EE682
HW4 - Performance Comparison between GA and QEA

Herman Kolstad Jakobsen
20196493

November 22, 2019

Contents

1 Introduction

The task is to compare the performance of a *Genetic Algorithm (GA)* with a *Quantum-inspired Evolutionary Algorithm (QEA)* for five given De Jong functions. First, an overview of the GA will be given. Then, the performance of the GA will be discussed using three different selection methods respectively. A literature survey of QEA will so presented. Finally, the performance of the GA will be compared to the performance of the QEA. Source code for both the GA and QEA was handed out.

2 Genetic Algorithm

The main idea behind this type of algorithm is that each good parent (solution) should produce a new and better offspring (solution). In this way, the algorithm should converge into a final and optimal offspring which will represent the optimal solution for a given function. The creation of an offspring bases itself on information exchange between parents and some randomness. Overall, genetic algorithms can be seen as a stochastic algorithm based on natural phenomena, including genetic inheritance and Darwinian strife for survival.

Genetic algorithms are derivative free, where the optimization consists of repeated evaluations of the objective function. Due to the derivative freeness, the convergence speed of genetic algorithms are generally slower compared to traditional derivative-based optimization algorithms. However, genetic algorithms can easily be implemented using parallel-processing, which can greatly reduce the computation time. As a counterweight to the relative longer computation time, genetic algorithms can easily deal with more complex optimization problems and objective functions without sacrificing even more computation time. This imposes some flexibility to genetic algorithms where this type of algorithm is applicable to many types of problems. Further, genetic algorithms are also injected with a stochastic component. By applying randomness in deciding the next search direction, this type of optimization is less likely to be stuck in local minima, which is a known problem for nonlinear objective functions.

An important aspect of genetic algorithms is the balancing of exploration and exploitation. The algorithm should be able to both exploit the currently best solution and explore the whole search space. The iteration from one generation of solutions to the next can essentially be divided into three steps:

1. Select the best solutions from the previous generation and store them.
2. A crossover where information between different potential solutions are exchanged.
3. A mutation of different potential solutions which introduces some extra variability into the population.

Tuning a genetic algorithm then boils down to deciding the balance between exploration and exploitation. Deciding this balance is done by choosing the amount of crossover and mutation of solutions. Crossover and mutation are often referred to as genetic operators.

Another aspect that affects the performance of a genetic algorithm is how the best solutions from the currently generation are selected. The selection methods base themselves on how fit a solution is, i.e. how close the objective function value is to the optimal value using this solution. In the source code, three different types of selection methods were implemented:

- Proportional selection - Calculates a cumulative fitness for each solution, which can be seen on as creating a roulette wheel consisting of the different solutions. The wheel is then spun and the solution it stops on is stored for further use. The roulette wheel is spun a total times equal to the population size.
- Ranking selection - Has several similarities with proportional selection. Before calculating the cumulative fitness, however, the solutions are ranked based on their fitness. The population is then divided into subsets, and a pre-decided cumulative fitness is given to each subset. The better solutions are given a higher cumulative fitness. The new population is then chosen from a roulette wheel, where the better solutions have a higher chance of getting picked.
- Tournament selection - The population is divided into subsets, where solutions from different subsets will compete against each other by comparing fitness. The better solution will be stored for use in the new generation.

3 Tuning of Genetic Algorithm

It was attempted to tune the genetic algorithm by changing the genetic operators. Using the algorithm with the De Jong 2 function and proportional selection, different combinations of crossover and mutation probabilities were tested out. The algorithm was ran 1000 times for each combination of genetic operators, and the average of the best fitness from each run was used as a measurement for the algorithm's performance. For the De Jong 2 function and proportional selection it was possible to obtain a better performance by increasing the probability of mutation by five times of the default value. However, an increase in the probability of mutation resulted in the algorithm using more generations before converging. The different configurations, with corresponding performances, are stated in *GA_tuning.txt.*

The configuration that gave a better performance for De Jong 2 and proportional selection was then tested out with different selection methods, and compared against the performance of the default tuning. For the De Jong 2 function, the new tuning gave a better performance for all the three selection methods. However, when the algorithm was tested out on the De Jong 1 function, the new tuning gave a worse performance.

From testing out different configurations of the genetic algorithm on the given De Jong functions, the effect of the genetic operators were clearly shown.

Functions that had few local minima required little exploration of the solution space, and it was therefore beneficial to reduce the probability of mutation and increase the probability of crossover. However, functions that had many local minima, like De Jong 4, required more exploration of the solution space. For these kind of functions, a better performance was obtained by increasing the probability of mutation.

It was therefore concluded that the tuning of the algorithm was dependent on the function it was used for. The default tuning introduces a good balance between exploration and exploration independent of the De Jong functions. Ultimately, the default tuning was chosen for further use of the genetic algorithm.

4 Performance of GA with different selection methods

Due to difficulties improving the performance of the algorithm further, the algorithm was used with the default tuning. The algorithm was ran 10 times for each De Jong function. Running the algorithm more times for each function would give a better estimation of the algorithm's performance. However, the number of runs were sufficient in order show the difference in performance between the selection methods. The raw results from the runs of the genetic algorithm with different De Jong functions and selection methods can be found in the *GA_output* folder.

4.1 De Jong 1

The optimal solution for this function is $x_i = 0$ and $f(\mathbf{x}) = 0$ for $i = 1, 2, 3$. The results for the genetic algorithm for the three different selection methods are shown in table 1. All of the selection methods yield the same optimal solution. Ignoring the computational error, these values are the same as the optimal value of the function. However, the average number of generations needed to obtain the optimal solution is far superior for the ranking selection method. The proportional selection method, on the other hand, has the highest number of generations needed to obtain the optimal solution. Based on this, it can be concluded that the ranking selection method gives the best performance while the proportional selection gives the worst.

To easier compare the selection methods, the generated results for one run of the algorithm are graphically shown in fig. 1. Here, the tournament selection method has a consistently lower average fitness and standard deviation among its solutions for each generation. The ranking selection method, however, has a large and fluctuating standard deviation, but its average fitness is still smaller compared to the proportional selection method. From fig. 1 it can be concluded that the tournament selection method has a more favourable performance.

The best selection method for this De Jong depends on which performance criteria that is most valued. If fast convergence is valued the most, then the ranking selection method is superior. The tournament selection method is favourable

Table 1: GA results for De Jong 1 with three different selection methods

GA results for De Jong 1			
	Proportional	Ranking	Tournament
Best fitness	10^{-9}	10^{-9}	10^{-9}
Mean of best fitness	$5.6 \cdot 10^{-8}$	10^{-9}	10^{-9}
Standard deviation of best fitness	$1.25 \cdot 10^{-7}$	0	0
Average generation for opt. sol.	422.5	101.1	263.8

if it is important to have a good fitness and small standard deviation between the solutions for each generation.

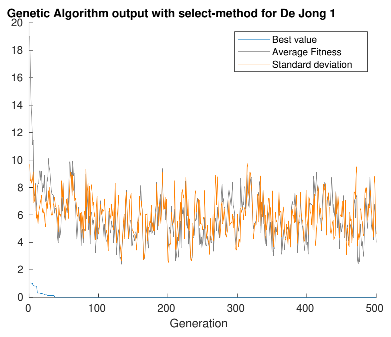
4.2 De Jong 2

The optimal solution for this function is $x_i = 1$ and $f(\mathbf{x}) = 0$ for $i = 1, 2$. The results for the genetic algorithm for the three different selection methods are shown in table 2. Ignoring computational errors, all of the selection methods are able to obtain the optimal solution of the De Jong 2 function. Similar to the results for the De Jong 1 function, the ranking selection method uses on average the least generations to obtain the optimal solution. However, this selection method also has the highest mean and standard deviation of the best fitness, something which is not desirable. The tournament selection method uses on average the most generations, but as a consequence the method has the best mean and standard deviation of the best fitness.

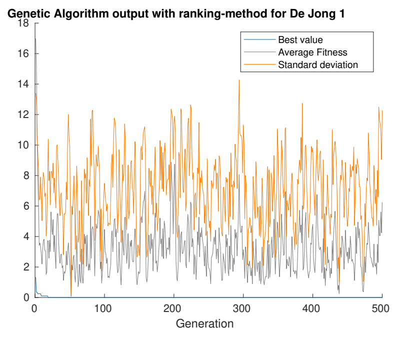
Table 2: GA results for De Jong 2 with three different selection methods

GA results for De Jong 2			
	Proportional	Ranking	Tournament
Best fitness	$1.4 \cdot 10^{-8}$	$2 \cdot 10^{-9}$	$6 \cdot 10^{-9}$
Mean of best fitness	$3.1 \cdot 10^{-4}$	$4.3 \cdot 10^{-4}$	$1.7 \cdot 10^{-4}$
Standard deviation of best fitness	$3.2 \cdot 10^{-4}$	$5.1 \cdot 10^{-4}$	$2.8 \cdot 10^{-4}$
Average generation for opt. sol.	432.5	409.5	447.1

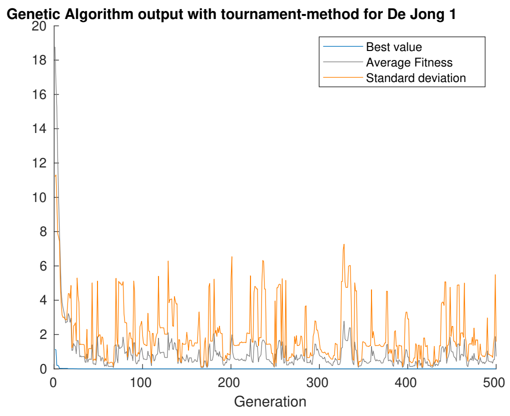
The results from an arbitrary run of the algorithm with the De Jong 2 function is shown in fig. 2. The ranking selection method has both the highest and most fluctuating standard deviation and average fitness. For this run, it also took some time before the optimal solution was found. Both the tournament and proportional selection method has rather similar results. However, the tournament selection method has on average a lower fitness and the stan-



(a) Proportional



(b) Ranking



(c) Tournament

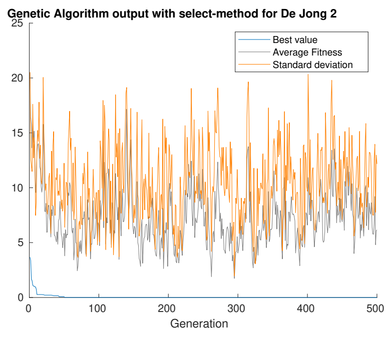
Figure 1: Generation of optimal solution for De Jong 1

dard deviation is consequently low when ignoring the spikes. In conclusion, the tournament selection method performs best for the De Jong 2 function.

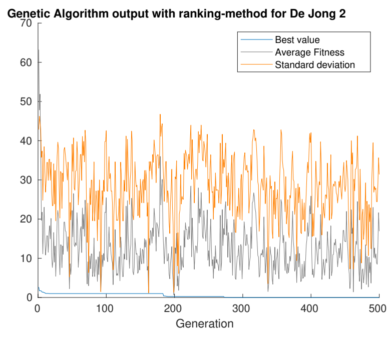
4.3 De Jong 3

The optimal solution for this function is $x_i = -5.12$ and $f(\mathbf{x}) = -30$ for $i = 1, ..., 5$. The results for the genetic algorithm for the three different selection methods are shown in table 3. All the selection methods are able to find the optimal solution. The only result that separates the selection methods is the average of how many generations that are needed to find the optimum. Here, the ranking selection method is superior, while the proportional selection method is far inferior.

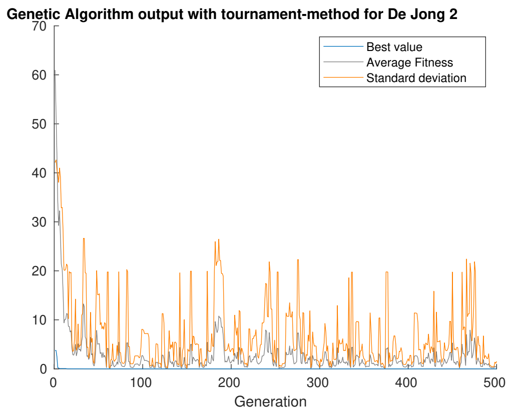
The results generated from an arbitrary run is shown in fig. 3. The ranking and tournament selection methods have pretty identical results. The standard deviation and average fitness for the tournament selection method is, however,



(a) Proportional



(b) Ranking



(c) Tournament

Figure 2: Generation of optimal solution for De Jong 2

slightly less noisy compared to the ranking selection method. The proportional selection method takes long to converge and has a rather noisy standard deviation. Taking the number of average generations needed to find the optimal solution into consideration, it can be concluded that the ranking selection method has a better performance for the De Jong 3 function.

4.4 De Jong 4

The optimal solution for this function is $x_i = 0$ and $f(\mathbf{x}) = 0$ for $i = 1, ..., 30$. The results for the genetic algorithm for the three different selection methods are shown in table 4. None of the selection methods are able to make the genetic algorithm converge to the optimal solution. This is clearly shown by the average number of generations used to obtain the optimal solution being the generation threshold. However, the ranking selection method was closest to the optimal solution and had the most desirable mean and standard deviation of

Table 3: GA results for De Jong 3 with three different selection methods

GA results for De Jong 3			
	Proportional	Ranking	Tournament
Best fitness	-30	-30	-30
Mean of best fitness	-30	-30	-30
Standard deviation of best fitness	0	0	0
Average generation for opt. sol.	343.7	49.5	87.3

the best fitness. For this function, the tournament method performed the worst compared to the other selection methods. This is a contrast to the results from the previous functions.

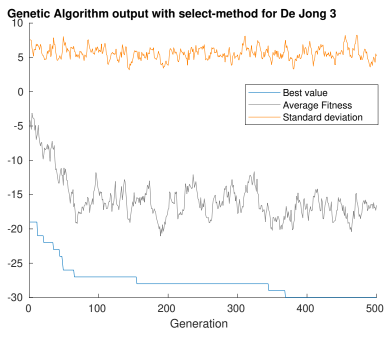
Table 4: GA results for De Jong 4 with three different selection methods

GA results for De Jong 4			
	Proportional	Ranking	Tournament
Best fitness	0.22	0.08	0.29
Mean of best fitness	0.53	0.14	0.66
Standard deviation of best fitness	0.18	0.04	0.28
Average generation for opt. sol.	500	500	500

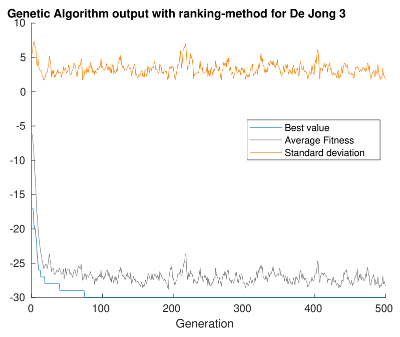
The results generated from an arbitrary run is shown in fig. 4. All the selection methods have a somewhat identical result. The standard deviation and mean fitness of the tournament selection is less noisy. The ranking selection method converges the fastest to a solution close to the optimum, while the tournament selection method converges the slowest. It is worth noting the ranking selection method uses roughly 25 generations to obtain its first acceptable solution. However, since the ranking selection method found solutions closest to the optimal solution it can be concluded that the this selection method has the best performance for the De Jong 4 function.

4.5 De Jong 5

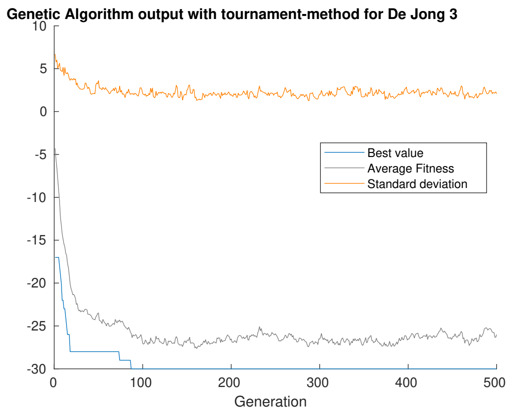
The optimal solution for this function is $x_i = -32$ and $f(\mathbf{x}) \approx 1$ for $i = 1, 2$. The results for the genetic algorithm for the three different selection methods are shown in table 5. All the selection methods make the genetic algorithm able to converge to approximately the optimal solution. The tournament selection methods has the most desirable mean and standard deviation of the best fitness, while also converging the fastest. The proportional selection method has



(a) Proportional



(b) Ranking



(c) Tournament

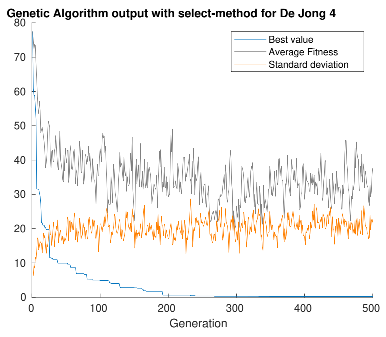
Figure 3: Generation of optimal solution for De Jong 3

better mean and standard deviation of the best fitness compared to the ranking selection method, but converges slower.

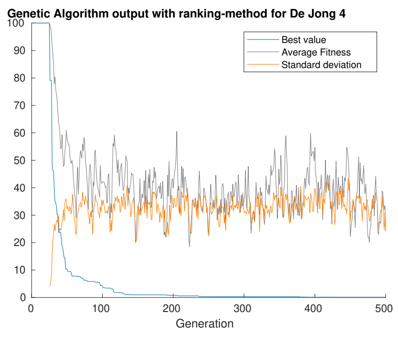
The results generated from an arbitrary run is shown in fig. 5. Clearly, the tournament selection method has the best results with both the most desirable average fitness and standard deviation. The proportional selection method has a high average fitness compared to the other two methods. In conclusion, the tournament selection method gives the best performance for the De Jong 5 function.

4.6 Performance summary

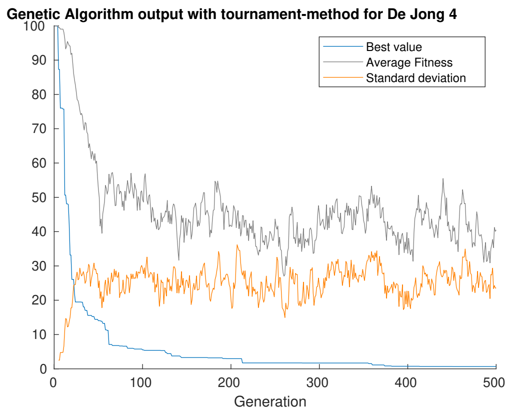
The selection methods that gave the best performance for the different De Jong functions is summarized in table 6. The ranking selection method and the tournament selection method gave the best performance for an equal number of De Jong functions. Overall, the ranking selection method had a faster convergence



(a) Proportional



(b) Ranking



(c) Tournament

Figure 4: Generation of optimal solution for De Jong 4

rate, and was able to find a solution closest to the optimal solution for the De Jong 4 function. The tournament selection method, on the other hand, had much more favourable characteristics when an arbitrary run was inspected. On average, both the fitness and standard deviation was less noisy and the fitness was generally closer to the optimal solution.

5 Quantum-inspired Evolutionary Algorithm

Quantum-inspired Evolutionary Algorithms draw their inspiration from both evolutionary and quantum computing, and differs from genetic algorithms by representing the variables as qubit chromosomes instead of strings of bits. Qubits are the smallest unit of information stored in a two-state quantum computer. A qubit differs from an ordinary binary bit by being able to be in three different states: ”0”, ”1” or any superposition of the two. This means that a system

Table 5: GA results for De Jong 5 with three different selection methods

GA results for De Jong 5			
	Proportional	Ranking	Tournament
Best fitness	0.998	0.998	0.998
Mean of best fitness	1.06	1.18	1.04
Standard deviation of best fitness	0.13	0.17	0.08
Average generation for opt. sol.	420.8	359.9	325.8

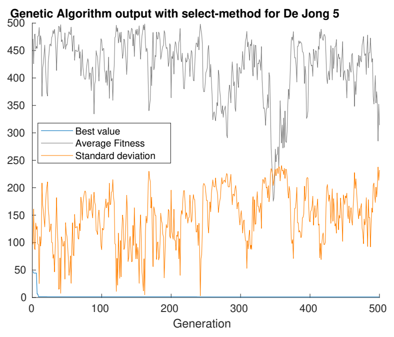
Table 6: Selection methods that gave best performance for GA

Best performing selection methods			
	Proportional	Ranking	Tournament
De Jong 1		✓	✓
De Jong 2			✓
De Jong 3		✓	
De Jong 4		✓	
De Jong 5			✓

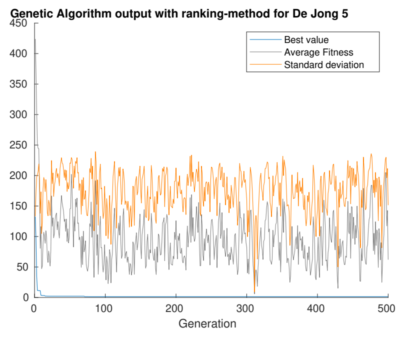
of m-qubits can represent 2^m states at the same time. In act of observing a quantum state, it collapses to a single state depending on the probability amplitudes of the quantum states. Each qubit individual is then a probability model, which can represent multiple solutions. This makes the representation of variables probabilistic, which is a contrast to the deterministic way of representing variables in genetic algorithms. A quantum population consists of probability distributions of sampling the search space, contrary to representing the population as exact points.

As another contrast to the genetic algorithms, Quantum-inspired Evolutionary Algorithms exclusively use quantum gates as operators instead of the genetic operators, crossover and mutation. Quantum gates are more flexible opposed to the many classical logical gates, where quantum gates allow any process to be reversible. The tuning of a QEA consists then of choosing the type and configuration of the quantum gates. Further, QEAs use elitism and migration as selection methods for finding the variables for the next generation.

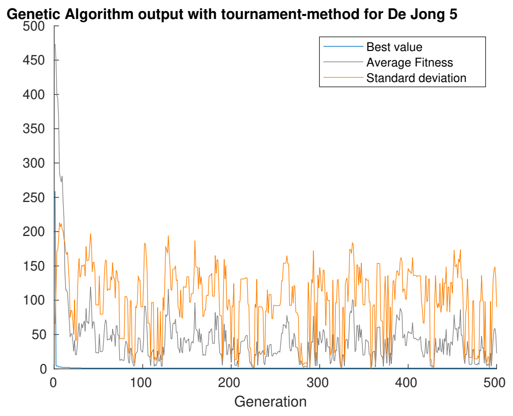
The structure of a QEA is divided into several steps and is shown fig. 6. Binary solutions are obtained by observing the quantum states. The observed solution depends on the probability amplitudes of the quantum states. The algorithm then replaces lesser good solutions with the best observed solution. This is called migration and can happen locally in the same local group or globally. Lastly, the qubits are updated using the quantum gates. The loop is



(a) Proportional



(b) Ranking



(c) Tournament

Figure 5: Generation of optimal solution for De Jong 5

then repeated until a specified condition is met.

An evolutionary algorithm based on the concepts and principles of quantum computing was first proposed in [5]. Other early examples of quantum-inspired genetic algorithms which use binary quantum representation based on qubits were introduced by Han and Kim [3, 2, 4]. In the following years, most papers have focused on reporting successful applications of Quantum-Inspired Evolutionary Algorithm in different examples. An example is presented in [1]. Other topics that have been introduced throughout the years are generalized genetic operators for quantum individuals, modifications of quantum individuals' representation and binary quantum coding. During this time period it has, however, been aimed little attention on in-depth theoretical analysis of Quantum-Inspired Evolutionary Algorithms. In total, it has only been written one book that is thoroughly devoted to Quantum-Inspired Evolutionary Algorithms [6]. An overview of some of the different papers linked to Quantum-Inspired Evolutionary Algorithms can be found in [7].

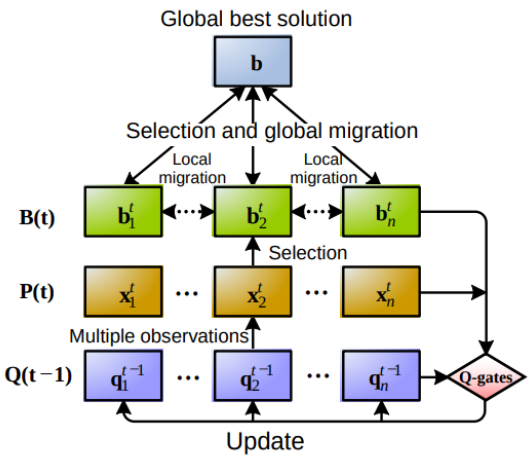


Figure 6: Overview of QEA structure

Further studies on how to use the new possibilities introduced by Quantum-Inspired Evolutionary Algorithms can be expected in the future. The additional randomness in the algorithm allows it to solve a huge variety of optimization problems. Such problems can include problems with high uncertainty and imprecise definition , where an example is to evolve fuzzy rules for an FLC.

A side of Quantum-Inspired Evolutionary Algorithms that has not yet been sufficiently explored is the theoretical analysis of the algorithm. These analysis can include dynamic systems analysis, machine learning or modern tuning techniques.

Lastly, and maybe the biggest challenge for the future regarding Quantum-Inspired Evolutionary Algorithms, is to implement the algorithm on a quantum computer. Only theoretical results has been achieved so far.

6 Tuning of Quantum-inspired Evolutionary Algorithm

Tuning of a QEA consists of deciding the type of quantum gate and its configuration. The QEA from the source code was implemented with a rotation gate. To tune the algorithm, different values of the rotation angle for the rotation gate was tested out for the De Jong 2 function. The algorithm was ran 1000 times for each quantum gate configuration, and the mean profit together with the standard deviation was used as a measurement for the algorithm's performance. The results from the different configurations can be seen in *QEA_tuning.txt*.

It was found out that a decrease in the rotation angle resulted in a better mean profit and standard deviation, but longer computation time. An increase in the rotation angle, however, resulted in a worse mean profit and standard deviation, but shorter computation time. Modifying the lookup table was also tested out, where some of the configurations gave the same effect as an increase in the rotation angle, and some configurations gave the same effect as a decrease in the rotation angle. Lastly, the algorithm was tested with a square-root-of-NOT (SRN) gate. This gate gave a worse result than the rotation gate and was therefore discarded.

Due to lack of experience and knowledge, the task of improving the performance of the QEA was considered difficult. Different configurations resulted in a trade-off between more desirable results and computation time. The default configuration offers a good balance between these two statistics. Hence, the default tuning was chosen for further use of the Quantum-inspired Evolutionary Algorithm.

7 Performance comparison between QEA and GA

Due to difficulties improving the performance of the QEA further, the algorithm was used with the default tuning. In order to achieve a fair comparison between the two algorithms, the QEA was also ran 10 times for each function, the maximum number of generations was reduced from 1000 to 500 and the population size increased from 15 to 50. The termination condition, *USER_TERMI_BEST*, for the QEA had a significant impact on the mean profit, standard deviation and the number of generations needed to obtain an optimal solution. Due to little knowledge of this variable, it was set to the default value of 0.9. For the De Jong 3 function and the De Jong 5 function the termination condition was tuned in order to get a decent result. The raw results from the runs of the QEA with different De Jong functions can be found in the *QEA_output* folder.

The performance of the genetic algorithm depended on the chosen selection method. To achieve a best possible performance for the genetic algorithm, and hence giving a fair comparison with the QEA, the ranking selection method was selected for the genetic algorithm. This selection method gave generally the fastest convergence rate and solutions closest to the optimal solution for the different De Jong functions.

7.1 De Jong 1

The generated results from both the GA and the QEA are shown in table 7. Both algorithms are able to obtain the optimal solution. The only result that separates the two algorithms is the number of generations needed to obtain the optimal solution. The GA algorithm only needs one quarter of the generations needed by the QEA.

Table 7: Comparison between GA and QEA for De Jong 1

Data generated from running algorithms with De Jong 1		
	GA	QEA
Best fitness	10^{-9}	10^{-9}
Mean of best fitness	10^{-9}	10^{-9}
Standard deviation of best fitness	0	0
Average generation for opt. sol.	101.1	458

7.2 De Jong 2

The generated results from both the GA and the QEA are shown in table 8. Ignoring computational errors, both algorithms are able to obtain the optimal solution. The GA has a slightly better best fitness, but this can be a consequence of the chosen termination conditions. On the other side, the QEA has a better fitness mean and standard deviation, while the generations needed by both the algorithms are pretty much identical.

Table 8: Comparison between GA and QEA for De Jong 2

Data generated from running algorithms with De Jong 2		
	GA	QEA
Best fitness	$2 \cdot 10^{-9}$	$3.9 \cdot 10^{-8}$
Mean of best fitness	$4.3 \cdot 10^{-4}$	$6.9 \cdot 10^{-5}$
Standard deviation of best fitness	$5.1 \cdot 10^{-4}$	$1.7 \cdot 10^{-4}$
Average generation for opt. sol.	409.5	413

7.3 De Jong 3

The generated results from both the GA and the QEA are shown in table 9. Both algorithms are able to obtain the optimal solution. Similar to the results from the De Jong 1 function, the only result to separate the two algorithms is the number of generations needed to obtain the optimal solution. The QEA algorithm only needs two thirds of the generations needed by the GA.

7.4 De Jong 4

The generated results from both the GA and the QEA are shown in table 10. This function is difficult to minimize for both the GA and the QEA, where none of the algorithms are able to obtain the optimal solution. This is shown in both

Table 9: Comparison between GA and QEA for De Jong 3

Data generated from running algorithms with De Jong 3		
	GA	QEA
Best fitness	-30	-30
Mean of best fitness	-30	-30
Standard deviation of best fitness	0	0
Average generation for opt. sol.	343.7	224

the best fitness measurement and that the algorithms meet the threshold of number of generations used. However, the QEA performs better than the GA in all the results. The QEA has a better best fitness, mean fitness and standard deviation.

Table 10: Comparison between GA and QEA for De Jong 4

Data generated from running algorithms with De Jong 4		
	GA	QEA
Best fitness	0.08	0.06
Mean of best fitness	0.14	0.09
Standard deviation of best fitness	0.04	0.02
Average generation for opt. sol.	500	500

7.5 De Jong 5

The generated results from both the GA and the QEA are shown in table 11. Both algorithms are able to obtain the optimal solution. However, the QEA performs far superior with no standard deviation and a low number of required generations. The GA also performs good, but the algorithm has some significant standard deviations in its best fitness.

7.6 Discussion

A summary of the functions the algorithms were able to optimize is shown in table 12. Both of the algorithms struggled to optimize the De Jong 4 function. For this function, however, the QEA was able to achieve better results and a solution closer to the optimal solution. For the simplest De Jong function, De Jong 1, the GA outperformed the QEA by converging faster to the optimal solution. Nonetheless, as soon as the functions became more complex, the performance of the QEA became superior to the one of the GA.

Table 11: Comparison between GA and QEA for De Jong 5

Data generated from running algorithms with De Jong 5		
	GA	QEA
Best fitness	0.998	0.998
Mean of best fitness	1.18	0.998
Standard deviation of best fitness	0.17	0
Average generation for opt. sol.	359.9	119

Table 12: Overview on whether an algorithm managed to optimize the given function

Comparison between GA and QEA		
	GA	QEA
De Jong 1	✓	✓
De Jong 2	✓	✓
De Jong 3	✓	✓
De Jong 4		
De Jong 5	✓	✓

One of the reasons for the GA performing worse for more complex functions could be the balancing between exploration of the search space and exploitation of the current solution, where population diversity and selective pressure is crucial. A GA must be initialized with a diverse population in order to obtain a fruitful exploration of the search space. Further, selective pressure is the ratio between the probability that the most fit member of the population is selected as a parent to the probability that an average member is selected as a parent. Too high a selection pressure would result in over-exploiting the current solution and result in the population converging too early.

The QEA overcomes this restriction due to its probabilistic representation of quantum states. By having the probabilistic representations, the QEA obtains a good diversity in its population which results in an excellent ability of global search. Further, the structure of a QEA introduces an automatic balance ability between global search and local search. The QEA then obtains a good balance between global and local search due to its nature, as opposed to the GA which must be configured with fine-tuned parameters in order to obtain the same ability.

Another weakness of the GA is the absence of individual's past history. From one generation to the next, the new generation has no "memory" on how it obtained its solutions. Because of this, GA often under-perform when used for optimizing complex functions with many local minima. This weakness is removed in the QEA by having a probabilistic representation of the quantum states. Since the quantum states store global statistical information of good solutions found previously in the search, QEA get a better ability to deal with complex functions with many local minima.

The qubits imposes another advantage of using QEA. By being able to represent quantum states as linear superpositions, the algorithm is able to represent more solutions with fewer bits compared to GA. Hence, QEA can have a smaller population without loss of performance. A smaller population would then result in less computation time. However, this characteristic of QEA was not tested out in this assignment.

8 Conclusion

A genetic algorithm is a stochastic algorithm based on natural phenomena, including genetic inheritance and Darwinian strife for survival. The performance of such an algorithm is heavily depended on how the solutions for the next generation is chosen. Through testing, it was concluded that the ranking selection method gave the best overall performance.

Quantum-inspired Evolutionary Algorithms draw their inspiration from both evolutionary and quantum computing, and differs from genetic algorithms by representing the variables as qubit chromosomes instead of strings of bits. This type of algorithm was first presented in [5] and later in [3, 2, 4]. One of the challenges for the future is to be able to implement this type of algorithms on a quantum computer.

Overall, the Quantum-inspired Evolutionary Algorithm had a better performance than the genetic algorithm, where it needed fewer generations and obtained solutions closer to the optimal solution. The main reasons for this is due to the probabilistic representation of quantum states.

References

[1] André V. Abs da Cruz et al. *Cultural operators for a quantum-inspired evolutionary algorithm applied to numerical optimization problems.* 2005.

[2] K.H. Han and J.H. Kim. *Analysis of Quantum-Inspired Evolutionary Algorithm.* 2001.

[3] K.H. Han and J.H. Kim. *Genetic quantum algorithm and its application to combinatorial optimization problem.* 2000.

[4] K.H. Han and J.H. Kim. *Quantum-inspired evolutionary algorithm for a class of combinatorial optimization.* 2002.

[5] A. Narayanan and N. Moore. *Evolutionary Computation.* 1996.

[6] Nadia Nedjah, Leandro dos Santos Coelho, and Luiza de Macedo Mourelle. *Quantum inspired intelligent systems.* Springer Verlag, 2008.

[7] Robert Nowotniak. *Survey of Quantum-Inspired Evolutionary Algorithms.*