one currently being considered. This was done as it is generally an accurate estimation and would prove satisfactory in a scenario of trying to find the minima of progressively bigger clusters. In other configurations, other estimations would have to be developed.

A summary of the strategies can be found in Table 1. It presents the strategies, a brief description, a formula (if necessary), and which category they belong to. In this research, only adaptation strategies in the population and individual scope have been considered. The primary reason for this is their lack of representation in literature. There aren't many resources talking about mutation or gene scope adaptation. This is primarily because most genetic algorithms utilise a single mutation operator which always works on the whole genome. Additionally, adding in those strategies would introduce additional implementation and testing challenges.

These strategies present a range of different ways of managing the mutation rate, so they should produce various results, which should shed insight on how they perform in the real world. The strategies were implemented into the Baseline Genetic Algorithm, explained in the next section.

3 Baseline Genetic Algorithm

A Baseline Genetic Algorithm (BGA) was developed by a team of students for a different project, which was adapted for use in this research while keeping the same execution sequence. It was made with the goal in mind of creating a simple algorithm that could be later modified to identify possible improvements among its parts. It incorporates commonly used mutation and crossover operators in the GA research field. The BGA is based on a standard GA architecture, where

Name	Summary	Formula	Category	Scope
Constant 0	Constant zero mutation	$p_m = 0$	Constant	Population
Constant 10	Constant low mutation	$p_m = 0.1$	Constant	Population
Constant 30	Constant high mutation	$p_m = 0.3$	Constant	Population
Time Deterministic	Mutation determined by linear function of time	$p_m(t) = 0.1 + 0.19 \times \frac{1-t}{T}$	Deterministic	Population
Average Fitness Heuristic	Uses distance to mean and max fitness	$p_m = k_2 \times \frac{f_{max}-f_i}{f_{max}-\bar{f}}$	Adaptive	Individual
Self-adaptive	Encodes mutation in genome	$p_m = p_{m_{old}} + \sigma_{sa} \times X \sim \mathcal{N}(0,1)$	Self-adaptive	Individual
Fuzzy Logic Controller	Uses FLC using a convergence metric	-	Adaptive	Population
Mutation Success Condition	Keeps the mutation rate at 20%	$p_m = p_{m_{old}} + \sigma_{msc}(x - s)$	Adaptive	Population
Distance to Estimation	Distance from individual to estimation of optimum	$p_m = 0.05 + 0.25 \times (1 - e^{-\lvert q_i - q_e \rvert / 10})$	Adaptive	Individual

Table 1: Summary of mutation adaptation strategies

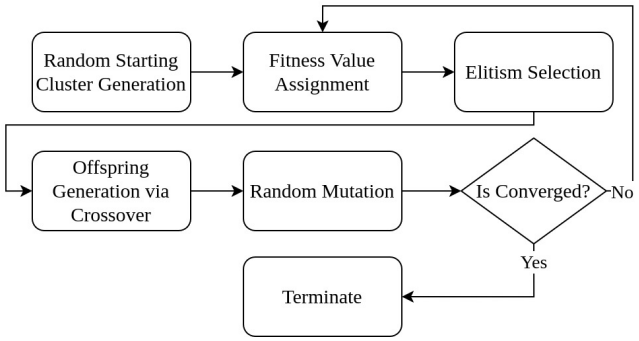

Figure 2: Baseline Genetic Algorithm Program Flow. The algorithm starts by creating a random population sample, then proceeds to the generation cycle. During each generation, individuals are scored according to a fitness function, then some of them are selected and new offspring are created using crossover and mutation. If at the end of the generation the convergence criterion has been met, the algorithm terminates.

during each generation individuals are selected, crossover is performed to produce children and some of those children get mutated. As the last step the individuals undergo local optimization. The algorithm was developed in Python 3.11 with the use of the Atomic Simulation Environment (ASE) [22] library. It allows for easily keeping track of atoms, performing energy calculations and local optimization. The general flow of the algorithm is presented in Figure 2

The detailed implementation of the BGA was chosen in accordance to standard practices found in the literature. The selection strategy used is elitism with a configurable number of surviving individuals, similar to that in [23]. The crossover operator is the cut and splice [9]. It involves randomly selecting two parents, aligning their atoms to the centre and using Principle Component Analysis (PCA) to align their principal axes. Next, a plane is generated through the clusters, and a child is created by combining the opposite sides of the cluster from each parent. This procedure is performed until the desired number of children is acquired. After any modification made to a cluster, such as crossover or mutation, the cluster is checked for validity. A configuration is deemed valid if it contains the correct number of atoms and no two atoms are closer than 0.15 angstrom (angstrom is the default distance unit in ASE, its symbol is Å and it is equal to $10^{-10}m$). This check is implemented because when atoms get too close together, the forces between them start to grow very rapidly, so no stable configuration is possible with atoms this close together. Discarding such configurations allows speeding up the computation time.

The child clusters may then undergo mutations, with a probability based on their mutation rate. A mutation rate of 0.1 means that there is a 10% chance that a cluster would undergo a specific mutation. Only one mutation can be applied to a certain cluster during a generation. The mutations present in the algorithm are as follows:

- The first implemented mutation is Random Displacement [24]. It involves moving some of the atoms in the cluster by a randomized distance. This algorithm's variation of it applies the mutation to every atom independently and (possibly) moves them in a random direction by a specified distance (default 0.1 angstrom). This mutation may be repeated until the cluster is in a valid state.
- The second mutation operator is Twinning [24], which consists of using a plane to divide the cluster and rotate one side of it. This is done here by calculating the normal of the plane, and randomly determining which side of it to consider. A random degree is drawn and the selected side is rotated thusly.
- The final implemented mutation is Etching [24]. The first variation of it introduces a new atom, performs local optimization and then removes the atom with the new highest potential energy. Another variation of it is the opposite; it removes the atom with the highest potential energy, performs local optimization and then introduces

Parameter	Symbol	Value
Base Population size	P	8
Local Optimizer (LO)	-	BFGS
Maximum LO steps	opt_{steps}	1000
Maximum number of iterations	max_iter	100
Iterations to convergence	$conv\_iter$	10
Parents selected per generation	$num_{selection}$	$P/2$
SA speed	σ_{sa}	0.3
AFH Default	k_4	0.6
AFH Aggresivenes	k_1	1
Length of FLC window size	flc_window	2
MSC window size	msc_window	3
MSC target	x	0.2
MSC aggressiveness	σ_{msc}	0.5

Table 2: Parameters of the Baseline Genetic Algorithm and of the mutation parameter control strategies

back a random atom.

The algorithm determines that it has converged if it has not seen any significant (defined as 10^{-6}eV) improvement in the best fitness found for a number of generations (default is $maximum\_iteration\_count/10$). The default local optimizer used in the algorithm is Broyden-Fletcher-Goldfarb-Shann (BFGS) [25], which utilizes a quasi-Newton approach to iteratively approximate a Hessian matrix to navigate the search space. The maximum number of local steps was by default 10,000, but was limited in this research to 1,000, as BFGS can sometimes get stuck in computation loops, leading to excessive computation times.

4 Experimental Setup and Results

The methodology of this paper is a comparative benchmarking approach. An experiment was designed, in which every presented mutation rate control strategy is used to solve the same problems. The clusters used for testing are the carbon clusters of 19, 31, and 47 atoms. Carbon clusters specifically were chosen, because they are well studied in literature, and they serve as a good representative of general clusters. The 19 atom cluster was chosen for its simplicity, to show the results of the algorithm for clusters where the global minima can be easily found. LJ31 clusters are known for their several deep local minima, so it showcases the algorithm's performance on a complex energy surface. The last cluster was chosen for the fact that it has a higher atom count, however its landscape is simple. This will showcase how well the algorithm can find a simple minimum in a large cluster. An algorithm is considered to have found the minimum if it is within $10^{-6} * num\_atoms^2$ eV of it. This threshold was created with the consideration of floating point numbers and estimates. Each atom has an interaction with every other atom, of which all are subject to floating point precision errors. Additionally, the relaxations and energy calculations present in ASE perform some estimations in order to speed up computation, which may have an effect on accuracy [22]. Lastly, when examining the forests provided by Doye et al. [17], it can be noted that for most clusters the difference between the global minimum and the deepest local minimum is on the order of approximately 0.5eV. Thus even if the algorithm doesn't find the exact global minimum, then if it is within the threshold set it is most likely in the funnel of the global optimum. The only step necessary to find the global minimum then would be extensive local optimization. However for the purposes of faster algorithm execution, the number of local steps was limited to 1000, which hinders its ability to get arbitrarily close to the global minimum. Each test was repeated on 20 different seeds to aid in providing statistically significant results. This number was chosen because it was the maximum that could be run in the allocated time on the supercomputer. All tests were run on the DelftBlue cluster [26] using the `compute-a` nodes. These nodes use the Intel XEON E5-6248R 3.0GHz processor with 3GB of memory per core. Experiments were run on 1 core each. All of the hyperparameters of the GA can be found in Table 2.

The execution of a test is done as follows. The algorithm is started with a selected adaptation strategy with the base population size (here 8). If it finds the global optimum early, then it terminates early. Otherwise it continues until the convergence criterion has been met. If after convergence the GA has not found the global optimum, then it runs again with a doubled population size. This continues until the population size limit (here 128). At this point, if the algorithm fails, its population is reset to 8 and its seed is incremented by 1000. This may continue until the upper time limit for an execution of 100 minutes, at which point the execution is reported as a failure. The purpose of this benchmark is to determine the algorithm's efficacy at finding the global optimum of a given cluster. Benchmarking this way allows to clearly demonstrate how much time, and what population size was needed to find a solution. Results from such tests can be used to effectively compare the adaptation strategies in the contexts of different cluster types.

The results are presented using boxplots for the time to find the optimum and stacked bar charts to show the population sizes needed to find the optimum. The green line in each box plot shows the mean time, the upper line presents the maximum time excluding outliers, and the lower line shows the minimum time. For the population sizes, the stacked bar charts can be read to see the number of times the algorithm found the global optimum at a specified population size. If the algorithm failed at 128 and had to restart, then regardless of the population size it ended with, it still gets reported as being 128.

For the LJ19 cluster times shown in Figure 3, it can be observed that the algorithms successful at it managed a median time of about 14 seconds. Cons. 1 and Time det. showed poorer results. In Figure 4, all algorithms aside from Cons. 0 finished almost all or all of their runs with a population size of 8. The sample size is not high enough to conclude whether the runs at 16 for some algorithms were anomalies or if they should be expected for all of them.

The LJ31 cluster is significantly bigger, and it also has a very complicated global minimum to find. The time results for it are presented in Figure 5. The median for algorithms which did well here is in the range of 230 to 300 seconds. Most algorithms show a big spread between the minimum

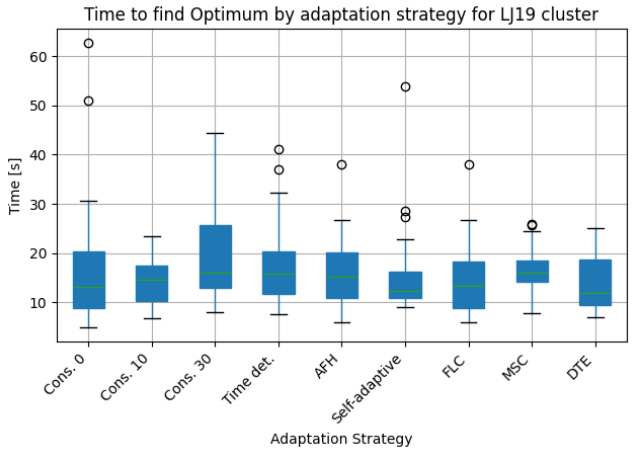

Figure 3: Boxplot graph of the time needed for the GA to find the global optimum for an LJ19 cluster varied by adaptation strategies

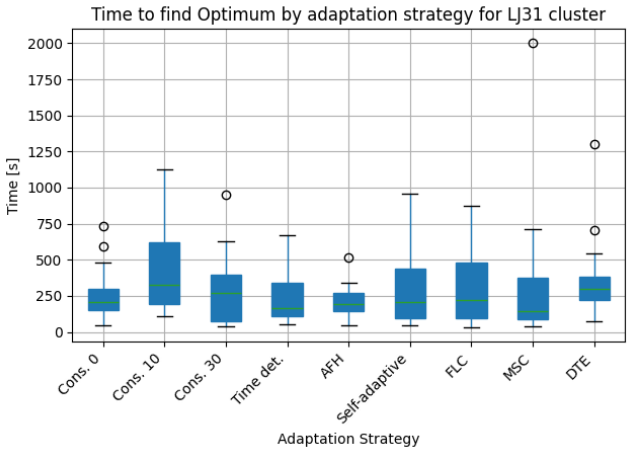

Figure 5: Boxplot graph of the time needed for the GA to find the global optimum for an LJ31 cluster varied by adaptation strategies

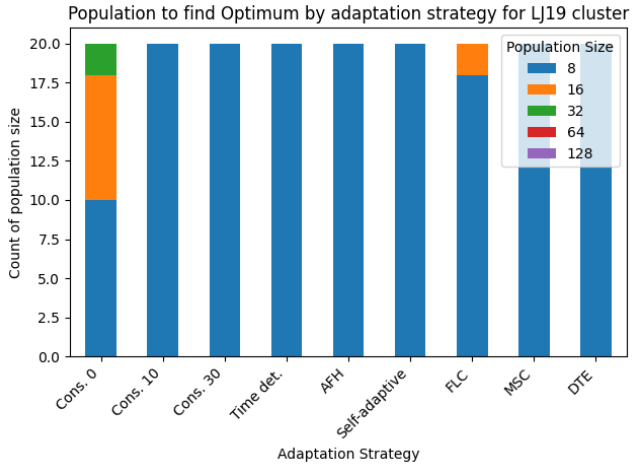

Figure 4: Stacked bar chart of the population used for the GA to find the global optimum for an LJ19 cluster varied by adaptation strategies. Each column represents an adaptation strategy, and each colour of the bar represents the cumulative count of how many times a certain population size was needed to find the global optimum.

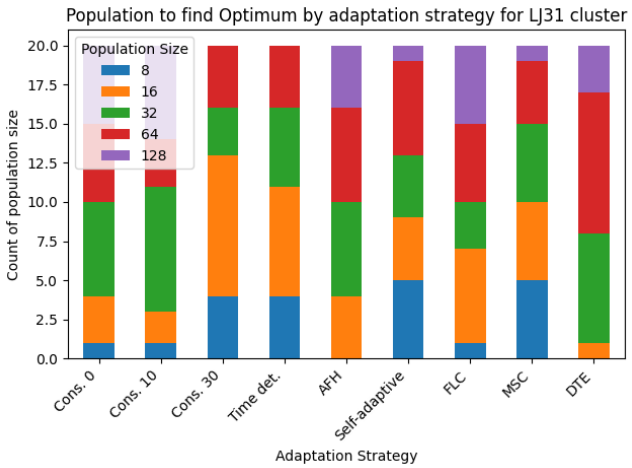

Figure 6: Stacked bar chart of the population used for the GA to find the global optimum for an LJ31 cluster varied by adaptation strategies. Each column represents an adaptation strategy, and each colour of the bar represents the cumulative count of how many times a certain population size was needed to find the global optimum.

and maximum, with there being some significant outliers as well. Population results shown in Figure 6 show a big change over LJ19. The spread between the different population sizes is a lot more even. Some algorithms had to especially use the maximum population size, especially AFH.

Lastly is the LJ47 cluster. It is a cluster with a simple global minimum. The spread in the times in Figure 7 is very large. There are some significant outliers for Cons. 2 and Time det. AFH shows the best results. As for the population chart in Figure 8, it can be observed that the population sizes needed were lower than those for LJ31. Most of the runs finished in the 8 and 16 population size.

The performance of each strategy for each cluster was compared against the baseline of Constant 10 and Constant 30. A one-tailed Mann–Whitney U test with $\alpha = 0.05$ was conducted to ascertain whether the difference in the results is statistically significant. The results of these tests are presented in Table 3. From the table it can be observed that the difference in performance for LJ19 and LJ47 is not high enough to be considered statistically significant. The performance for LJ31 has all algorithms except for DTE with p-values less than 0.05 for Constant 10, making it significantly better performing.

5 Responsible Research

Material science is a neutral field. The discoveries can be used in any area from providing higher quality resources in humanitarian aid, to developing new materials for the construction of warships. This research attempts to advance the

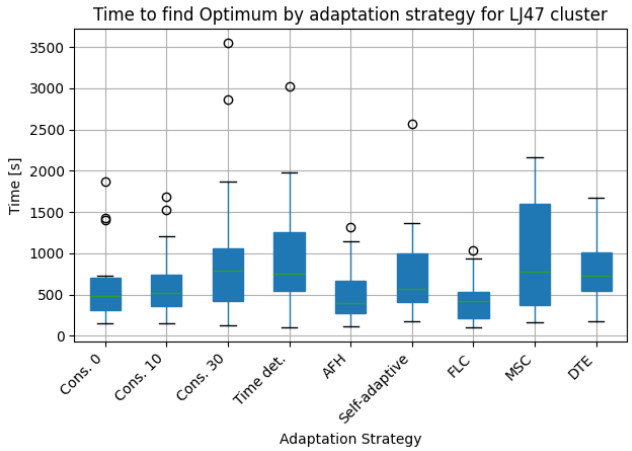

Figure 7: Boxplot graph of the time needed for the GA to find the global optimum for an LJ19 cluster varied by adaptation strategies

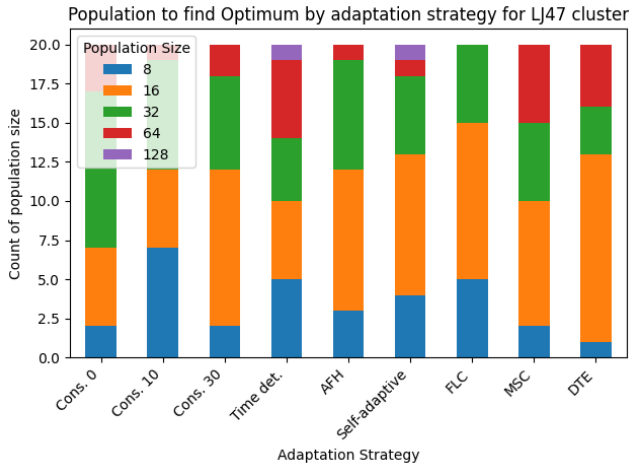

Figure 8: Stacked bar chart of the population used for the GA to find the global optimum for an LJ19 cluster varied by adaptation strategies. Each column represents an adaptation strategy, and each colour of the bar represents the cumulative count of how many times a certain population size was needed to find the global optimum.

field of global optimization, and any discoveries are provided, as most science is, to everyone who may find it. Thus it can equally benefit those seeking to use it for good or to cause harm. There is no reasonable solution for the researcher to undertake to diminish any possible negative outcomes of this research. Limiting the reach of the study to only actors wanting to use it for ethically positive purposes is not feasible, as it would require intensive information gathering about them, and then still the actor could misrepresent their intentions.

In order to aid in reproducibility, all the results were ran using the same seeds supplied to the `numpy.random.seed()` function. This sets the results of all random number generations to be deterministic from the provided seeds. Doing this ensures that the results can be replicated, by providing the program with those seeds. The seeds used in this research were integers in the range 0 to 19 (inclusive). The results were verified to be consistent; running the same seeds again did produce the same results. Any researchers willing to reproduce the results may do the same. The code used can be obtained in the author's GitHub repository[1]. The dataset provided by Wales et al. [1] is free to use for other researchers, who may also independently verify the results there.

	Cluster Size					
	LJ19		LJ31		LJ47	
	vs 10	vs 30	vs 10	vs 30	vs 10	vs 30
T. det.	0.825	0.237	0.005	0.495	0.955	0.559
AFH	0.729	0.162	0.005	0.271	0.280	0.014
SA	0.452	0.063	0.031	0.692	0.746	0.197
FLC	0.338	0.036	0.045	0.755	0.082	0.003
MSC	0.886	0.271	0.010	0.516	0.937	0.538
DTE	0.347	0.02	0.245	0.0896	0.940	0.347

Table 3: Mann-Whittney U test of performance of strategies vs the performance of the Constant 10 and 30 mutation rate. These values can be interpreted as regular p-values, with statistical significance usually being considered at a value lower than 0.05

6 Discussion

Overall, the adaptation algorithms seem to have performed more consistently than the constant mutation rates. Constant zero did not perform well in terms of population sizes numbers. This is likely due to it terminating early often, which also leads to it having a lower mean time result. Cons. 15 and Cons. 30 perform acceptably, though their performance varies significantly between the tested clusters. Time det. did not perform well. This is likely due to the choice of the maximum number of iterations. 100 was chosen because it is a value that works well for all of the tested clusters. LJ47 runs never stopped because they reached the iteration limit, only sometimes exceeding 90. This consequently meant that time det. was able to show good results in LJ47, as it utilised its full range. MSC also generally performed poorly. The way a mutation was judged to be a success or not was by looking at the cluster immediately before and immediately after a mutation. Whether this would improve the fitness of a cluster is a random and difficult process. The cluster had already undergone local optimization, so it is likely that after a mutation it will be in a lower fitness position. This issue could be resolved by comparing the cluster before the mutation, and after mutation and local optimization (may still be implemented). This was not done due to it being difficult in the framework of how the BGA was designed. The self-adaptive strategy increased the size of the genome and increased the number of operations necessary to perform during each crossover and mutation. This does increase the time it takes to perform a generation. However, due to most of the execution time of this genetic algorithm being taken up by local optimisation, this does not end up being an issue.

Data was gathered about the mutation rates supplied by the adaptation strategies and is presented in Figure 9 for LJ47,

[1] https://www.github.com/Senteran/go-ga-adaptative

the graphs for the other clusters can be found in Appendix A. It can be observed that most strategies follow the trend of higher mutation rates at the start, and lower later on. AFH acts on individuals so its behaviour is almost opposite. As the average individual's fitness improves, then individuals with lower fitness values are going to receive higher and higher mutation rates. The self-adaptive strategy doesn't show any convergence towards one mutation rate over time, but rather oscillates between different values. This might be a quirk of the specific implementation of it that was chosen.

The different handling of the mutation rates lead to different generations the algorithms took to converge. Higher mutation rates likely lead to more generations to converge. This can be a positive, as it can prevent the algorithm from converging prematurely but it can also needlessly cause more computation, where a couple of generations of simple local optimization could have found the global minimum. The graph of convergence generations for LJ47 is shown in Figure 10, with more graphs of generations to convergence being available in Appendix B. This graph shows that it is hard to find a correlation between the generations of successful solutions, and the quality of an control strategy. For example, AFH has the lowest median time for this cluster, however in terms of the median generation count, it is among the highest of the algorithms. Further research would be needed to derive any conclusions about optimal convergence generation counts.

In matters of the local optimizer used, it was considered to change it to the Fast Inertial Relaxation Engine (FIRE). This algorithm uses a velocity adaptation method similar to physical inertia to dynamically update the velocities of each atom based on the dot product of the gradient and momentum vectors [27]. This change was considered because FIRE's execution time on a cluster is much more consistent (and generally much lower) than BFGS, leading to total results which are less varied as well. However, after preliminary testing, FIRE reduced the number of global minima successfully found by about 4 times, while only decreasing the computation time by 30%. Because of this the default BFGS algorithm was kept.

In this research the adapted mutation rate was applied equally to three mutation operators. Undoubtedly those mutations would benefit from having different mutation rates. This consideration was excluded from this research due to the increased complexity that it would add. Better results can be acquired by adding this as a consideration. For example, for the LJ31 cluster, etching proves very useful due to its many deep minima, while it may come in less use in LJ19.

The results presented in this paper show a large degree of variance. The biggest reason for this is the algorithms heavy use of local optimization. It acts in a similar way to a mutation operator, where it takes an individual and moves it along the energy surface in a probabilistic manner. It does it by purely looking at the functions of the potential between the atoms. That is a different approach than the mutation operators, which try to manoeuvrer the atoms in sensible ways that might produce better clusters. Accounting for the factor of the local optimizer in the results is difficult. There has been little research done on the impact this has on the Genetic Algorithm so giving any results with confidence is impossible. Research using some of these strategies without the use of local optimizers has shown good results in the past, like in Herrera et al. [13]. They have shown that their FLC implementation greatly outperformed the other solutions they considered.

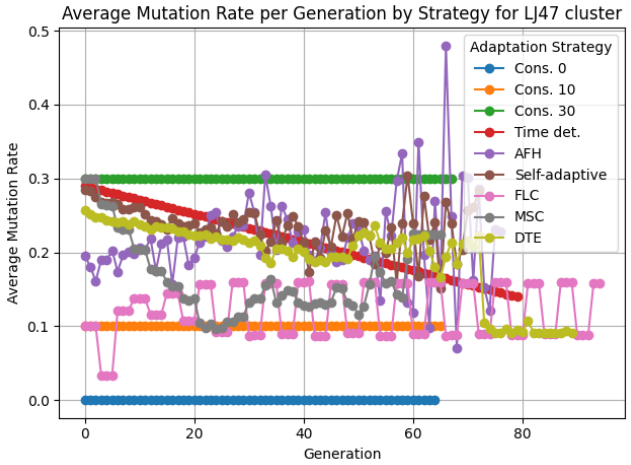

Figure 9: Graph of the average mutation rate over time for each of the adaptation strategies. Some of the strategies work on individuals, while some work on the whole population so the interpretation of their supplied mutation rates differ. The strategies take different approaches to the mutation over time, some decrease it, and some keep it stable.

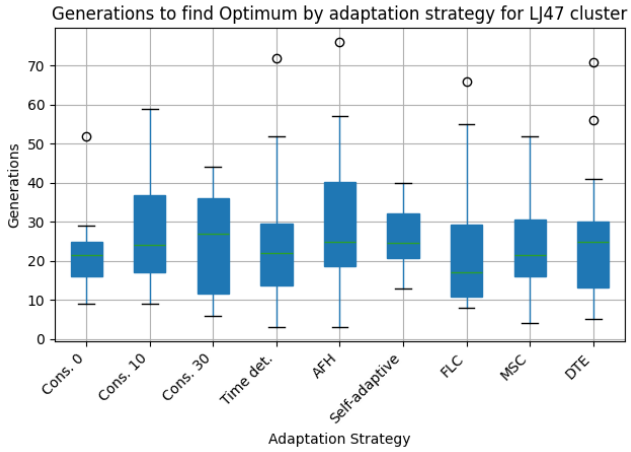

Figure 10: Boxplot of the generations needed for each of the algorithms to converge for LJ47 clusters for all population sizes.

7 Conclusions and Future Recommendations

The adaptation strategies presented in this paper show a small improvement to the performance of the algorithm when compared to the chosen constant mutation rates. The degree of improvement depends heavily on the specific cluster chosen. For simple or small clusters the change doesn't appear to be

large. It becomes more prominent in clusters when the algorithm runs for a larger number of generations, as then they total differences in the number of mutations can take effect. The choice of strategy for cluster is also important to note. Some perform better for different types of clusters.

These results are not fully conclusive however. The heavy use of local optimisation makes analysing the results of this research difficult. Further research should be undertaken in the field of analysing genetic algorithms utilising local optimizers, or more broadly, more operations than just crossover and mutation.

Additionally, this research only considered a benchmark of the algorithm's efficiency at finding a global minimum. This was done to make analysing the results easier. Different evaluation metrics or application domains may reveal deeper insights into the benefits of dynamic control.

References

[1] D. Wales, "Decoding the energy landscape: Extracting structure, dynamics and thermodynamics," *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 370, p. 2877–99, June 2012.

[2] D. J. Wales and J. P. K. Doye, "Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms," *The Journal of Physical Chemistry A*, vol. 101, p. 5111–5116, July 1997.

[3] X. Wang, S. Ramírez-Hinestrosa, J. Dobnikar, and D. Frenkel, "The lennard-jones potential: when (not) to use it," *Physical Chemistry Chemical Physics*, vol. 22, p. 10624–10633, May 2020.

[4] M. K.-H. Kiessling, "Testing lennard-jones clusters for optimality," *The Journal of Chemical Physics*, vol. 159, p. 014301, July 2023.

[5] A. Nashim and K. Parida, "A glimpse on the plethora of applications of prodigious material mxene," *Sustainable Materials and Technologies*, vol. 32, p. e00439, July 2022.

[6] J. H. Kim, T. V. Pham, J. H. Hwang, C. S. Kim, and M. J. Kim, "Boron nitride nanotubes: synthesis and applications," *Nano Convergence*, vol. 5, p. 17, June 2018.

[7] L. T. Wille and J. Vennik, "Computational complexity of the ground-state determination of atomic clusters," *Journal of Physics A: Mathematical and General*, vol. 18, p. L419, June 1985.

[8] S. Goedecker, "Minima hopping: An efficient search method for the global minimum of the potential energy surface of complex molecular systems," *The Journal of Chemical Physics*, vol. 120, p. 9911–9917, June 2004.

[9] D. M. Deaven and K. M. Ho, "Molecular geometry optimization with a genetic algorithm," *Physical Review Letters*, vol. 75, p. 288–291, July 1995.

[10] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, p. 122–128, Jan. 1986.

[11] R. Haupt, "Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors," in *IEEE Antennas and Propagation Society International Symposium. Transmitting Waves of Progress to the Next Millennium. 2000 Digest. Held in conjunction with: USNC/URSI National Radio Science Meeting (C*, vol. 2, p. 1034–1037 vol.2, July 2000.

[12] M. A. Lee and H. Takagi, "Dynamic control of genetic algorithms using fuzzy logic techniques," in *Proceedings of the 5th International Conference on Genetic Algorithms*, (San Francisco, CA, USA), p. 76–83, Morgan Kaufmann Publishers Inc., June 1993.

[13] F. Herrera and M. Lozano, "Adaptive control of the mutation probability by fuzzy logic controllers," in *Parallel Problem Solving from Nature PPSN VI*, p. 335–344, Springer, Berlin, Heidelberg, 2000.

[14] B. W. Goldman and D. R. Tauritz, "Meta-evolved empirical evidence of the effectiveness of dynamic parameters," in *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, (New York, NY, USA), p. 155–156, Association for Computing Machinery, July 2011.

[15] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statistical Science*, vol. 8, no. 1, p. 10–15, 1993.

[16] A. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 3, p. 124–141, July 1999.

[17] J. P. K. Doye, M. A. Miller, and D. J. Wales, "Evolution of the potential energy surface with size for lennard-jones clusters," *The Journal of Chemical Physics*, vol. 111, p. 8417–8428, Nov. 1999.

[18] T. Bäck, "Self-adaptation in genetic algorithms," Oct. 1994.

[19] T. Bäck and M. Schütz, *Intelligent mutation rate control in canonical genetic algorithms*, vol. 1079 of *Lecture Notes in Computer Science*, p. 158–167. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996.

[20] M. Srinivas and L. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, p. 656–667, Apr. 1994.

[21] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Feb. 1996.

[22] A. Hjorth Larsen, J. Jørgen Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. Bjerre Jensen, J. Kermode, J. R. Kitchin, E. Leonhard Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. Bergmann Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, and

K. W. Jacobsen, "The atomic simulation environment-a python library for working with atoms," *Journal of Physics. Condensed Matter: An Institute of Physics Journal*, vol. 29, p. 273002, July 2017.

[23] J. Zhao, R. Shi, L. Sai, X. Huang, and Y. S. and, "Comprehensive genetic algorithm for ab initio global optimisation of clusters," *Molecular Simulation*, vol. 42, no. 10, p. 809–819, 2016.

[24] M. D. Wolf and U. Landman, "Genetic algorithms for structural cluster optimization," *The Journal of Physical Chemistry A*, vol. 102, p. 6129–6137, July 1998.

[25] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical Programming*, vol. 45, p. 503–528, Aug. 1989. Company: Springer Distributor: Springer Institution: Springer Label: Springer number: 1 publisher: Springer-Verlag.

[26] Delft High Performance Computing Centre (DHPC), "DelftBlue Supercomputer (Phase 2)." https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2, 2024.

[27] E. Bitzek, P. Koskinen, F. Gähler, M. Moseler, and P. Gumbsch, "Structural relaxation made simple," *Physical Review Letters*, vol. 97, p. 170201, Oct. 2006.

A Mutation Rate Graphs

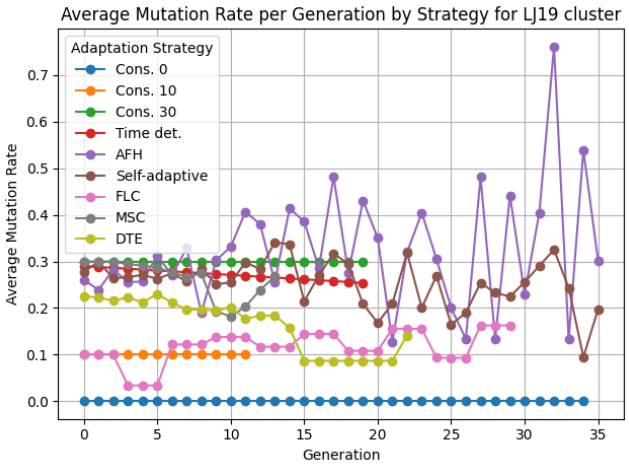

Figure 11: Graph of the average mutation rate over time for each of the adaptation strategies for LJ19.

B Convergence Generations Graphs

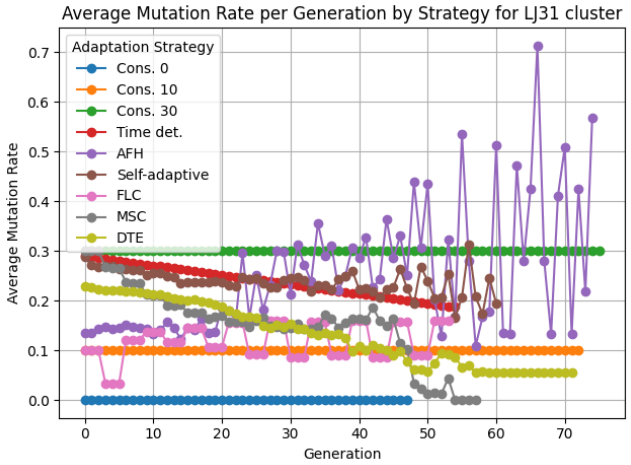

Figure 12: Graph of the average mutation rate over time for each of the adaptation strategies for LJ31

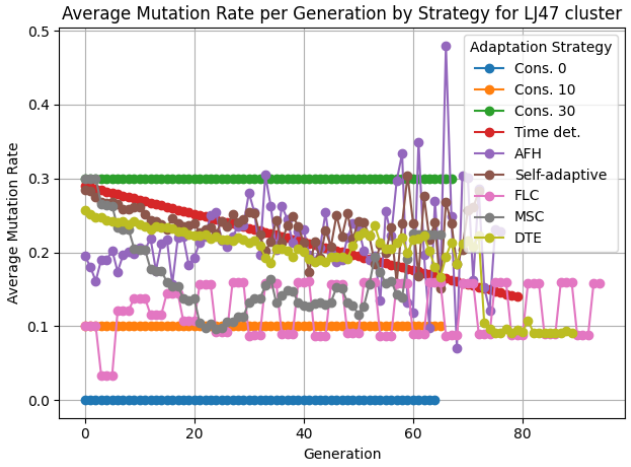

Figure 13: Graph of the average mutation rate over time for each of the adaptation strategies for LJ47.

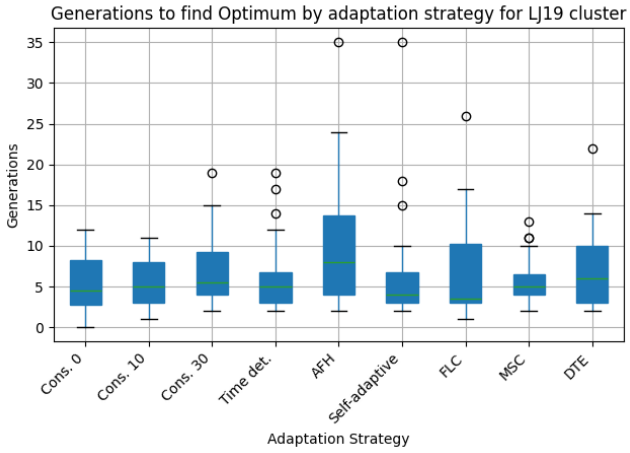

Figure 14: Graph of the convergence generations for each of the adaptation strategies for LJ19.

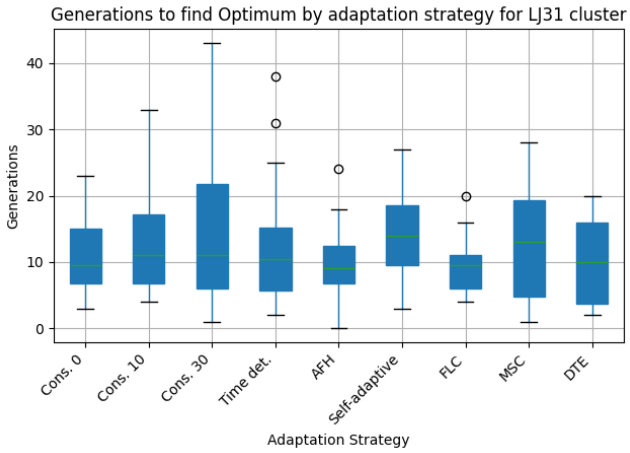

Figure 15: Graph of the convergence generations for each of the adaptation strategies for LJ31.

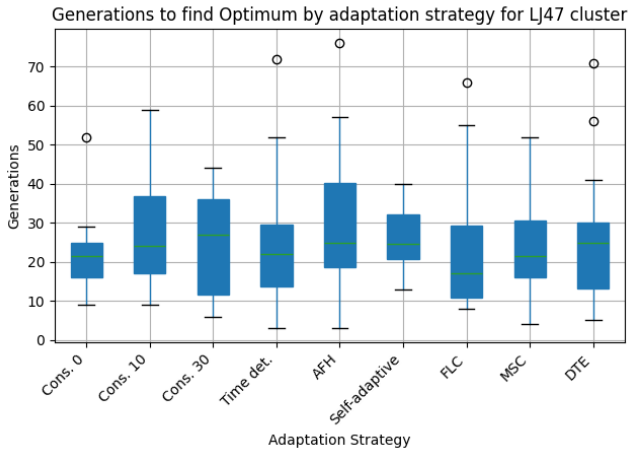

Figure 16: Graph of the convergence generations for each of the adaptation strategies for LJ47.