Name: CHAN King Yeung
SID: 1155119394
CSCI3230 Term Paper

Question 1

a) Discuss the importance of convergence and divergence for genetic algorithm in detail.

Unlike other searching algorithms, genetic algorithm (GA) consists of both convergence and divergence for an optimisation problem. Both components make GA beats other algorithms. In general, convergence means things come together while divergence refers to things moving apart. If we apply the concept into GA, we can imagine as a set of individuals getting closer and there are several groups of individuals playing around the solution space.

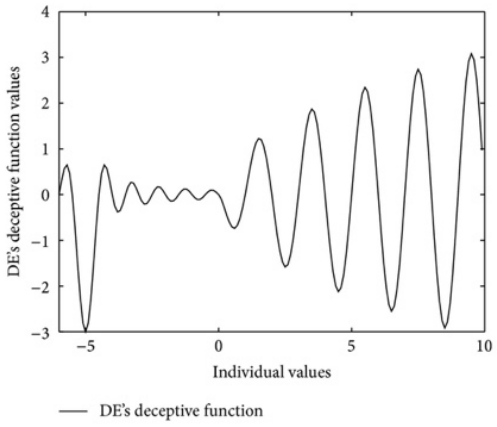
In an optimisation problem, our objective is finding a solution which is either maximised or minimised. GA generates a set of the population to find the optimal point in the solution space. The population will experience several processes to reproduce the next generation for searching the optima. We say convergence is important for GA because we want individuals in the population to obtain a similar fitness which is a measurement from the objective function. The score of fitness determines how good is an individual in the solution space. Through a selection process, individuals with higher fitness score are more likely to survive and reproduce the next generation. As such, individuals are becoming more similar to each other at some points. In other words, it converges individuals into a local optimal which provide an optimisation solution for GA. Yet, it does not guarantee GA finds an optimised solution since individuals may only find a local solution and get stuck at that point. Because individuals have already converged into a local optimum, it is not an easy job to escape the local optimum for some fitness of score. Individuals are less likely to find global maxima or minima under the situation. The phenomenon is known as premature convergence (T. Gabor, 2018). If GA commits premature convergence and output non-global optima, the algorithm has not differenced compare to other searching algorithms.

Due to the premature convergence, GA also considers divergence to help individuals break out the local optima. Diversity enables individuals to explore the solution space and locate multiple optima to find a better solution by pausing convergence (M. Bhattacharya, 2014). In short, GA decreases the similarity of individuals by driving off some individuals from the local optima and let them walk around the search space. Meanwhile, some individuals are required to populate discovered optima to avoid finding the same optima while driving off others. Both convergence and divergence empower GA to find a global optimum by searching as much as possible. However, both convergence and divergence are treated equally important, and we need to balance convergence and divergence in GA to achieve a better solution. For a small population size, it is easy to converge prematurely because there are not enough individuals to cover the search space. On the other hand, it is very time-consuming to converge a solution if the size of the population is too large. Hence, the population size should be defined effectively to ensure the quality of solution and computation time.

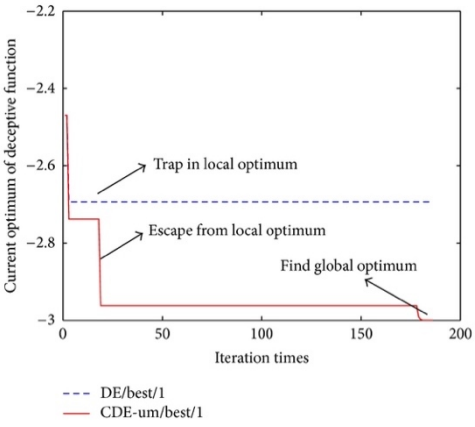
Besides, Bhattacharya believes the searchability of GA depends on the variation among individuals. Even the population size is adequate to find a global optimum, GA still converges prematurely to a non-optimal solution if the loss of diversity or loss of genetic variation for individuals (2014). In other words, divergence is a way to introduce bias for individuals to cover the solution space. B.M. Kim suggests bias is the key to improve the efficiency of convergence, and mutation operation is decisive to introduce random walk method for the chromosomes of each individual (1997). Increase the rate of mutation operator allows more variation among individuals, which reduce the chance of premature convergence. In addition, Gabor says diversity help GA to reach a solution which satisfies a multi-objective optimisation problem when a solution is able to adapt the environment changes (or change in objective function). It makes GA more favourable to some advance topic when another method is not easy to solve it. For example, Markov Chain Monte Carlo method may be a tough process to find an optimal solution when it comes to high dimensional problem. Here GA can be an alternative to achieve the same goal.

b) Illustrate the convergence and the divergence of genetic algorithm with real examples.

To describe the convergence and the divergence of genetic algorithm (GA), we adopt an example from a research paper authored by Z. Hu. Given an objective function, $f(x) = -3\,\mathrm{sinc}(2x + 10)$ if $x \in [-10, 0)$ and $f(x) = -\sqrt{x}\sin(x\pi)$ if $x \in [0, 10]$. If we eyeball the curvature of the function in Figure 1, we can see that the function is quite naughty and playing up and down. The global minimum is $f(x \coloneqq -5) = 8.506$. The convergence helps GA to find a local minimum quickly by updating the chromosomes of each individual. However, there are several hills in the function which may trap individuals at local minima. Under the situation, the divergence allows GA to drive off some individuals to climb other hills to escape from those traps. Figure 2 illustrates the process of GA. In the beginning, the convergence immediately finds a local optimum somewhere at -2.7. However, the algorithm gets stuck at that point for several generations. The divergence helps GA to escape from the trap and find another local optimum. The procedure continues until the global optimum is reached. The convergence and the divergence play different roles in GA, but both of them help GA to find local optima efficiently and effectively.



(Figure 1, captured from Z. Hu)

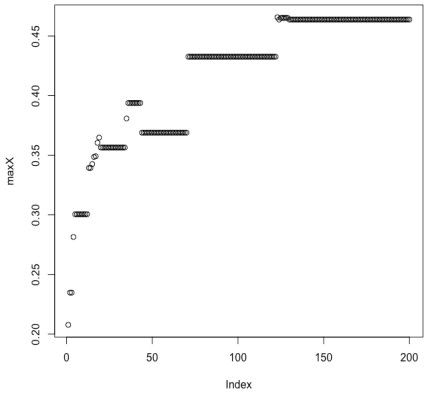


(Figure 2, captured from Z. Hu)

In the above ideal situation where GA consists of both the convergence and the divergence can eventually find the global minimum. What if GA lacks either the convergence or the divergence? GA will only converge to a single local optimum if GA absences diversity. It is almost impossible to output a global optimum if there is support from diversity. On the other hand, GA will never converge to a local optimum if GA is missing the convergence. The programme will only let individuals walk through the solution space but never concentrate on one point. The following example is captured from my previous course project to demonstrate GA without diversity property. Given an objective function in Formula 1, it is almost impossible to plot a nice curve to describe the function. In the problem statement, we need to maximise the objective function and find the global maximum. The developed GA only consists of the convergence component, and the divergence part is missing in the algorithm. As a counter-example, the GA cannot reach to the global maximum somewhere at 0.7116. In Figure 3, we can see that the GA converges a solution around 0.4657. Because the GA is missing diversity to find another local optimum, even the iteration increases, the returned outputs are near the local maximum. In the counter-example, we can see that the divergence is as important as the convergence in GA. In general, both features are crucial for GA to search though the solution space and find multiple local optima.

$$\left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|, \quad \text{if } (\forall i,\ 0 \le x_i \le 10) \text{ and } \left(\prod_{i=1}^{n} x_i \ge 0.75\right)$$

(Formula 1)



(Figure 3)

Question 2

a) Select five most notable algorithms that we learnt in this course. Justify your selections and describe their advantages and disadvantages.

The first one must be A* search algorithm because we do not want to be eaten by that lion. Similar to the uniform cost search algorithm, it expands the lowest cost node to find a path which achieves the minimal cost. However, A* search algorithm also considers the estimated cost during the search procedure. Formally speaking, the evaluation function is defined as $f(n) = g(n) + h(n)$ where $g(\cdot)$ is a function which records the total cost spent to n node, and $h(\cdot)$ is a heuristic value function which estimates the minimal cost from n node to the goal. With the shortest path, it saves a lot of unnecessary resources in real life. For example, it saves your fuel while driving to some other places. Or it helps the lion to eat the hunter. It speeds up the process and makes things more effective. Because A* uses admissible heuristic, for which the estimated cost must smaller or equal to the actual cost, it guarantees an optimal solution if it does exist. It effectively finds the shortest path for the application. In the search process, however, the time complexity is exponential to the depth of solution for unbounded search space in the worse case. Besides, it expends all the nodes with a minimal cost which requires exponential space to store all the records. The memory required for a search is not significant. Fortunately, there are some other search algorithms overcomes the time complexity and memory complexity, such as reclusive best-first search.

Do you believe there is a person with the similar ability of Beth Harmon[1] in real life? Beth spends a second to checkmate her opponent in go game, which is crazy if the story is true. This reminds me of an algorithm which speeds up the computation time by cutting unnecessary search branches. That is the second worth mentioning algorithm is alpha-beta pruning for the minimax algorithm. Without checking all the nodes in a search tree, the algorithm eliminates the searching process if there is enough information to draw a conclusion. It reduces the exponent term in time complexity by half. In most cases, however, it is still feasible to search the entire game tree for a depth limit. The algorithm considers all the legal move in a round which has wasted some time on it. In other words, alpha-beta pruning only performs well when the number of subtrees is insignificant. In such a situation, the algorithm can find a better move when time is enough to search the game tree more. In AI, we often search a relative best solution among all the possible solutions to achieve a problem statement. The concept of alpha-beta pruning emphasis the importance to search effectively and efficiently. Although there is no perfect way to find the best solution yet, it provides a precise way to search.

The third one is a stochastic algorithm which makes use of the mechanism of evolution. That is a genetic algorithm (GA). GA has a wide range of application, but it often appears in the optimisation problem because GA is robust to find multiple local optima. GA initiates a population to search through the solution space of an objective function. Individuals are more likely to survive into the next generation if it has a higher fitness score. During the evolution, individuals will experience selection, crossover and mutation to go into the next population. Although the idea of GA is simple, there are imperfect ways to implement GA. Lots of parameters, such as population size and mutation rate, are problem-dependent. GA balance convergence and divergence to improve the efficiency and effectiveness of the algorithm to run through the solution space. Also, GA supports multi-objective optimisation, which finds a solution that can satisfy multiple objective functions. Although similar things can be done by Markov Chain Monte Carlo, it involves much in-depth knowledge of statistic. Having said that, the time complexity for GA varies on the definition of parameters. It can be a disaster to wait for a solution even the convergence and the divergence are balanced, which depends on the problem statement.

The fourth worthwhile mention is a decision tree. In inductive learning, it is a simple but robust learning algorithm to forecast unseen data. It is a popular learning algorithm to deal with a variety of problem, and we can see decision trees everywhere in Kaggle. Because of the tree-like structure, the model complexity is not very complicated, which makes the interpretation crystal clear. Moreover, the use of a decision tree is not limited to the classification problem, but also the regression problem. The node splitting algorithm is slightly different, but the idea is almost the same. A classification tree measures impurity of exploratory variables using entropy, which is formulated as $f(X_i) = \sum_n -\Pr(X_i)\log_2 \Pr(X_i)$, whereas a regression tree minimises the residual sum of squares, which is defined as $f(X_i) = \sum(y - \bar{y}_R)$. Yet, it is easy to overfit a decision tree when we utilise all the variables in train data. Hence, we often rely on the Ockham's Razor principle to prune a tree which tolerates more noise in the fitted model for better accuracy. Although it seems to be a dummy algorithm, it can produce a comparable result to other learning

[1] A character in the Queen's Gambit

algorithms. Furthermore, the state of art for the decision tree is random forest and boosting, which build some amount of trees to predict the response variable. Those state of art methods is much more powerful in prediction.

Inspired by the functionality of the human brain, the last algorithm to discuss is a deep neural network (DNN). There are multiple nodes in DNN connected by links. Each link weights an input from one node to another node(s). Links will be updated during the learning process to improve the accuracy of outputs. Nodes in DNN are are an analogy to neural which aggregate weighted inputs and evaluate activation values by the built-in activation functions. The structure of the neural network is also important in model development since it determines whether the network is a feed-forward network or a recurrent network. DNN is a hot topic all around data science without a doubt because of its wide application and robustness in prediction. For image process, convolutional NN can generalise the structure of items numerically and classify unseen data, just like what we have done in the course project. In statistical learning, moreover, we often encounter regression problems and classification problems. Deep feed-forward NN can achieve the goal by learning from the train data as well. However, DNN requires much subjective knowledge to build a model. In feed-forward NN, for example, the number of nodes and layers are problem-dependent. It requires time to study the best combination to learn the data. For a large number of nodes or layers, computation time and overfitting are the major concern on the model development. Yet, DNN is still popular in the market because of the adaptive application.

b) Demonstrate one of the selected algorithms in depth with one application scenario of this algorithm. (*Please do not use DNN as one of the selections in this question.)

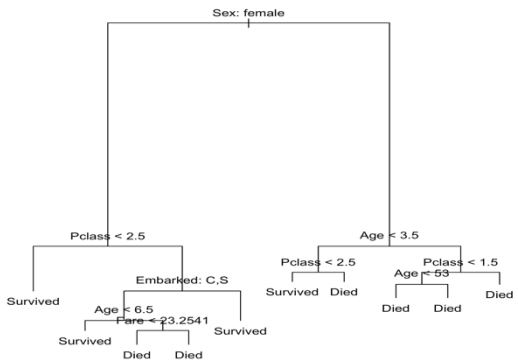
As mentioned, a decision tree is a simple but robust learning algorithm to predict unobserved data. We are going to apply the algorithm with “Titanic” data to predict whether a passenger survived or not when titanic sank after colliding with an iceberg.

PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
Min. : 1.0	Min. :0.000	Min. :1.000	female:185	Min. : 0.75	Min. :0.000	Min. :0.00	Min. : 0.000	: 1
1st Qu.:125.8	1st Qu.:0.000	1st Qu.:2.000	male :315	1st Qu.:20.12	1st Qu.:0.000	1st Qu.:0.00	1st Qu.: 7.925	C: 92
Median :250.5	Median :0.000	Median :3.000		Median :28.00	Median :0.000	Median :0.00	Median : 14.456	Q: 45
Mean :250.5	Mean :0.386	Mean :2.326		Mean :29.20	Mean :0.574	Mean :0.38	Mean : 31.782	S:362
3rd Qu.:375.2	3rd Qu.:1.000	3rd Qu.:3.000		3rd Qu.:37.75	3rd Qu.:1.000	3rd Qu.:0.00	3rd Qu.: 30.071	
Max. :500.0	Max. :1.000	Max. :3.000		Max. :71.00	Max. :8.000	Max. :5.00	Max. :512.329	
				NA's :102				

(Table 1)

The “Titanic” data consists of eight variables, excluding the passenger id. Table 1 summarises the statistics for the available variables. We will omit the details in data cleansing, and directly replace the missing data with 90% trimmed mean. For sure, it is a classification problem which we want to find a passenger who survived in the accident. To access the performance of the classification tree, we divide the dataset into two parts. One will be train data, and the other one will be test data.

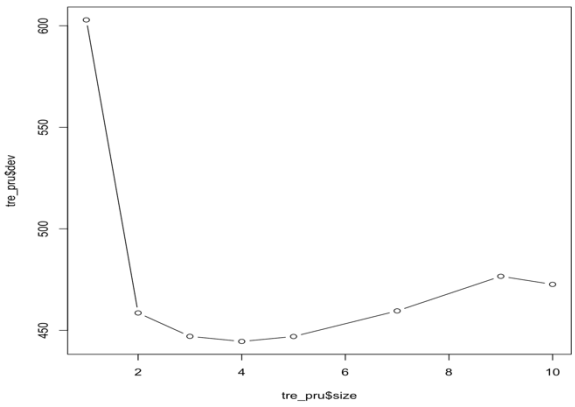
A classification tree computes an entropy for each variable to compare which variables consist of the least impurity. The one with the least value in entropy is the most informative to say whether a passenger did survive in the accident. Entropy is defined as $\sum_n -\Pr(X_i)\log_2\Pr(X_i)$. We usually utilise entropy to calculate the information gain of each variable for determining the splitting node. The higher the gain, the less impurity contains in a variable. More formally, the information gain is defined as $\text{Gain}(A) = initial\ entropy - (weight\ average) \times entropy(A)$, where A is an attribute. Calculating all the information is a tedious process, so we pass the job to software to aid the demonstration. The software suggests “Sex” has the largest values in information gain. Thus, we choose “Sex” as the first splitting node in the classification tree. The process continues until all the variables are used in the tree. Visually, Figure 4 shows how the classification tree looks like. We can easily classify a passenger who survived in the accident are more likely to be women with upper tickle class. However, the software says the prediction accuracy is only 60% from the test data. Because the fitted model consists of too many possible hypotheses, overfitting may happen in the classification tree. In the view of inductive learning, we often rely on Ockham’s Razor principle to avoid overfitting a model. Simple is good enough, which implies more noise in the example to generalise the data pattern.



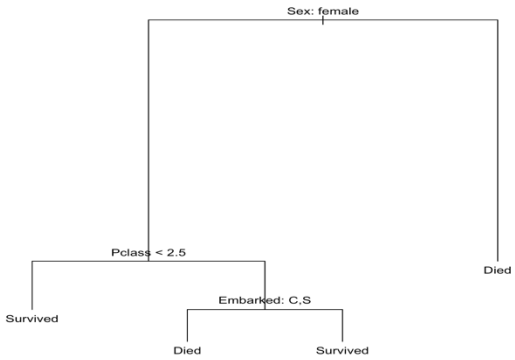
(Figure 4)

With support from software, we can easily find the cross-validation errors of each pruned tree. Cross-validation is a mythology which divides the train data into K folds and computes errors of possible models. Figure 5 illustrates the cross-validation errors at different complexity of trees. We notice a tree model with only four leaf nodes fits the train data best, which achieve minimal cross-validation errors. We, hence, prune the classification tree and leave four leaf nodes in it. Figure 6 describes how the tree looks like after pruned some other branches. Again, the software coincidently reports the same prediction accuracy as the previous tree model. It implies a simpler model already achieve the same results as the complicated one, which matches Ockham’s Razor principle in inductive learning. However, 60% may not be a satisfactory number. It motivates to some more advanced tree-based learning

models, such as random forest and boosting, to improve the prediction accuracy. Yet, those methods are out of scope, so we are not going the discuss further.



(Figure 5)

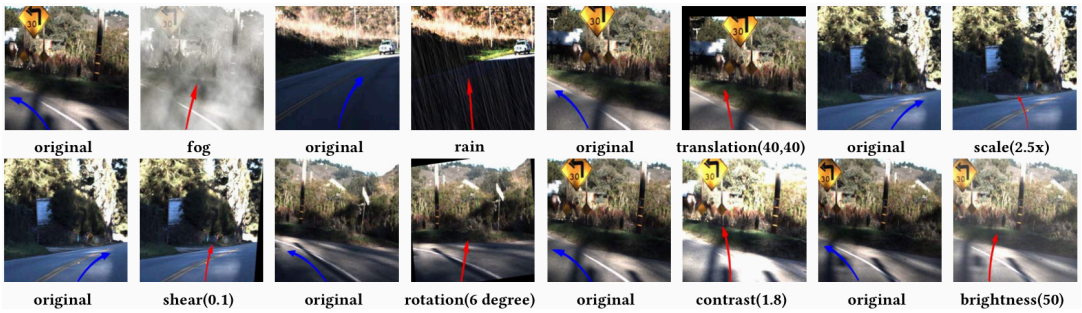


(Figure 6)

Question 3
Suppose you are trying to build the software system for a self-driving car. Describe how to apply the algorithms (multiple) in this course for the agent. (Hints: vision, route planning, decision making etc. by appropriate algorithms.)

When talking about learning algorithms for a self-driving car, it recalls me a YouTube video[2] which describes how a self-driving vehicle learns to park using reinforcement learning. It is therapy when looking at a self-driving car to get in the right place. However, there are a ton of aspects to consider for an automatic car. Choosing the right intelligent agent is also crucial. Otherwise, it may lead to more potential issues, such as AI fighting[3]. Yet, we are not going to dig deep into those topics but how to apply learning algorithms in a self-driving car.

Every agent requires a sensor to receive percept from the real world. Usually, self-driving cars use camera images as input to predict the next action on the road. The images will be processed by a trained convolutional neural network (CNN) model and determine the steering angle as a control to the vehicle (K. Jelena, 2019). CNN is robust in recognising images by finding unique features for a particular item. In self-driving cars, the built-in CNN model extracts new images in real-time to control over the vehicle. The training process of CNN being important at this stage since the output depends on how to fit the model are. The ideal way to feed a model is giving all the possible situation to it. However, it is impossible because the environment is dynamic, and the space of situation is infinite. Hence, it is more often to let a model learn the pattern of situations, so that is can apply to a verity of situations. Up to this stage, it is still ideal to feed a model with lots of examples. Data augmentation is a methodology to deal with insufficient train data. The idea of data augmentation converts examples by changing the scale of them. It can also be understood as introducing noise into the train data, but the converted train data is expected to have the same output. This enlarges the size of the train data to feed a model. Yet, a CNN model may fail to learn those data and lead to potentially fatal accidents. Tian and her team implemented a DeepTest on recognising streets and determining steering angle in 2018. Tian found out that a DNN model behaved erratically when facing synthetic images. The blue arrows in Figure 7 refer to the correct steering angle, while arrows in red colour indicating a working wrong steering angle in the converted data. Self-driving cars heavily rely on the decision made by the built-in CNN. If CNN incorrectly predicts a direct, it may lead to terrible collisions. As safety places in the first place of automatic cars, the fitness of built-in CNN should be almost perfectly developed.



(Figure 7)

Instead of remote a vehicle, self-driving cars should also provide the best and the quickest way to reach the destination. An effective route planning can substantially save driving costs. To achieve the goal, A* search algorithm helps to find the shortest path. Unlike other algorithms searching all the possible path, A* algorithm only search for a path with the lowest cost to reach the destination. In addition, A* algorithm considers heuristic value, as known as estimated cost to the target address, when finding a route plan. Since only paths with minimal costs will be reached, A* algorithm guarantees an optimal solution which is the shortest path with minimal costs. Yet, it is not enough if self-driving cars only propose the shortest path. Passengers in the self-driving cars should enjoy their time free from driving. Szczerba suggests a robust route path should take account of the real-time situation by considering some constraints in the learning algorithm. A path with minimum route leg length restricts automatic cars to have some straight distance before taking a steering angle, which avoids unnecessary turns that makes passengers feel sick. Similarly, the number of steering angles should have an upper limit which prevents collision with other cars. The length of the path should be less than or equal to the maximum distance that a self-driving car can go to. This ensures the road trip will be a place but not a middle of nowhere. The moving direction should stick into the direction of the destination, which reduces the risk to go the long way round (2000). With the above constraints, self-driving cars can make occupant more comfortable

[2] AI Learns to Park: https://youtu.be/VMp6pq6_QjI
[3] Two AI Figth for the same Parking Spot: https://youtu.be/CqYKhbyHFtA

during the trips with the shortest path. Yet, Szczerba recommends Sparse A* Search, an extension of A* algorithm, which we did go through in the course, and we stop the discussion here.

Would you buy a self-driving car just because of its auto control and well-planned route? Self-driving cars can be even more functionality in term of user experience. Imagine you were in a self-driving car, what would you like to do in the car? If self-driving cars build in with a decision tree model, it can predict what passengers want to do in the travelling. A decision tree model is simple but robust in forecasting the corresponding action. For example, an automatic car with built-in decision tree trained with passenger music preference. The vehicle plays music according to the weather, the distance of a journey, or some other factors. Moreover, there can be a driving assistant which built-in with a deep neural network to talk with passengers. Natural language processing is an application of deep neural network which recognises human language and provide some response like a human being. It is a good idea to kill time with the assistant. Learning algorithms not only operate a self-driving car but also improve the quality of driving by providing a variety of functions to entertain passengers in the travel.

Reference

BM. Kim, YB. Kim, CH. Oh (1997). *A Study on the Convergence of Genetic Algorithms*. ScienceDirect. https://www.sciencedirect.com/science/article/pii/S0360835297001988

K. Jelena, J. Nenad, D. Vujo (2019). *An End-to-End Deep Neural Network for Autonomous Driving Designed for Embedded Automotive Platforms*. MDPI. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6539483/#!po=11.4583

M. Bhattacharya (2014). *Diversity Handling Evolutionary Landscape*. arXiv. https://arxiv.org/pdf/1411.4148.pdf

R.J. Szczerba, P. Galkowski, I.S. Glickstein, N. Ternullo (2000). *Robust Algorithm for Real-Time Route Planning*. IEEE. https://www3.nd.edu/~cpoellab/teaching/cse40463/rconnaug1.pdf

T. Gabor, L. Belzner, T. Phan, K. Schmid (2018). *Preparing for the Unexpected: Diversity Improves Planning Resilience Evolutionary Algorithms*. arXiv. https://arxiv.org/pdf/1810.12483.pdf

Y. Tian, S. Jana, K. Pei, B. Ray (2018). *DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars*. IEEE. https://dl.acm.org/doi/pdf/10.1145/3180155.3180220

Z. Hu (2013). *Sufficient Conditions for Global Convergence of Differential Evolution Algorithm*. Hindawi. https://www.hindawi.com/journals/jam/2013/193196/

Appendix

Source code for Question 2b

```
titanic = read.csv("Titanic.csv", header = TRUE)
summary(titanic)

titanic[which(is.na(titanic$Age)), "Age"] = ifelse(titanic$Pclass == 1,
round(mean(na.omit(titanic[which(titanic$Pclass == 1), "Age"]), trim = 0.1)),
ifelse(titanic$Pclass == 2, round(mean(na.omit(titanic[which(titanic$Pclass == 2), "Age"]),
trim = 0.1)), round(mean(na.omit(titanic[which(titanic$Pclass == 3), "Age"]), trim =
0.1))))[which(is.na(titanic$Age))]
titanic[which(titanic$Embarked == ""), "Embarked"] = "S"
titanic$Embarked = as.factor(as.character(titanic$Embarked))
titanic$Sex = as.factor(titanic$Sex)
titanic$Survived = factor(ifelse(titanic$Survived == 1, "Survived", "Died"))
titanic = titanic[-1]

index = sample(1:nrow(titanic), nrow(titanic) * 0.1)
train = rep(T, nrow(titanic))
train[index] = F
test = !train

library(tree)
tre_cla = tree(Survived ~ ., data = titanic, subset = train)

plot(tre_cla)
text(tre_cla, pretty = 0)

tre_pro = predict(tre_cla, newdata = titanic[test, ])
tre_pre = rep("Died", nrow(tre_pro))
tre_pre[tre_pro[2] > 0.5] = "Survived"
mean(tre_pre == titanic$Survived[test])

tre_pru = cv.tree(tre_cla)
plot(tre_pru$size, tre_pru$dev, type = "b")

tre_cla = prune.tree(tre_cla, best = 4)

tre_pro = predict(tre_cla, newdata = titanic[test, ])
tre_pre = rep("Died", nrow(tre_pro))
tre_pre[tre_pro[2] > 0.5] = "Survived"
mean(tre_pre == titanic$Survived[test])
```