
SCHOOL OF ENGINEERING AND TECHNOLOGY**ASSESSMENT FOR THE MASTER OF DATA SCIENCE**


SUBJECT CODE AND TITLE:		MAI5013 ARTIFICIAL INTELLIGENCE
ASSESSMENT DUE DATE:		20/06/2024
NO.	STUDENT ID	STUDENT NAME
1	14086334	KAN JUN FAI

IMPORTANT

The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorised late submission of work. Coursework submitted after the deadline will be subjected to the prevailing academic regulations. Please check your respective programme handbook.

Academic Honesty Acknowledgement

"I **KAN JUN FAI** (name) verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realise the penalties (*refer to the student handbook and undergraduate programme handbook*) for any kind of copying or collaboration."

.......... (Student' Signature / Initial)

Introduction

Customs clearance of goods into and out of the country is an essential process of trading between international borders. The clearance process is important to facilitate efficient movement of goods between countries while complying with strict international & local regulations in order to safeguard the health and safety of the countries' citizens. A smooth and effective customs clearance process boosts the economy of countries by reducing the likelihood of delay in supply chain, facilitating logistical processes, and increases the efficiency of international trades, leading to increased economic growth. Moreover, proper documentation of customs duties and taxes ensures timely collection of revenue stream towards the country's government that further supplements economic growth.

Problem Definition

Based on the literature by Hoffman and colleagues, customs clearance delays in the Sub-Saharan African region were mainly due to several factors: more than 90% of supposedly low-risk shipments were stopped and subjected to inspections of shipments without proper justification, resulting in preventable customs clearance delay due to inefficient risk management systems. Secondly, another reason for customs delay is unnecessary requests for further documentation from the shipper or importers when the initially provided documents had sufficient information, resulting in avoidable clearance delays. Thirdly, there were inconsistencies in document processing procedures, resulting in varied performance across different employees of different offices, further adding onto the volume of unjustified inspections of shipments that ultimately leads to higher volume of delayed shipments that unnecessarily require additional processing time for customs clearance (Hoffman et al., 2018).

Due to an already overwhelming amount of shipments being handled at customs, there is a need for standardization and optimization of customs clearance processes to enhance the efficiency of the customs clearance process and ultimately minimize clearance delays. Proper risk management standard operating procedures (SOPs) need to be established by having justified inspections of shipments according to newly determined risk levels parameters,

thereby allocating more manpower to focus on shipments with truly higher risk, while also increasing the clearance efficiency of shipments with lower risk. Accessibility to clear documentation preparation guidelines need to be readily available to shippers and importers in order to allow proactive ample preparation time of required documents prior to customs clearance of shipments to smoothen the clearance process. The customs clearance department needs to achieve justified processing time commitment to minimize clearance delays, while also complying with local & international compliance & regulations.

Selection and Justification of algorithms

A* search algorithm and Genetic algorithm are selected as the search algorithm and evolutionary algorithm respectively and to optimize customs clearance processes. Based on the problem statement discussed above, the variables to optimize within the customs clearance process are shipment risk levels, document completeness levels, and processing time.

A* search algorithm can optimize the customs clearance process by designing an efficient heuristic to identify the risk level of shipments, determine document completeness levels, and evaluate the processing time cost accordingly. This heuristic is also flexible and can be tweaked to suit different conditions given the complexity of customs compliance and regulations. By generating an effective heuristic that generates efficient pathing – from shipments with lower risk and higher document completeness, to shipments with higher risk and lower document completeness – the algorithm determines solutions that consistently have the least cost to reach the goal, providing optimal and consistent solutions to streamline the decision-making processes. Applying A* search algorithm in the customs clearance process will bring benefits by optimizing the inspection sequence by determining each shipments' risk levels and document completeness, providing solutions that minimize shipment clearance times through selected justified inspections, ultimately reducing clearance delays.

Genetic algorithm is selected to suit the customs clearance process, where the algorithm considers multiple variables that resembles real world customs processes, simulates every possible solution, and generate optimization solutions accordingly. Genetic algorithm is known for its uniqueness in generating solutions using its mutation, crossover, and selection mechanics. By crossing over multiple important features, the genetic algorithm can simulate different scenarios simultaneously to come up with solutions that are most optimized to reduce customs clearance delays. The genetic algorithm can also evolve solutions over time through its mutation feature, which is highly advantageous in the customs clearance sector that involves everchanging compliance and regulations. Genetic algorithms can provide benefits in the customs clearance sector by considering all input features, and schedule operating procedures

that optimizes the efficiency of customs clearance process by minimizing shipment processing time.

Ultimately, the selected algorithms of A* search algorithm and genetic algorithm may potentially minimize the clearance time by completing lower risk shipments with higher document completeness levels, freeing up significantly more time and manpower to focus on higher risk shipments that require more attention and decision-making time.

Algorithm Implementation

For both the A* search algorithm and Genetic algorithm, a generic code was used to synthetically generate a dataset of 100 shipments in order to test its performance and effectiveness respectively. In the code used to generate the synthetic dataset, a function that generates “risk level”, “documentation completeness”, and “processing time (hours)” was used. Both A* search algorithm and Genetic algorithm were tested on the same synthetic dataset.

```
import numpy as np

# Generate synthetic shipment data
def load_shipments_data(num_shipments=100):
    np.random.seed(42) # For reproducibility
    shipments = []
    for i in range(num_shipments):
        shipment = {
            'id': i,
            'risk_level': np.random.rand(), # Random risk Level between 0 and 1
            'documentation_completeness': np.random.rand(), # Random completeness between 0 and 1
            'processing_time': np.random.randint(1, 10) # Random processing time between 1 and 10 hours
        }
        shipments.append(shipment)
    return shipments
```

Figure: Code used to generate the synthetic dataset of 100 shipments.

A* Search Algorithm

The heuristic function was first defined to simulate the customs clearance process. In this case would be to estimate the total time cost needed, such that a lower “risk level” and higher “documentation completeness” of shipments is set as the most favourable condition. This heuristic function helps in deciding the pathing of the algorithm to consider.

```
import heapq
import numpy as np

# Heuristic function to estimate the remaining cost/time
def heuristic(shipment, goal):
    return shipment['risk_level'] + (1 - shipment['documentation_completeness'])
```

Figure: Heuristic function used.

Next, using the package “heapq” in Python, nodes for the algorithm to explore are stored in the heapq. The “came from” function stores historical memory of which path was previously taken by the algorithm, and “cost so far” stores information on the time cost to explore each node. The “heappop” function ensures that the algorithm prioritizes nodes with the lowest time cost first. The algorithm continues to search for open nodes until the current shipment is equal to the goal shipment, which then breaks the loop function.

The neighbor function is then defined, where “new cost” helps the algorithm to calculate from the current shipment, what the estimated cost to reach the neighbor shipment. An “if” statement was added to ensure that if the “new cost” is lower than the “cost so far”, that cheaper time cost shipment is updated to reach its neighbor. The “priority” code line combines the new time cost with the heuristic estimate time cost of shipments. The overall A* search algorithm is defined this way such that the algorithm ensures the overall loop explore all potential paths from the current shipment, then updates the time cost and priorities respectively, which ultimately causes the algorithm to provide the solution with the shortest path possible. Lastly, a “reconstruct path” is defined to allow the algorithm to backtrack from the goal to the starting point by utilizing the “came from” codes.

```

# Define the A* search algorithm
def a_star_search(start_shipment, goal_shipment, shipments):
    open_list = []
    heapq.heappush(open_list, (0, start_shipment))
    came_from = {}
    cost_so_far = {start_shipment['id']: 0}

    while open_list:
        _, current_shipment = heapq.heappop(open_list)

        if current_shipment == goal_shipment:
            break

        for neighbor in get_neighbors(current_shipment, shipments):
            new_cost = cost_so_far[current_shipment['id']] + get_cost(current_shipment, neighbor)
            if neighbor['id'] not in cost_so_far or new_cost < cost_so_far[neighbor['id']]:
                cost_so_far[neighbor['id']] = new_cost
                priority = new_cost + heuristic(neighbor, goal_shipment)
                heapq.heappush(open_list, (priority, neighbor))
                came_from[neighbor['id']] = current_shipment

    return reconstruct_path(came_from, start_shipment, goal_shipment)

# Function to get neighbors (next possible steps in the clearance process)
def get_neighbors(shipment, shipments):
    index = shipments.index(shipment)
    return shipments[index + 1:index + 4]

# Function to calculate the cost between two shipments
def get_cost(current, neighbor):
    return abs(current['processing_time'] - neighbor['processing_time'])

# Function to reconstruct the path from start to goal
def reconstruct_path(came_from, start, goal):
    current = goal
    path = [current]
    while current != start:
        current = came_from[current['id']]
        path.append(current)
    path.reverse()
    return path

```

Figure: A* search algorithm.

In order to implement the A* search algorithm into the dataset, the following code was used. The algorithm will print the solution that provides the best path prioritizing on the least time cost needed to complete all shipments of customs clearance. The results will be discussed in the Results and Discussions section.

```

# Example usage
shipments = load_shipments_data()
start_shipment = shipments[0]
goal_shipment = shipments[-1]

best_path = a_star_search(start_shipment, goal_shipment, shipments)
total_cost = sum(get_cost(best_path[i], best_path[i+1]) for i in range(len(best_path)-1))
print("Best path:", best_path)
print("Total cost:", total_cost) # Time Cost in Hours

```

Figure: Code used to provide best optimized pathway and the least time cost needed.

Genetic Algorithm

A fitness function is first defined in order to set the priorities for the algorithm to analyse and calculate the best pathway, which will hereby be denoted as “best schedule”. Similar concept of how the A* search algorithm heuristic function was designed, the fitness function here will treat a lower processing time “total time” as the more favourable condition, giving the smaller “total time” a higher fitness score. This ensures that the genetic algorithm prioritises and selects pathways that have shorter shipment processing times as the optimal pathway.

```
# Fitness function to evaluate a schedule
def fitness_function(schedule, shipments):
    total_time = sum(shipments[shipment]['processing_time'] for shipment in schedule)
    return -total_time
```

Figure: Fitness Function defined for the genetic algorithm .

Then, a test population is initialized by using a random permutation of custom shipments. This creates a wide search space for the genetic algorithm to capitalize on in order to look for the best optimized solutions considering all pathways. The code “select” is then defined and used to select the population with fitness scores above 50% to be act as the “parent” for the next generation. This selection method ensures the overall fitness score of the population is improved over generations over time. The “crossover” of parents are then used to create two new “child” schedules from each parent (denoted as offspring), creating new variation of solutions for the genetic algorithm to explore. Next, the “mutate” function is used to swap elements in the offspring at random to further create variability within the population, reducing the likelihood of the population becoming too homogeneous.

```

# Initialize population
def initialize_population(pop_size, num_shipments):
    return [np.random.permutation(num_shipments) for _ in range(pop_size)]

# Select the best individuals
def select(population, fitness_scores):
    selected_indices = np.argsort(fitness_scores)[-len(population)//2:]
    return [population[i] for i in selected_indices]

# Crossover operation to combine parents
def crossover(parents):
    offspring = []
    for i in range(0, len(parents), 2):
        if i + 1 < len(parents):
            cross_point = np.random.randint(len(parents[0]))
            child1 = np.concatenate((parents[i][:cross_point], parents[i + 1][cross_point:]))
            child2 = np.concatenate((parents[i + 1][:cross_point], parents[i][cross_point:]))
            offspring.extend([child1, child2])
    return offspring

# Mutation operation to introduce variability
def mutate(offspring, mutation_rate):
    for individual in offspring:
        if np.random.rand() < mutation_rate:
            swap_indices = np.random.randint(0, len(individual), 2)
            individual[swap_indices[0]], individual[swap_indices[1]] = individual[swap_indices[1]], individual[swap_indices[0]]
    return offspring

```

Figure: The initialization, selection, parent crossover, and mutation of population to create new solutions while ensuring variability.

Lastly, the adaptive genetic algorithm is defined and implemented to allow the algorithm to continue looping the cycle of selecting, crossover, and mutation of the population, improving the likelihood of the best solution being found.

```

# Adaptive Genetic Algorithm
def adaptive_genetic_algorithm(pop_size, generations, mutation_rate, shipments):
    population = initialize_population(pop_size, len(shipments))

    for generation in range(generations):
        fitness_scores = [fitness_function(individual, shipments) for individual in population]
        selected = select(population, fitness_scores)
        if len(selected) < 2:
            break
        offspring = crossover(selected)
        population = mutate(offspring, mutation_rate)

    best_solution = max(population, key=lambda x: fitness_function(x, shipments))
    return best_solution

```

Figure: Adaptive Genetic Algorithm defined to find the best optimized solution across generations.

To implement the adaptive genetic algorithm created a population size of 100 shipments, 50 generations, and a mutation rate of 1% was proposed and used as an example to run the algorithm. Similar to A* search algorithm, the goal is to find the solution containing the “best schedule” with the least amount of “total processing time”. The results are to be discussed in the Results and Discussion section.

```
# Example usage
best_schedule = adaptive_genetic_algorithm(pop_size=100, generations=50, mutation_rate=0.01, shipments=shipments)
total_time = -fitness_function(best_schedule, shipments)
print("Best schedule:", best_schedule)
print("Total processing time:", total_time) # Time in Hours
```

Figure: Code used to define the adaptive genetic algorithm.

Results

```
# Example usage
shipments = load_shipments_data()
start_shipment = shipments[0]
goal_shipment = shipments[-1]

best_path = a_star_search(start_shipment, goal_shipment, shipments)
total_cost = sum(get_cost(best_path[i], best_path[i+1]) for i in range(len(best_path)-1))
print("Best path:", best_path)
print("Total cost:", total_cost) # Time Cost in Hours

Best path: [{'id': 0, 'risk_level': 0.3745401188473625, 'documentation_completeness': 0.9507143064099162, 'processing_time': 8}, {'id': 3, 'risk_level': 0.6011150117432088, 'documentation_completeness': 0.7080725777960455, 'processing_time': 6}, {'id': 6, 'risk_level': 0.3042422429595377, 'documentation_completeness': 0.5247564316322378, 'processing_time': 9}, {'id': 8, 'risk_level': 0.38246199126716274, 'documentation_completeness': 0.9832308858067882, 'processing_time': 9}, {'id': 11, 'risk_level': 0.9656320330745594, 'documentation_completeness': 0.8083973481164611, 'processing_time': 9}, {'id': 14, 'risk_level': 0.9093204020787821, 'documentation_completeness': 0.2587799816000169, 'processing_time': 8}, {'id': 15, 'risk_level': 0.7553614103176525, 'documentation_completeness': 0.4251558744912447, 'processing_time': 6}, {'id': 18, 'risk_level': 0.5978999788110851, 'documentation_completeness': 0.9218742350231168, 'processing_time': 8}, {'id': 21, 'risk_level': 0.5867511656638482, 'documentation_completeness': 0.965255307264138, 'processing_time': 9}, {'id': 23, 'risk_level': 0.9868869366005173, 'documentation_completeness': 0.7722447692966574, 'processing_time': 8}, {'id': 26, 'risk_level': 0.926300878513349, 'documentation_completeness': 0.6510770255019445, 'processing_time': 7}, {'id': 29, 'risk_level': 0.6659223566174967, 'documentation_completeness': 0.5912977877077271, 'processing_time': 6}, {'id': 31, 'risk_level': 0.9717120953891037, 'documentation_completeness': 0.8489138242660839, 'processing_time': 5}, {'id': 34, 'risk_level': 0.03142918568673425, 'documentation_completeness': 0.6364104112637804, 'processing_time': 4}, {'id': 36, 'risk_level': 0.24929222914887494, 'documentation_completeness': 0.41038292303562973, 'processing_time': 4}, {'id': 39, 'risk_level': 0.10549425983027061, 'documentation_completeness': 0.45653457048291024, 'processing_time': 4}, {'id': 40, 'risk_level': 0.8925589984899778, 'documentation_completeness': 0.5393422419156507, 'processing_time': 7}, {'id': 43, 'risk_level': 0.8180147659224931, 'documentation_completeness': 0.8607305832563434, 'processing_time': 7}, {'id': 45, 'risk_level': 0.6924360328902703, 'documentation_completeness': 0.2694123337985215, 'processing_time': 8}, {'id': 48, 'risk_level': 0.2539154139343447, 'documentation_completeness': 0.2468760628386012, 'processing_time': 7}, {'id': 51, 'risk_level': 0.5026790232288615, 'documentation_completeness': 0.05147875124998935, 'processing_time': 5}, {'id': 53, 'risk_level': 0.5309345833171364, 'documentation_completeness': 0.44778316457309164, 'processing_time': 6}, {'id': 56, 'risk_level': 0.4703006344460384, 'documentation_completeness': 0.983423140894843, 'processing_time': 6}, {'id': 59, 'risk_level': 0.5908929431882418, 'documentation_completeness': 0.6775643618422824, 'processing_time': 3}, {'id': 62, 'risk_level': 0.3975720210875223, 'documentation_completeness': 0.5177513505274801, 'processing_time': 8}, {'id': 65, 'risk_level': 0.22855002179729966, 'documentation_completeness': 0.17495492709593619, 'processing_time': 9}, {'id': 68, 'risk_level': 0.8826363431893397, 'documentation_completeness': 0.18870710834137938, 'processing_time': 9}, {'id': 71, 'risk_level': 0.6420316461542878, 'documentation_completeness': 0.08413996499504883, 'processing_time': 8}, {'id': 73, 'risk_level': 0.3722827665617431, 'documentation_completeness': 0.9401334424577784, 'processing_time': 9}, {'id': 75, 'risk_level': 0.4856137535862266, 'documentation_completeness': 0.44842414298624733, 'processing_time': 9}, {'id': 77, 'risk_level': 0.1788227092213288, 'documentation_completeness': 0.3664687845828599, 'processing_time': 8}, {'id': 80, 'risk_level': 0.26520236768172545, 'documentation_completeness': 0.24398964337908358, 'processing_time': 6}, {'id': 83, 'risk_level': 0.5769038846263591, 'documentation_completeness': 0.4925176938188639, 'processing_time': 5}, {'id': 86, 'risk_level': 0.994550510797341, 'documentation_completeness': 0.46994451399094295, 'processing_time': 5}, {'id': 87, 'risk_level': 0.883494022266259, 'documentation_completeness': 0.7477187738974139, 'processing_time': 5}, {'id': 90, 'risk_level': 0.2944488920695857, 'documentation_completeness': 0.38509772860192526, 'processing_time': 7}, {'id': 93, 'risk_level': 0.570061170089365, 'documentation_completeness': 0.09717649377076854, 'processing_time': 6}, {'id': 96, 'risk_level': 0.7024840839871093, 'documentation_completeness': 0.35949115121975517, 'processing_time': 7}, {'id': 99, 'risk_level': 0.6278944149486361, 'documentation_completeness': 0.19427395351204224, 'processing_time': 8}]
Total cost: 44
```

Figure: Results of A* search algorithm use case example.

The results of A* search algorithm in the figure above show the most optimal pathway to complete the shipment customs clearance, with a total time cost of 44 hours. The solution of best path and total cost were generated based on the prioritization of lower “processing time” and higher “document completeness”, allowing the shipments with the least “risk level” to have its customs clearance completed first, then moving to those of higher “risk level”, showcasing effective and optimized processing SOPs, reducing time taken to complete customs clearance.

```
# Example usage
best_schedule = adaptive_genetic_algorithm(pop_size=100, generations=50, mutation_rate=0.01, shipments=shipments)
total_time = -fitness_function(best_schedule, shipments)
print("Best schedule:", best_schedule)
print("Total processing time:", total_time) # Time in Hours

Best schedule: [27 48 13 67 88 68 85 96 16 39 10 5 19 17 94 84 73 64 33 57 77 80 84 51
23 60 18 29 94 32 81 16 33 86 17 6 92 72 70 48 76 79 53 63 68 85 12 59
24 57 66 8 5 88 1 27 76 60 67 12 35 79 91 88 85 95 56 72 82 54 32 90
6 5 36 33 84 58 41 52 26 75 57 80 46 15 27 83 64 59 98 42 28 23 53 11
16 2 24 34]
Total processing time: 477
```

Figure: Results of Genetic algorithm use case example.

The figure shows potentially the best pathway (labelled as best schedule) with the total processing time of 477 hours. The increased processing time compared to A* search algorithm indicates the time taken to process all potential solutions generated by the selecting, crossover, and mutation of the population.

```
import heapq
import numpy as np
from scipy.stats import mannwhitneyu

# Collect performance metrics for A* search
performance_a_star = []
for _ in range(30):
    path = a_star_search(start_shipment, goal_shipment, shipments)
    total_cost = sum(get_cost(path[i], path[i+1]) for i in range(len(path)-1))
    performance_a_star.append(total_cost)

# Collect performance metrics for Genetic Algorithm
performance_genetic_algorithm = []
for _ in range(30):
    best_schedule = adaptive_genetic_algorithm(pop_size=100, generations=50, mutation_rate=0.01, shipments=shipments)
    total_time = -fitness_function(best_schedule, shipments)
    performance_genetic_algorithm.append(total_time)

print("A* Search Performance:", performance_a_star)
print("Genetic Algorithm Performance:", performance_genetic_algorithm)

# Perform Mann-Whitney U test
stat, p_value = mannwhitneyu(performance_a_star, performance_genetic_algorithm)

print('Mann-Whitney U Test Statistic:', stat)
print('P-Value:', p_value)

A* Search Performance: [44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44]
Genetic Algorithm Performance: [477, 482, 467, 489, 494, 480, 468, 479, 478, 478, 493, 476, 476, 488, 484, 466, 494, 486, 481, 468, 484, 466, 466, 486, 479, 475, 462, 485, 485, 474]
Mann-Whitney U Test Statistic: 0.0
P-Value: 1.1999610007865912e-12
```

Figure: Mann-Whitney U Test Results.

Both the A* search algorithm and Genetic algorithm were repeated 30 times respectively using the same mentioned synthetic dataset and its solutions were recorded. The Mann-Whitney U Test was run on each algorithm respectively and the results show that the A* search algorithm

consistently found the same pathway with 44 hours processing time, whereas the genetic algorithm explored and used between 462 to 494 hours processing time. The p-value is less than 0.05, therefore the null hypothesis is rejected as there is a significant difference between the performance of the two algorithms. For this scenario where only three attributes of “risk level”, “document completeness”, and “processing time”, the A* search algorithm is better suited to optimize customs clearance processes due to its shorter processing time.

Discussion

In this dataset where it consist of 100 shipments with randomly generated attributes of “risk level”, “document completeness”, and “processing time”, A* star search gave better results than genetic algorithm in overall optimization of pathfinding by selecting lower “processing time” and higher “document completeness” as priority shipments. A* search algorithm exploits the same path with minimal processing time of 44, confirmed by algorithm’s performance test before Mann-Whitney U Test. The p-value from Mann-Whitney U test was less than 0.05, indicating that there were significant differences between the two algorithms where A* search is superior in terms of performance and thus better optimized solutions with reduced processing times were achieved as opposed to those obtained under genetic algorithm. The summary of this data set also indicated that the genetic algorithm performed poorly with a processing time of 477, compared to A-star search algorithm’s processing time of 44, revealing that it is more effective at minimizing customs clearance process duration. The A* search algorithm was consistent in its patterning as well as dependability.

As a conclusion, the A* search algorithm is significantly better than the genetic algorithm at optimizing the customs clearance process within this dataset, with a clear problem definition and straightforward yet effective heuristic function to instruct the algorithm. However, in real-world data of customs clearance involving the movement of millions of shipments, and considering a more complex and everchanging compliance and regulations between countries, there are scenarios that contain multi-dimensional attributes to consider within the customs clearance sector. The genetic algorithm may be the preferred choice due to the algorithm’s dynamic ability to adapt to multi-dimensional search space, generating solutions that considers and explores all potential pathways within its increased population size and generations.

References

1. Hoffman, A. J., Venter, W. C., Grater, S., Maree, J., & Liebenberg, D. (2018). An explorative study into the effectiveness of a customs operation and its impact on trade. North-West University ac.za . <https://repository.nwu.ac.za/handle/10394/31423>