BEYKOZ UNIVERSITY

GRADUATE PROGRAMS INSTITUTE

WAREHOUSE PICKING PATH OPTIMIZATION USING METAHEURISTIC ALGORITHMS

Tarık HASSA

MASTER'S THESIS

İstanbul, August 2023

WAREHOUSE PICKING PATH OPTIMIZATION USING METAHEURISTIC ALGORITHMS

Tarık HASSA

MASTER'S THESIS
COMPUTER ENGINEERING DEPARTMENT
COMPUTER ENGINEERING MASTER'S PROGRAM
WITH THESIS

MASTER'S THESIS ADVISOR
Asst. Prof. Dr. Özlem Feyza ERKAN

MASTER'S THESIS JURY MEMBERS
Asst. Prof. Dr. Burçin KÜLAHÇIOĞLU
Assoc. Prof. Dr. Onur CİHAN

PREFACE

At first, I would like to thank my dear advisor Asst. Prof. Dr. Özlem Feyza ERKAN from my heart for being a candle lighting my journey at master and a role model through my study. Thank you for your help and patience and I wish you more and more success in your life.

In addition, I would like to thank my wife and family for their support, encouragement and patience and I hope that this will be just a start for more achievements in both my educational and professional life.

CONTENTS

ABSTRACT

WAREHOUSE PICKING PATH OPTIMIZATION USING METAHEURISTIC ALGORITHMS

Warehouse picking path optimization is a critical issue in the logistics industry as it can greatly affect the efficiency and productivity of warehouse operations. The optimization process can be divided into two sub-problems: shortest path problem which can be solved using well-known algorithms such as Dijkstra's algorithm and Bellman–Ford algorithm, and traveling salesman problem, which will be the primary focus of this thesis. The Traveling Salesman Problem is a combinatorial optimization problem and belongs to the class of NP-hard problems. Since, it is usually impossible to obtain an exact solution to the problem within a polynomial time, alternative methods such as metaheuristic algorithms are utilized. In this thesis, we explore several metaheuristic algorithms, including Particle Swarm Optimization, Genetic Algorithm, Archimedes Optimization Algorithm, Honey Badger Algorithm, and Harris Hawks Optimization, and compare them in terms of their computational efficiency and effectiveness in finding optimal solutions. Our experimental results show that Harris Hawks Optimization and Honey Badger Algorithm yield better results compared to other algorithms. These results will contribute to the development of efficient and effective techniques for optimizing warehouse picking routes in the logistics sector.

Keywords: Picking Path Optimization; Shortest Path Problem; Traveling Salesman Problem; Metaheuristic Algorithms; Logistics

Date: 03.08.2023

ÖZET

METASEZGİSEL ALGORİTMALAR KULLANARAK DEPO TOPLAMA YOLU OPTİMİZASYONU

Depo toplama yolu optimizasyonu depo operasyonlarının verimliliğini ve üretkenliğini büyük ölçüde etkileyebileceğinden lojistik endüstrisinde kritik bir konudur. Optimizasyon süreci iki alt probleme ayrılabilir: Dijkstra algoritması ve Bellman-Ford algoritması gibi iyi bilinen algoritmalar kullanılarak çözülebilen en kısa yol problemi ve bu tezin ana odak noktası olacak gezgin satıcı problemi. Gezgin satıcı problemi, kombinasyonel optimizasyon problemidir ve NP-zor problemler sınıfına aittir. Problemin kesin çözümü çoğu zaman polinom zamanda elde edilemediğinden metasezgisel algoritmalar gibi diğer yöntemler kullanılır. Bu tezde, Parçacık Sürü Optimizasyonu, Genetik Algoritma, Arşimet Optimizasyon Algoritması, Bal Porsuğu Algoritması ve Harris Şahini Optimizasyonu dahil olmak üzere çeşitli metasezgisel algoritmalar araştırılmış ve bunlar hesaplama verimliliği ve optimal çözümler bulmadaki etkinlikleri açısından karşılaştırılmıştır. Deneysel sonuçlarımız, Harris Hawks Optimizasyonu ve Bal Porsuğu Algoritmasının diğer algoritmalara kıyasla daha iyi sonuçlar verdiğini göstermektedir. Bu sonuçlar, lojistik sektöründe depo toplama yollarını optimize etmek için verimli ve etkili tekniklerin geliştirilmesine katkı sağlayacaktır.

Anahtar Sözcükler: Toplama Yolu Optimizasyonu; En Kısa Yol Problemi; Gezgin Satıcı Problemi; Metasezgisel Algoritmalar; Lojistik

Tarih: 03.08.2023

ABBREVIATIONS

Nondeterministic Polynomial Time	: NP
Traveling Salesman Problem	: TSP
Genetic Algorithm	: GA
Particle Swarm Optimization	: PSO
Archimedes Optimization Algorithm	: AOA
Honey Badger Algorithm	: HBA
Harris Hawks Optimization	: HHO

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Efficient warehouse management is critical for the success of supply chains. In particular, fulfillment and distribution warehouses play an important role in the processes of goods receiving, storage, retrieval, order processing, and shipping. However, the activities involved in retrieving and fulfilling orders alone account for fifty to seventy percent of the entire operational costs of warehouses. These expenses result from the labor-intensive nature of the manual picker-to-part order picking system commonly used, wherein pickers have to travel to specific aisle locations to collect the items in the quantities specified on their pick list. As a result, warehouse professionals have placed a high priority on the task of order picking as a potential area for performance enhancement, as it can contribute to reduce expenses (Aboelfotoh, Singh, and Suer, 2019).

Warehouse picking path optimization is one of the major challenges encountered in the logistics industry. This task involves finding the most efficient path for workers or automated systems to pick items from warehouse shelves. The optimization of the picking path carries substantial importance for the overall performance of the supply chain. Inefficient picking paths can lead to longer travel times, increased labor costs, and reduced order accuracy. Thus, optimizing picking paths is crucial for the success of the logistics sector.

Logistics companies have been under increased pressure to enhance their warehouse operations due to the rapid expansion of e-commerce in recent years. The rise in the prevalence of online shopping has resulted in an increase in the number of orders that must be processed and a greater demand for faster delivery times. In response, many logistics companies have invested in automation and robotics technology to improve the efficiency of their warehouse operations. However, even with these technologies, picking path optimization remains a challenging problem, and the development of effective algorithms is crucial to ensure efficient and timely order fulfillment. The process of determining a picking path involves solving two of the most typical issues encountered in graph theory: the shortest path problem and the TSP (Key and Dasgupta, 2015).

1.1 PROBLEM STATEMENT

The task of determining the shortest route between a particular vertex and all the other vertices is referred to as the shortest path problem. In the context of warehouse picking path optimization, the shortest path problem pertains to finding the most optimal path for a picker to travel between two distinct locations within the warehouse. Various well-known algorithms are commonly utilized to solve the shortest path problem including Dijkstra's algorithm, the Bellman-Ford algorithm and the Floyd-Warshall algorithm (Zhao L., and Zhao J., 2017). TSP presents the challenge of designing a route that visits n different locations while minimizing the distance traveled, ensuring each location is visited exactly once and beginning and terminating at the same location. When considering the optimization of warehouse picking paths, the TSP entails locating the most optimal path for a picker to travel between multiple locations within the warehouse and eventually return to the starting point. The TSP poses a challenge as it falls within the category of problems known as NP-hard problems, making it computationally challenging to discover the ideal answer especially for large instances of the problem. (Gademann, Van Den Berg, and Van Der Hoff, 2001).

This thesis mainly addresses the problem of optimizing warehouse picking paths by solving the underlying TSP. To tackle this problem, we have utilized two of the well-known metaheuristic algorithms: The Genetic Algorithm (GA) and the Particle Swarm Optimization (PSO). Additionally, we have employed three recent algorithms, namely the Archimedes Optimization Algorithm, the Honey Badger Algorithm, and the Harris Hawks Optimization Algorithm. Furthermore, a comparative study has been conducted to evaluate the performance of these optimization techniques across datasets of varying sizes.

1.2 MOTIVATION

With the rise of large-scale warehouses and the ever-growing demand for fast and accurate order fulfillment, efficient warehouse picking has become one of the major concerns. As a result, the necessity for efficient algorithms capable of optimizing picking paths in real-world warehouse scenarios has become apparent. This study aims to contribute to the existing knowledge on warehouse picking path optimization by evaluating the performance of new metaheuristic algorithms (that were not used in this

field before) along with the traditional algorithms on datasets of varying sizes.
The valuable insights derived from this thesis can be leveraged by logistics and e-commerce companies to enhance their warehouse operations and improve the overall performance of their supply chains.

1.3 ORGANIZATION OF THESIS

The rest of the thesis is organized as follow:
Chapter 2 presents a comprehensive review on warehouse picking path optimization and the different techniques used.
Chapter 3 describes the assumptions of the studied warehouse, details the methodology employed for warehouse picking path optimization and provides the details of used metaheuristic algorithms.
Chapter 4 discusses the algorithms parameter settings side by side with the experimental results with and a review about our demo application.
Chapter 5 summarizes the achieved results and presents future directions in the field of warehouse picking optimization.

2 LITERATURE SURVEY

Warehouse picking path optimization plays a crucial role in supply chain management, directly influencing overall system efficiency and productivity. Over recent years, several approaches have been implemented to improve the efficiency of a warehouse's picking path, with a primary focus on minimizing travel distances during picking tours which significantly reduces travel time. Many of these approaches are rooted in on the TSP. In this literature survey, we review some of the important contributions made in this area.

Ene, S., and Öztürk, N. (2012) proposed a study that attempts to improve warehouse activities in the automotive sector by developing a system for order picking and storage assignment based on a stochastic evolutionary optimization method and a mathematical model. The study involved two phases. Initially, the storage location assignment problem was addressed using integer programming. This was followed by batching and routing problems, which are used to reduce the travel costs within warehouse operations. The suggested method is based on Genetic Algorithms and is adaptable to any warehouse design used in the automotive sector. The storage assignment optimization findings demonstrated that the strategy was able to optimize storage assignments and minimize the total amount of time spent traveling throughout an assembly supply warehouse. Furthermore, the results of optimizing the route and batching process indicated that the proposed Genetic Algorithm can efficiently optimize the order picking process. The algorithm achieved this by encoding solutions through order locations and calculating the distance between locations in the warehouse using the proposed approaches.

Azadnia et al. (2013) worked on minimizing travel distance and order tardiness during order picking process in managing warehouses using a novel approach named ATGH. The ATGH approach comprises a total of four stages: using a weighted association rule mining method to determine relationships between orders based on expected delivery dates, a model for order batching that seeks to strengthen the

connections between orders within each batch, the order picking stage which involved employing a Genetic Algorithm to solve the Traveling Salesman Problem to identify the best route, and employing a Genetic Algorithm to sort the generated batches in order to reduce the amount of lateness. The results showed that ATGH outperforms other approaches by considering the weight parameter of due dates in order association mining, which improves solution quality in terms of minimizing tardiness.

Kordos, M., Boryczko, J., Blachnik, M., and Golak, S. (2020) introduced a comprehensive and fully automated solution based on Genetic Algorithms for improving product placement and optimize order picking paths in a warehouse. The warehouse layout and the list of orders are both taken into consideration to generate an efficient product placement strategy that reduces the total amount of time spent on order picking. The Genetic Algorithm utilized in this study incorporates a multi-parent crossover operator to optimize the order choosing pathways. In experiments, the optimization of order picking routes resulted in a 54% reduction in the overall amount of time required for order picking. Additionally, the optimization of product placement resulted in a 26% reduction in that time, while the optimization of product placement using proposed methods yielded 21% reduction. The techniques presented in this study can accelerate the order picking and reduce warehouse operating costs, enabling an increase in sales and profitability as well as serving more clients at the same time.

Zhang, D., and Zhang, J. (2021) discussed the problem of optimizing the picking routes in an automated warehouse in order to increase the efficiency of the picking operations in distribution center. The authors used the Simulated Annealing algorithm to optimize the picking path, comparing its performance against the nearest-neighbor heuristic and the nearest-insertion heuristic. The outcomes showed that the picking performance of the distribution center has significantly increased using the Simulated Annealing algorithm, reducing the traveling distance by 25% and 16.67% compared to the nearest-neighbor heuristic and the nearest-insertion heuristic, respectively. The research demonstrates that the picking time is substantially reduced, improving the efficiency of picking operations, efficiency of the distribution center's operations and the quality of the service it provides to customers. The results of the study indicated that the Simulated Annealing algorithm is an effective approach to optimize outbound picking routes in an automated warehouse, providing a significant improvement in the picking efficiency of the distribution center.

Önüt, S., Tuzkaya, U. R., and Doğaç, B. (2008) addressed the problem of constructing multi-level warehouses for a distribution-type strategy with the aim of minimizing the annual carrying costs. Their study proposed a class-based storage approach involving three product categories and considered turnover rates and distances between shelves and docks. Given the NP-hard nature of the problem, the researchers developed a novel heuristic, the particle swarm optimization (PSO) algorithm, to determine the optimal layout. The PSO algorithm is modified to handle constrained problems using funnel effect and penalty methods. The study also investigated the impact of increased dock numbers cost management and the architecture of three-level warehouses. The results indicated that with an increase in dock numbers, the overall handling cost has grown, regardless of architecture modifications. This proposed model enhanced the two-dimensional warehouse design to a multiple-level warehouse design contributing to warehouse layout problem by integrating the class-based storage strategy. Results demonstrated that PSO algorithm is an efficient heuristic for solving the NP-hard problem of warehouse design.

Chen et al. (2013) proposed a routing method called A-TOP, based on Ant Colony Optimization (ACO), to reduce picker crowding within order picking systems characterized by narrow aisles. A-TOP considered congestion by using a Tabu List which records unavailable time, and employs specific rules to construct picking routes for two order pickers coexisting in the same area without contributing overcrowding in picking aisles. The effectiveness of A-TOP is evaluated through a comprehensive simulation study that considers the design of the warehouse, the number of items per order, and the pick:walk-time ratio as factors. Simulation results show that A-TOP achieved the shortest total picking time in most instances and performed well in reducing picker overcrowding within order-picking system.

Cheng et al. (2015) introduced an efficient hybrid algorithm to address the joint batch picking and picker routing problems in warehouses. The algorithm merges aspects of both PSO and ACO algorithms and is designed to calculate the batch size, order sorting in a batch, as well as the distance traveled. The proposed method outperforms known optimal solutions and current practices in the industry, improving picking performance and meeting customer demands more efficiently. Overall, the proposed hybrid algorithm provides an efficient solution for the studied problem, considering the characteristics of existing logistics centers and enabling planners a greater degree of

flexibility in allocating orders.

Yu et al. (2020) used a parallel ranking ant colony optimization algorithm (P-RACO) for warehouse routing, specifically aiming to enhance the operational efficiency of automated guided vehicles (AGVs) and order fulfillment within intelligent warehouses. The algorithm employed multiple sub-colonies that interact through pheromones to co-evolve and enhance the ability of the algorithm to perform its functions. A multi-objective function with the objectives of the shortest route possible with the fewest number of twists required by the AGV was implemented. The proposed technique generated routes aligned with these two objectives. During the driving process of the AGV, the path was made substantially smoother by cutting down on the number of intermediate nodes, which in turn significantly decreased the count of twists and the overall traveling distance. The study concluded that the P-RACO algorithm has improved the efficiency and reduced the count of the twists compared to the methods that only focuses on the shortest path.

Chen, F., Xu, G., and Wei, Y. (2019) introduced a customized routing algorithm that blends ant colony optimization (ACO) and integer-coded Genetic Algorithm (GA) in order to overcome the challenges presented by ultra-narrow aisles and access restrictions in a large e-commerce enterprise. The proposed algorithm generates initial chromosomes using ACO and then uses GA to search for solutions that are close to ideal scenario for picker-routing. An innovative chromosomal design is adopted, recording access modes and sub-aisles. The results of the simulations demonstrated that the hybrid algorithm produces higher quality solutions compared to dedicated heuristics. The research also analyzed the influence that the warehouse layout characteristics have on the efficiency of picking and suggested increasing the count of the connect aisles and cross aisles to increase the efficiency. The proposed method can be applied to warehouses with traditional aisles by minimizing access restrictions.

Reda et al. (2022) presented a novel algorithm, called Discrete Damped Cuckoo Search algorithm (DDCS), to solve the order picking routing problem within warehouse environments. The algorithm is specifically designed for autonomous trolleys operating in warehouses and accounting for diagonal movement constraints. Several enhancements were introduced to the standard cuckoo search algorithm, including the use of random key encoding to represent TSP solutions, 2-opt moves and adaptive Levy Flight Random Walk (LFRW) to update solutions with diverse mutation strategies for different population subgroups. The proposed DDCS algorithm was compared with GA,

ACO, and PSO. The results showed that the DDCS algorithm is superior than all other algorithms, particularly when handling challenging tasks. Moreover, the study recommended that the DDCS algorithm is appropriate for usage in big warehouses due to its centralized architecture which.

3 METHODOLOGY

In this section, we will discuss the studied warehouse assumptions and the techniques used to optimize the warehouse picking routes based on two main problems: the Shortest Path Problem and Traveling Salesman Problem.

3.1 WAREHOUSE ASSUMPTIONS

In this thesis, we make several assumptions about the layout and design of the warehouse. These assumptions serve as a foundation for developing the optimization techniques in order to find a solution for the picking path problem. The following paragraphs describe the assumptions in more detail.

3.1.1 Warehouse Layout

Suppose that we have a traditional rectangular warehouse layout with 9 double-sided racks and 3 single-sided racks. Every rack side is partitioned into six sections, each measuring 2 meters in width and 1 meter in depth. This setup results in a total of 126 sections in the warehouse. These sections are vertically labeled with letters and horizontally with numbers. This labeling convention helps to locate products quickly and accurately where we assume that each product corresponds to its designated section. The layout of the studied warehouse is illustrated in Figure 3.1.

Aisles

Between the racks, we consider 4 meter wide aisles that allow the product picker to move in a comfortable manner within the warehouse.

Figure 3.1 Studied Warehouse Layout

Packaging Area

It is assumed that every product picking process starts and ends at the packaging area, which is located near section A1. This dedicated area is used for product packaging and preparation for shipment. By assuming that the picking process starts and ends at the same point (A1), we simplify the picking path problem and make it more tractable.

Figure 3.2 provides a simplified representation of the warehouse's design. The graph weights are calculated by starting from the midpoint of each section. For instance:

- The distance from A1 to A2 spans 2 meters.
- The path from A1 to B1 covers 4 meters.
- The path from A1 to C1 is 8 meters.
- Lastly, the path from A6 to A7 is 6 meters.

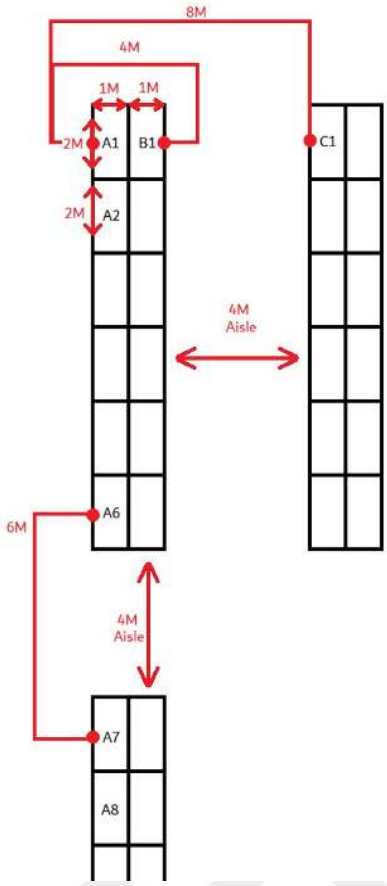


Figure 3.2 Warehouse Racks, Sections and Aisles Dimensions

3.1.2 Warehouse Graph

In the field of graph theory, a graph includes a group of vertices, which are also commonly referred to as nodes, and edges, which connect every pair of vertices. The edges indicate the relationships or connections among the nodes. Graphs offer a versatile framework to model a vast number of different real-world systems, such as social networks, transportation networks, and computer networks.

The vertices in a graph may represent different entities, such as people, cities, or websites. The edges indicate the links or connections between the vertices, like friendships, roads, or hyperlinks. Edges come in both directed and undirected forms. While the edges in an undirected graph can be traversed in any direction, the edges in a directed graph have a definite direction.

The graph can be represented through an adjacency matrix which is a square matrix that describes the graph by listing all of its vertices as rows and columns. The entries of matrix are filled with a value denoting the weight of the edge between the corresponding pair of vertices. If edge does not exist between a pair of vertices, the

entry is set to infinity (Deo, 2017).

With the warehouse design assumptions given in Figure 3.2, our warehouse can be represented with the undirected weighted graph in Figure 3.3 where each node represents a rack section whereas the edges represents the distances between the nodes.

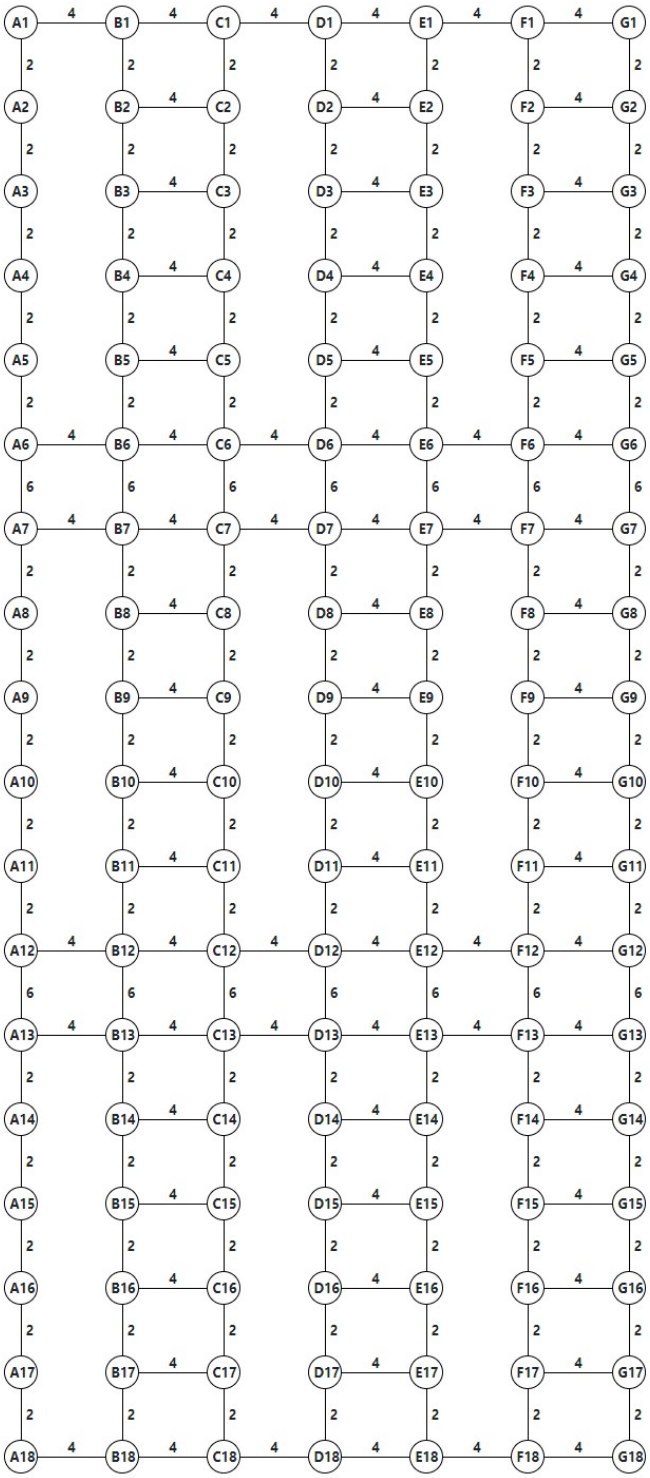


Figure 3.3 Warehouse Graph

3.2 SHORTEST PATH PROBLEM

The shortest path problem is one of the most fundamental problems in graph theory and computer science. This problem is defined as finding the path in a graph with the smallest distance between two nodes. It has applications in a wide variety of sectors, including logistics, transportation, network routing, and robotics. For instance, within transportation networks, determining the route with the minimum travel distance between two locations can significantly reduce travel time, fuel consumption, and cost.

Over the years, various algorithms have been introduced to address the shortest path problem. Dijkstra's algorithm, the Bellman-Ford algorithm, the A* algorithm, and the Floyd-Warshall algorithm are examples of well-known algorithms used for this problem (Magzhan and Jani, 2013).

In this thesis, we focus on the shortest path problem in the context of warehouse picking path optimization. We aim to find the shortest path for a product picker to visit a set of sections in a warehouse in order to complete an order. To achieve this goal, we have implemented Dijkstra's algorithm, a widely employed technique for determining the shortest path between two nodes in a graph.

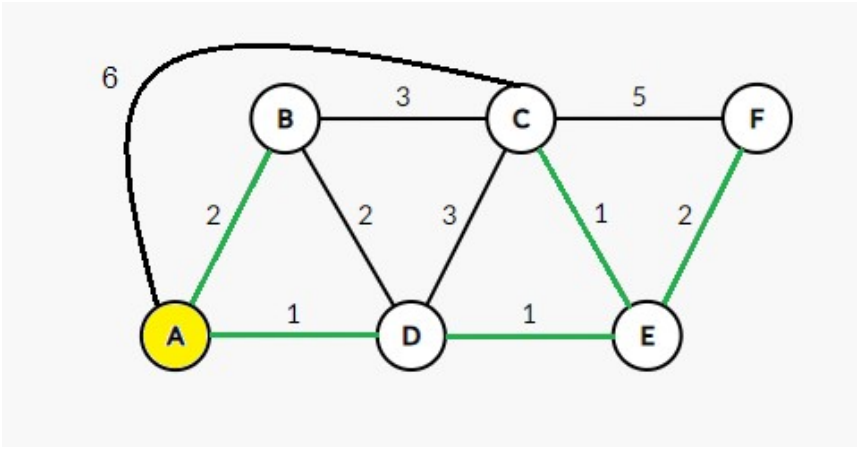


Figure 3.4 Shortest Path From Node (A) to Other Nodes

3.2.1 Dijkstra's algorithm

Dijkstra's algorithm is a well-known technique for identifying the shortest path within a graph connecting any two locations. It was proposed by Edsger W. Dijkstra and is considered one of the most popular and efficient algorithms that address the shortest path problem. Its appeal stems from its simplicity, efficiency, and applicability to a

diverse set of challenges, including routing in computer networks, map navigation, and transportation logistics.

The algorithm works by iteratively visiting the vertices of the graph and updating the shortest distances from the source vertex to each of the visited vertices. In each iteration, Dijkstra's algorithm chooses the closest unvisited vertex to the initial vertex and then adds it into the set of explored vertices. Subsequently, it updates the distances of all its neighboring vertices by comparing whether the distance across the current vertex is shorter than the distance that was calculated previously. This iterative process is repeated until either all vertices have been traversed or the destination vertex has been reached (Javaid, 2013).

Figure 3.5 provides the pseudo code for Dijkstra's algorithm (Al Shammary and Saudagar, A. K. J., 2015).

```
1  function Dijkstra(Graph, source):
2      dist[source]  := 0                    // Distance from source to source
3      for each vertex v in Graph:           // Initializations
4          if v ≠ source
5              dist[v]  := infinity          // Unknown distance function from source to v
6              previous[v]  := undefined     // Previous node in optimal path from source
7          end if
8          add v to Q                        // All nodes initially in Q (unvisited nodes)
9      end for
10
11     while Q is not empty:                 // The main loop
12         u := vertex in Q with min dist[u] // Source node in first case
13         remove u from Q
14
15         for each neighbor v of u:          // where v has not yet been removed from Q.
16             alt := dist[u] + length(u, v)
17             if alt < dist[v]:              // A shorter path to v has been found
18                 dist[v]  := alt
19                 previous[v]  := u
20             end if
21         end for
22     end while
23     return dist[], previous[]
24 end function
```

Figure 3.5 Dijkstra's Algorithm Pseudocode

For the studied warehouse graph, our objective is to compute the distances between all pairs of nodes within the graph. This process helps us to create a distance matrix and determine the sets of shortest paths. Using the adjacency matrix and the distance matrix, we can create a model of the warehouse that can be used for the TSP.

3.2.2 Subgraphing

In graph theory, a subgraph ***g*** refers to a smaller graph that is constructed by selecting a subset of vertices and edges from a larger graph ***G***. The concept of subgraphing can be applied finding patterns in a larger graph, reducing the size of a

graph for more efficient computation, and focusing on specific areas of interest within the graph. Subgraphing is particularly useful in analyzing large and complex graphs, as it allows researchers to focus on smaller, more manageable components of the graph (Deo, 2017).

In the domain of warehouse picking path optimization, creating a subgraph that contains the desired products sections while minimizing the required edges is an important step in the problem formulation. By creating a subgraph, the complexity of the optimization problem can be reduced significantly. In other words, instead of optimizing the entire warehouse layout, we can focus on optimizing the paths within the subgraph that contains the desired products. This approach yields the potential for more efficient and faster optimization algorithms. Moreover, it can also reduce the search space for the optimization problem, thereby improving the scalability and feasibility of the solution.

The following pseudocode given in Figure 3.6 represents the proposed algorithm to create the subgraph:

```
1 Function subgraph(products):
2   adjacency_matrix = Create a matrix of infinity values with the length of products
3     For each pair of nodes (i, j) in products:
4       If there is a shortest path between i and j that doesn't pass through the other products nodes:
5           Add the distance between i and j to adjacency_matrix[i][j]
6   Return a new Graph object with products nodes and adjacency_matrix as parameters
```

Figure 3.6 Pseudocode of Used Subgraphing Algorithm

Figure 3.7 illustrates an example of graph partitioning. The left-hand side shows the original graph, while the right-hand side shows a subgraph containing nodes 0,1,2,4,5 and the shortest paths between them.

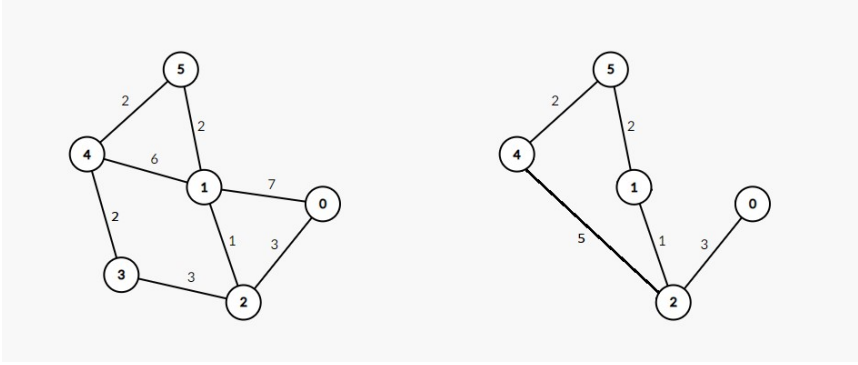


Figure 3.7 Creating a Subgraph

3.3 TRAVELING SALESMAN PROBLEM

TSP is a well-known problem that has a straightforward description but is characterized by intricate solution. It requires finding the route with the least traveling distance for a salesman to visit a list of cities once only, returning to the starting point. Figure 3.8 represents the TSP for number of cities. The TSP falls under the category of combinatorial optimization problems, specifically NP-hard problems. If an efficient algorithm to solve the TSP is discovered, it could lead to efficient solutions for other NP-hard problems. However, no polynomial-time algorithm for the TSP has been found so far. The TSP can be mathematically formulated as a complete graph, with cities represented as nodes and distances between cities as edges. A tour or Hamiltonian cycle is defined as a round-trip of the cities, with the objective of identifying the tour with the minimum total distance (Hoffman et al., 2013).

The complexity of TSP attracted significant research interest, leading to its study from various perspectives such as graph theory, linear programming, dynamic programming, and local optima. Branch-and-bound strategies, dynamic programming, local search, heuristic algorithms, and metaheuristic algorithms (which are what we will primarily be focusing on) are some of the approaches used for obtaining exact solutions (Abeledo et al., 2013).

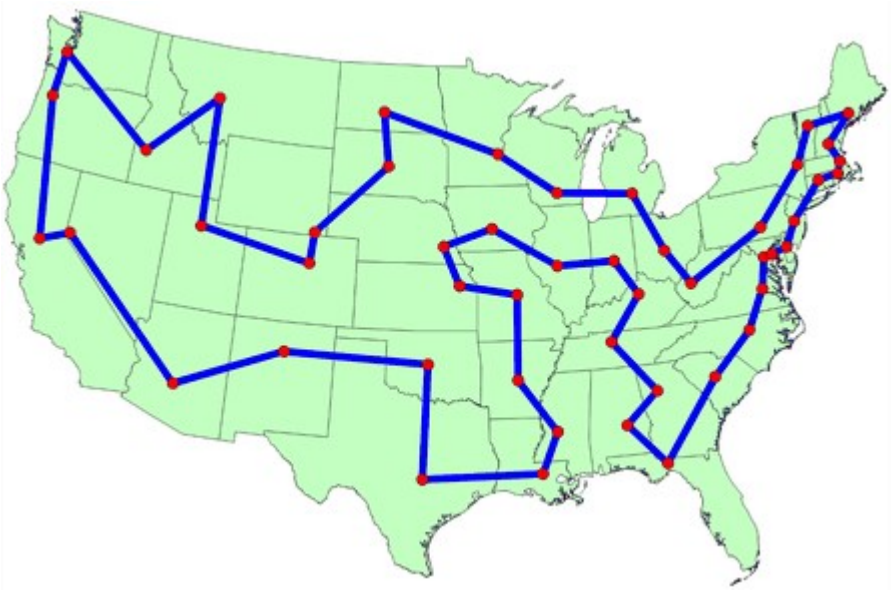


Figure 3.8 Travelling Salesman Problem

The warehouse picking path problem represents a TSP where the nodes corresponds the warehouse rack sections or product positions with one exception, revisiting a node multiple times is permissible provided that the total cost of picking

tour is minimized.

3.3.1 Metaheuristic Algorithms

Metaheuristics are advanced heuristics employed to find solutions a wide variety of optimization problems. They have gained popularity in addressing complex problems because these algorithms generate solutions that are close to optimal despite the scale of the task over a short period of time. Metaheuristic algorithms offer practical and elegant solutions for NP-Hard optimization problems, providing approximate or optimal solutions within reasonable execution times. Metaheuristics posses broad applicability, including single and multi-objective, continuous and discrete, constrained and unconstrained problems. The complexity of these problems makes exact algorithms impractical due to their non-polynomial execution times or computational requirements. In contrast, metaheuristic algorithms present efficient solutions by approximating the optimal solutions.

Metaheuristic algorithms can be broadly categorized into the following classes: evolution-based, swarm intelligence-based, physics-based, and human behavior-related algorithms given as in Figure 3.9. Evolution-based algorithms emulate the natural process of evolution, generating new population through mutation, crossover, and choosing the best solutions. Genetic Algorithm (GA) is a popular example in this category. Swarm intelligence-based algorithms mimic collective behaviors observed in nature. For instance PSO is inspired by the movement of bird flocks. Physics-based algorithms like Simulated Annealing follow principles from physics. Human behavior-related algorithms derive insights from human activities and performance. One prominent example is Teaching Learning-Based Optimization Algorithm (TLBO) (Agrawal et al., 2021).

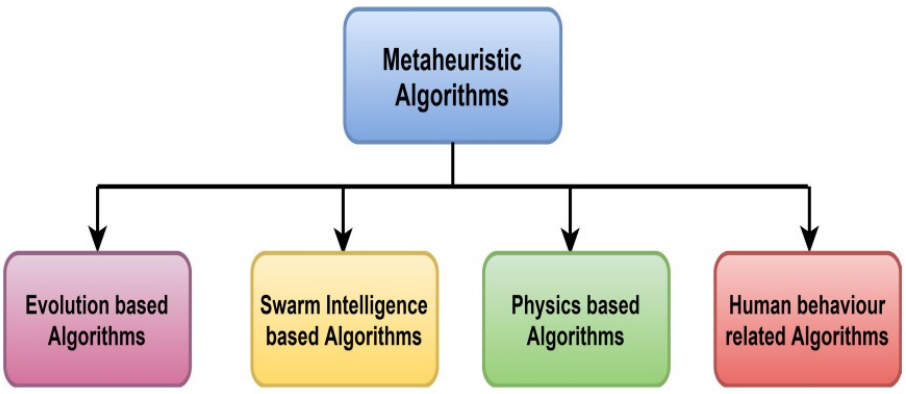


Figure 3.9 Classification of Metaheuristic Algorithms

Metaheuristic algorithms have common characteristics that render them well-suited for addressing scientific and engineering problems. These algorithms do not require complex mathematical expressions, and they can handle constraints easily compared to classical techniques. The main objective of metaheuristic algorithms is to search for the global optimal solution rather than local solutions.

Although specific details may vary among different algorithms, the fundamental sequence of steps for metaheuristic algorithms are as follows (Toklu, 2013):

Step 1: Creation of base vector(s): Generate initial solutions.

Step 2: Evaluation of base vector(s): Compute the objective function and evaluate constraint satisfaction for each solution, and assign evaluation values.

Step 3: Create a new set of candidate vectors: Generate a new set of solutions based on algorithm-specific rules.

Step 4: Evaluation of candidate vector(s): Evaluate the new solutions.

Step 5: Comparison of base vectors and candidate vectors: Compare the evaluation results of the base and candidate solutions.

Step 6: Re-defining the step size in the search area: Adjust the step size based on the comparison. If candidate vectors are better, increase the step size; otherwise, decrease it gradually.

Step 7: Deciding on ending the computations: Determine the stopping criteria, which could rely on the number of cycles or convergence criteria such as stabilization of the best evaluation value or reaching a point where the search space cannot be significantly reduced.

Step 8: Redefining base vectors and rearranging rules and parameters: Define new base vectors, modify rules and modify parameters based on the algorithm's results from the previous cycle(s).

The classical metaheuristic algorithms, including GA, PSO, ACO, and SA, were proposed prior to the year 2000 and have since been widely used. However, over the last two decades, new and novel metaheuristic algorithms have emerged, inspired by evolutionary or behavioral paradigms. These new generation metaheuristics have shown promising results, often outperforming the classical algorithms. Algorithms like Archimedes Optimization Algorithm, Honey Badger Algorithm, Harris Hawks Optimization are the representatives of this new generation metaheuristics.

3.3.1.1 Genetic Algorithm

GA is a computational model inspired by Darwinian natural selection theory that designed to address a large number of challenges. This model works by encoding initial solutions into a data structure resembling a chromosomes and using recombination operators to keep important information safe. The GA begins with a population of randomly generated chromosomes, evaluates their performance, and allocates reproductive opportunities based on their fitness. The fittest chromosomes have a higher chance of reproducing, mimicking natural selection. GA derive inspiration from the mechanisms of nature, where survival of the fittest leads to the propagation of superior traits. In natural ecosystems, individuals with better survival traits survive longer and have more opportunities to pass on their genes. Gradually, the population becomes dominated by genes from the superior individuals, resulting in the extinction of less fit individuals. This fundamental phenomenon is referred to as natural selection (Mathew, 2012).

The algorithm consists of several key elements: chromosome representation, selection, crossover, mutation, and fitness function computation. The process begins with an initial random population (with a size of N) of solutions represented as chromosomes with genes which is spread across the search space to increase diversity. The results of each population is evaluated using the fitness function. Following this evaluation, two chromosomes are chosen from the population based on their fitness values which simulates natural selection, favoring the fittest individuals. Subsequently, crossover operator is applied to these selected chromosomes, yielding offspring. Crossover allows for exploration of different regions in the search space by exploiting the characteristics of the parent solutions. A mutation operator is then applied to the new generation, resulting a mutated offspring. This mutated offspring is then added to a new population. Mutation introduces randomness by altering one or more genes with the child solutions which helps maintaining diversity in the population and increases the chance of avoiding local optima. Mutation rates are typically set low to prevent excessive random changes. The aforementioned steps are iteratively executed until the new generation is fully formed. The algorithm dynamically adjusts the probabilities of crossover and mutation, leading to the discovery of optimal solutions (Katoch, Chauhan and Kumar, 2021). GA is represented with the pseudocode in Figure 3.10.

Input:
Population Size, n
Maximum number of iterations, MAX
Output:
Global best solution, Y_{bt}

begin
Generate initial population of n chromosomes $Y_i \ (i = 1, 2, ..., n)$
Set iteration counter $t = 0$
Compute the fitness value of each chromosomes
while $(t < MAX)$
Select a pair of chromosomes from initial population based on fitness
Apply crossover operation on selected pair with crossover probability
Apply mutation on the offspring with mutation probability
Replace old population with newly generated population
Increment the current iteration t by 1.
end while
return the best solution, Y_{bt}
end

Figure 3.10 GA Pseudocode

The algorithm iteratively applies the selection, crossover, and mutation operators to improve the population over successive generations. In addition, elitism strategy where the best solutions are preserved without modification, can also be incorporated to maintain high-quality solutions. The performance of GA is influenced by adjustment of the rates of selection, crossover, and mutation. These rates can be fixed or dynamically changed during the optimization process. The algorithm continues until an end criterion is met, and the best result in the final generation is returned as the best approximation of the global optimum for the given problem (Mirjalili, S., 2019).

With the context of warehousing, chromosomes (solutions) are generated by encoding the selected products through their indices within the generated subgraph. After generating the random initial population and selecting two solutions, the crossover procedure is executed by selecting a random index x, and switching the product indexes from 0 to x between the selected solutions and filling the remaining items at the same order as shown in Figure 3.11(a) and 3.11(b). Later, the mutation is done randomly by selecting two items (product indexes) and switching them as shown in Figure 3.11(c).

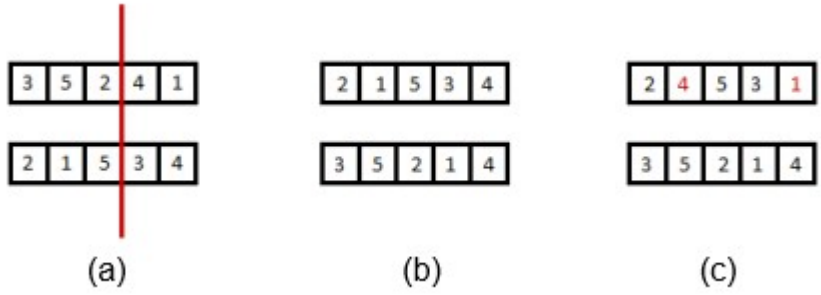


Figure 3.11 Crossover & Mutation Process in GA

3.3.1.2 Particle Swarm Optimization

PSO, a mathematical optimization algorithm introduced by James Kennedy and Russell Eberhart in 1995, draws inspiration from the concept of swarm intelligence. It is grounded in the behavior exhibited by groups of fish and bird, and it has been extensively employed to solve nonlinear, nonconvex, and combinatorial optimization problems across diverse domains. The motivation behind PSO lies in the social behavior of birds that form flocks for various benefits. Flocking allows them to share information about food sources and protect themselves from predators.

Unlike evolutionary algorithms in which individuals with lower fitness may be eliminated, PSO promotes cooperation among individuals throughout the search process. PSO focuses on aggregation and homing, where agents fly within a particular region while maintaining a maximum distance from each other and have the potential to reach the global optima (Bansal, 2019).

In PSO, the particles constituting the swarm continuously change their search pattern based on their own learning experiences and the experiences of fellow particles. The algorithm allows particles to update their positions and velocities according to environmental changes, while ensuring proximity and quality requirements. The swarm in PSO explores the solution space without limiting its movement, while maintaining stable movement and adapting to environmental changes.

The origin of the PSO algorithm can be traced back to the Boid model, which was designed to simulate bird behavior. The Boid model employs a simple rule, called the "nearest proximity velocity match", to synchronize the velocities of individual birds in a flock. This model inspired the development of the PSO algorithm, wherein particles are represented as points in a Cartesian coordinate system with velocities and positions. The algorithm updates the velocities and positions of particles based on their own optimal positions as well as the swarm's optimal position (Wang, D., Tan, D., and Liu, L. 2018). The algorithm starts with a random population (with size N) with initial positions X:

$X = \{x_1, x_2, x_3, ..., x_n\}$ and velocities V: $V = \{v_1, v_2, v_3, ..., x_n\}$.

The velocity of particles control their movement within the search space and consist of three components: inertia, cognitive, and social. The inertia component maintains the particle's previous direction to prevent abrupt changes. The cognitive element induces particles to return to their previously identified best locations. The social component drives particles towards the best location of the whole swarm or a local neighbourhood.

Based on these components, the velocity is formulated as in Equation (1):

$$V_i(t+1) = V_i(t) + c_1 r_1(pbest_i - X_i(t)) + c_2 r_2(gbest_i - X_i(t)) \quad (1)$$

where r_1 and r_2 are random decimal values in the range (0,1), $pbest_i$ represents the coordinates of the best solution visited so far by itself, whereas $gbest_i$ represents the coordinates of the best solution visited so far by any particle. c_1 and c_2 are integer values between 0 and 4 representing the cognitive and social scaling components respectively. During the search for the optimal solution, particles iteratively update their positions with the Equation (2):

$$X_i(t+1) = X_i(t) + V_i(t+1) \quad (2)$$

where t is the current iteration number and t+1 is the next iteration, and V_i (t+1) represents the position of particle i in the next iteration (Bai, 2010).

The algorithm can be represented with pseudocode provided in Figure 3.12 (Hudaib, A. A., and Hwaitat, A. K. A. L. 2017).

```
For each particle
   Initialize particle
END
Do
   For each particle
      Calculate fitness value
      If the fitness value is better than the best fitness value (pBest) in history
         set current value as the new pBest
   End

   Choose the particle with the best fitness value of all the particles as the gBest
   For each particle
      Calculate particle velocity according equation          (2)
      Update particle position according equation             (1)
   End
While maximum iterations or minimum error criteria is not attained
```

Figure 3.12 Particle Swarm Pseudocode

3.3.1.3 Harris Hawks Optimization

The Harris Hawks Optimization (HHO) algorithm is a swarm intelligence optimization algorithm proposed by Heidari et al. in 2019. The algorithm takes its cues from the hunting behaviour of Harris hawks in desert conditions, where food is scarce. In order to locate, encircle, attack, and successfully catch their prey, the hawks engage in complex behaviors of cooperative hunting and cooperation with one another. The HHO algorithm mimics this behavior, particularly the "surprise pounce" strategy, where multiple hawks collaborate to target and converge upon a single prey. The attack is executed swiftly, but based on the prey's escape capabilities, the hawks might engage in

several dives near the prey for a few minutes. The chase tactics employed by the hawks adapt according to the prey's behaviour and environmental factors.

The algorithm comprises three stages: the first stage, known as discovery or exploration (global search), followed by the second stage, known as transformation from exploration to exploitation, and finally, known as exploitation (local search). During discovery phase, the algorithm explores multiple locations and techniques to find relevant solutions. The transformation phase is crucial for the success of the algorithm and involves utilization of the prey's escape energy to shift from exploration to exploitation. The exploitation phase focuses on utilizing the identified resources effectively (Alweshah et al., 2022).

According to Heidari et al. in 2019, these stages can be defined as following:

Exploration phase:

This phase discusses the relocating strategy of hawks when searching for prey, which is based on two strategies. The first one focuses on how hawks identify prey based on the locations of actual participants X_i whereas the second one involves the way hawks locate the prey by perching on a random tree X_{rand} which is expressed in Equation (3):

$$X_i(t+1) = \begin{cases} X_{rand}(t) - r_1|X_{rand}(t) - 2r_2X(t)|, & q \geq 0.5 \\ X_{rabbit}(t) - X_m(t) - r_3(LB + r_4(UB - LB)), & q < 0.5 \end{cases} \quad (3)$$

In Equation (3), $X_i(t+1)$ represents the next location of hawks in next iteration, $X_{rand}(t)$ represents the current position of a random selected hawk, q, r_1, r_2, r_3, r_4 are random numbers in the range (0,1), X_{rabbit} is the current rabbit location which represents the best position of the hawks so far, LB and UB are the lower and upper bounds of the solution. X_m represents the average position of the current hawks population of size *N* which is calculated using Equation (4):

$$X_m(t) = \frac{1}{N}\sum_{i=1}^{N} X_i(t) \quad (4)$$

Transition from exploration to exploitation:

When the hawks start chasing the prey, the prey's energy start decreasing while escaping which is represented by Equation (5):

$$\mathrm{E} = 2\mathrm{E}_0\left(1 - \frac{\mathrm{t}}{\mathrm{T}}\right) \quad (5)$$

where E_0 is the initial state of the energy which is defined with a value inside the range (-1,1), t represents the current iteration, whereas T represents the total number of

iterations. When prey's energy *E* falls below 1 then the exploitation phase will start.

Exploitation phase:

In this phase, four strategies are applied based on both the escaping behaviour of the prey and pursuing approaches of the hawks. These strategies are mathematically formulated using two parameters: *r* which is a random number within the range of (0,1) representing the likelihood of successful escaping and *E* which denotes the energy level of the prey.

a. Soft besiege (When $r \geq 0.5$ and $|E| \geq 0.5$): implies that the prey is unable to escape effectively as its energy is drained while trying to escape from the hawks. This behaviour is represented by Equations (6) and (7):

$$\mathrm{X(t+1)} = \Delta\mathrm{X(t)} - \mathrm{E|J}X_{rabbit}(t) - X(t)| \quad (6)$$

$$\Delta\mathrm{X(t)} = X_{rabbit}(t) - X(t) \quad (7)$$

where *J* stands for prey's random jump power while trying to escape

$\mathrm{J} = 2(1 - \mathrm{r}_5)$ and r_5 is a random number within the range of (0,1)

b. Hard besiege (When $r \geq 0.5$ and $|E| < 0.5$): implies that the prey becomes extremely tired and its energy for escaping decreased significantly so that it is unable to escape successfully. This condition is represented in Equation (8):

$$\mathrm{X(t+1)} = X_{rabbit}(t) - E|\Delta\mathrm{X(t)}| \quad (8)$$

c. Soft besiege with progressive rapid dives (When $r < 0.5$ and $|E| \geq 0.5$): implies that the prey still possesses sufficient energy to escape but a gentle soft besiege is established by the hawks prior to the sudden pounce. Initially, the hawks make multiple moves and evaluate them (Equation 9). Hawks then evaluate the results of their own moves, if the outcome does not result in identifying the optimal dive towards the prey, the swarm tends to execute rapid dives using Levy flight (LF) to enhance the exploitation process (Equation 10).

$$Y = X_{rabbit}(t) - E|JX_{rabbit}(t) - X(t)| \quad (9)$$

$$Z = Y + S \times LF(\mathrm{D}) \quad (10)$$

In Equation (10) D represents the dimension of the problem, S represents a random vector by size $1 \times \mathrm{D}$ and LF represents levy flight function given by Equation (11):

$$LF(D) = 0.01 \times \frac{u \times \sigma}{|v|^{\frac{1}{\beta}}}, \quad \sigma = \frac{\Gamma(1+\beta) \times sin(\frac{\pi\beta}{2})}{\Gamma(\frac{1+\beta}{2}) \times \beta \times 2^{\frac{(\beta-1)}{2}}} \quad (11)$$

where *u,v* are random values within the range (0,1), β is a constant with a value 1.5. Based on the values of Y and Z which are defined in Equations (9) and (10), the

position at soft besiege with progressive rapid dives phase will be updated as shown in Equation (12) where F represents the fitness function:

$$X(t+1) = \begin{cases} Y \; if \; F(Y) < F(X(t)) \\ Z \; if \; F(Z) < F(X(t)) \end{cases} \quad (12)$$

d. Hard besiege with progressive rapid dives (When $r < 0.5$ and $|E| < 0.5$):

$$X(t+1) = \begin{cases} Y \; if \; F(Y) < F(X(t)) \\ Z \; if \; F(Z) < F(X(t)) \end{cases} \quad (13)$$

where Y and Z are obtained using Equations (14.a) and (14.b):

$$\mathrm{Y} = X_{rabbit}(t) \; - \; E|JX_{rabbit}(t) \; - \; X_m(t)| \quad (14.a)$$

$$\mathrm{Z} = \mathrm{Y} + \mathrm{S} \times \mathrm{LF(D)} \quad (14.\mathrm{b})$$

The algorithm's pseudocode is given in Figure 3.13:

Inputs: N and T
Outputs: prey location and its fitness value
Initialize $(X_i, i = 1, 2, 3, 4 \ldots, N)$
while (stopping condition is not met) **do**
Calculate the fitness values of hawks
Set $\mathrm{X_{prey}}$ as the location of prey (best location)
  for (each hawk (X_i)) do
    Update the initial energy $\mathrm{E_0}$ and jump strength J
    $\mathrm{E_0}$=2rand()-1, J=2(1-rand())
    Update the E using Eq. (5)
      if ($|\mathrm{E}| \geq 1$) **then**
        Update the location vector using Eq. (3)
      if ($|\mathrm{E}| < 1$) **then**
        if (r $\geq$0.5 and $|\mathrm{E}| \geq 0.5$) **then**
          Update the location vector using Eq. (6)
        else if (r $\geq$0.5 and $|\mathrm{E}| < 0.5$) **then**
          Update the location vector using Eq. (8)
        else if (r < 0.5 and $|\mathrm{E}| \geq 0.5$) **then**
          with progressive rapid dives
          Update the location vector using Eq. (12)
        else if (r < 0.5 and $|\mathrm{E}| < 0.5$) **then**
          with progressive rapid dives
          Update the location vector using Eq. (13)

Figure 3.13 HHO Pseudocode

3.3.1.4 Archimedes Optimization Algorithm

Archimedes' principle is a fundamental concept in physics and fluid mechanics that claims an object submerged in a liquid experiences an upward force referred to as the buoyant force which is equivalent to the weight of the fluid the object has displaced (Mohazzab, 2017). This principle inspired Hashim et al. to develop the Archimedes Optimization Algorithm (AOA) in 2021. The AOA maintains a balance between

exploration and exploitation, making it appropriate for resolving challenging optimization issues with multiple local optima.

In the AOA, multiple objects, representing candidate solutions, are submerged in the same liquid, each object aiming to attain a state of equilibrium. These objects have different densities, volumes, and accelerations. The equilibrium state is reached when the buoyant force equals the weight of the object, taking into account any additional forces such as collisions with neighboring objects.

The AOA is a population-based algorithm, similar to other metaheuristic algorithms. It starts with an initial population of objects, each with random volumes, densities, and accelerations. The fitness of the initial population is evaluated, and then the algorithm proceeds through iterations until a termination condition is satisfied.

In each iteration, the AOA updates the density and volume of each object and determines its acceleration based on collisions with neighboring objects. These updated parameters then influence object new location in the fluid. This iterative process is repeated to search for optimal solutions.

The algorithm starts with initializing the position of the objects using Equation (15):

$$O_i = lb_i + rand \times (ub_i - lb_i);\ i = 1,2,\ldots,N \quad (15)$$

where O_i is the position of object i in a population of N object, ub_i, lb_i represent the upper and bottom limits of the search space, respectively.

Each object has different initial density and volume which are assigned randomly as represented in Equation (16) and has an initial acceleration represented in Equation (17):

$$den_i = rand \qquad vol_i = rand \quad (16)$$

$$acc_i = lb_i + rand(ub_i - lb_i) \quad (17)$$

where ***rand*** is a D dimensional vector populated with random values within range (0,1).

After the initialization, the population is evaluated using the fitness function to get the best position x_{best}, and it's density, volume and acceleration: den_{best}, vol_{best} and acc_{best}.

Subsequently, the densities and volumes are updated using Equation (18):

$$\left.\begin{aligned} den_i^{t+1} &= den_i^t + rand \times (den_{best} - den_i^t) \\ vol_i^{t+1} &= vol_i^t + rand \times (vol_{best} - vol_i^t) \end{aligned}\right\} \quad (18)$$

In AOA, exploration to exploitation transition is maintained through the utilization of transfer operator TF, which represents the collision between the objects followed by their gradual movement towards an equilibrium state over time. The definition of TF is provided in Equation (19):

$$\mathrm{TF} = \exp\left(\frac{\mathrm{t} - \mathrm{t}_{\max}}{\mathrm{t}_{\max}}\right) \quad (19)$$

where t and t_{max} are the current iteration number and the total count of iterations. From Equation (19), of TF we can notice that the operator increases gradually until it reaches the value of 1. Another operator is used to assist the transition process which is the density decreasing factor d defined in Equation (20):

$$\mathrm{d}^{\mathrm{t}+1} = \exp\left(\frac{\mathrm{t}_{\max} - \mathrm{t}}{\mathrm{t}_{\max}}\right) - \left(\frac{\mathrm{t}}{\mathrm{t}_{\max}}\right) \quad (20)$$

The key difference between exploration and exploitation phases in AOA is the collision, which is represented by the value of TF. $\mathrm{TF} \leq 0.5$ denotes the occurrence of the collision occurs that corresponds to exploration phase. In this phase a random object *mr* is selected to collide (update) the other objects accelerations using Equation (21):

$$acc_i^{t+1} = \frac{den_{mr} + vol_{mr} \times acc_{mr}}{den_i^{t+1} \times vol_i^{t+1}} \quad (21)$$

When $\mathrm{TF} > 0.5$, there will be no collision meaning the exploitation phase. The acceleration in this phase is represented by Equation (22):

$$acc_i^{t+1} = \frac{den_{best} + vol_{best} \times acc_{best}}{den_i^{t+1} \times vol_i^{t+1}} \quad (22)$$

where *best* is the object with the best position according to the fitness function.

In both exploration and exploitation phases, the acceleration is normalized to calculate the change percentage using Equation (23). The normalized acceleration determines the step change of each object.

$$acc_{i-norm}^{t+1} = u \times \frac{acc_i^{t+1} - min(acc)}{max(acc) - min(acc)} + l \quad (23)$$

In Equation (23), u and l are constants with 0.9 and 0.1, , values representing the normalization range.

Finally, the position update for both exploration and exploitation phases are calculated with Equations (24), (25), (26) and (27) respectively:

$$\mathrm{x_i^{t+1}} = \mathrm{x_i^t} + \mathrm{C1} \times \mathrm{rand} \times acc_{i-norm}^{t+1} \times d \times (x_{rand} - x_i^t) \quad (24)$$

$$\mathrm{x_i^{t+1}} = \mathrm{x_{best}^t} + \mathrm{F} \times \mathrm{C2} \times \mathrm{rand} \times acc_{i-norm}^{t+1} \times d \times (T \times x_{best} - x_i^t) \quad (25)$$

$$\mathrm{T} = \mathrm{C3} \times TF \quad (26)$$

and F is a flag used to change the motion direction:

$$F = \begin{Bmatrix} +1 \; if \; P \leq 0.5 \\ -1 \; if \; P > 0.5 \end{Bmatrix} \text{ where p } = 2 \times rand - C4 \quad (27)$$

Where *C1, C2, C3* and *C4* are constants lying within values: (0,1), (2,4,6), (0,1) and (0.5,1) respectively. Figure 3.14 represents AOA Algorithm:

```
procedure AOA(population size N, maximum iterations t_max, C1, C2, C3, and C4)
    Initialize objects population with random positions, densities and volumes using (15), (16), and (17), respectively.
    Evaluate initial population and select the one with the best fitness value.
    Set iteration counter t = 1
    while t≤ t_max do
        for each object i do
            Update density and volume of each object using (18)
            Update transfer and density decreasing factors TF and d using (19) and (20), respectively.
            if TF ≤ 0.5 then  ▷ Exploration phase
                Update acceleration using (21) and normalize acceleration using (23)
                Update position using (24)
            else              ▷ Exploitation phase
                Update acceleration using (22) and normalize acceleration using (23)
                Update direction flag F using (27)
                Update position using (25)
            end if
        end for
        Evaluate each object and select the one with the best fitness value.
        Set t = t+1
    end while
    return object with best fitness value.
end procedure
```

Figure 3.14 AOA Pseudocode

3.3.1.5 Honey Badger Algorithm

The Honey Badger Algorithm (HBA) is a global optimization method introduced by Hashim e1 al., 2022. It mimics the honey badgers' foraging patterns and is designed to maintain a balance between exploration and exploitation, making it effective for tackling challenging optimization problems involving a large number of local optima.

The honey badger is able to track down its food source by moving slowly relying on its strong sense of smell. It digs multiple holes in a day within a large radius while foraging. Although honey is a preferred food for honey badger, it is not very good at finding beehives. On the other hand, the honey-guide bird, has the ability to detect beehives but is unable to reach the honey. Honey badgers and honey-guide birds have a symbiotic relationship where the bird takes honey badgers to beehives, and in return, the honey badgers use their sharp claws to break open the hives.

This collaborative effort is beneficial to both species.

HBA operates in two modes: the digging mode and the honey mode. During the digging mode, the algorithm approximates the location of the food source using exploration techniques. Once the location is reached, the algorithm explores the neighborhood to determine the best spot for obtaining the food. In the honey mode, the algorithm directly follows a guide to the target location, analogous to the way a honey badger following the honey-guide bird.

Similar to other metaheuristic algorithms, HBA starts with an initial population of N-sized solutions, where each solution represent a honey badger at a different location x_i. These solutions are generated with the Equation (28):

$$x_i = lb_i + r_1 \times (ub_i - lb_i) \tag{28}$$

where r_1 is a number chosen at random within the range (0,1), lb_i and ub_i represent the bottom and upper limits of the search space, respectively.

The next step involves calculating the intensity *I*, which is associated with the concentration strength of the prey and the distance that separates each honey badger from the prey. The intensity determines the motion speed of the honey badger, where higher smell intensity resulting in faster motion and vice versa. This is defined by Equation (29):

$$\begin{Bmatrix} I_i = r_2 \times \dfrac{S}{4\pi d_i^2} \\ S = (x_i - x_{i+1})^2 \\ d_i = x_{prey} - x_i \end{Bmatrix} \tag{29}$$

where r_2 is a number chosen at random within the range (0,1), *S* is the strength of the concentration and d_i refers to the distance between the prey and the ith badger.

Utilizing the density factor α ensures that the shift from exploration to exploitation is as smooth as possible. This factor is responsible for controlling time-varying randomization that goes down as the number of iterations goes up, leading to less randomness over time as shown in Equation (30):

$$\alpha = \mathrm{C} \times \exp\left(\frac{-\mathrm{t}}{\mathrm{t}_{\max}}\right) \tag{30}$$

where t and $t_{\max}$ represent the current and the total number of iterations respectively, *C* is a constant equals to 2.

The latest step in the algorithm is updating the agents' position. As previously mentioned, the process of updating the position is divided into two distinct phases:

Digging phase - which represents the exploration phase - where the badger moves in a Cardioid shape as shown in Figure 3.15. This can be simulated using Equation (31):

$$x_{new} = x_{prey} + F \times \beta \times I \times x_{prey} + F \times r_3 \times \alpha \times \mathrm{d_i} \times |\cos(2\pi r_4)| \times |1 - \cos(2\pi r_5)| \quad (31)$$

where x_{prey} represents the preys position (the best solution so far), β is a constant equals to 6 represents the ability of the badger to acquire food for itself, d_i is the amount of space that exists between the badger and its prey. r_3, r_4, r_5 are different random numbers within the range (0,1). F is a flag responsible for changing the search direction which represents a disturbance that the badger may face:

$$\mathrm{F} = \begin{cases} 1 \; if \; r_6 \leq 0.5; \; r_6 \text{ is a random number inside } (0,1) \\ -1 \; else \end{cases} \quad (32)$$

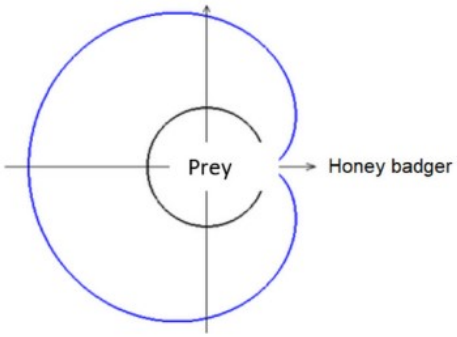


Figure 3.15 Digging phase

The other phase is the Honey phase where the honey-guide guides the badger to the beehive which can be simulated by the Equation (33):

$$x_{new} = x_{prey} + F \times r_7 \times \alpha \times d_i \quad (33)$$

where r_7 is a random number inside (0,1).

Figure 3.16 shows the discussed previous steps for HBA:

Set parameters t_{max}, N, β, C.
Initialize population with random positions.
Evaluate the fitness of each honey badger position x_i using objective function and assign to f_i, $i \in [1, 2, ..., N]$.
Save best position x_{prey} and assign fitness to f_{prey}.
while $t \leq t_{max}$ **do**
  Update the decreasing factor α using (30)
  for $i = 1$ to N **do**
    Calculate the intensity I_i using Eq. (29)
    if $r < 0.5$ **then** ▷ r is random number between 0 and 1
      Update the position x_{new} using Eq. (31)
    else
      Update the position x_{new} using Eq. (33)
    end if
    Evaluate new position and assign to f_{new}.
    if $f_{new} \leq f_i$ **then**
      Set $x_i = x_{new}$ and $f_i = f_{new}$.
    end if
    if $f_{new} \leq f_{prey}$ **then**
      Set $x_{prey} = x_{new}$ and $f_{prey} = f_{new}$.
    end if
  end for
end while Stop criteria satisfied.
Return x_{prey}

Figure 3.16 HBA Pseudocode

3.3.2 Random-Key Encoding

Since the warehouse picking path is a permutation problem where each solution contains an ordered set of integer numbers representing the nodes (products) index in the graph, an additional step is required after generating the initial/new population of solutions known as encoding step. The choice of encoding algorithm depends on the specific problem and search space. In our study, we have used the random-key encoding algorithm which is capable of transferring the solutions from continuous to discrete spaces (Ouaarab et al., 2015).

The encoding process involves assigning a random key value to each element or position in the solution. These keys determine the order or arrangement of the elements, thus representing a permutation. The mapping from keys to permutations can be based on various strategies, such as sorting the keys or using a specific mapping functions. To ensure a valid permutation, the keys are typically sorted or rearranged in ascending order. The corresponding permutation is then derived based on the sorted key values. This mapping guarantees that each element appears exactly once in the permutation (Gharehchopogh, and Abdollahzadeh, B. 2022).

Figure 3.17 illustrates an example of Random-Key encoding.

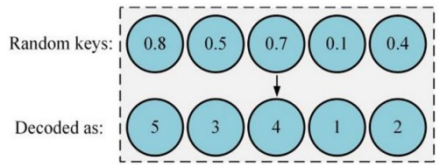


Figure 3.17 Example of Random-Key Encoding

3.3.3 Fitness Function

In the context of optimization and metaheuristic algorithms, a fitness function is a mathematical representation for the studied problem that is used to evaluate the performance or quality of a solution within the given problem domain.

The fitness function assigns a fitness value to each individual or candidate solution in a population, indicating how well the solution performs in relation to the problem's objectives.

Related to our warehouse, our object is to minimize the walking distances as much as possible, where the start and end point is section A1. This objecting can be represented

in Equation (34):

$$F(X) = \sum_{i=1}^{N-1} W_{x_i x_{i+1}} + W_{x_N x_1} \quad (34)$$

where $W_{x_i x_{i+1}}$ represents the edge weight from x_i to x_{i+1}, $W_{x_N x_1}$ is the edge weight from the latest product at the solution to the starting point (A1) and *N* is the dimension of our problem which represents the number of different product being collected.

4 EXPERIMENTAL RESULTS

To investigate the effectiveness of the examined algorithms, we conducted experiments using various random datasets of differing sizes. For each dataset size, we tested it with 50 different cases and the results were measured with the warehouse fitness function. The following section outlines the used parameters for each algorithm.

4.1 PARAMETERS SETTINGS

To ensure a fair comparison, all the algorithms were executed on $T_{max} = N * 10$ iterations, with the same population size N which is equal to the half of the number of nodes at the dataset. In addition, the experiments were conducted on a PC with windows 11 64-bit OS, AMD Ryzen 5 3600 CPU, 32GB of RAM. The algorithms were implemented using Python 3.10.

Regarding the algorithm specific parameters, we ran many tests to determine the best parameters for each algorithm. A total of 100 tests were performed over 50 size datasets chosen randomly and the results were as follow:

Genetic Algorithm

In GA, the crossover and mutation probabilities were explored within the range between 0 and 1. We ran the tests with (0.5, 0.6, 0.7, 0.8, 0.9) for crossover and (0.05, 0.1, 0.15, 0.2) mutation and found that the best combination was 0.8 for crossover and 0.05 for mutation. The detailed results can be observed in Table 4.1:

Table 4.1 GA Parameter Tests

Crossover Probability	Mutation Probability	Cost
0.5	0.05	895
0.5	0.1	958
0.5	0.15	974
0.5	0.2	983
0.6	0.05	895
0.6	0.1	945
0.6	0.15	971
0.6	0.2	983
0.7	0.05	894
0.7	0.1	947
0.7	0.15	972
0.7	0.2	983
0.8	**0.05**	**889**
0.8	0.1	953
0.8	0.15	969
0.8	0.2	986
0.9	0.05	899
0.9	0.1	942
0.9	0.15	967
0.9	0.2	987

Particle Swarm Optimization

In PSO, C1 and C2 are integers between 0 and 4. We did the tests for all the possible combinations and had the best results using C1 = 4 and C2 = 2. The detailed results can be found in Table 4.2.

Table 4.2 PSO Parameter Tests

C1	C2	Cost
0	0	1748
0	1	991
0	2	933
0	3	919
0	4	868
1	0	1647
1	1	802
1	2	813
1	3	868
1	4	818
2	0	1486
2	1	785
2	2	758
2	3	773
2	4	779
3	0	1361
3	1	759
3	2	702
3	3	743
3	4	759
4	0	1308
4	1	769
4	**2**	**678**
4	3	707
4	4	696

Harris Hawks Optimization

HHO is characterized by a single parameter, denoteed as E_0, which spans a range from 0 to 1. This parameter is assigned randomly, so there is no need to make tests for this algorithm.

Archimedes Optimization

In AO, we have 4 parameters to test: C1 in (0,1), C2 in (2,4,6), C3 in (0,1) and C4 in (0.5,1). So we tested all the possible combinations and got the best results using C1=1, C2=6, C3=1 and C4=0.5. The full results can be seen in Table 4.3:

Table 4.3 AO Parameter Tests

C1	C2	C3	C4	Cost
1	2	1	0.5	824
1	2	1	1	821
1	2	2	0.5	904
1	2	2	1	870
1	4	1	0.5	804
1	4	1	1	808
1	4	2	0.5	822
1	4	2	1	872
1	**6**	**1**	**0.5**	**784**
1	6	1	1	792
1	6	2	0.5	823
1	6	2	1	841
2	2	1	0.5	807
2	2	1	1	810
2	2	2	0.5	894
2	2	2	1	868
2	4	1	0.5	805
2	4	1	1	803
2	4	2	0.5	872
2	4	2	1	858
2	6	1	0.5	792
2	6	1	1	796
2	6	2	0.5	807
2	6	2	1	835

Honey Badger Algorithm:

In HBA, we have 2 parameters: C which ranges between 0.5 and 2.5 and β which ranges between 0.5 and 8. We did our tests for C across the range of (0.5, 1, 1.5, 2, 2.5), while β was evaluated within (0.5, 2, 4, 6, 8). Following the analysis, the most optimal parameter combination emerged as C=2.5 and β=0.5. The detailed findings are given in Table 4.4:

Table 4.4 HBA Parameter Tests

C	**β**	**Cost**
0.5	0.5	553
0.5	2	549
0.5	4	551
0.5	6	558
0.5	8	555
1	0.5	527
1	2	538
1	4	546
1	6	541
1	8	540
1.5	0.5	524
1.5	2	526
1.5	4	530
1.5	6	529
1.5	8	526
2	0.5	516
2	2	516
2	4	514
2	6	518
2	8	520
2.5	**0.5**	**504**
2.5	2	517
2.5	4	520
2.5	6	521
2.5	8	515

4.2 STATISTICAL RESULTS

Following the implementation of these five algorithms across datasets of dimensions 25, 50 and 75, different outcomes were obtained. We found that HBA exhibited the best performance for 25 dataset size with 332m. On the other hand, HHO was able to outperform the other algorithms for the larger size datasets with costs 503m and 586m. In addition, both HBA and HHO outperformed by more than 24% in the case of 50 sized datasets the other algorithms for 50 and 75 size datasets. In conclusion, we can use both of the algorithms according to the dataset size. Table 4.2 presents the results summary for the studied cases.

Table 4.5 Tests Cost Summary

Dataset Size	AOA		HBA		GA		PSO		HHO	
	conv	avg	conv	avg	conv	avg	conv	avg	conv	avg
25	195	398	157	332	151	411	141	402	102	339
50	431	767	336	504	261	887	264	669	265	492
75	668	1151	329	594	429	1411	429	967	165	573

4.3 CONVERGENCE ANALYSIS

To introduce a better understanding of how the studied algorithms are progressing through the iterations until the final result, the convergence curve which demonstrates the minimum value of the objective function for each population in comparison to the iteration number during the process of minimization should be examined. The faster convergence means that the algorithm was able to reach its best solution in lower number of iterations but does not necessarily refers to the superiority of the algorithm since our main reference is the value of the objective function itself.

The Harris Hawks Optimization (HHO) algorithm, for instance, exhibited rapid convergence, achieving minimal costs sooner compared to other algorithms for the 25-size dataset. Additionally, HHO demonstrated the best convergence performance for the 75-size dataset, coinciding with the attainment of the minimum cost—underscoring its overall superiority.

Conversely, Genetic Algorithm (GA) showcased superior convergence for the 50-size dataset, but resulted with the highest cost among the algorithms. This scenario underscores GA's under-performance for this particular dataset size. For a comprehensive summary of the convergence analysis, refer to Table 4.5. Figures 4.1, 4.2 and 4.3 augment the comprehension of the convergence dynamics across diverse dataset sizes, depicting sample convergence curves for the considered cases.

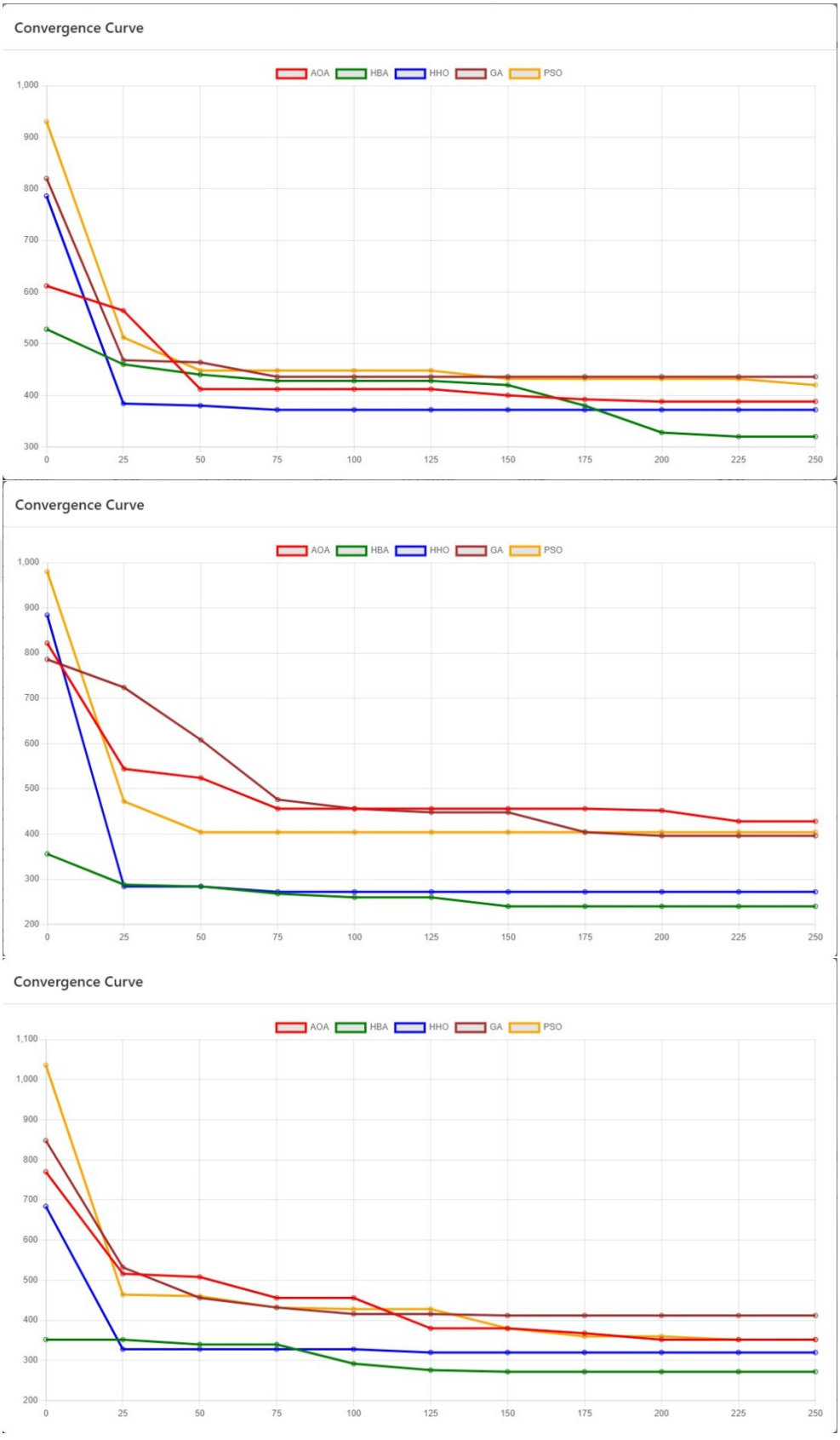


Figure 4.1 Convergence Curve Samples For 25 Size Datasets

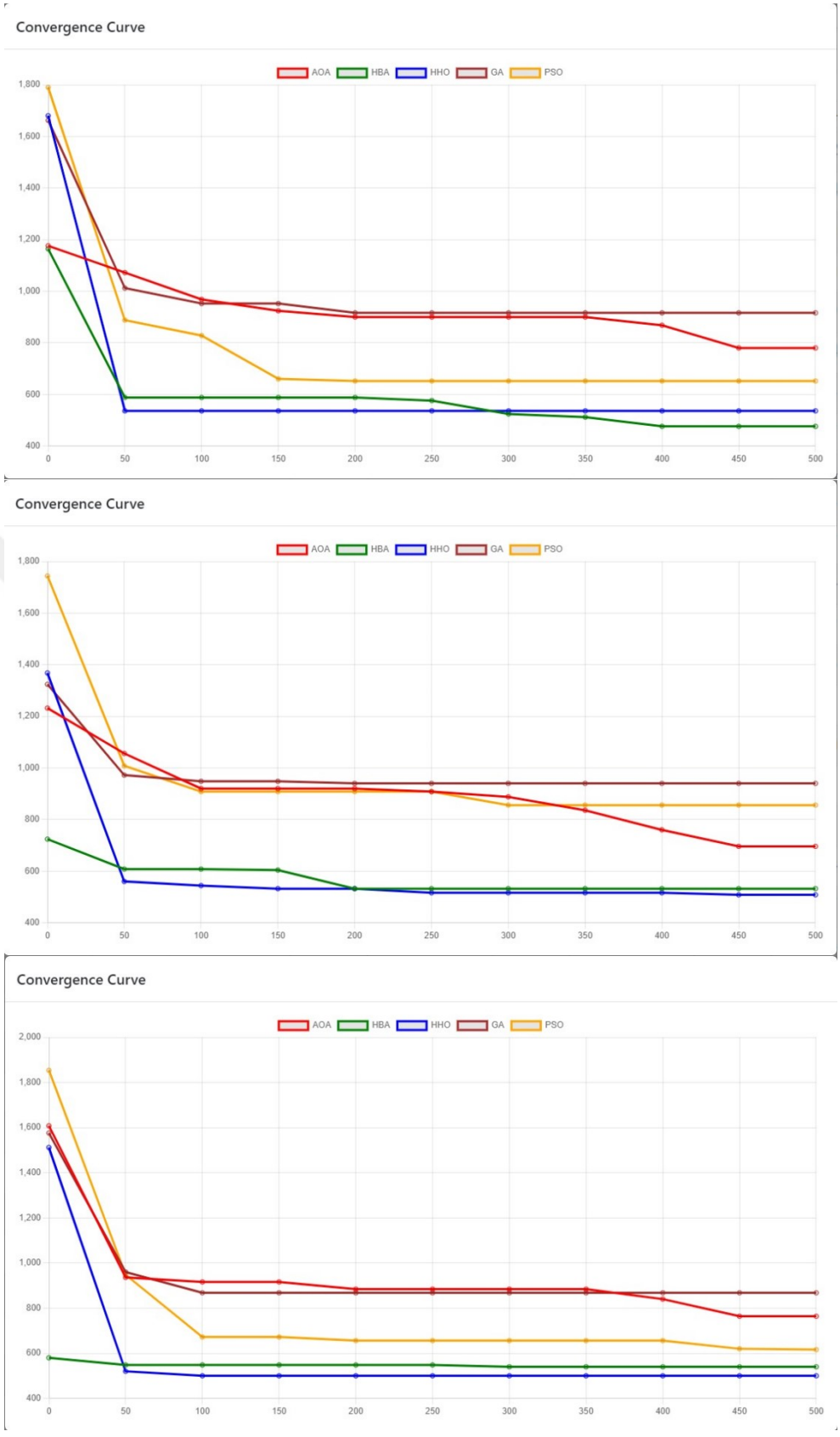


Figure 4.2 Convergence Curve Samples For 50 Size Datasets

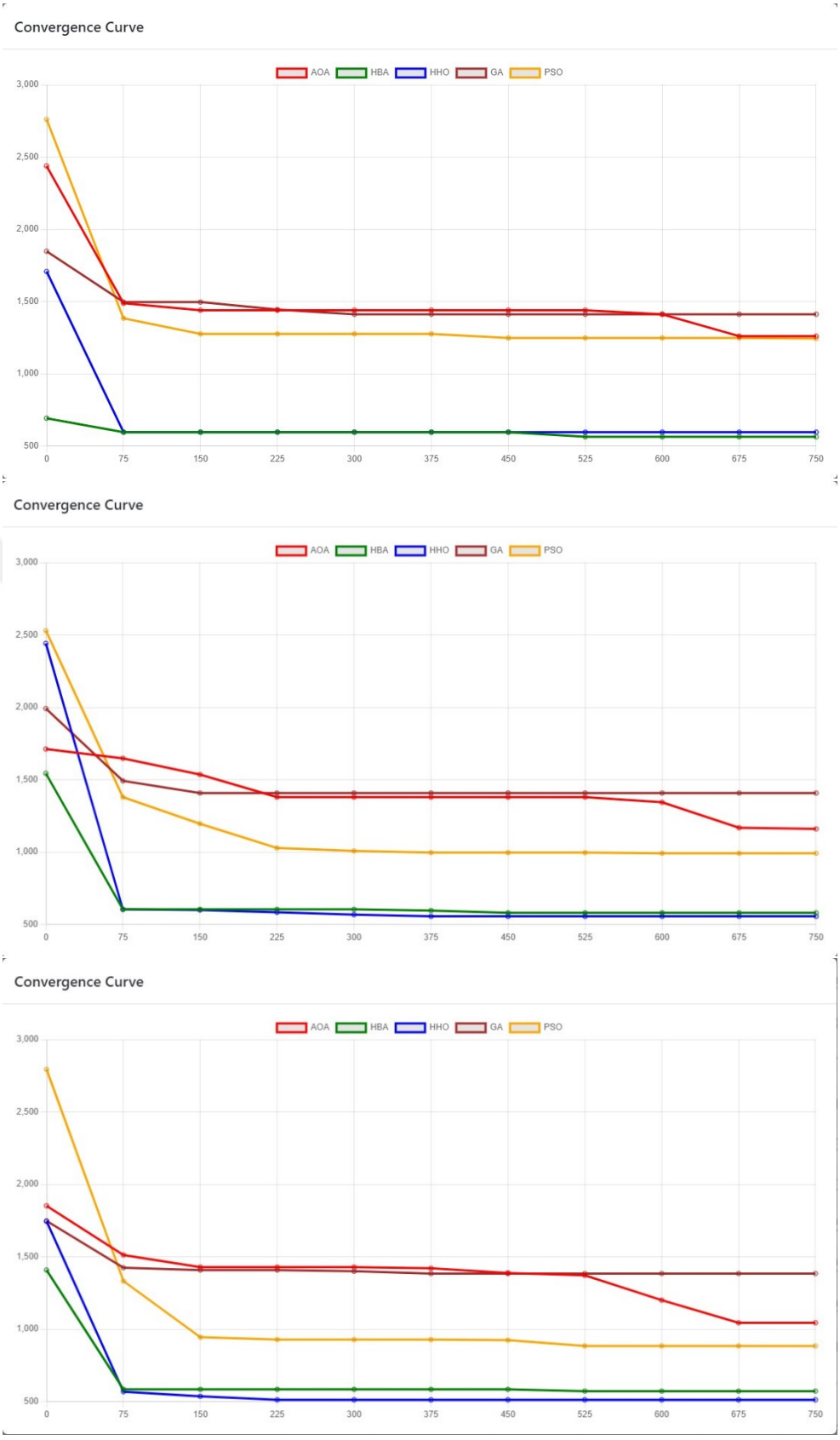


Figure 4.3 Convergence Curve Samples For 75 Size Datasets

4.4 DEMO APPLICATION

In this thesis, we also developed a simulation to model the optimization process for product picking paths. Our simulation was constructed using the Django framework for the API side, and we utilized JQuery and Bootstrap to enhance the user interface (UI). The application starts by loading and generating the warehouse graph, which is depicted in Figure 4.4.

The simulation offers the flexibility to enter the number of nodes (denoted as d) that need to be collected and the number of test runs to be conducted. Within each test run, we randomly selected nodes from the graph and consecutively apply all the algorithms to the same dataset of nodes. At the end of each test run, we recalculate and present the minimum, maximum, and average outcomes for each algorithm.

Furthermore, Figure 4.5 illustrates a comprehensive overview of the simulation results. In addition to the final cost corresponding to each algorithm, a more detailed breakdown is available. By clicking on the cost value associated with a particular algorithm, users can visualize the suggested path and even simulate the picking process itself.

This simulation framework serves as an interactive and insightful tool to analyze the performance of the algorithms in various scenarios, fostering a better understanding of their efficacy in optimizing product picking paths.

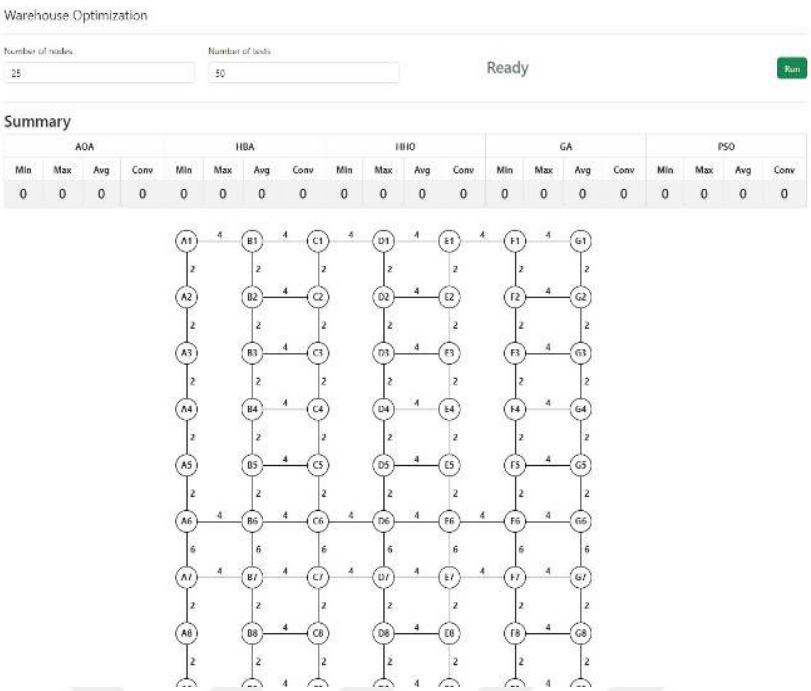
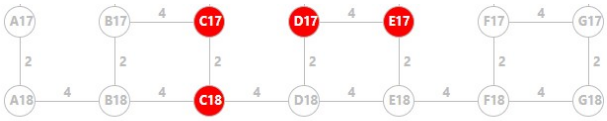


Figure 4.4 Demo Application



Results

#	Nodes	AOA		HBA		HHO		GA		PSO		
		Cost	Time	Cost	Time	Cost	Time	Cost	Time	Cost	Time	
1	Hide D6, F10, E14, B6, F3, E11, A7, A1, D7, D16, C15, F15, E10, A4, B7, E1, C11, D1, G15, G7, B14, A8, C9, E8, E9	352	0.15ms	304	0.17ms	276	0.19ms	372	0.04ms	360	0.13ms	Convergence
2	Show	356	0.18ms	332	0.20ms	328	0.23ms	400	0.05ms	404	0.15ms	Convergence
3	Show	396	0.18ms	356	0.20ms	336	0.22ms	444	0.05ms	392	0.15ms	Convergence
4	Show	658	0.18ms	332	0.22ms	340	0.24ms	416	0.06ms	396	0.15ms	Convergence
5	Show	400	0.18ms	280	0.21ms	368	0.24ms	368	0.05ms	400	0.15ms	Convergence
6	Show	416	0.18ms	328	0.21ms	316	0.23ms	408	0.05ms	476	0.16ms	Convergence
7	Show	384	0.18ms	308	0.20ms	348	0.23ms	376	0.05ms	400	0.15ms	Convergence
8	Show	368	0.18ms	296	0.20ms	324	0.23ms	412	0.05ms	388	0.15ms	Convergence
9	Show	436	0.18ms	344	0.20ms	384	0.23ms	408	0.05ms	428	0.15ms	Convergence
10	Show	368	0.18ms	368	0.20ms	380	0.23ms	432	0.05ms	440	0.15ms	Convergence
11	Show	432	0.18ms	368	0.20ms	380	0.23ms	416	0.05ms	456	0.15ms	Convergence
12	Show	380	0.18ms	372	0.21ms	368	0.23ms	428	0.06ms	392	0.15ms	Convergence
13	Show	400	0.17ms	264	0.20ms	296	0.23ms	384	0.05ms	420	0.15ms	Convergence

Figure 4.5 Detailed Test Results

5 CONCLUSION AND FUTURE WORK

In this thesis, we addressed the problem of warehouse picking path optimization. We first started by solving the shortest path problem to obtain the shortest path between each pair of nodes in the warehouse graph using Dijkstra's algorithm. Using these shortest paths, we created a subgraph that contains the desired nodes to speed-up the process of finding the best permutation of the nodes which leads to the minimum cost for picking the required products. Subsequently, we applied several metaheuristic algorithms—namely AOA, HBA, HHO, GA, and PSO to solve the TSP, a representation of the sequence in which the products should be picked to minimize the distance traveled. To obtain the best outcome, we conducted multiple tests to determine the best parameter values for each algorithm.

The results showed the ability of the new metaheuristic algorithms, namely HBA and HHO. These algorithms outperform the traditional algorithms like GA and PSO algorithms showcasing cost reduction more than 24%. This reduction can lead to faster and lower product picking effort in the logistics industry.

There are many promising directions that require further research. For instance, a hybrid version of HBA and HHO can be investigated to obtain lower picking costs for the solution of the same problem. Batching methods represent another avenue for research, breaking down the warehouse into smaller units to address the scale of the problem. In addition, more subproblems can be stated in the field of dividing the picking process between multiple product pickers.

REFERENCES

Abeledo, H., Fukasawa, R., Pessoa, A., & Uchoa, E. (2013). The time dependent traveling salesman problem: polyhedra and algorithm. Mathematical Programming Computation, 5(1), 27-55.

Aboelfotoh, A., Singh, M., & Suer, G. (2019). Order batching optimization for warehouses with cluster-picking. Procedia Manufacturing, 39, 1464-1473.

Agrawal, P., Abutarboush, H. F., Ganesh, T., & Mohamed, A. W. (2021). Metaheuristic algorithms on feature selection: A survey of one decade of research (2009-2019). Ieee Access, 9, 26766-26791.

Al Shammary, N. M., & Saudagar, A. K. J. (2015). Smart transportation application using global positioning system. International Journal of Advanced Computer Science and Applications, 6(6), 49-54.

Azadnia, A. H., Taheri, S., Ghadimi, P., Mat Saman, M. Z., & Wong, K. Y. (2013). Order batching in warehouses by minimizing total tardiness: a hybrid approach of weighted association rule mining and Genetic Algorithms. The Scientific World Journal, 2013.

Chen, F., Wang, H., Qi, C., & Xie, Y. (2013). An ant colony optimization routing algorithm for two order pickers with congestion consideration. Computers & Industrial Engineering, 66(1), 77-85.

Chen, F., Xu, G., & Wei, Y. (2019). An integrated metaheuristic routing method for multiple-block warehouses with ultranarrow aisles and access restriction. Complexity, 2019.

Cheng, C. Y., Chen, Y. Y., Chen, T. L., & Yoo, J. J. W. (2015). Using a hybrid approach based on the particle swarm optimization and ant colony optimization to solve a joint order batching and picker routing problem. International Journal of Production Economics, 170, 805-814.

Deo, N. (2017). Graph theory with applications to engineering and computer science. Courier Dover Publications.

Dokeroglu, T., Sevinc, E., Kucukyilmaz, T., & Cosar, A. (2019). A survey on new generation metaheuristic algorithms. Computers & Industrial Engineering, 137, 106040.

Ene, S., & Öztürk, N. (2012). Storage location assignment and order picking optimization in the automotive industry. The international journal of advanced manufacturing technology, 60, 787-797.

Gademann, A. J. R. M., Van Den Berg, J. P., & Van Der Hoff, H. H. (2001). An order batching algorithm for wave picking in a parallel-aisle warehouse. IIE transactions, 33(5), 385-398.

Hashim, F. A., Hussain, K., Houssein, E. H., Mabrouk, M. S., & Al-Atabany, W. (2021). Archimedes optimization algorithm: a new metaheuristic algorithm for solving optimization problems. Applied Intelligence, 51, 1531-1551.

Hoffman, K. L., Padberg, M., & Rinaldi, G. (2013). Traveling salesman problem. Encyclopedia of operations research and management science, 1, 1573-1578.

Huang, H., & Gartner, G. (2012). Collective intelligence-based route recommendation for assisting pedestrian wayfinding in the era of Web 2.0. Journal of location based services, 6(1), 1-21.

Javaid, A. (2013). Understanding Dijkstra's algorithm. Available at SSRN 2340905.

Key, R., & Dasgupta, A. (2015). Warehouse pick path optimization algorithm analysis. In Proceedings of the International Conference on Foundations of Computer Science (FCS) (p. 63). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Kordos, M., Boryczko, J., Blachnik, M., & Golak, S. (2020). Optimization of warehouse operations with Genetic Algorithms. Applied Sciences, 10(14), 4817.

Magzhan, K., & Jani, H. M. (2013). A review and evaluations of shortest path algorithms. International journal of scientific & technology research, 2(6), 99-104.

Mirjalili, S. (2019). Genetic algorithm. Evolutionary Algorithms and Neural Networks: Theory and Applications, 43-55.

Önüt, S., Tuzkaya, U. R., & Doğaç, B. (2008). A particle swarm optimization algorithm for the multiple-level warehouse layout design problem. Computers & Industrial Engineering, 54(4), 783-799.

Reda, M., Onsy, A., Elhosseini, M. A., Haikal, A. Y., & Badawy, M. (2022). A discrete variant of cuckoo search algorithm to solve the Travelling Salesman Problem and path planning for autonomous trolley inside warehouse. Knowledge-Based Systems, 252, 109290.

Yu, J., Li, R., Feng, Z., Zhao, A., Yu, Z., Ye, Z., & Wang, J. (2020). A novel parallel ant colony optimization algorithm for warehouse path planning. Journal of Control Science and Engineering, 2020, 1-12.

Zhang, D., & Zhang, J. (2021). Research on picking route optimization based on simulated annealing algorithm. In Journal of Physics: Conference Series (Vol. 1972, No. 1, p. 012086). IOP Publishing.

Zhao, L., & Zhao, J. (2017, December). Comparison study of three shortest path algorithm. In 2017 International Conference on Computer Technology, Electronics and Communication (ICCTEC) (pp. 748-751). IEEE.

BIOGRAPHY

Tarık HASSA

Tarık Hassa is a student of last year of Master's program of Computer Engineering at Beykoz University, Turkey. He completed his Computer Engineering and Automation bachelor's degree from Damascus University, Syria. He worked as a software developer at an e-commerce company called Incehesap and currently is a full-stack web developer at Thomson Reuters. He is interested in music, especially the oud and like to travel to different countries and meet people from different cultures.