Kingdom of Saudi Arabia
Ministry of Education
College of Computer
Computer Science

المملكة العربية السعودية
وزارة التعليم
كلية الحاسب
قسم علوم الحاسب

LIBRARIAN ROBOT USING SEARCH-BASED ALGORITHMS

Students

Abdullah Sulaiman Alhabib **371111359**
Abdullah Rashed Alhasan **382124116**

Supervisor
Dr Faisal Alhwikem

This project report is submitted to Qassim University in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science

Qassim University - Saudi Arabia
1443 (2022)

CERTIFICATE

It is certified that this project report has been prepared and written under my direct supervision and guidance. This project report is approved for submission for its evaluation.

Dr Faisal Alhwikem

DEDICATION

To my teacher, my light, my mother, may Allah have mercy on her: You may have left us physically, but your influence is carried for the rest of my life; I will make you proud.

Abdullah Alhabib

To my teachers, my family, my parents you have been my inspiration my whole life you have pushed me to do better, believed in me and I can never repay you all I can say is thank you.

Abdullah Alhasan

CONTENTS

LIST OF TABLES

List of Figures

ABSTRACT

Humans aspire to be more and more efficient in all everyday tasks. We appreciate making complex tools that do very specific tasks to save time, money, and energy. Now some of that time, money, and energy are being mismanaged by having people do manual and dull tasks that can be automated and done by robotics; heavy objects that need to be moved from one point to another, sewing, filtering, etc. Thus, tasks can be automated and optimized using multi-objective search-based algorithms that can be embedded into robotics to do specific work. This research project takes this narrative and models a system in which a librarian robot sorts library resources such as books, journals, etc. into shelves without human intervention. The project report considers energy consumption and shortest path computation of objectives into the planned optimization algorithm and goes ahead to distinguish related requirements and design elements to the project proposal.

CHAPTER ONE

INTRODUCTION

1.1 Overview

There is no doubt that computers and robotics play a significant role in our everyday lives. They are adopted for the everyday task that we do, for example, for transportation, for industrial factories, and even for critical tasks such as medical surgeries. The adoption has become in recent decades more popular because of an extensive range of dedicated research work. Computers and robotics can be programmed to do tasks that can be done or impossible to be done by humans.

1.2 Project Scope

This project aims to model a librarian task of sorting library resources using an EV3 LEGO Mindstorms robot. The task can be objectively optimized for energy consumption and the shortest path to target shelve locations. It can be extended to include further objectives.

For optimization, this project considers search-based algorithms to use for optimizing the robot performance according to energy consumption and optimal path.

1.3 Aim and Objectives

As mentioned in the last section, the aim is to model a librarian robot system that can accomplish the sorting of library resources in the most efficient manner. Resources can have weights that can contribute to the robot's energy consumption. The aim is to sort these resources back to shelves as short trips as possible. The task must be carried out and achieved the best outcomes using multi-objective search-based and optimization algorithms. The objectives of this project, therefore, are:

- Create a simulated library environment
- Create a small physical robot model
- Have a system that can fetch instructions for the robot
- Utilize optimization techniques to efficiently sort resources
- Provide a Graphical User Interface to interact with the robot

1.4 Motivation

Tedious and repetitive tasks are one type of many types that are commonly targeted and automated by computers and robotics because they often do not require human intelligence to be accomplished. The librarian's task of sorting resources can be considered a dull and tedious task to do. With the necessary information and using task optimization techniques, the task can be automated and done using robotics. However, optimization required objectives and some parameters to accomplish a task in the most effective and efficient manner; and that can be a challenge.

This project is intended to solve the task using an appropriate search-based algorithm and this motivates this project to go on this path. Also, this project indented to model the solution of paths to resource locations using a robot.

1.5 Project Plan and Schedule

A very important element to development and one of the main software development factors that we wish to improve on is workflow and roadmaps, for this project we have

written down a project schedule and have tried to adhere to, but while we have tried to give ourselves more time than we think needed, we ended up underestimating the amount of research needed, along with the unpredictability of other courses, although we only missed the plan by two weeks.

Task	Planned weeks	Actual weeks
Project planning and setup	1-2	1-2
Research	3-9	3-11
Requirement engineering	10-11	12-13
System design	12-13	14-15
Report fixes and finalization	14-END	16-END

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

Looking for techniques to automate aspects of our everyday tasks has always been a topic of research and innovation. If we take a self-automatic parking system that is nowadays fitted in modern automobile cars, it has been evolving since the beginning of this century through rigorous experimentation and testing towards the best outcomes.

Researchers and innovators have targeted tasks that are dull and do not require sophisticated intelligence and processing of a large amount of data. Examples of such tasks include autopilot systems, autonomous self-driving vehicles, and robotic vacuum cleaners. This research project contributes to this trend and models a robotic system in which optimization techniques are used for the multi-objective task management of a librarian.

This section describes the state-of-the-art optimization techniques, their strongholds, and weaknesses and provides a comparison of reviewed techniques. In addition, this section reviews any related work in optimization techniques and their usage to operate robotics to perform task management.

2.2 Problems and Optimization Algorithms

Computer-based problems can be categorized into three main groups: trackable, decision, and optimization [1]. Trackable problems are those that can be solved in polynomial time, whereas decision problems are those with only true or false outputs. Optimization problems are those concerns about maximizing or minimizing real function

outputs by selecting from a set of a permitted range of values. The selection is carried out using various optimization algorithms, which is one pillar focus of this research project.

Some problems are very hard to solve and therefore need to be reduced [1]. This means either slicing the problem into smaller, more manageable problems or ignoring factors that should not affect the result too greatly.

The problem can be reduced in a systematic way to apply tested techniques. For example, a problem of a robot in a library that moves books between shelves and tables can be reduced to a popular problem like the traveling salesman problem where a salesman wishes to find the most optimized way to go between cities (nodes) [1]. The next section presents this problem in detail.

2.2.1 Traveling Salesman Problem

We start by illustrating the usability of optimization algorithms in one of the famous problems that we usually come across in the computer science field: the Traveling Salesman Problem. It is a problem where imaginable salesmen wish to find the most optimal way to go between cities (represented as nodes) without revisiting a city again. The problem is considered NP-hard (a class of specification that is used to label problems that are non-deterministic polynomial problems) [2]. The traveling salesman problem (TSP) is a well-known, historically significant, and extremely difficult combinatorial optimization problem and it has applications in different fields of engineering, operations research, discrete mathematics, graph theory, and other subjects.

TSP has received a great deal of attention in the last three decades, and several ways to solve it have been developed, including branch, and bound, cutting planes, particle swarm, simulated annealing, ant colony, neural network, tabu search, and genetic algorithms, among others.

It is a generalization for problems wherein a need for optimizing paths is present. Thus, it is the most relevant generalization to this project and its objectives since this project

models and optimizes pathing of traveling robot in a library and navigate between shelves.

There are two main types of TSP. One type is symmetric and asymmetric, and the other type is Euclidean and Dimensional. The former type concerns with the distance between nodes. In symmetric the distance is equal. Where in asymmetric the distance is not necessarily equal. The latter type concerns with constraints. In Euclidean problem only one constraint is considered. However, when the problem considers multiple constraints such as geographical distance, time, and cost then the problem become dimensional type problem [2].

2.3 Algorithm Comparisons

There are numerous algorithms that are useful in some situations but become useless in others. Thus, choosing an optimization algorithm to apply must be carefully studied. Algorithms are divided into two categories, algorithms that only assess complete solutions, and algorithms that need partial or estimated solutions to be evaluated [1].

You can stop an algorithm that treats entire solutions at any time, and you will always have at least one possible answer to try. In contrast, if you stop a partial-solution method, you might not be able to use any of its output at all.

With Algorithms that only assess complete solutions, we can simply compare two entire solutions using an evaluation function; many algorithms rely on such comparisons, manipulating only one complete solution at a time; after evaluation, we can look for the best solution.

Partial solutions come in two forms: an incomplete solution to the problem, or a complete solution to a reduced (simpler) problem. Incomplete solutions only use a portion of the total search space (an example would be in the travelling salesman problem, we consider only nodes that have X quality, this only takes a subset of the nodes that have X from the set of all nodes). A reduced problem is one in which the main problem has been broken down into a series of smaller and easier problems. We solve

each problem to obtain a partial solution, then aggregate all partial solutions to obtain the final answer.

In our project, we avoid algorithms that only partially resolve problems because if you do, you'll have to create a system for organizing the subspace so that they may be effectively searched and create a new evaluation function that can judge the quality of partial solutions, which is difficult to do in real-world situations. Therefore, we have organized a set of criteria to choose an optimization algorithm:

1. Hard criteria: stops/target places are finite. The robot will have a preset starting position (the place it's already in, it can't jump quickly to another starting position). The target/s cannot be beyond the borders of a library. The paths must respect obstacles and shelves.

2. Soft criteria: there should not be more than a few stops at one time (the exact number needs studying). Estimated travel should not take more than a specified amount of time (the exact amount needs to be studied).

3. Complexity criteria: how many calculations do an algorithm need; a simple big O notation will suffice.

Based on the above criteria, we have found four algorithms and techniques that satisfy the objectives of this research project.

2.3.1 Divide and conquer

This algorithm breaks the original problem into multiple sub-problems until it can be solved by “hand”, sometimes you lower your chances of getting only one solution with this approach. Other options are Lin-Kernighan algorithm (LK), and adaptive resonance theory (ART). But some think the best approach is a combination of the two for larger TSP problems.

2.3.2 Dynamic programming

A problem is decamped to multiple decisions at various stages. We can use a recursive call with every state or city starting from the beginning and choosing the lowest cost and

best decision, or we can go backward by starting from the end and checking the best decision for the second to last spot and so on until we reach the current state which would speed it up.

2.3.3 Greedy algorithms

These algorithms attack the problem by constructing the complete solution in a series of steps. A greedy algorithm assigns values to the decision variables at every step and makes the best decision. Greedy algorithms assume heuristics for decisions and usually is shortsighted.

SAT: some solutions for short sighting either consider the lowest common variables first or consider the length, and frequency, even with the solutions it does not solve all the problems. There is no Greedy algorithm for SAT.

TSP: the most intuitive Greedy algorithm for TSP is based on the nearest neighbor heuristic. There is often a high price to pay for choosing the greediest decision at the beginning.

Another approach is assigning the main city and going from it to another city then returning and going to another unvisited city and returning. This approach can also consider not returning to the main city if needed.

NLP: there is no efficient greedy algorithm for NLP, we can design an algorithm that displays some greedy characterization.

2.3.4 Genetic algorithms

Genetic algorithms apply optimization strategies by simulating species evolution through natural selection. Genetic algorithms generally consist of two processes, the selection of individuals to produce the next generation. the manipulation of individuals chosen to form the next generation with crossover and mutation techniques.

The major objective of the selection mechanism is to determine the greatest individual with the best probability of becoming a parent, as well as how many children each

individual produces. Genetic algorithms are well-known for being an excellent method for optimizing TSP.

To create a pure genetic algorithm, Michalewicz and Fogel (2000) have proposed the following steps [1]:

1. Create an initial population of P chromosomes
2. Assess each chromosome's fitness.
3. Using proportional selection, select P/2 parents from the current population.
4. Using the crossover operator, choose two parents at random to generate offspring.
5. To make modest adjustments to the results, use mutation operators.
6. Steps 4 and 5 should be repeated until all parents have been chosen and matched.
7. Replace the old chromosomal population with a new one.
8. Assess each chromosome's fitness in the new population.
9. If the number of generations reaches a certain limit, stop; otherwise, continue to Step 3.

Selection criteria, crossover, and mutation are all important operators in GAs and affect how they perform. Thus, selecting mutation operators and crossovers is very critical, and therefore, it should be carefully studied.

2.4 Related Work

A paper by Nurdiawan [3] shows the use of genetic algorithms to solve TSP, where they have multiple points. They want a tourist to visit but the tourists are unable to finish all the points in one try due to the lack of time and the long distance. Thus, they have used a genetic algorithm to find a solution to the long distance. They claim they were able to reduce the distance from 170,706 KM to 17,706 KM.

Another paper by Cheng et. al. (Cheng, 2020) tries to find the best way for path plan on a multi-objective reconfigurable robot. These robots are used in rough environments, and path planning is a problem they faced. Through their research and implementation,

they have found that instead of executing a greedy search for the shortest distance to the next configuration, the algorithm can determine plausible paths in maps with complex obstacle setups.

2.5 Summary

The travel salesman problem is a problem affecting multiple fields and has been studied for decades. Choosing a method to solve it is not as easy as it seems. There is no perfect choice or complete method. Choosing genetic algorithms, divide and conquer algorithms, or dynamic programming algorithms or any other option depends on a set of objectives and requirements. Optimization methods are good when applied to the right problem.

CHAPTER THREE

PROBLEM ANALYSIS

3.1 Introduction

To create an efficient, safe, and effective task management system, we must properly engineer the system by making sure we understand our goal, how we reach it, and where to start. As such, this chapter provides the problem specification that this project is trying to solve, the way we chose to solve them, and list all requirements, whether functional or non-functional.

The functions in this chapter are only there to give an idea of the functional requirements presented in a way that is both easily written and understood, and do not represent the implemented function names.

3.2 Problem Specification

The problem that this research project addresses is to model the task of arranging items back on the shelf in a library. The task should be accomplished in the most efficient and convenient manner considering the distance of travel and the weight of items to accomplish the task to save time and energy consumption, respectively. This should be achieved using optimization techniques.

Reducing the energy cost is, arguably, more important than saving time by reducing the distance traveled, since saving energy also contributes to saving overall time by reducing

overhead from having to charge the robot in the middle of its work. This factor becomes less and less relevant the more battery capacity or the more energy efficient the robot is. This project, however, aims to increase that efficiency in a generic situation.

Optimizing the process to save both time and energy requires us to look beyond simple single windowed techniques like greedy algorithms or divide and conquer, but instead utilize multi-windowed algorithms, and therefore, this research project intended to use a genetic algorithm for optimizing and modeling the sorting task.

3.2.1 System overall specification

The system will only work in specific environments and only after it is provided with specific information (inputs) on its resources. These requirements cannot be expressed in functions, nor can they be tested and measured. Therefore, they are not functional nor non-functional requirements but merely system specifications, such specifications are:

1. The library resources the robot is meant to sort must be provided in an XML format file and must be loaded and processed by the robot.
2. The initial robot position and idle state is always (0,0)
3. The resources must be represented as nodes
4. There should be two types of paths. One is the paths that represent the corridors or *lanes* on the map (library). The other one is the *traversing path* that the robot is going to take (this path must adhere to the paths of lanes).
5. The resources must be organized (put back on the shelves). This is modelled as the robot visiting the nodes only once. Also, the path to a visited node must not be retaken again by the robot.
6. The traverse path to visit the nodes must be optimized to the best output using a genetic algorithm. The optimization must consider the distance and weight of the resources.

3.2.2 Movement cost

The estimated energy cost of each 1 cm movement of the robot is modeled in the table below.

Resource weight (grams)	Cost (Joule)
1 - 50	10
51 - 100	20
101 - 150	30
151+	40

This will directly affect the working of the genetic algorithm in the system, though further testing of power costs will make the algorithm more effective, this is to be considered in the next phase of this project.

3.3 Resources Specifications

This section identifies resources that need to be provided in order to run the system as it is planned, the following is a list of these resources and coming subsections (3.3.x) will provide details:

1. A library map.
2. Items that are to be delivered.
3. An EV3 robot.

3.3.1 The map

The model robot will run on a physical miniature map of a library with paths the robot's sensors can follow, we're currently thinking that it will be a 2x2m map.

The map must be provided to the system through the GUI as an XML file containing coordinates for the shelves and other locations in the library that the robot is meant to move to (such as charging and idling spots).

3.3.2 The items

The system will not physically deliver items but will go to their locations following an efficient path chosen by the genetic algorithm. The system needs Item information such

as location (x and y coordinates), item weight and name to be provided through use of the GUI as an XML file.

3.3.3 The robot

An EV3 robot will adhere to the instructions given by the system and must be able to move, follow a path using censors, and have a wireless connection to the system.

3.4 Functional and Non-Functional Requirements

This section will describe the various functions that will be implemented into the system in the functional requirements subsection, along with the needed input, output, and sometimes use cases for specific functions when needed. Then specify non-functional requirements afterward, which are requirements that are measurable and testable but do not have functional expression.

3.4.1 Functional requirements

The following table describes the various functional requirements the system needs functions. Functional requirements specify system needs that can be expressed as a function.

In the table, **func#** refers to the unique function number that is split into three parts, F refers to it being a functional requirement, the second part is the module of the system to which this function belongs, and the third and final part is the function number within the module. For example, F1.5 refers to the 5th function in the main controller module.

The table, however, does not represent the implementation of the functions written therein, and code implementation was specified in chapter five section four.

Func#	Function ID	Description
1. Main Controller functional requirements		
F1.1	start	Start the software
F1.2	Acquire list of books	Get a workable list of books from input XMI file.
F1.3	Set up algorithm	Set the algorithm up with the proper settings that would result in the best outcome for book delivery
F1.4	Run Genetic Algorithm	Run the genetic algorithm, with input from F1.2.
2. Optimization algorithm functional requirements		
F2.1	Encode input as a workable problem	Take the input list of books and encode it as a solvable problem.
F2.2	Specify parameters for the algorithm	Specify parameters to properly run the algorithm, such as population size and number of evaluations.
F2.3	Create offspring	Generate offspring according to genetic algorithm.
F2.4	Evaluate and rank offspring	Evaluate each child chromosome and rank them based on the appropriate fitness function.
F2.5	Run the crossover	Do two-point crossover on the offspring based on their rank.
F2.6	Get solution	Return the top ranked solution as output

3. Input/Output functional requirements		
F3.1	`Get input from file`	Read XMI file and get book input
F3.2	`Give book information`	Return, when requested, relevant book information (such as coordinates, book name and author)
F3.3	`Provide output`	Provide testable output after running the algorithm

3.4.2 Non-functional requirements

While the specifics of non-functional requirements are not completely agreed upon by software engineers, they are of paramount importance to a system, and in this subsection, we describe system requirements that can be measured and tested but cannot be expressed as a function [4].

The structure of the table below is like the one above describing functional requirements, wherein **Req#** is the unique number for the requirement description.

Req#	*Requirement*	*Description*
1. Robot non-functional requirements		
R1.1	Power efficiency	The system must increase the robot's power efficiency by picking efficient routing.
R1.2	Time efficiency	The robot delivery must make sure not to waste too much time in its pathing, the algorithm must strike a good balance between saving time and saving energy
2. System non-functional requirements		
R2.1	Readability	The output must be clear to users of all reasonable backgrounds, as well as clarity on what each element means and is used for.
R2.2	Responsive	Users need immediately responsive feedback to their commands.
R2.3	Availability	Achievable by minimizing errors and testing the system thoroughly and vigorously in different environments.

R2.4	Security	The system must be secure and must not be a vulnerability to another system, this can be achieved by the system not having (nor needing) writing permissions or database access.

CHAPTER FOUR

SYSTEM DESIGN

4.1 Introduction

To create a robust, secure, and easy to develop a system, that system must be planned and designed with system evolution and compatibility in mind; this chapter goes over the design of the proposed system through architecture and UML class, state, and sequence diagrams, all of the diagrams in this document are made using the lucid app.

And although that planning is important, it is also imperative that we keep flexibility in system implementation. Since complete foresight is unrealistic and impossible, the design in this chapter is an approximation for the implementation and will not represent the exact result.

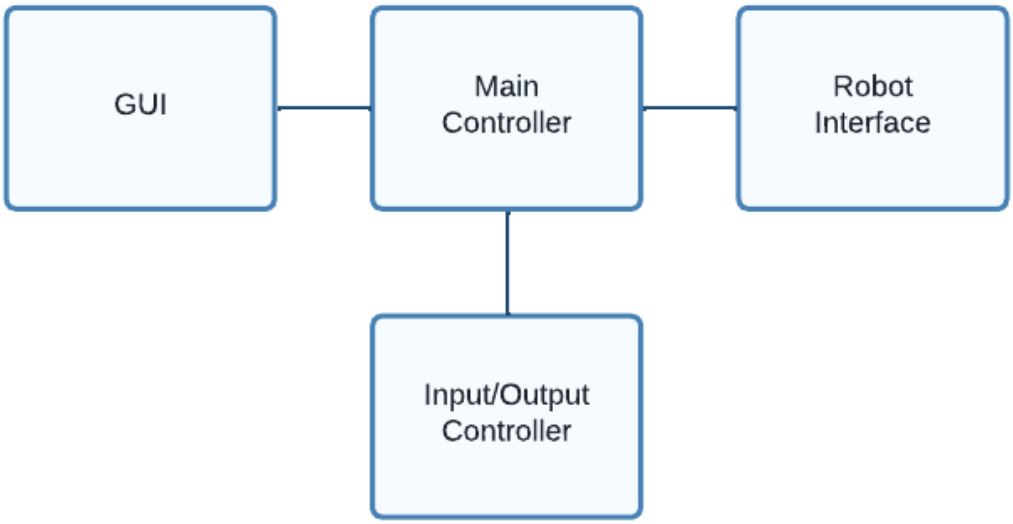
The system is split into components for good development and to produce a quality system that is easy to maintain and develop further. To design each component, we need to theorize how they will work and interact together through use case and sequence diagrams, and these are going to clarify the potential and objectives of each component as well as help imagine the system in real use.

This chapter focuses on the development of the necessary diagrams and models that will be used for implementation in phase two of this project, and this chapter is designed to be a roadmap for system implementation.

4.2 System Architecture

System architecture is the design of system components and the overall idea of how modular the system is going to get, what each module does and interactions with other systems (the robot in this project).

The system architecture for this project is divided into four components. The first component is the main controller which controls the flow of data between components. The second component is the input/output controller that is concerned with data retrieval from the library database (basically XML reader and writer). The third component robot integration component that include necessary interface between the software part and hardware part of the physical robot. The last component is the GUI component which connects the software with a graphical user interface application for direct input from users.

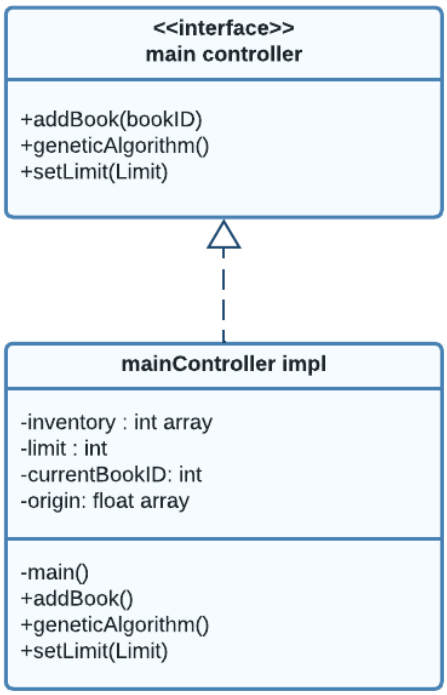


4.3 UML Class Diagrams

UML class diagrams are a universal way to communicate general implementation strategies of classes and their contents. Such descriptive diagrams will help in the implementation stage of phase two of this project. The following subsections contain diagrams of the different components of the proposed system.

Each subsection describes the module's functionality along with the functionality of each attribute. To avoid redundancy, the subsections will not describe module functions, since that is detailed in section 3.4.

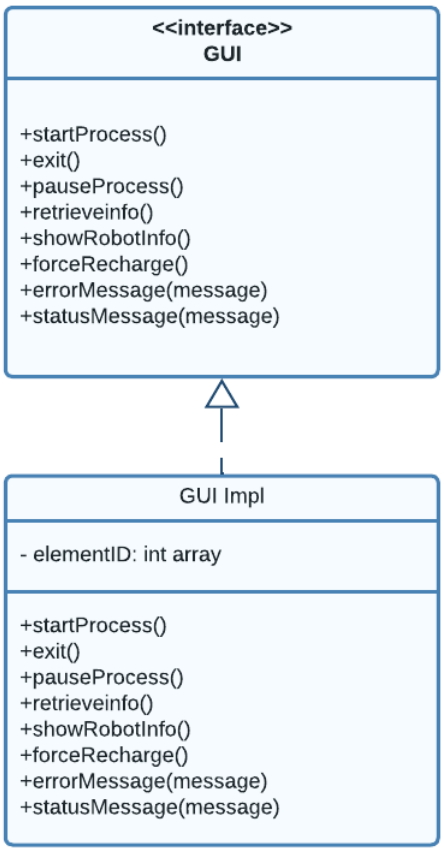
4.3.1 Main Controller



The main controller components control the flow of the system, as the figure shows, the attributes of this module are: inventory, which contains a list of books for the genetic algorithm to iterate through, limit, which is a parameter for the limit of books that can be iterated through, then comes currentBookID, the ID of the book that is being iterated through in the genetic algorithm, then finally comes origin, which is the attribute that holds the robot's idle coordinates.

As the system starts, the main function calls `checkCoords()` to set the origin attribute, which contains the coordinates for the robot's idle position. Then start a loop and wait for `startProcess()` to be called, then begin by calling `addBook()` from the list of books in the file selected, then call `geneticAlgorithm()`, then send the list of coordinates to the robot through calling `startDelivery()`, sending successful delivery through a `statusMessage()` call, and looping back to the beginning until `exit()` or `pauseProcess()` are called.

4.3.2 GUI

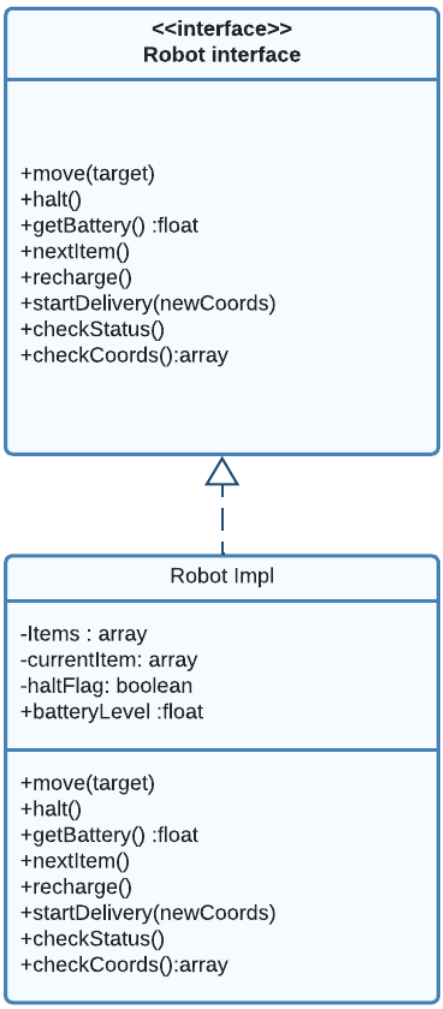


Direct user interaction should be allowed to functionalities that are necessary to load XML files containing book and map information, starting, and terminating the robot remotely.

The GUI should be designed and implemented using an Event-Driven approach and should be extensible to allow more integration of new features and the system evolution, the UML class on the left shows only the functions necessary for the GUI to contain, the detailed design of the GUI will be determined in phase two.

As for the attribute elementID, it will hold the ID of each object of GUI elements.

4.3.3 Robot interface

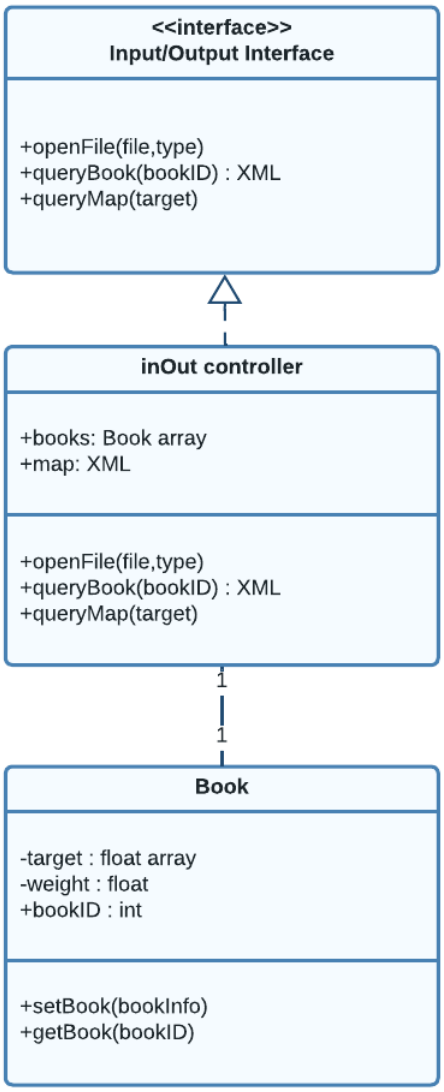


This module will be the interface between the software and the robot's API, the Application Programming Interface provided by the developers of the robot.

The attribute *items* hold a two-dimensional array of coordinates the robot will deliver to, the method of iterating through this array will be either through saving it all in the robot and iteration within the robot, or that the software part will send each coordinate individually. The choice of which method is to be used will be determined in phase two of this project.

currentItem holds the coordinates of the current iteration, haltFlag is the flag that determines whether the system is going through its shutdown routine or not. The final attribute, batteryLevel, holds the current battery percentage.

4.3.4 input/output



Input/Output module controls the flow of data through the input and output channels, its main concern is to retrieve information from files about books and the map.

The module has the input and output controller as well as the book class, the latter is the class that is concerned with creating book objects that contain coordinates and weight of each book, the relationship between the two classes is one to one because each book ID in books contains one book object, and each book object has a single ID.

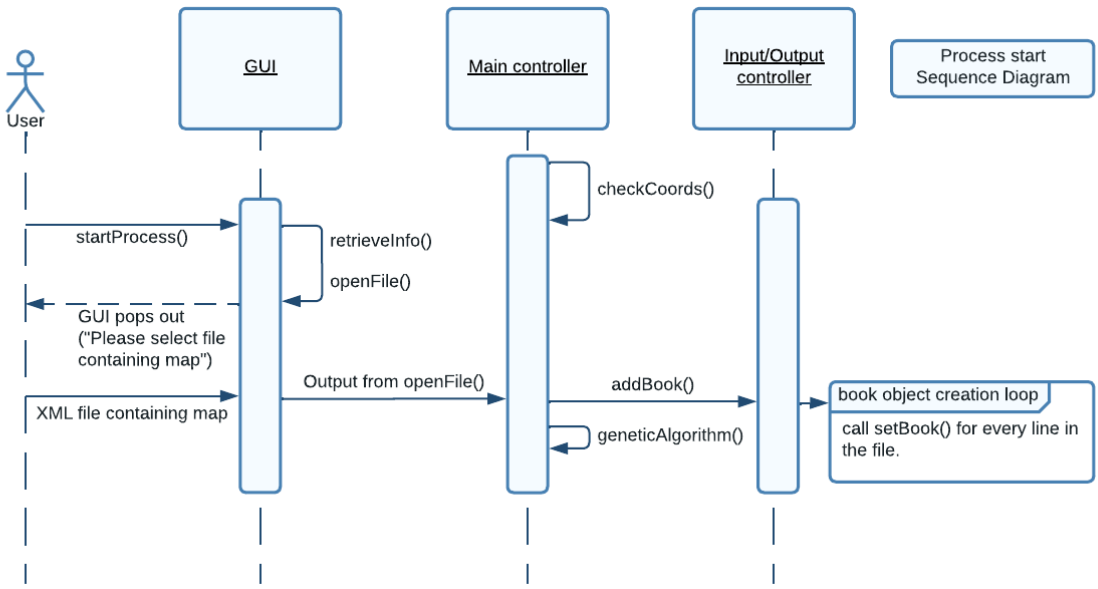
The attribute books contains the IDs of all of the book objects created after file retrieval, while the attribute map contains the map information.

4.4 Sequence and State Diagrams

To detail the flow of data and functions, software engineers utilize sequence and state diagrams. This section will use sequence diagrams to describe the start process and the delivery process, then make use of a state diagram to describe the different states of the robot while the system is running.

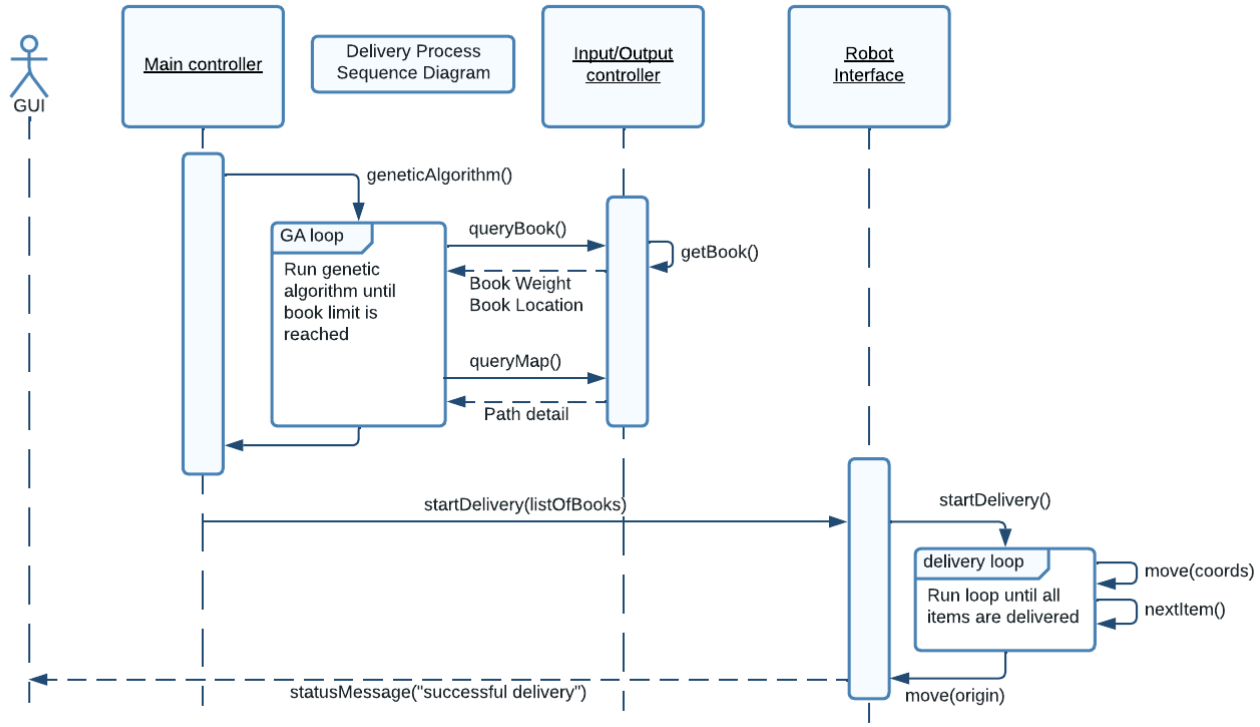
4.4.1 Item delivery sequence diagram

Once the system starts, the main function will first call `checkCoords()` to determine the location of the robot's idle position, then await for the user to call `startProcess()` from the graphical user interface, which will start the process by prompting the user to select the files containing the map and the list of books to be delivered. After this selection, the main controller will then direct that file input to the input and output model to store each book in a list for later use by the genetic algorithm. The main controller will then call `geneticAlgorithm()`.



4.4.2 Delivery process sequence diagram

After the previous sequence diagram, the main function will start the genetic algorithm, which will loop through each book to determine the most efficient path, sending an array of those coordinates to the robot interface which will start the delivery by sending the robot to each of those locations, then call move(origin) to move the robot back to its idling spot, and finishing the process by sending the status message “successful delivery” to the GUI (displayed directly as the user in this diagram for brevity).

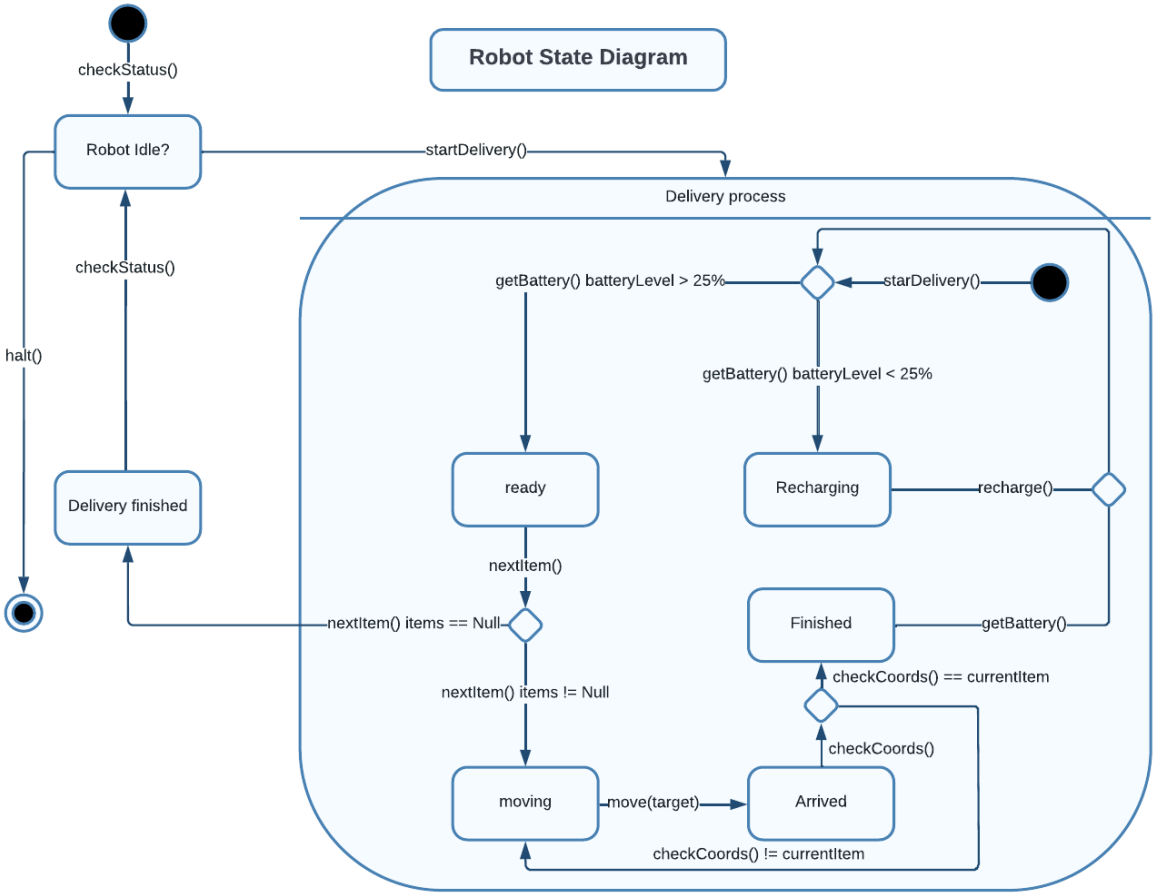


4.4.2 Robot state diagram

This diagram explains the states the robot will go through, remember that the system is intended to provide the path for this robot and does not fully consider the detailed implementation of the robot, which will be done through the application programming interface provided by the robot manufacturer.

The state starts with the robot idle in the origin position (0,0), and it will move out of idle mode once it receives the command `startDelivery()`, which is initiated through

the sequence explained earlier. It will then start the process of delivery, which will first check if the battery is sufficient, then, if it isn't, will initiate `recharge()`, otherwise, the robot will be ready to start delivery by initiating `nextItem()`, the function that will move the robot to the target of the item, finishing an iteration of this process and moving back to checking the battery until there are no more items, or a `forceStop()` command is initiated via the GUI, wherein the latter will also set the halt flag of the robot, which means that once the robot arrives back to the origin position, it will exit the process and the system will stop.



4.5 System Evolution

The evolution of the system is to improve its general capabilities and to set goals for further system development, there are possible improvements that can be the next step to this system but not yet its full potential, the following list contains what we currently think the next evolution of this system can be:

1. Fully automated system:
    - Through the utilization of barcode technologies, the software can be made to retrieve item information and send it to the robot without any human intervention.
    - The system can be made to recognize new or unrecognized books and put them on designated general shelves.
2. Retrieval and delivery of items:
    - As well as moving to target locations, the system can be improved to retrieve or drop off items using the same optimization principles.
3. Support for multiple robots:
    - Larger libraries may need multiple robots, the current system only supports one.
4. Delivery and retrieval of items other than books:
    - Improving the system to fit it into environments other than libraries is possible.

The system still has a long way to go before being implemented in real environments, which is the reason why we avoided moving too quickly from small test robots and environments.

CHAPTER FIVE

SYSTEM IMPLEMENTATION

5.1 Introduction

To finally implement the software designed in Chapter 4 of this document, an implementation plan is needed as well as an idea of which tools and technologies will be used for the implementation.

This chapter will go over what is necessary for the accomplishment of this project in terms of said technologies, as well as code structure and snippets to go over important parts of the code. The main programming that is used for implementing major parts of the project is Java language. It was chosen since it is a standard language used in a great multitude of companies here in the Saudi Arabian market. An additional script was also developed to plot the result of the outputs of the genetic algorithm.

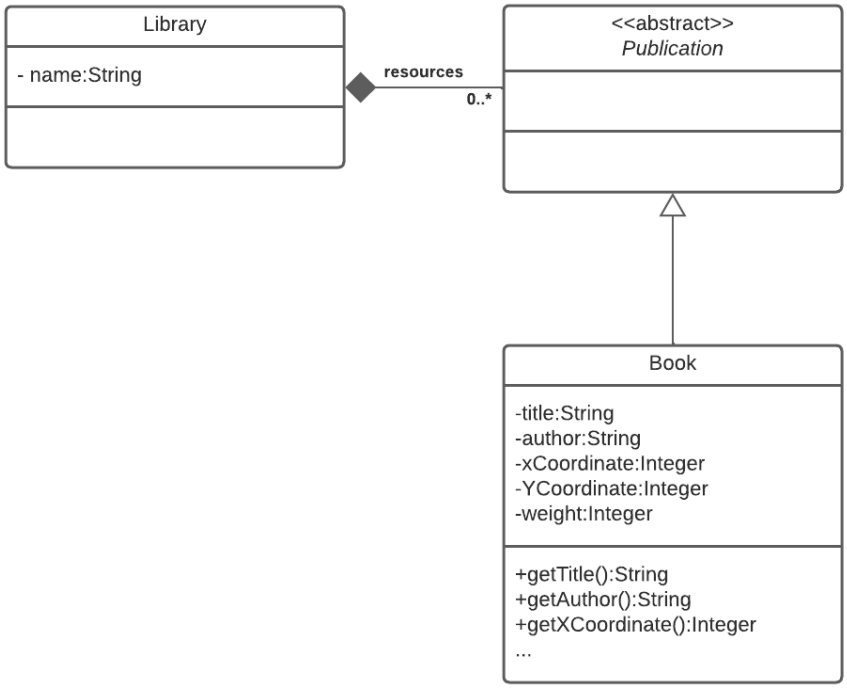
5.2 Implementation and Evaluation Plan

To implement the architecture, we first started creating the plan by dividing the structure into components, therefore we divided the “main controller” component into two different packages (Optimization and Controllers) for example. Component divisions and additions were put when needed, since we require flexibility instead of rigidly planning components and sticking to it regardless of need.

After working on each component, testing them based on input and output is the next clear step, so we have utilized a model generator to get many sample books (that are randomly generated), to test our algorithm against. The results of this are discussed further in Chapter 6.

5.3 Model Generator

To study the output of the implementation of our genetic algorithm, we have generated a different dataset of library models with 100, 500, 1000, 2500, and 5000 instances of books. The models were created with the help of Eclipse Modeling Framework (EMF) [5], and Ecore (like UML implementation). The metamodel of the library (to which the generated models conform) is illustrated in the figure below.



The data that is used for creating book instances of the library models were randomly generated and obtained from Mockaroo[1], which is a customized engine for randomly generated values. The data include books' titles and names of the authors, x and y coordinates, and books' weight.

[1] https://www.mockaroo.com

It is important to note that the generated models are sterilized in XMI format (XML metadata interchange), which is a standard that EMF uses to save models in the file system.

5.4 Code Structure

The software uses the JMetal framework [5], which facilitates multi-objective optimization experiments, and is divided into three main components: 1) the **main controller**, 2) the **Input/Output controller,** and 3) the **Optimization Algorithm**. This document will focus on the detailed structure of the optimization algorithm as it is the focus of the project, and briefly mention the needed detail of other components.

The main controller is where the program takes in inputs, then creates algorithm objects. It is made as a static class with static components since it will not have an object.

The Input/Output controller reads XMI files containing book information and turns it into a **List** data structure of **Book**s, usable by the algorithm that optimizes the travel of the robot according to distances and energy cost.

The optimization algorithm is a package of three important classes, the **genetic algorithm** class, the **GA settings** class, and the **NSGAII** class. The genetic algorithm creates the object containing the list of books object -- read by the input controller-- and has the evaluation function needed for evaluating distance and energy costs for a chromosome. Whilst the GA settings class sends the object created from the genetic algorithm (the problem) to the NSGAII after applying specific settings we desire, such as population size and the maximum number of evaluations, along with some default settings detailed further in the next section. Finally, we have the MyNSGAII class, which is our implementation of the powerful NSGAII [6], a most elite algorithm that implements the genetic algorithm very efficiently, applying binary tournament selection [7], Swap mutation [8] and two point crossover [9].

5.5 Code Snippets

To go over specifics of what section 5.4 went briefly over, this section will provide code snippets along with an explanation of what they do and why are they needed, starting with the main controller:

```
1.    //maximum number of evaluations for the algorithm, with the default of 500.
2.    private static int maxEvaluations      = 500;
3.    //Input/output controller objects, in order to get the list of books and write
    results.
4.    private static LibraryReader reader    = LibraryReader.getInstance();
5.    private static LibraryWriter writer    = LibraryWriter.getInstance();
6.    //list of books, each an object of class Book, which has methods to return each
    book's information.
7.    private static List <Book> books;
```

Followed by the Prepare function, which will be called by the main function of the software, this function is what manages the I/O classes, sending a dataset and getting a list of books back, then assigning the attributes above as such, ending in a call to the next function, execute.

```
1.    public static void prepare(int max, DataSet inputData) throws
    ClassNotFoundException, JMException {
2.              maxEvaluations = max;
3.              //Instantiate a library object
4.              Library library = reader.loadLibraryFromXMI(inputData);
5.              List<Book> books = new ArrayList<>();
6.
7.              for(Publication p:library.getResources()) {
8.                        books.add((Book) p);
9.              }
10.             MainController.books = books;
11.
12.             execute();
13.
14.   }
```

The Execute function will instantiate objects for the other algorithm classes, the for loop isn't necessary for the final product but it is needed for testing.

```
1.    public static void execute() throws ClassNotFoundException, JMException {
2.        GeneticAlgorithm ga = new GeneticAlgorithm(books,maxEvaluations);
3.        Algorithm algorithm = null;
4.
5.        for (int i = 1000; i<= 25000; i+=2500) {
6.              //this line exists to make the number of evaluations tidy, needed since
    i starts from 1000 instead of 0.
7.        if (i == 3500) i = 2500;
8.        maxEvaluations = i;
9.        System.out.println("\nMax Evaluations= "+maxEvaluations);
10.       //create an object of GA_Settings to specify the NSGA II settings.
11.       //Parameters: Problem name, problem object, population size and number of
    evaluations.
12.       GA_Settings nsgaiiSettings = new
    GA_Settings("GeneticProblem",ga,books.size(),maxEvaluations);
13.       algorithm = nsgaiiSettings.configure();
```

This part runs the NSGAII algorithm, which is why it is surrounded by time stamps to record the time taken for the function to run.

```
14.       //execute the algorithm and record the time taken
15.       long initTime = System.currentTimeMillis();
16.       SolutionSet solutionset = algorithm.execute();
17.       long estimatedTime = System.currentTimeMillis() - initTime;
18.
19.       //print out the distance cost and the energy cost, respectively, separated by
    space.
20.       System.out.println(solutionset.get(0).getObjective(0) + " "
    +solutionset.get(0).getObjective(1));
21.       System.out.println("Time= "+estimatedTime+" ms");
22.       }
23.                //print out the order in which books are to be delivered from the
    top ranked solution, according to book coordinates
24.                int index= 0;
25.                Solution solution = solutionset.get(0);
26.                System.out.println("Coordinate pairs of solution:");
27.                for ( i= 0; i <
    ((Permutation)solution.getDecisionVariables()[0]).getLength(); i ++)
28.                {
29.                      //set the index to the i-th element of the permutation list
30.                      index =
    ((Permutation)solution.getDecisionVariables()[0]).vector_[i];
31.                      //print out the set of coordinates
32.
      System.out.print("["+books.get(index).getXCoordinate()+","+books.get(index).getYC
    oordinate()+"] ");
33.                }//for
34.                //add an empty line
35.                System.out.println();
36.   }
```

The next few snippets will show how the algorithm runs in the order we found most intuitive. We start with GA_Settings class that holds the necessary attributes settings to run our own implementation of the NSGA II algorithm, the constructor of the class assigns those values.

```
1.  public class GA_Settings extends Settings{
2.
3.    public int populationSize_                ;
4.    public int maxEvaluations_                ;
5.    public double mutationProbability_        ;
6.    public double crossoverProbability_       ;
7.    public double mutationDistributionIndex_  ;
8.    public double crossoverDistributionIndex_ ;
9.
10.   /**
11.    * Constructor
12.    */
13.   public GA_Settings(String problemName, Problem problem, int popSize, int maxEval)
    {
14.             super(problemName) ;
15.             problem_ = problem;
16.
17.             // Attributes holding algorithm settings.
18.       populationSize_             = popSize;
19.       maxEvaluations_             = maxEval;
20.       mutationProbability_        = 1.0/problem_.getNumberOfBits() ;
21.       crossoverProbability_       = 0.9   ;
22.       mutationDistributionIndex_  = 20.0  ;
23.       crossoverDistributionIndex_ = 20.0  ;
24.              } // NSGAII_Settings
25.
```

The configure member configures the algorithm and then returns it ready for execution.

```
1.               /**
2.                * Configure NSGAII with default parameter experiments.settings
3.                * @return A NSGAII algorithm object
4.                * @throws jmetal.util.JMException
5.                */
6.               @Override
7.             public Algorithm configure() throws JMException {
8.                 Algorithm algorithm ;
9.                 Selection  selection ;
10.                Crossover  crossover ;
11.                Mutation   mutation  ;
12.
13.                HashMap  parameters ; // Operator parameters
14.
15.                // Creating the algorithm.
16.                algorithm = new MyNSGAII((GeneticAlgorithm) problem_) ;
17.
18.                // Algorithm parameters
19.                algorithm.setInputParameter("populationSize",populationSize_);
20.                algorithm.setInputParameter("maxEvaluations",maxEvaluations_);
21.
22.                // Mutation and Crossover operators fit for Permutation
    codification
23.                parameters = new HashMap() ;
24.                parameters.put("probability", crossoverProbability_) ;
25.                parameters.put("distributionIndex", crossoverDistributionIndex_) ;
26.                crossover =
    CrossoverFactory.getCrossoverOperator("TwoPointsCrossover",parameters);
```

```
27.
28.                 parameters = new HashMap() ;
29.                 parameters.put("probability", mutationProbability_) ;
30.                 parameters.put("distributionIndex", mutationDistributionIndex_) ;
31.                 mutation = MutationFactory.getMutationOperator("SwapMutation",
    parameters);
32.
33.                 // Selection Operator
34.                 parameters = null ;
35.                 selection =
    SelectionFactory.getSelectionOperator("BinaryTournament", parameters) ;
36.
37.                 // Add the operators to the algorithm
38.                 algorithm.addOperator("crossover",crossover);
39.                 algorithm.addOperator("mutation",mutation);
40.                 algorithm.addOperator("selection",selection);
41.
42.                 return algorithm ;
43.               } // configure
```

After configuration, the NSGA II algorithm is run by calling its execution function, implemented in class MyNSGAII.

The following snippets are taken from within the execute function, the whole function will not be included for brevity.

This next code creates the initial population.

```
86. // Create the initial solutionSet
87.       Solution newSolution;
88.       for (int i = 0; i < populationSize; i++) {
89.         newSolution = createDecisionVariable((GeneticAlgorithm) problem_);
90.         //newSolution = new Solution(problem_);
91.         problem_.evaluate(newSolution);
92.         problem_.evaluateConstraints(newSolution);
93.         evaluations++;
94.         population.add(newSolution);
95.       } //for
96.
97.       // Generations
98.       while (evaluations < maxEvaluations) {
99.
100.                 // Create the offSpring solutionSet
101.                 offspringPopulation = new SolutionSet(populationSize);
102.                 Solution[] parents = new Solution[2];
103.                 for (int i = 0; i < (populationSize / 2); i++) {
104.                   if (evaluations < maxEvaluations) {
105.                     //obtain parents
106.                     parents[0] = (Solution)
    selectionOperator.execute(population);
107.                     parents[1] = (Solution)
    selectionOperator.execute(population);
108.                     Solution[] offSpring = (Solution[])
    crossoverOperator.execute(parents);
109.                     mutationOperator.execute(offSpring[0]);
110.                     mutationOperator.execute(offSpring[1]);
111.                     problem_.evaluate(offSpring[0]);
112.                     problem_.evaluateConstraints(offSpring[0]);
113.                     problem_.evaluate(offSpring[1]);
114.                     problem_.evaluateConstraints(offSpring[1]);
115.                     offspringPopulation.add(offSpring[0]);
116.                     offspringPopulation.add(offSpring[1]);
117.                     evaluations += 2;
118.                   } // if
119.                 } // for
```

Calling `Problem_.evaluate` above calls the fitness function, we will jump to it and then return to the execute function.

The following is the Fitness Function for the algorithm, present in the GeneticAlgorithm class, it will return the two window costs of the input, the input being “problem” and “solution”; where the problem defines the decision variables of the problem along with their attributes, such as coordinates and weight. Whilst solution defines the current solution being evaluated, this object contains the permutation of book instances and is a part of a set of solutions, each being evaluated and ranked in a similar way.

```
1.    private static double[] fitnessFunction( GeneticAlgorithm problem, Solution
    solution) {
2.              //cost[0] is the distance cost, whilst cost[1] is the weight cost.
3.              double []cost = new double[2];
4.
5.              //This takes the proposed index representations from the SolutionSet,
    I.E. permutations of book IDs.
6.              int[] indexRep =
    ((Permutation)solution.getDecisionVariables()[0]).vector_;
7.              // Divided by 1000 to calculate the weight from G to KG for
    readability.
8.              double currentWeight = problem.weightSum/1000;
9.
10.             // Holders for current and previous books
11.             Book prev = problem.parent.get(indexRep[0]);
12.             Book current = problem.parent.get(indexRep[1]);
13.
14.             double dist=Math.abs((problem.parent.get(indexRep[0]).getXCoordinate()
    - 0) + (problem.parent.get(indexRep[0]).getYCoordinate() - 0));;
15.             // Initiate cost with first gene cost.
16.             cost[0] = dist;
17.             cost[1] = 0.1*dist*currentWeight;
18.
19.
20.             //Evaluate fitness for each gene.
21.             for (int i = 1; i < problem.parent.size();i++) {
22.                                 current = problem.parent.get(indexRep[i]);
23.                                 dist =  Math.abs((current.getXCoordinate() -
    prev.getXCoordinate()) + (current.getYCoordinate() - prev.getYCoordinate()));
24.                                 //Distance cost of moving from the previous book to
    the current one.
25.                                 cost[0] += dist;
26.                                 //Energy cost= the current weight for each unit of
    distance traveled.
27.                                 cost[1] += 0.1*currentWeight* dist;
28.                                 //Take off the weight of the dropped book out of
    the total weight carried now.
29.                                 currentWeight -= current.getWeight()/1000;
30.                                 prev = current;
31.                         }
32.
33.             return cost;
34.   }
35.
```

Now, back to MyNSGAII class, in the function execute after evaluating each solution, ranking the solutions, and finding the Pareto front.

```
128.  int remain = populationSize;
129.                  int index = 0;
130.                  SolutionSet front = null;
131.                  population.clear();
132.
133.                  // Obtain the next front
134.                  front = ranking.getSubfront(index);
135.
136.                  while ((remain > 0) && (remain >= front.size())) {
137.                    //Assign crowding distance to individuals
138.                    distance.crowdingDistanceAssignment(front,
    problem_.getNumberOfObjectives());
139.                    //Add the individuals of this front
140.                    for (int k = 0; k < front.size(); k++) {
141.                      population.add(front.get(k));
142.                    } // for
143.
144.                    //Decrement remain
145.                    remain = remain - front.size();
146.
147.                    //Obtain the next front
148.                    index++;
149.                    if (remain > 0) {
150.                      front = ranking.getSubfront(index);
151.                    } // if
152.                  } // while
```

```
165.  // This piece of code shows how to use the indicator object in the code
166.                  // of NSGA-II. In particular,it finds the number of evaluations
    required
167.                  // by the algorithm to obtain a Pareto front with a hypervolume
    higher
168.                  // than the hypervolume of the true Pareto front.
169.                  if ((indicators != null) &&
170.                      (requiredEvaluations == 0)) {
171.                    double HV = indicators.getHypervolume(population);
172.                    if (HV >= (0.98 * indicators.getTrueParetoFrontHypervolume()))
    {
173.                      requiredEvaluations = evaluations;
174.                    } // if
175.                  } // if
176.                } // while
177.
178.                // Return as output parameter the required evaluations
179.                setOutputParameter("evaluations", requiredEvaluations);
180.
181.                // Return the first non-dominated front
182.                Ranking ranking = new Ranking(population);
183.                ranking.getSubfront(0).printFeasibleFUN("FUN_NSGAII") ;
184.
185.                return ranking.getSubfront(0);
186.              } // execute
```

CHAPTER SIX

SYSTEM TESTING AND RESULT DISCUSSION

6.1 Introduction

In this section we will try to answer the questions of what the best number of evaluations is, to get the best result in the genetic algorithm. Is there a relation between the number of evaluations, time, cost, and energy?

6.2 Testing Description and Test Cases

For testing, we have five cases which we will explain as follows

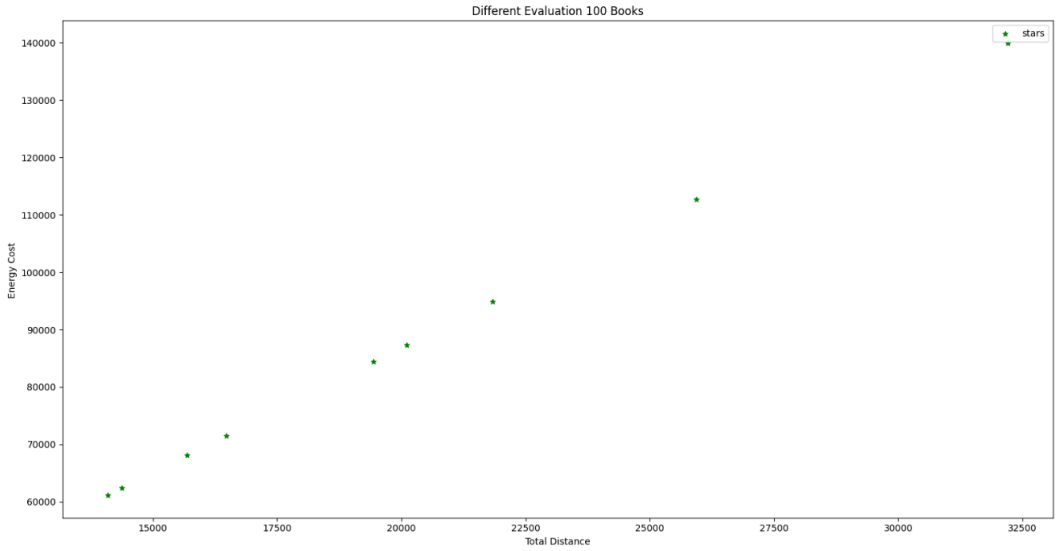
1. 100 books
2. 500 books
3. 1000 books
4. 2500 books
5. 5000 books

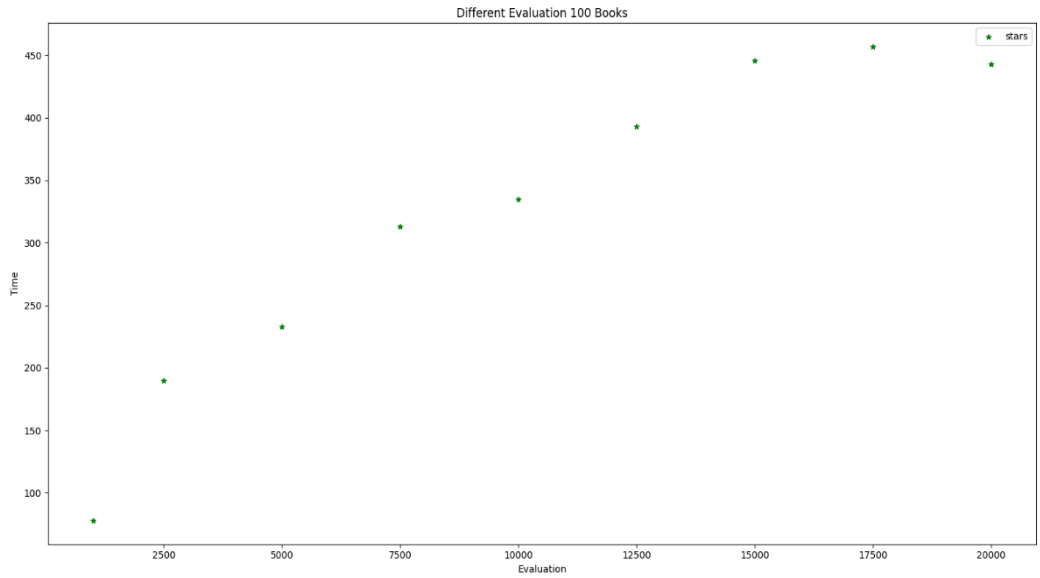
We will find the total cost and travel distance of each case as well as the time and the best number of evaluations based on the results.

6.3 Results and Discussion

The first plot of each case will show the total distance and energy cost, and the second plot will show the time and number of evaluations. We based our choice on the best number of evaluations after taking the results into consideration.

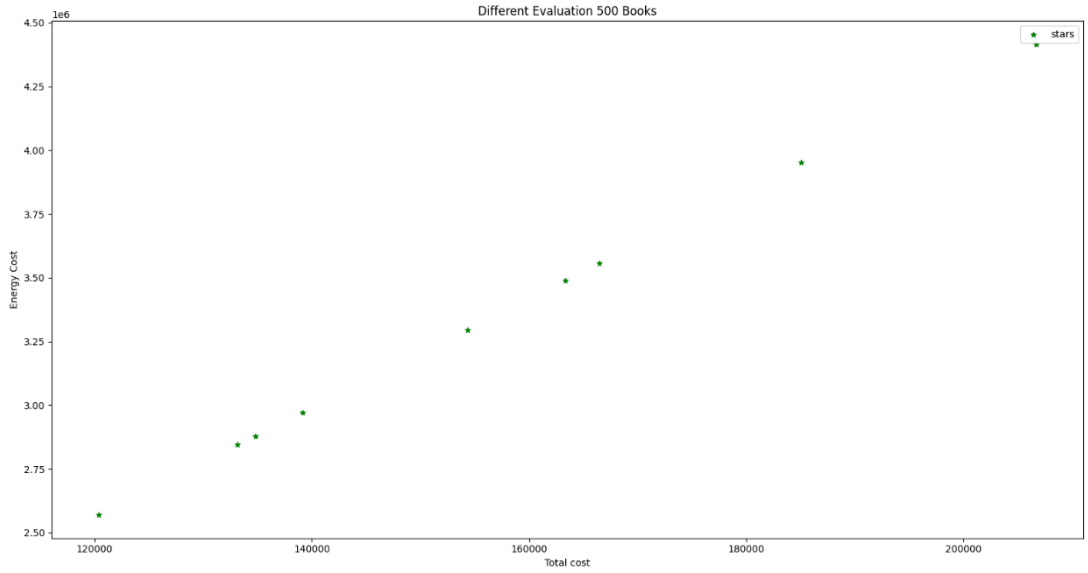
6.3.1 First case

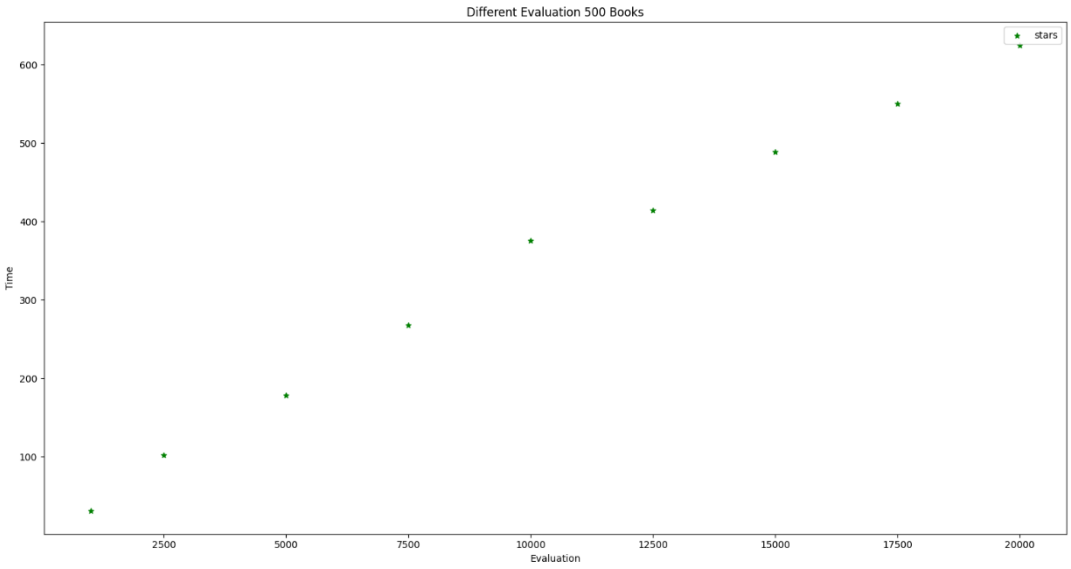




Based on the results of the two graphs of the first case the total distance and energy cost decrease as we increase the number of evaluations, but the time increase. For the first case the best result if we take time, cost and energy would be a maximum evaluation of 17500 since the difference in energy cost is not big enough to choose 20000.

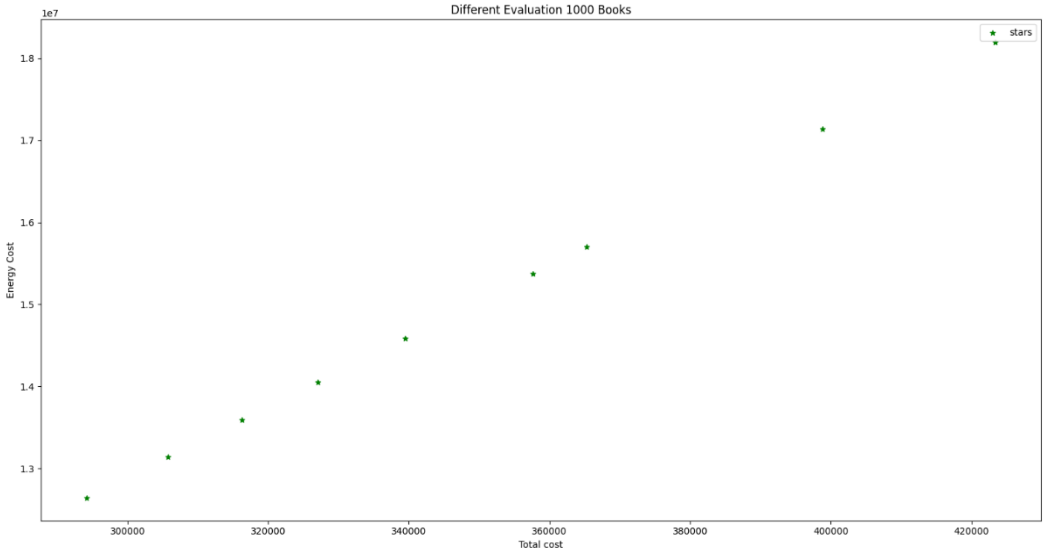
6.3.2 Second case:

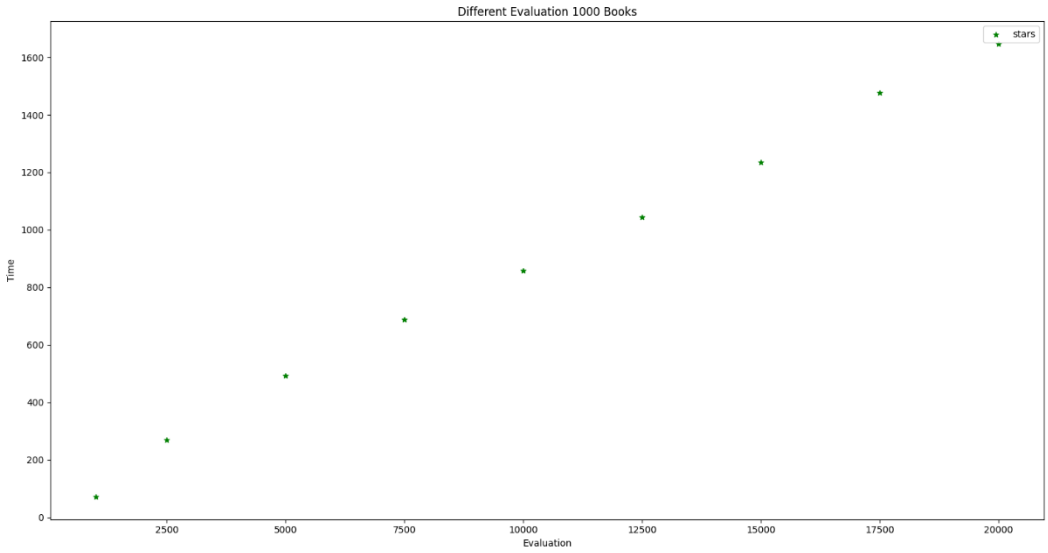




In the second case which we tested on 500 books, the total distance and energy cost decreased again as we increase the number of evaluations. The best number of evaluations is 20000 after we take cost, energy, and time into consideration.

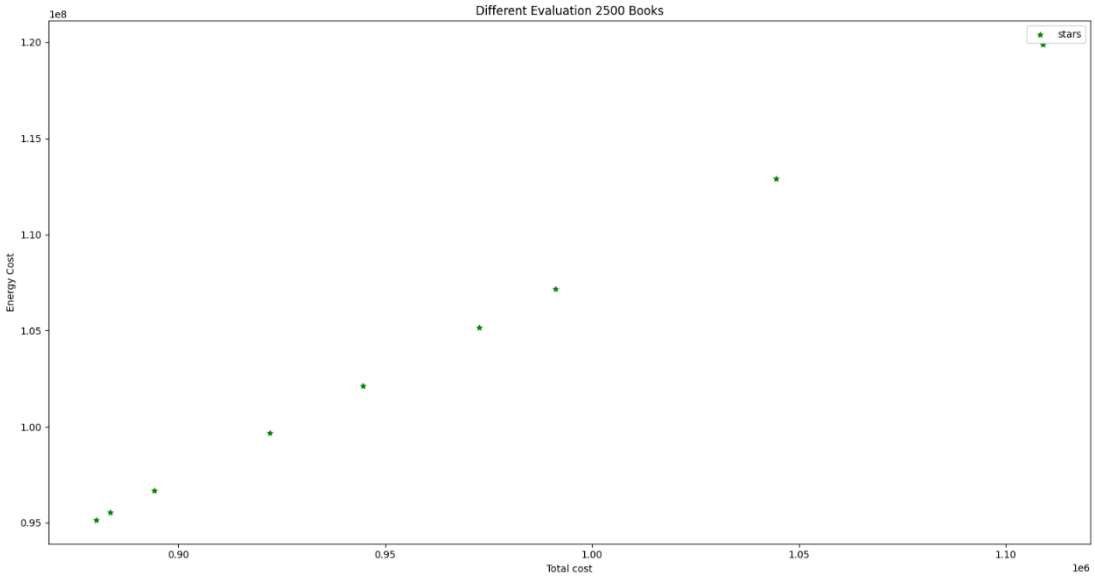
6.3.3 Third case

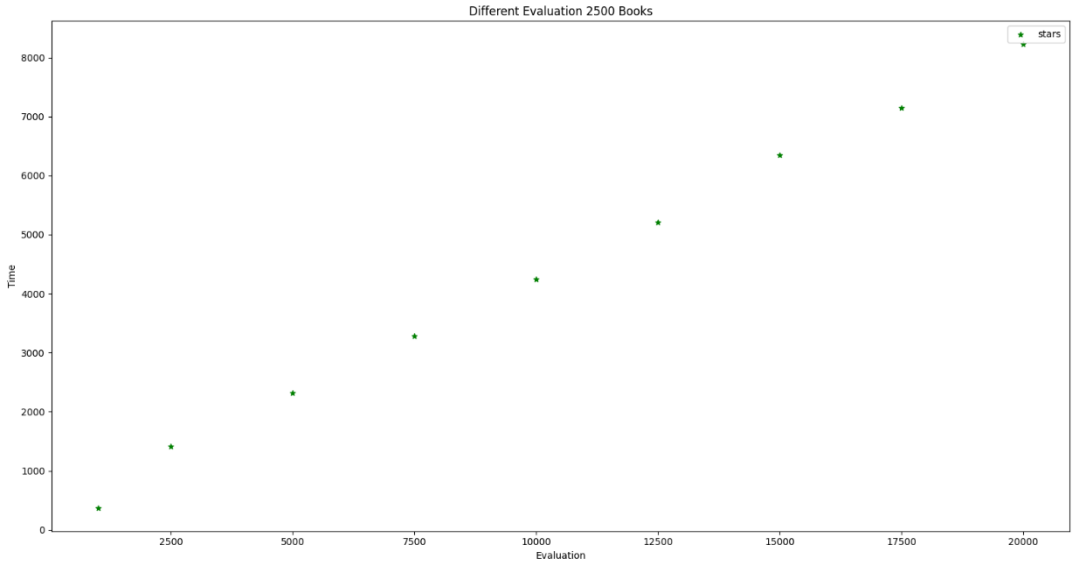




In the Third case which we tested on 1000 books, the total distance and energy cost decreased as we increase the number of evaluations. The best number of evaluations is 20000 after we take cost, energy, and time into consideration.

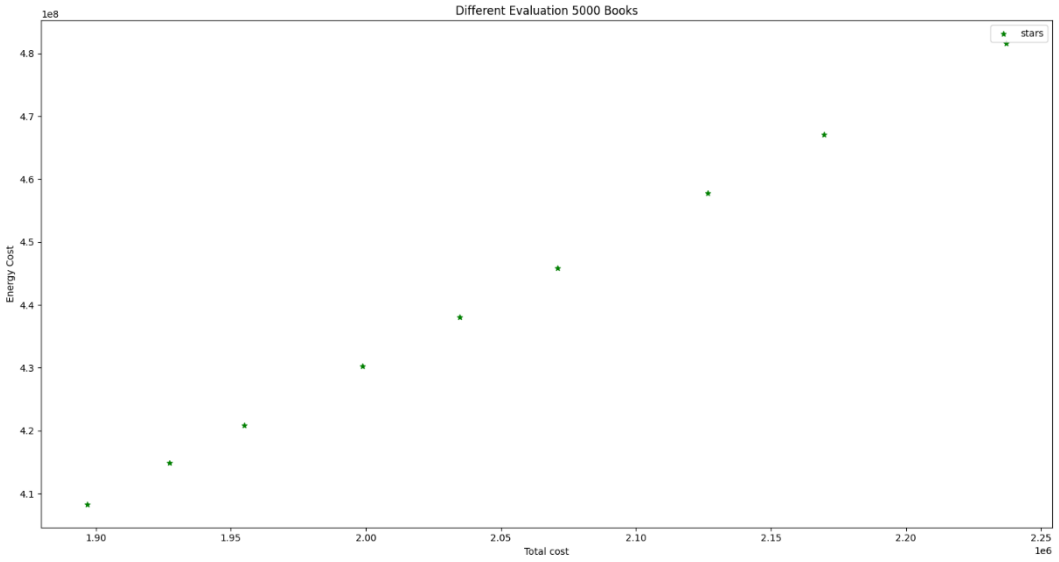
6.3.4 Fourth case:

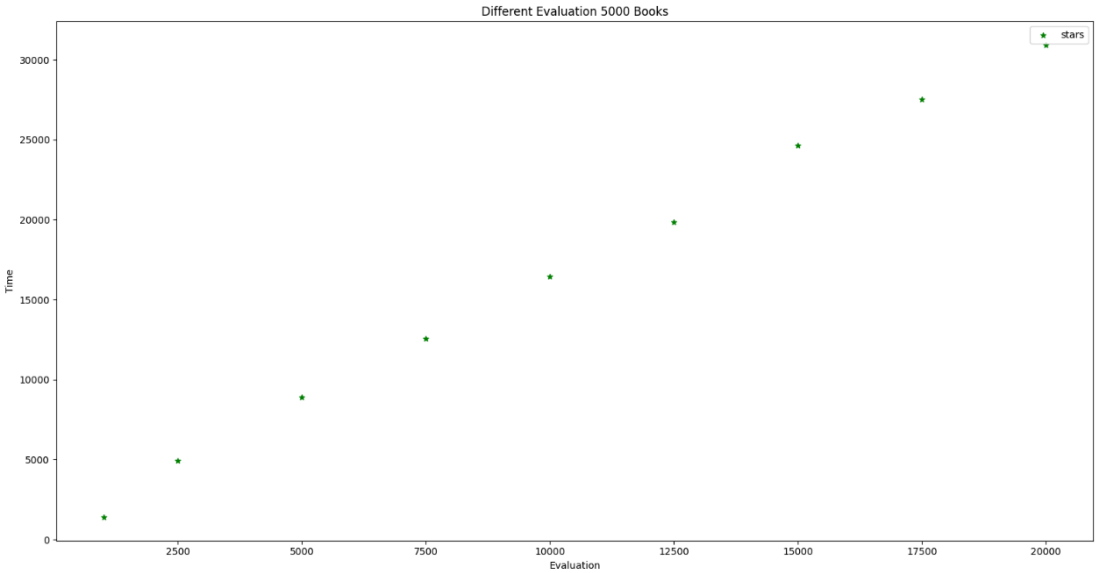




In the fourth case which we tested on 2500 books; the testing showed similar results to the first case. The best choice would be the 17500 evaluations, rather than 20000 after taking everything into consideration.

6.3.5 Fifth case





The best answer to the question of how many evaluations should we run for on the fifth case would be 20000 after taking time, cost, energy, and distance.

6.4 Summary

The results of the testing on our cases showed that the number of evaluations, cost and distance are related as each time we increased the evaluations our cost and energy decreased but the time increased in most cases.

In the end, each time we need to use a genetic algorithm we must answer the question of how much time we have.

If we have unlimited time, we can choose the height number we can think of, but this is not the case in the real world. So, we ask again how much time you have.

CHAPTER SEVEN

CONCLUSION AND FUTURE WORK

7.1 Conclusion

The genetic algorithm has shown to be a very good solution to the traveling salesman problem in general, and to the robot librarian problem specifically. As shown in chapter 6, it becomes increasingly efficient the greater the number of evaluations used, therefore applying non-functional requirements.

The aim of this project was to provide an algorithmic approach to sorting books in a library, along with proof that this approach is effective and efficient. Chapter two shows why the genetic algorithm is an excellent candidate for this project and chapter 3 specified the problem's details, as well as how we approach it. Chapters four and five have designed and implemented the project, respectively. And finally, chapter six shows how effective it is and what's the best approach and use case that this project fits the best.

As for the requirements, section four of chapter six provides code snippets for functional requirements, whilst chapter six provides tests that prove accomplishment of non-functional requirements.

The robot hardware integration, however, has been cut off from this project currently and has fully been moved to future work. That was done because of the semester reducing in its size, and we have chosen to cut out robot implementation since it was only meant as a proof of concept and is not imperative to the software's result, because it was a model robot, not a real delivery bot.

7.2 Future Work

This project sets the stepping stone to creating a robot capable of applying the genetic algorithm in its deliveries, it can be used on any sort of delivery whether it be books, or it can also be developed to deliver other types of items.

The software can be improved by implementing a clear Graphical User Interface, where book inputs can be selected more easily as well as robot control.

A map writer module can be implemented to quickly set the book details that would fit the criteria needed for the map reader module and make it into a bar code to be physically stuck onto the book to fully automate the process.

The system would benefit greatly from parallel computing, so the application of this algorithm using parallel computing techniques will yield great results.

7.3 Final Year Project Closing Remarks

This project was an honor to work with, we are proud to have had the opportunity to develop it. We have learned so much and have honed many a skill that we already know, we have been blessed to have Dr. Faisal Alhwikem as our supervisor as he was always helpful and quick in his responses and input.

The genetic algorithm was a very fascinating research subject that would, for as long as we live, hold a special place in our minds. Taking insight from the natural world, it very efficiently and effectively approaches a feasible solution, all the while being flexible to be applied to many types of problems with a very small number of necessary changes to it.

We extend our appreciation to the College of Computer for this opportunity and for all their help on the path, and we are thankful for you, dear reader, for allocating the time of your day to thoroughly read this project report.

Kindest regards, Abdullah Alhabib and Abdullah Alhasan, graduate computer science students at the College of Computers, Al-Qassim University, Saudi Arabia.

BIBLIOGRAPHY

[1] Z. Michalewicz and D. Fogel, How to solve it: Modern Heuristics, Springer, 2000.

[2] I. T. Hoda A. Darwish, "Reduced Complexity Divide and Conquer Algorithm for Large Scale TSPs," 2014.

[3] O. Nurdiawan, "Optimization of Traveling Salesman Problem on Scheduling Tour Packages using Genetic algorithm," 2020.

[4] M. Glinz, "On Non-Functional Requirements,," *15th IEEE International Requirements Engineering Conference,* pp. 21-26, 2007.

[5] Juan J. Durillo and Antonio J. Nebro, "jMetal: A Java framework for multi-objective optimization," *Advances in Engineering Software,* pp. 760-771, 2011.

[6] A. P. S. A. a. T. M. K. Deb, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation, vol. 6, no. 2,* pp. 182-197, 2002.

[7] T. Blickle, "Evolutionary computation 1," 2000, pp. 181-186..

[8] M. J. S. a. D. A. Torrey, "Genetic algorithms for control of power converters," in *Power Electronics Specialist Conference*, 1995.

[9] K. A. a. W. M. S. De Jong, "A formal analysis of the role of multi-point crossover in genetic algorithms.," *Annals of mathematics and Artificial intelligence 5.1,* pp. 1-26, 1992.

[10] R. E. M. N. H. K. N. a. A. V. L. K. P. Cheng, "Multi-Objective Genetic Algorithm-Based Autonomous Path Planning for Hinged-Tetro Reconfigurable Tiling Robot," 02 July 2020.

[11] W. D. 2. Mulder SA, "Mulder SA, Wunsch DC 2nd. Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks. Neural Netw.," 2003.

[12] S. Datta, "Traveling Salesman Problem – Dynamic Programming Approach," 2020.

[13] V. Joshi, "Speeding Up The Traveling Salesman Using Dynamic Programming," 2017.