CSCI3230 Fundamentals of Artificial Intelligence – Term Paper
Jonathan Lam Pui Hong (1155093597)

QUESTION ONE

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

Genetic algorithm (GA) is a stochastic group search for the global optimum. It usually starts from a multi-point and randomly generated set of population, which is called chromosomes, that evolve over multiple generations by genetic operations to search for the optima. These genetic operations, such as crossover, mutation, selection, and fitness (evaluate) function, ensure the chromosomes to evolve and compete with one another in multiple generations. Through the survival of the fittest, chromosomes with favourable traits, i.e., those obtained the highest value in the fitness function, are chosen to produce offspring that will eventually reach the problem's local or global maxima.

With the idea of the best solution (the fittest chromosome) in mind, convergence is the utmost important feature of a genetic algorithm. Convergence describes the process of individual chromosomes, by competition and selection, evolve and *converge* to reach a maximum – the desired output of the algorithm. As the fittest chromosomes are favoured to be proportionately selected to carry out genetic operations, known as the selection pressure, it is more likely their traits are mixed with other fit chromosomes, mutated to different offspring chromosomes, and carried through onto the next generation. GA will continue to execute until either a minimal threshold (desired quality of the solution) is reached or, after a set amount of generation, the chromosomes are converged to a point where further computation is not beneficial.

The convergence of GA ensures the chromosomes reach local maxima, but it does not guarantee to reach a global maximum. If the chromosomes converge too quickly, the algorithm will be skewed to converge to a few local maximum and trapped into a region where the global optimum is absent. This is called premature convergence [1].

Divergence then comes into play to resolve premature convergence. Divergence describes the process of having a diversified set of chromosomes in each generation, especially in the initial population. It ensures chromosomes are dispersed within the search space so global optimum is included when they converge. While genetic operations such as crossover already promote diversity, it may eventually fallout in later generations where structurally similar chromosomes are being operated. In these cases, some other methods such as reinitialization can ensure the search space is covered [1]. Furthermore, if GA treats diversity as a component of fitness, with enough chromosomes populate all the local maxima sufficiently, the still peripatetic chromosomes are then driven off by those already discovered local maxima to search after undiscovered a global maximum [2]. The level of diversity therefore drives a better performance goal towards finding a global maximum.

The diversity mentioned above is *phenotype diversity*, which counts the number of unique fitness values in a population, and *entropy diversity*, which is calculated by the proportion of the population partition against the whole population [3]. In general, these two enforces the selection pressure in different stages that distinguish between better performing chromosomes and the lesser ones [3]. Meanwhile, a higher *genetic diversity* of the initial set generally means a higher probability for each chromosome scattered across different polarities of the search space to compete with each other, producing a more varied and diversified environment [4].

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

Intuitively, a higher degree of diversity (to a certain generation into the algorithm) means a better performance of the GA. This is not necessary for the cases in all situations. There may be a correlation between diversity and performance, but there is not a direct infer causation between the two, particularly when there are multiple diversity measures that GA can utilize and different problem spaces to be searched [3]. Following are a few examples to illustrate the convergence and divergence of GA from [3] and [4].

In [3], three problems domains are being tackled: 1) the artificial ant problem, a classical evolutionary problem where ants find the best route to get food along the trail for survival; 2) the regression problems, where GA finds the best fit to the quadric and Rastrigin function; and 3) the parity problem, where GA finds the correct number of even 1s.

We start by looking into the convergence of these problems. In all of the problems of [3], through mere crossover (without mutation) and tournament selection, the best fitness value of each problem decreases and almost put to a stop in about 15-20 generations, with the exception of the parity problem continually decreasing. As the fitness function is evaluating with minimum as the best fitness value (e.g., in the ant problem, the fitness of a chromosome is measured as the number of food pellets missed), the model is continually improving. This means the GAs are converging towards their local/global optimum and that the fitness functions are evaluating the performance of chromosomes adequately for such convergence. The convergence rate towards a *critical point,* the "run-length beyond which it seems irrational to plan to do runs", as stated by Luke [5], is also dependent on the problems. While GA has the ability to continually improve on the mean fitness, the variance may stay similar (at high, which means local optima is close). Thus, it makes more sense to run *fewer* runs rather than pressing forward for more generations in problems with diminishing returns, and more runs in difficult problems as the parity problem.

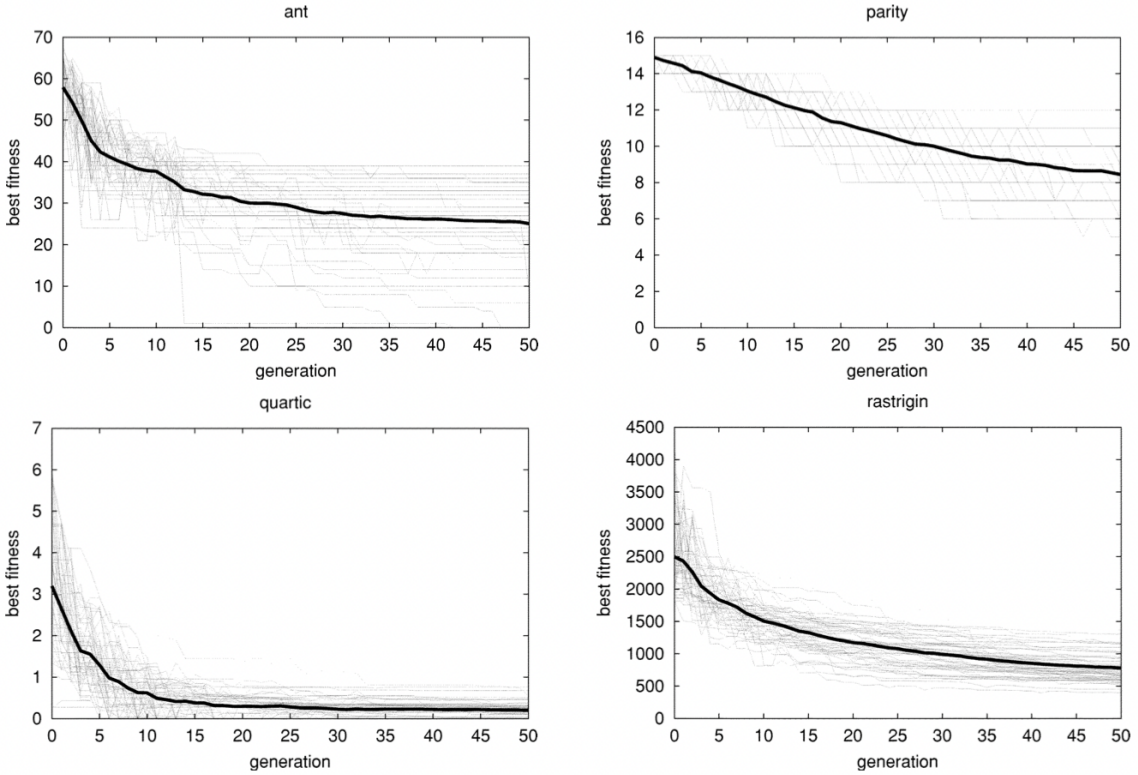

Fig. 1. Ant, parity, quartic, and Rastrigin best fitness per population, plotted against the generation number. Fifty independently random runs of each problem are shown.

Not only the uniqueness of each problem, hence their search space, affects the convergence of GA, it also affects the strength of correlation between diversity and fitness. As with the above section, we focus on the phenotype and entropy diversity. From Fig. 2 and 3 we can see there is a steady increase of diversity on the parity problem, a lesser extent of increase on the ant problem, and a not-so-notable difference of diversity change on the regression problems. Comparing this to Fig. 1, it is noticeable that low (good) fitness is correlated with high diversity. The extend of increase is dependent on the problem's *critical point*. This is because in later generations, the constant genetic operations, which increase diversity, do minimal effort in improving the fitness value as the critical point has passed [3]. In cases such as the parity problem where only 32 unique fitness values (for 5-bit-parity) exist, diversity comes into play to distinguish between good and bad fitness value, thus higher diversity still becomes impactful in later generations on getting better fitness.

The above already illustrates the importance of convergence and divergence in correlation to thes fitness of chromosomes and, ultimately, the success of GA. There is still one more thing to address is that correlation is not causation. Following will use an example from [4], in which the more complex form of 16x16 Iterated Prisoner Dilemma (IPD), where chromosomes find the best way to cooperate or denies of the other participant to a certain degree, is tackled.

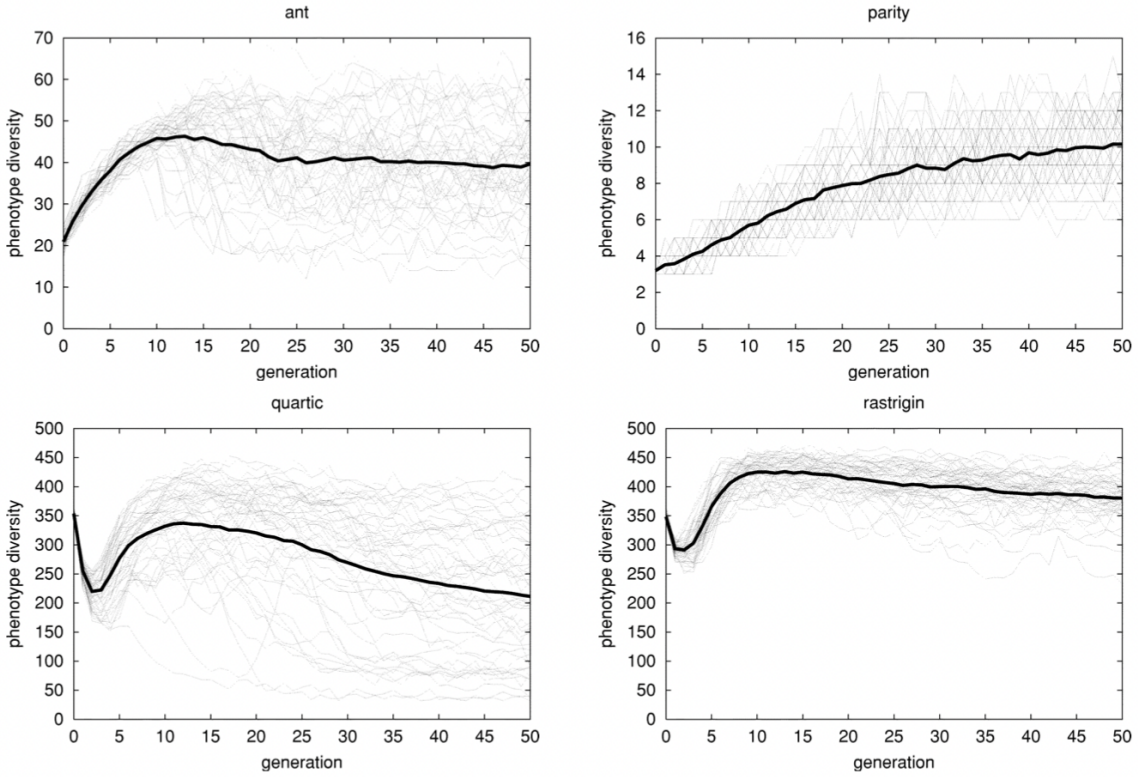

Fig. 2. Ant, parity, quartic, and Rastrigin phenotype diversity, plotted against the generation number. Fifty independently random runs of each problem are shown.

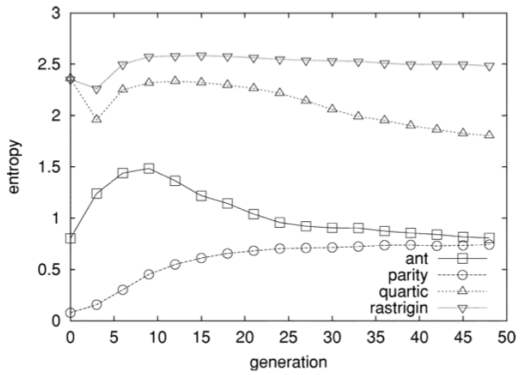

Fig. 3. Average of 50 runs of the entropy measures for ant, parity, quartic, and Rastrigin.

Consistent with the above observation, Darwen and Yao discovered that higher genetic diversity in co-evolutionary learning of GA produces better learning and convergence towards an optimum, signifying a possible increase of phenotype diversity. Upon further investigation, as with the example above, the diversity that produces better learning is highly dependent on the problem/search space. In IPD, a higher genetic diversity merely shifts the initial population choices the chromosomes make [Fig. 4]. Despite the evident success in improving fitness, it has resulted from a biased initial choice; genetic diversity, in *this* particular problem space, results in *lower* behavioral and phenotype diversity.

In conclusion, convergence and divergence play significant roles in genetic algorithm. As illustrated above, it is essential to note that the effect of the two are not consistent in all problem/search spaces and different approaches and applications are needed to be configured to tackle individual problems.

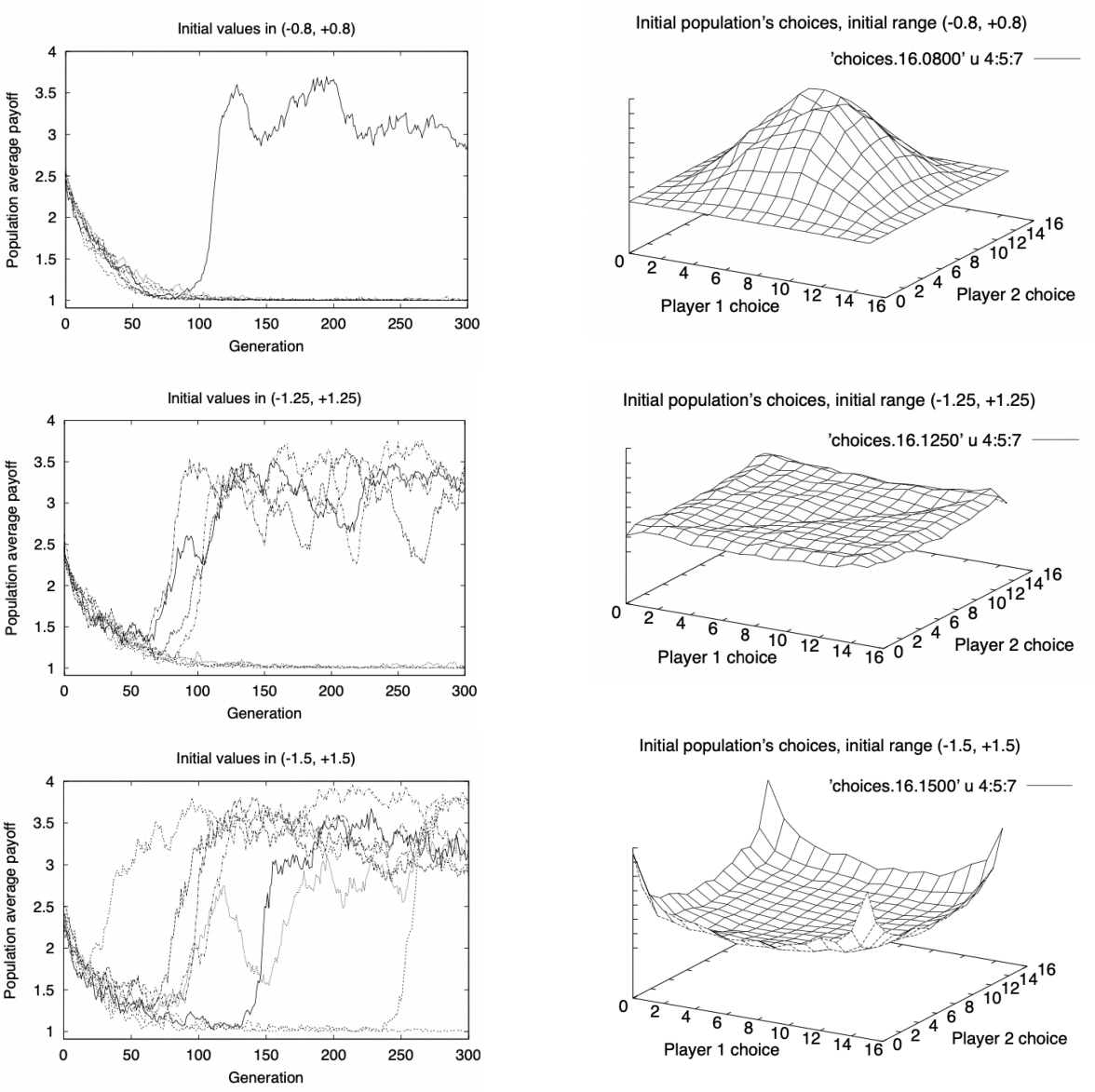

Fig.4. IPD runs with respective low, moderate, and high initial genetic diversity.
(Left column) Ten co-evolutionary runs of IPD with 16 choices. A higher payoff means a higher degree of mutual cooperation in IPD. In high initial values, almost all discover some degree of mutual cooperation.
(Right column) The distribution of outcomes in the 16x16 matrix. The diagonal symmetry is an artifact. In low and high diversity, only the center region and corners are explored respectively.

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

Aside from DNN, the chosen algorithms are 1) Iterative deepening searching algorithm, 2) Recursive best first search, 3) Genetic algorithm, 4) Minimax with alpha-beta pruning, and 5) Decision tree algorithm.

1. Iterative Deepening Searching Algorithm (IDS)

Among all uninformed search strategies, Iterative Deepening Search (or Iterative Deepening Depth-first Search, IDDFS) combined all the advantages of other searching algorithms. It dissects graphs in an increasing number of layer, and perform Depth-first Search (DFS) with that subtree until a subtree with a solution is found – basically, it is performing DFS with Breath-first Search (BFS). Combining the two, IDS is optimal and complete just as BFS and has a linear space *bd* similar to that of DFS, where *b* is the branching factor and *d* is the solution depth. It fixes the exponential space complexity b^d that BFS requires and the non-guaranteed optimality and completeness in DFS, all while maintaining a time complexity b^d. Therefore, it can deal with deep or infinite trees without little concern of space, and perform searches in graphs where the depth of solution is unknown (in which depth-limited search fails).

The trade-off of IDS is that, since it is a deepening search, each loop will evaluate the top level of nodes again. That means a bit of redundancy exists per loop to go over the searched nodes, wasting computational time and calculation which results in a longer searching time comparing to BFS or DFS. The nature of IDS being the combination of BFS and DFS also signifies that loops or cycles, while solvable by the algorithm (unlike DFS), are not specifically detected and conquered; in the tree/graph that these occurs, IDS will go through multiple repeated nodes as intended, again increasing the time needed.

IDS is particularly useful in places like game solution finding where time and space are usually fixed and limited. It can search for a current best solution within a set amount of turns (depth) within the time constrain.

2. Recursive Best First Search (RBFS)

Among all informed search strategies, A* search is an improvement on greedy best-first search in terms of completeness, optimality, and time. By using a cumulative cost *g(n)* and an admissible heuristic *h(n)* to form the evaluation function, it never expands any nodes with *f(n)* larger than optimal path cost and, as a result, A* effectively goes towards the goal while keeping the cost optimal. However, both greedy search and A* leaves one issue unsolved, which is the space requirement. All nodes must be kept in memory in order for the algorithm to transverse the tree with the second best choice.

In light of this, Recursive Best First Search comes into play. It preserves everything good mentioned in A* searching, but keeps the memory complexity at *bd*. It works by storing a best-alternative value, the *f-limit*, on each parent node of each layer; if the children have a larger value than *f-limit,* this value is prone to be updated, the subtree is "forgotten", and unwinding occurs to search for the alternate route. Hence, only one node is expanded at one depth, drastically reduce the space requirement.

Intuitively, as RBFS "forgets" a subtree to save space, it also means that upon any larger value than *f-limit* in the alternate route *R*, it will have to "regenerate" the subtree that was discarded (as it is the alternative to the once-alternate route *R*). There is also a sorting process and local *f*-limit value storage in place that was absent in A* [6]. This inevitable trade-off costs more time for the algorithm if constant revisiting occurs.

3. Genetic Algorithm (GA)

While A* and RBFS are some of the deterministic, single-point searching algorithms, Genetic Algorithm is a stochastic, multi-point searching algorithm. It is beneficial in problems that require searching for a global optimum where hill-climbing and single-point algorithms usually fail (landing on sub-optimal or local optimum). Based on the natural evolution of species where mating (crossover), mutation, and survival of fittest (elitism) occurs, the GA (the chromosomes) are always subject to change and improvement, thus it is crucial in works that require modelling and predicting the behaviour of species or machines.

The freedom and stochastic nature that benefits GA also hinders the application of them. There are many hyper-parameters in GA that can be changed, such as the number of chromosomes, mating occurrence, mutation rate, repetitiveness, and elitism. Above all, evaluation methods, for example the fitness function, are of the most important and the hardest to configure, as a good fitness function will determine a model's ability to converge or diverge to a certain extent and, ultimately, determine the success of GA of finding the global optimum. That said, GA's ability to self-evolve could allow the possible configuration of unsupervised learning. In the words of Darwin and Yao, "if the famous artist Picasso was available, and you wanted a nice work of art, would you rather have Picasso write a fitness function for evolving art works, or instead just let him paint paintings directly [4, p.3]?" Precisely of its continuous improvement, *co*-evolution in evolutionary algorithm (in which GA is a sub-class) may be a way to overcome the disadvantage of setting an adequate fitness function.

4. Minimax (with alpha-beta pruning)

When it comes to devising a solution for games, due to its usual large branching factor and time constrain, the algorithm has to adapt accordingly to contain and search such a large space within a limited time. Minimax works particularly well in perfect-information and deterministic, i.e., players take turns to move, games. It works by predicting the opponent's move if a move is taken, and returns a move with the highest payoff; in other words, it tries to

maximize the player's payoff while minimizing the opponent's. With the help of cut-off (depth-limited) search and alpha-beta pruning, where the algorithm ignores and eliminates the branches that will not be considered by the opponent and itself, minimax can find an optimal solution effectively.

The disadvantage of minimax is, foremost, the space and time requirement for the searching. Despite the help of cut-off and pruning, it still has to tackle a large number of choices and, therefore, usually, only a few layers are searched for an approximate, local optimal move. The second disadvantage is coming up with a good evaluation function to access the utility value of each state. A poor evaluation function may incorrectly access the situation and return undesirable moves. Moreover, minimax works by predicting a move by the opponent and assuming the opponent chooses the same, most beneficial move to them. If any unpredicted moves are made by the opponent, it may negatively affect the effectiveness of the algorithm.

5. Decision Tree Algorithm

In the cases that require logic decision and binary classification, decision tree algorithm is selected for its simplicity and intuitiveness. Given a set of data, the algorithm decides what attributes are essential to classify the data after a series of decisions. It usually just take entropy or information gain of each decision into account to find an optimal solution for the set of data, hence it is easy to implement while effectively solving everyday problems.

However, it has several downsides. Decision tree algorithm is only useful in discrete data and are unable to handle continuous data, so data prepossessing is needed for such conversion. Not only the data need to be large enough for the algorithm to pick up traits, but it is also subjected to noise which leads to overfitting. Careful pruning, approximations, generalizations, and specializations are therefore needed to devise a mere probably approximately correct solution.

b. Demonstrate one of the selected algorithms in depth with one application scenario of this algorithm.

Following is a demonstration of the decision tree algorithm using examples from [7], represented in Tables I and II. Given a list of weather attributes, the aim of the algorithm is to find a decision tree that decides whether one will enjoy sport or will play tennis.

Ex.	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes
*	Sunny	Warm	Normal	Weak	Warm	Same	No

Table I. *EnjoySport* dataset from [7, p.21] with additional example (*) from [7, p.78].

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Table II. *PlayTennis* dataset from [7, p.59].

To begin the algorithm, a list of information gain is calculated for every attribute. This gain, the difference between the original information requirement and the new requirement, tells us how much more information is needed after the test of attribute A.

$$Gain(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$$

where

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

The largest gain, i.e., the most information *gained*, will be the first node in the decision tree. This process is repeated until all the data is categorized, i.e., one of the *remainder* (summation) part of all the *Gain(A)* reaches 0. One of the calculations, the gain of humidity after *Sky is Sunny* is chosen, is demonstrated below. The information gain for the two datasets is shown in Tables III and IV.

$$Gain(Humidity \mid Sky_{Sunny}) = I\left(\frac{3}{4}, \frac{1}{4}\right) - \left(\frac{2}{4} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{4} I\left(\frac{2}{2}, \frac{0}{2}\right)\right)$$

$$= -.75\log_2.75 - .25\log_2.25 - (.5(-.5\log_2.5 - .5\log_2.5) + .5(0))$$

$$\approx .811278 - .5 \approx .31128 \text{ bits}$$

	Sky	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*
Gain	.81128^	.81128^	.12256	0	.12256	.31128
Gain with *	.32193	.32193	.01997	.32193	.17095	.01997
	Sunny	0	.31128	.81128^	.12256	.12256
	Rainy^	-	-	-	-	-

Table III. Information gain for *EnjoySport* dataset. ^ denotes the remainder reaches 0.

	Outlook	*Temperature*	*Humidity*	*Wind*
Gain	.24675	.02922	.15184	.04813
	Sunny	.57095	.97095^	.01997
	Overcast^	-	-	-
	Rain	.01997	.01997	.97095^

Table IV. Information gain for *PlayTennis* dataset. ^ denotes the remainder reaches 0.

For *EnjoySport* without the new example (*), the decision tree simply has one node: if *Sky is Sunny,* then *EnjoySport is Yes.* If the new example is included, the first attribute is *Sky* and followed by *Wind* in the branch of Sky_{Sunny} (Fig. 5). For *PlayTennis*, the decision tree first selects *Outlook*, then *Humidity* is selected on the branch of $Outlook_{Sunny}$, and *Wind* is selected on the branch of $Outlook_{Rain}$ (Fig. 6).

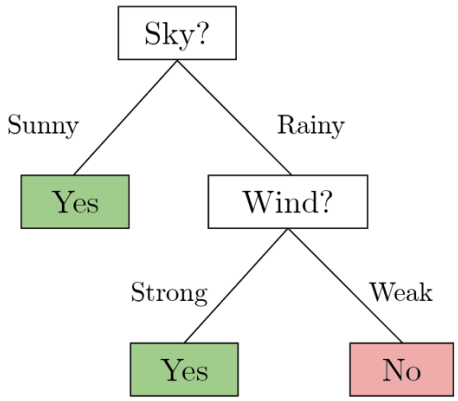

Fig. 5 Decision tree for *EnjoySport* with the new example (*).

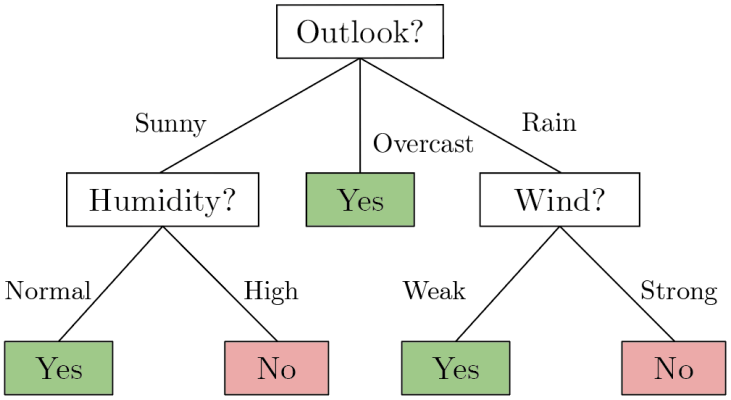

Fig. 6 Decision tree for *PlayTennis*.

One can notice in the initial *EnjoySport* dataset with 4 entries, the lack of data causes underfitting which incorrectly predicts the new example. It shows the algorithm requires some amount of data to find patterns within. This is true in the *PlayTennis* dataset: even if the last few entries are removed and a new hypothesis is formed by the algorithm, the new one will still be able to correctly categorize the data. This signifies an adequate amount of data is provided for the algorithm to train the model.

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.

To reach from one destination to the next, a self-driving car should have the ability to plan the most efficient route from point *A* to point *B*. It should also be able to observe the environment at any given point en route and make suitable adjustments to controls and routes. In summary, a self-driving car should be equipped with 1) High-Level Path Planning, 2) Perception and Localization, 3) Behavior Arbitration, and 4) Motion Controllers [8]. Based on these four main components, the following will apply the algorithms learnt for the agent.

1. High-Level Path Planning

To compute an efficient route from a start position to a target destination, since the estimated distances between destinations are already known and all route costs are given, an informed deterministic search such as RBFS (with A*) can be implemented. Ideally, with the aid of data from cooperations like Google Map API, the route costs are subject to update when routes are crowded or less-populated according to time and events, for example on morning and evening hours where people go on and off work; otherwise, all of which can also be learnt by constant data gathering and processing from the learning agent in the car itself. Since RBFS are implemented, segmentation of routes are also recommended lest the car went astray from the intended route and the best alternative is required on impulse to lead the car back on track, all while making sure the remaining part of the route intact for the driver (and lessen the computational time of the agent to find a completely new route).

2. Perception and Localization

There are a lot of obstacles and road conditions, including pedestrians, cars, traffic lights, turns, and road labelling, that affects the decision of an automated car. A good perception of the environment is essential for the car to access its current condition and make adjustments. When driving, radars and cameras should be used to capture the signal from the surroundings and feed into a trained network to recognize these symbols on the road. To process 2D photos (or even combined 3D images) from the sensors, a convolutional neural network (CNN) is beneficial to concatenate the images and learn their features in advance, in order to highlight different kinds of obstacles and bound them in collision boxes on the go to prevent possible crashing. Where image collection and object detection fails, such as the issue of weather conditions and lighting problems, radars should work together and act as an additional input to the neural network to compute a map that simulates the environment. CNN could also be used for this multi-dimension implementation.

3. Behaviour Arbitration (or low-level path planning) and Motion Controllers

While high-level path planning plans a route from point *A* to *B*, there are things like overtaking, giving way, merging, taking turns, and navigating unstructured urban roadways that require an on-the-spot decision and short-term path planning with other agents (vehicles) in mind [8]. Aside from DNN, to learn the behaviour of a self-driving car and other agents, genetic algorithm is useful in taking many variables (such as safety, communication with other agents, opportunity to make moves etc.) into account to continuously improve in producing the globally optimal solution. The agent can be trained in advance in simulators where sets of conditions and evaluation functions are given; through generations, with the combination of the perception data mentioned above, it can learn to adjust motion controls (i.e., the terminal sets in GA), simulate real-life scenario, and hence make suitable arbitration on the road. (That said, there are other more suitable algorithms such as Imitation Learning and Deep Reinforcement Learning (DRL) that would be much more malleable and reliable in migrating from a virtual environment to real-life [8]. In fact, the trajectories that GA produces are highly dependent on the state of the car and are difficult to applicate on versions of cars in various conditions [9]. However, GA does show promising results in predicting and performing standard trajectories such as turning and navigating through complex courses [9]. Details are not discussed here.)

REFERENCE

[1] C. Garcia-Martinez, F. J. Rodriguez, and M. Lozano, "Genetic Algorithms," in *Handbook of Heuristics*, R. Marti, P.M. Pardalos, and M.G.C. Resende, Eds. Cham: Springer International Publishing, 2018, pp. 431-464.

[2] K.S. Leung, Class Lecture, Topic: "Informed and Stochastic Searching Algorithms." CSCI3230, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, 2020.

[3] E. Burke, S. Gustafson, and G. Kendall, "Diversity in Genetic Programming: An Analysis of Measures and Correlation With Fitness," *IEEE Transactions on Evolutionary Computation,* vol 8, pp. 47-62, March 2004. DOI: 10.1109/TEVC.2003.819263.

[4] P. Darwen and X. Yao, "Does Extra Genetic Diversity Maintain Escalation in a Co-Evolutionary Arms Race," *International Journal of Knowledge-Based Intelligent Engineering Systems*, vol. 4, pp. 191-200, 2000.

[5] S. Luke, "When short runs beat long runs," in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, L. Spector et al., Eds. CA: San Francisco, 2001, pp. 74–80.

[6] M. Hatem, S. Kiesel, and W. Ruml, "Recursive Best-First Search with Bounded Overhead," in *Proceedings of the 29th AAAI Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 1151-1157.

[7] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[8] S. Grigorescu et al., "A Survey of Deep Learning Techniques for Autonomous Driving," *Journal of Field Robotics,* vol. 37, October 2009. DOI: 10.1002/rob.21918.

[9] Y. Saez et al., "Driving Cars by Means of Genetic Algorithms," in *Parallel Problem Solving from Nature*, G. Rudolph et al., Eds. Berlin Heidelberg: Springer-Verlag, 2008, pp. 1101-1110. DOI: 10.1007/978-3-540-87700-4_109.