A COMBINED ALGORITHM FOR PLACEMENT OF RECTANGULAR VEHICLES IN A FERRY

BÜŞRA PAŞALI
B.S., Industrial Engineering, Isik University, 2010

Submitted to the Graduate School of Science and Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
in
Industrial Engineering

IŞIK UNIVERSITY
2013

ISIK UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

A COMBINED ALGORITHM FOR PLACEMENT OF RECTANGULAR VEHICLES IN A FERRY

BÜŞRA PAŞALI

APPROVED BY:

Assoc. Prof. Çaglar AKSEZER Işık University _____
(Thesis Supervisor)

Assoc. Prof. Seyhun ALTUNBAY Işık University _____

Assoc. Prof. Aydın YÜKSEL Işık University _____

APPROVAL DATE: ..../..../....

A COMBINED ALGORITHM FOR PLACEMENT OF RECTANGULAR VEHICLES IN A FERRY

Abstract

Management of inland maritime transportation is an important task since it has a costly operation environment, as well as public service responsibility with safety and speed concerns. A typical operation involves multiple ferryboats, assigned to transport both passengers and vehicles between two stationary points. Effective management of resources (ferries, crew, fuel etc.) to meet the increasing demand has become the primary objective of planners working in this area.

This research focuses on an uninvestigated part of the general problem: Finding the ideal layout of vehicles on ferryboats. Optimal placement initiative will remedy both the trip utilization rate and financial indicators of the organization. However, an optimal solution is usually not available due to complicated nature of the problem, such as sequencing and embarking restrictions. Here, a heuristic approach is proposed in order to find the best solution by abiding the restrictions of vehicle placement algorithm.

The proposed procedure seeks the best position of a given sized (or categorized) vehicle inside a ferryboat under first come first served sequencing rule restriction. The problem at hand may be thought as a sub-echelon of the well-known knapsack and bin-packing algorithms, and benefits from both philosophies in the proposed algorithm. Economical and operational effects of the proposed procedure were illustrated by comparing its application on a real ferry line data gathered from the Sirkeci-Harem route operating in the city of Istanbul.

Key Words: ferryboat layout, vehicle placement heuristic, maritime

DİKDÖRTGEN ARAÇLARIN FERİBOTLARA YERLEŞTİRİLMESİ İÇİN BİRLEŞİK BİR ALGORİTMA

Özet

Deniz ulaşımı güvenliği ,hızlı oluşu ve düşük maliyetli olması dolayısıyla önemli bir konudur. Feribotlar iki kıyı arasında yolcu ve araç taşıyabilen deniz taşıtlarıdır.Deniz ulaşımındaki beklenen büyüme feribotlara olan talebi de arttıracaktır.Bu alandaki kaynakların (feribot, çalışanlar,yakıt v.b) etkin bir şekilde yönetimi bu alanda çalışan uzmanların ana amacıdır.Bu çalışmada daha once çalışılmamış bir konu olan feribota araçların en iyi şekilde yerleştirilmesi üzerinde çaılşılmıştır.Araçların en uygun şekilde yerleştirilmesi kurum kârını da arttıracaktır. Fakat problemin karmaşık yapısından dolayı en iyi çözümü bulmak her zaman mümkün değildir. Bu yüzden bu çalışmada en iyi çözümü bulabilmek için sezgisel bir algoritma önerilmiştir.

Bu algoritma "ilk gelen yerleştirilir" kuralına uygun olarak yerleştirilecek araçlara en uygun konumu bulmaktadır. Bu algoritma çok bilinen Sırtçantası Algoritması ve Paketleme Algoritmalarının bir uzantısı olarak düşünülüp değiştirilerek geliştirilmiştir. Bulunan bu çözümün ekonomik ve operasyonel etkisi gerçek bir feribot hattı olan Eskihisar-Yenikapi hattı verisiyle test edilmiştir.

Anahtar Kelimeler: Deniz ulaşımı, sezgisel algoritma, araç yerleştirme, arabalı feribot

Acknowledgements

I would like to thank to my supervisor, Assoc. Prof. Çağlar AKSEZER for his guidance. I am very grateful to my family and my friends for their encouragements during my graduate studies.

To my family...

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AS	Automated System
BL	Bottom Left
MS	Manual System
VPP	Vehicle Placement Problem
VPA	Vehicle Placement Algorithm
BPP	Bin Packing Problem
FCFS	First Come First Serve

Chapter 1

Introduction

1.1 Motivation

Vehicle Placement Problem (VPP) involves assignment of parking or storage location to cars, vans, and trucks in a two dimensional closed environment such as garages, barges, and ships. Placing vehicles in a ferry is a complicated task and to the best of our knowledge there isn't any study regarding VPP in the literature.

Achieving the highest occupancy rate in a ferryboat is an important task for managers and captains to raise the operational efficiency and profitability. This study aims to analyze whether there is a need for an automated vehicle placement system instead of random assignment procedure, in order to obtain higher amount of space utilization in inland waterway ferries. In this thesis, a new algorithm is suggested for solving vehicle placement problem that is guaranteed to give solutions that are not too far away from the optimal solution.

Because of the fact that algorithms in the literature couldn't solve this problem directly we have suggested a combined algorithm for solving vehicle placement problem that is guaranteed to give solutions that are not too far away from the optimal solution.

Development of an algorithm for solving the VPP in a ferry has difficulties, which stem from geometry of vehicles and ferries. The constraints, that should be considered are; types of vehicles to be placed, capacity of the ferry, and balance of

the ferry and distance between vehicles. In addition, vehicles should be handled according to the First Come First Served (FCFS) sequencing rule, it is the concept of an online algorithm is used to formalize the realistic scenario, where the algorithm receive its input incrementally and need to make decisions without knowing the rest of the input. Such algorithms are required in situations where solutions need to be generated over time and the input is only completely known at the end of processing.

1.2 Contributions

This thesis introduces a combined vehicle placement algorithm consisting of three algorithms, Bin-Packing algorithm, Bottom-Left algorithm and Knapsack algorithm. When we place vehicles according to this combined algorithm and automate the vehicle placement algorithm, it provides lots of benefits to organization and people.

1.3 Thesis Organization

The remainder of the thesis report is organized as follows. Chapter 2 overviews the previous work and literature, which is related to the general concept of the problem. Chapter 3 introduces the methodology and later Chapter 4 provides details of the problem and describes the steps of the developed algorithm. Finally, we present conclusions drawn from this research, as well as applied results and possible future work.

Chapter 2

Literature Review

Placement is an important topic at Operations Research area; it may be used in box placement, pallet placement, rectangular placement etc. Packing problems are optimization problems that are concerned with finding a good placement of multiple items in larger containing regions.[1] The usual objective of the allocation process is to maximize the material utilization and hence to minimize the "wasted" area. The research in this thesis is about placing vehicles to a ferry in two-dimensional environment. Placement problems are solved mainly via heuristic and meta-heuristic algorithms. This type of problems are NP-hard non-deterministic polynomial-time hard), means "at least as hard as any NP-problem," although it might, in fact, be harder. A problem is NP-hard if and only if there is an NP-complete problem. NP-complete problem is means that it cannot be solved in polynomial time in any known way. NP-Hard and NP-Complete is a way of showing that problems are not solvable in realistic time.[2] Heuristic algorithms are used in this research; the reason of the usage of heuristic algorithms will be explained at the forthcoming sections.

2.1 Heuristic Algorithms

Heuristic techniques have long been used to quickly solve optimization problems to find an exact solution for an optimization problem in real practice is sometimes

less practical in comparison to using an easily computed method of acquiring near-optimal solutions. When the problems grow larger in size, obtaining the exact solutions can take excessive computational time and storage space. In such cases, the results obtained by a complex, time consuming method may be no more attractive than near optimal solutions. Further considering the imprecision of the real-world problem data, and the approximate nature of some formulations, obtaining a precise solution in reality may seem meaningless. Obtaining a near-optimal solution in a reasonable computational time may be advantageous and more practical.[3]

It is essential to note that it would be useful to develop algorithms that can ultimately be used in real systems. An exact algorithm produces optimal solutions but may have a running time that could make it infeasible in real systems. On the other hand, a heuristic could run fast but there are no warranties on the solution excellence.

In this chapter we review several heuristic algorithms for solving placement problems. These algorithms not directly refer to placement problems but they should be assimilated to this problem. The proposed algorithms are first fit Bin Packing Algorithm, Bottom Left Algorithm and Knapsack Algorithm. They will be detailed at following sections.

2.2 First Fit Bin Packing Algorithm

Two-dimensional bin packing problems has been studied in the literature exceedingly.[4] The researchers have solved that kind of problems with a variety of algorithms. Especially, a huge amount of work has been done on on-line and off-line approximation algorithms. See [5] for a survey on approximation algorithms and [6] for an overview of on-line algorithms. Bin packing is an optimization problem in which we are given an instance consisting of a sequence of items and the goal is to pack these items into the smallest possible number of bins of unit size.[7] First

fit bin packing algorithm packs each item into the first bin where it fits, possibly opening a new bin if the item does not fit into any currently open bin.

The objective of the BPP is to load all the items while minimizing the number of used bins. The problem has been extensively studied in the past decades, producing several exact and heuristic methods.[8]

2.3 Bottom Left Algorithm

The Bottom-Left (BL) algorithm makes the layout as stable as possible so that placed items cannot move farther downward or leftward. Baker et al. defines The Bottom-Left (BL) algorithm like sorting the items by non-increasing width, and packs the current item in the lowest possible position, left just. This method can not be used at online algorithms because the input is not been known at the beginning of solution.[9]

Two dimensional BL problems can be categorized into orthogonal problems (where pieces are rectangular) and irregular problems.[10] Orthogonal problems have received greater attention from the academic community, as they are less geometrically complex. The best-known results for the established benchmark problems have been achieved using the best-fit heuristic which is presented, discussed and evaluated by researchers.[11]

There are some heuristics which belongs to the class of bottom-left (BL) packing heuristics to combine an order-based genetic algorithm. In order to reduce computational complexity the heuristic does not necessarily place an item at the lowest available BL position. However, it preserves BL stability in the layout.[12] There is an evolutionary algorithm, which is combined with a heuristic routine. This routine is similar to the BL-heuristic and places items in the position that is closest to the lower-left corner of the object. Comparisons with a mathematical programming algorithm show that the evolutionary approach is computationally more efficient.[13]

2.4 Knapsack Algorithm

The Knapsack problem is a general resource allocation problem in which a single resource is assigned to a number of alternatives with the objective of maximizing the total return.This model is also know as the fly-away kit problem or the cargo-loading problem.[14]

The Knapsack algorithm has been used to model various decision making processes and finds a variety of real world applications: processor allocation in distributed computing systems, cutting stock, capital budgeting, project selection, cargo loading, and resource allocation problems. Industrial applications find the need for satisfying additional constraints such as urgency of requests, priority and time windows of the requests, and packaging with different weight and volume requirements.[15]

Chapter 3

Methodology

The algorithm presented in this thesis provides solution for vehicle placement into a ferry by a proposed combined algorithm. The primary objective is to place all vehicles at the queue into ferry of various sizes such that the total area of the used bins is minimized. This chapter describes the algorithm at a high level and presents rationale for the approach.

3.1 The Algorithm

We introduce a heuristic algorithm to solve the problem. Use of a modeling approach is not appropriate since there are certain operational restrictions. One of the reasons is the enforced First Come First Serve (FCFS) sequencing rule. In Roll on-Roll off (RORO) ships; the placement problem may be solved with modeling approach because the input scheme is known before placement by reservations or long embarking windows and FCFS rule is not mandated. In the ferries however, we couldn't know what will be the input and inter-arrival rate of the vehicles. The other reason for solving the problem with a heuristic algorithm is the time constraint of the problem. The vehicles are waiting at the queue and the ferry must depart at its scheduled time. So the problem should be solved simultaneously.

3.1.1 Vehicle Placement Algorithm: No Balance

In this approach; the algorithm firstly detects the type of the vehicle and then places it to the bottom left of the ferry. When the next vehicle comes in, it checks the type again and then searches for empty space at the first row and then places the vehicle to the right of the first vehicle. Then algorithm restarts from the first step for each vehicle. Flow chart of Vehicle Placement Algorithm: No Balance Constraint and Pseudo-code illustrates the steps of the Vehicle Placement without balance constraint.

Algorithm 1 VPA:No Balance

```
Look at the type of the vehicle N
while Ferry is not full do
  for row zero to row n do
      search for empty cells if there is an empty cell then
      Place N to the empty cell
    else
      Go to next cell
    end if
  end for
end while
```

3.1.2 Vehicle Placement Algorithm: Balance Constraint

Balance is important for safety of the ferry, if one side of the ferry is bulkier than the other than this creates a dangerous state for all passengers and vehicles. Also, balance negatively affects the speed of the ferry resulting in lower fuel efficiency. Placement of type 1 vehicles in this algorithm is similar to "Vehicle Placement Algorithm: No Balance Constraint" It checks the type of the vehicle and then search for empty places in the ferry. If it finds an empty place at the bottom left then it checks the number of the vehicles at the left and right side. If both sides are in balance then the vehicle is placed to the first proper place. When the ferry is full and there is no empty space the algorithm ends. At Flow chart of Vehicle Placement Algorithm With Balance Constraint and pseudo-code the steps of the algorithm may be seen.

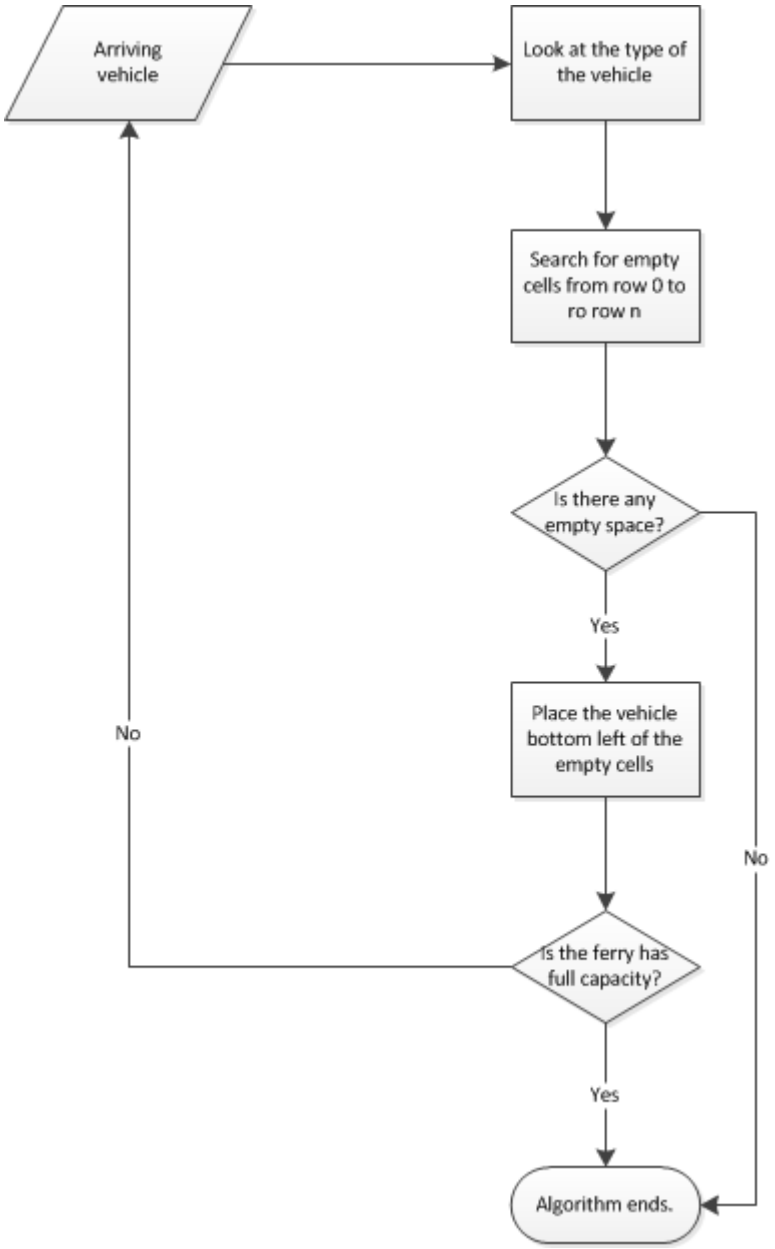

Figure 3.1: Flow chart of Vehicle Placement Algorithm: No Balance Constraint

3.1.3 Vehicle Placement Algorithm: Balance at the Peripheries Constraint

At the previous section the balance is calculated according to number of vehicles at both sides of the ferry. But that algorithm may not provide an accurate balance according to momentum laws. So at this algorithm; the heavy vehicles placed at the peripheries of the ferry layout (first and last column of the ferry). Firstly

Algorithm 2 VPA:Balance Constraint

```
Look at the type of the vehicle N while Ferry is not full do
for row zero to row n do
    search for empty cells if N is type 1 then
    Place N to the empty cell which is dedicated to type 1
  else
    if Number of Type 2 at left> Number of Type 2 at right then
      Place N to the right
    else
      Place N to the left
    end if
  end if
end for
end while
```

these grids were reserved to type 2 vehicles, after they have placed and if there is no truck at the queue, type 1 vehicles are placed to reserved cells.The details of this algorithm may be seen at the below pseudo-code and figure 3.3.

Algorithm 3 VPA: Balance at the Peripheries

```
Look at the type of the vehicle N while Ferry is not full do
for row zero to row n do
    search for empty cells if N is type 1 then
    Place N to the empty cell which is dedicated to type 1
  else
    Place N to the empty cell which is dedicated to type 2
  end if
  if there is no type 2 at the queue and type 2 cells are empty  then
    Take type 1 from queue and place to these cells
  end if
end for
end while
```

3.1.4 Vehicle Placement Algorithm: Balance at the Center Constraint

This model is similar to VPA:Balance at the Peripheries; at this model heavy vehicles are placed to the center of the ferry. Two columns at the center are reserved to type 2 vehicles; firstly they are are placed if there is empty cell and no other type 2 at the queue; at that time type 1 vehicles fill the empty cells. The details of this algorithm may be seen at the below pseudo-code and figure 3.4.

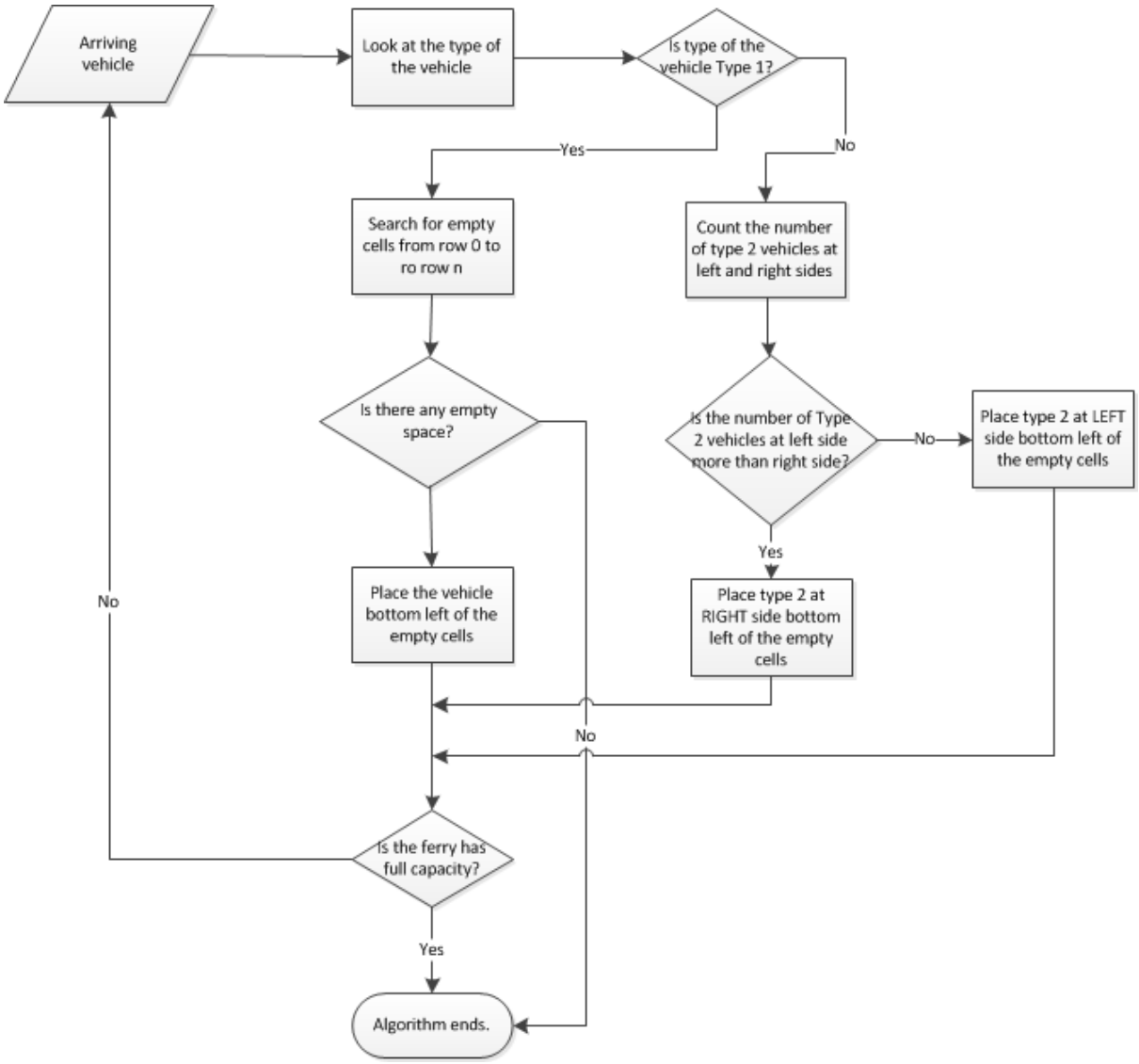

Figure 3.2: Flow chart of Vehicle Placement Algorithm: Balance Constraint

3.1.5 Vehicle Placement Algorithm: Momentum Law

This model takes into consideration momentum laws practically. Same type of vehicles are placed to the ferry layout equidistantly. According to algorithm, type 2 vehicles are started to placing from center line of the ferry and type 1 vehicles are started from peripheries. Firstly they are placed to the left side and then right side. Also; the number of each type of vehicles are equal at each side of the ferry. It can be said that this algorithm places vehicles more balanced. The details of this algorithm may be seen at the below pseudo-code and figure 3.5.

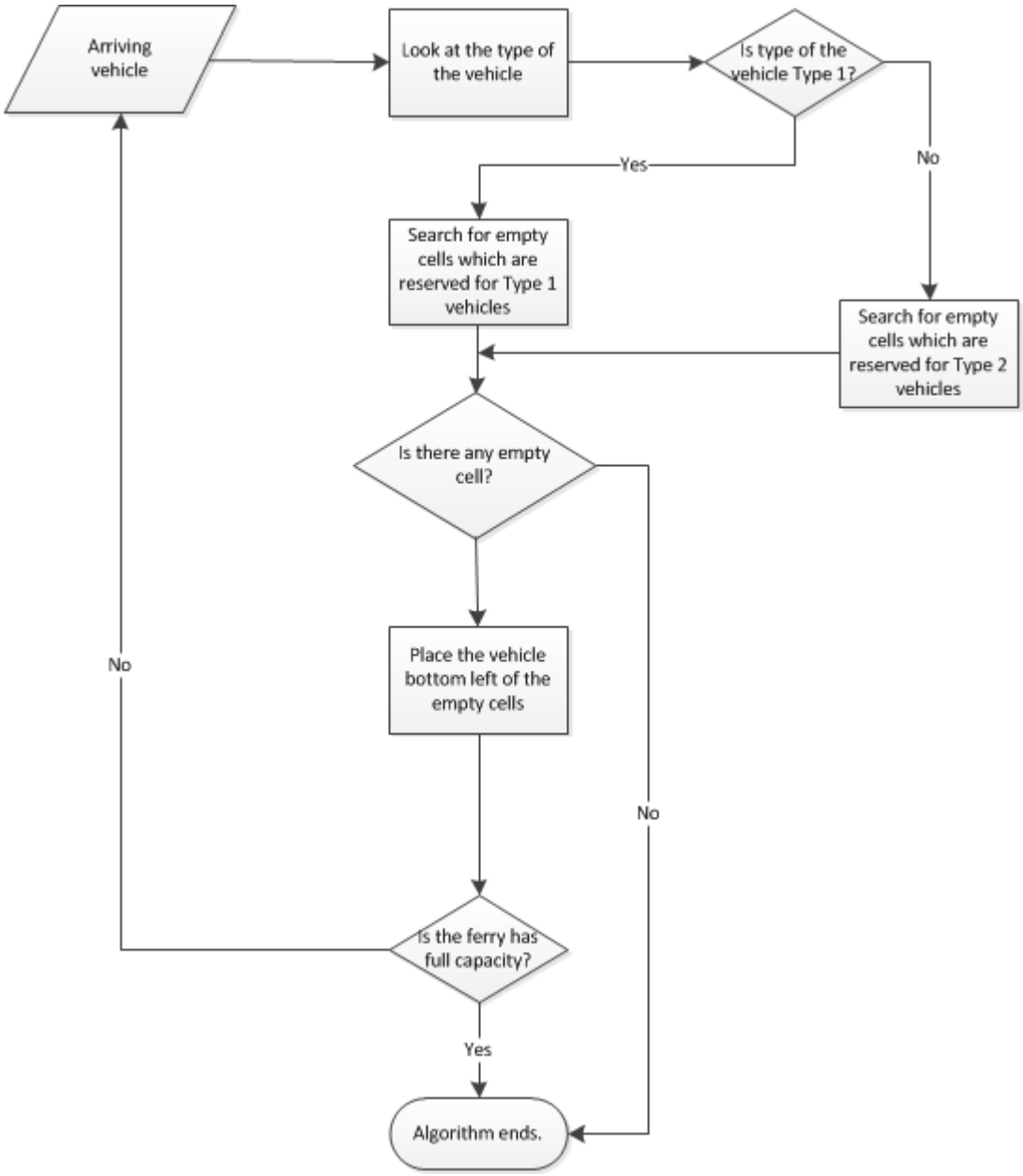

Figure 3.3: Flow chart of Vehicle Placement Algorithm: Balance at the Peripheries

3.2 Solution Process of the VPP

This study consists of analysis, design and development phases as shown in figure .

Before developing the algorithm, we have talked with the organization that is responsible for marine transportation in Istanbul. Also, we have learned the

Algorithm 4 VPA: Balance at the Center

```
Look at the type of the car N while Ferry is not full do
for row zero to row n do
    search for empty cells if N is type 1 then
    Place N to the empty cell which is dedicated to type 1
  else
    Place N to the empty cell which is dedicated to type 2
  end if
  if there is no type 2 at the queue and type 2 cells are empty  then
    Take type 1 from queue and place to these cells
  end if
end for
end while
```

Algorithm 5 VPA: Momentum Law

```
Look at the type of the vehicle N while Ferry is not full do
for row zero to row n do
    search for empty cells if N is type 1 then
    if left side > right side then
      Place N to the bottom right
    else
      Place N to the bottom left
    end if
  end if
  if N is type 2 then
    if left side > right side then
      Place N to the right cell of center
    else
      Place N to the left cell of center
    end if
  end if
end for
end while
```

manual vehicle placement process from ferry flagmen, which is responsible for placement within the ferry floor. We have developed requirements and constraints of the problem via meetings. After analysis of the problem; the algorithm was developed and code is written in Java programming language. The results were tested via a case study, which is from line Sirkeci-Harem ferry. The picture of that ferry is shown at figure 3.7

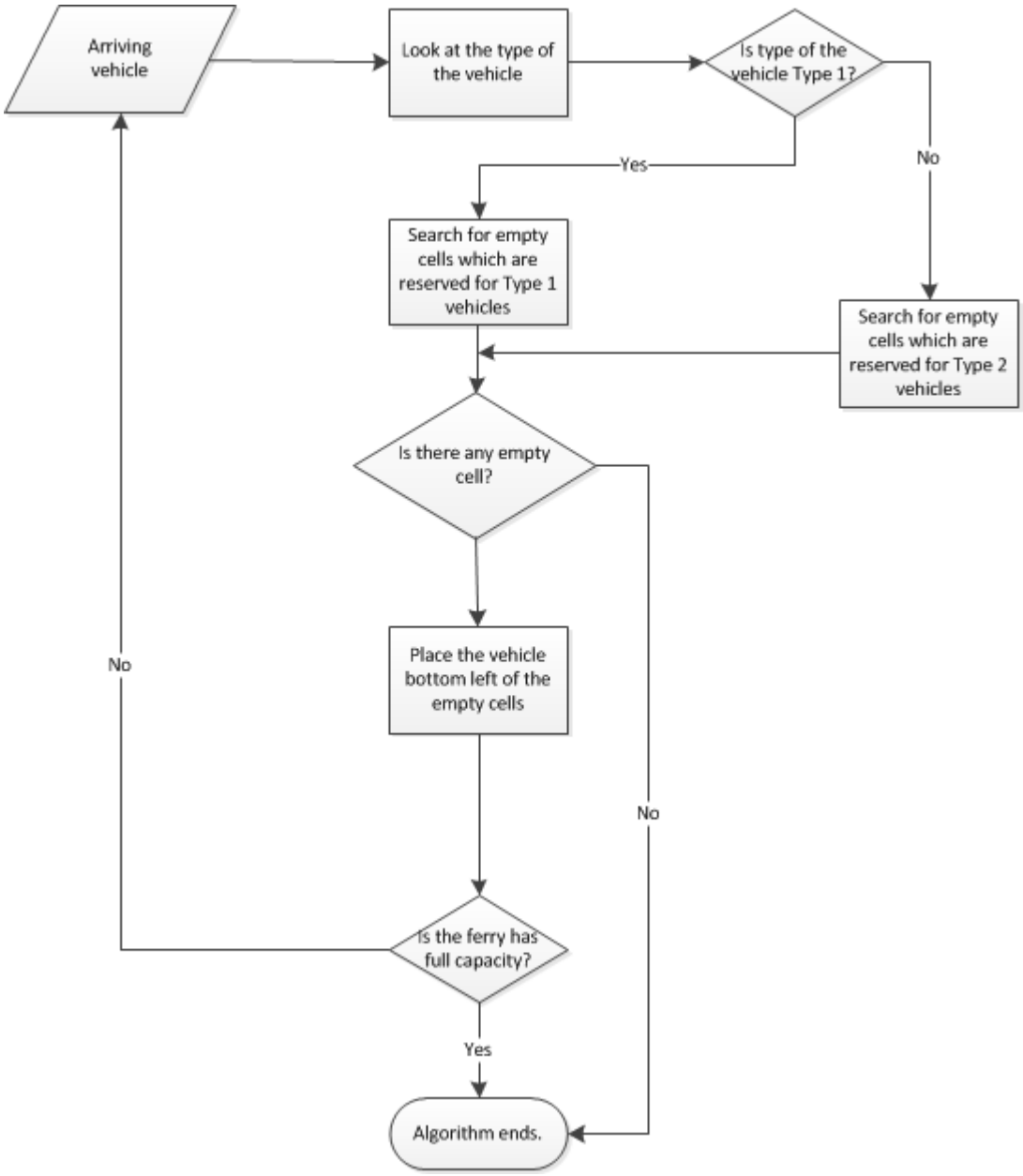

Figure 3.4: Flow chart of Vehicle Placement Algorithm: Balance at the Center

Figure 3.7: Ferryboat:SADABAT

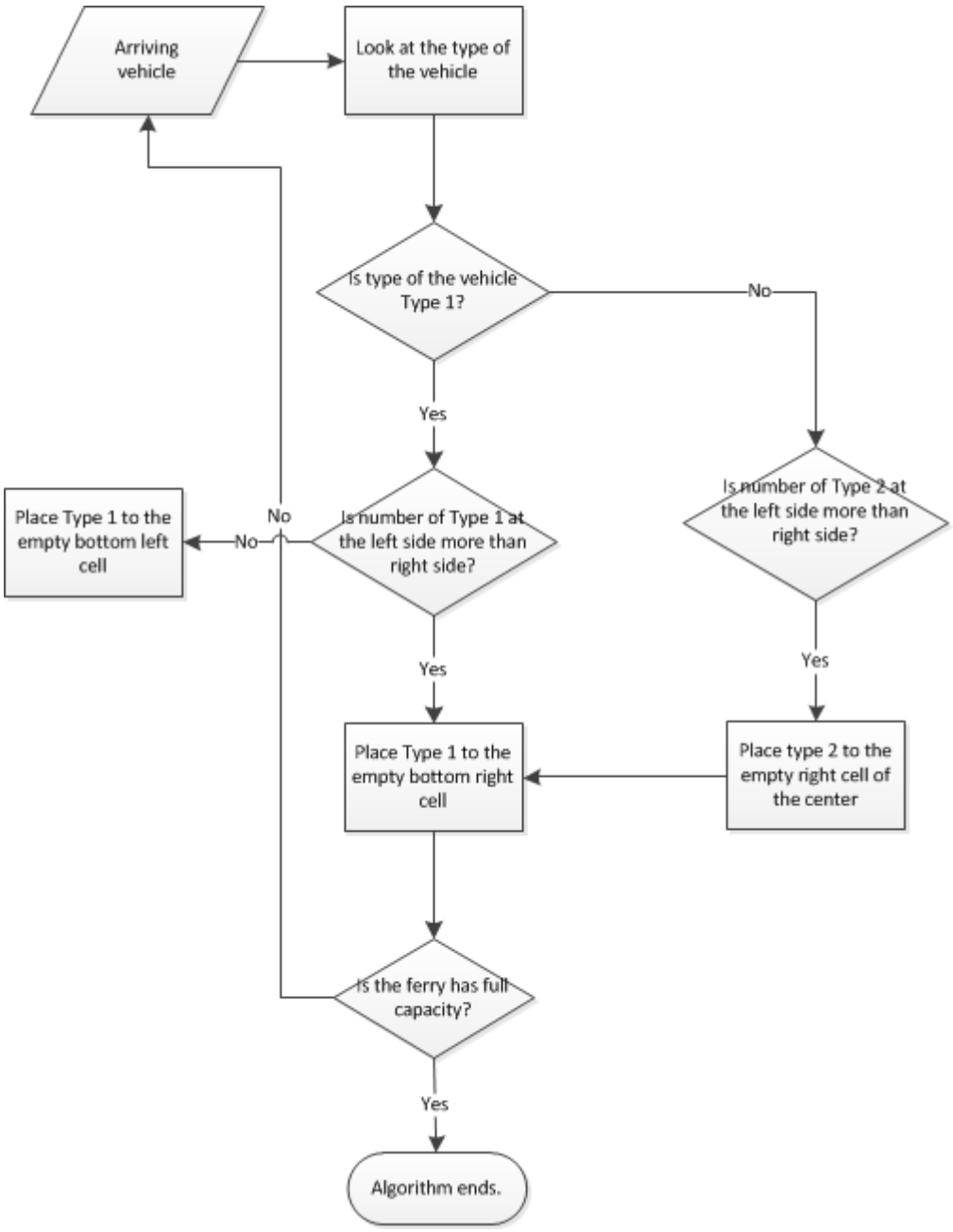

Figure 3.5: Flow chart of Vehicle Placement Algorithm:Momentum Law

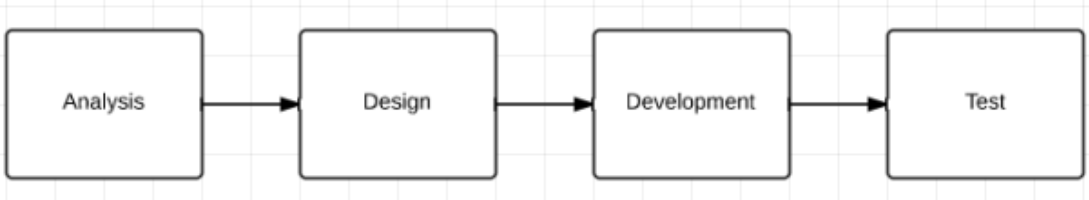

Figure 3.6: The Solution Procces

Chapter 4

Vehicle Placement Algorithm (VPA)

4.1 Maritime Transportation

Transportation is an important issue for economic and social relations of a country. Maritime transportation has a noteworthy place in the transportation sector and it is the largest carrier of freight throughout the world. It is also valuable for metropolises like Istanbul where traffic jams and environmental problems occur frequently. Also, low risk of accidents and less air pollution increases the importance of maritime transportation. Ship, ferry, sea-bus, RORO ship are some of the maritime transportation vehicles which are used to carry passengers, vehicles and cargo. Ferryboats carries both vehicle and passenger for transportation. Although Turkey has surrounded by water on three sides, maritime transportation is effective just in Marmara region.

4.2 Inland Ferryboat System

In Turkey, ferry industry varies widely because of its geographical conditions and there are lots of factors that affect public benefits. These factors are; cost effectiveness, transportation demand, safety, economic development and environmental issues.

Our study is based on Sirkeci-Harem ferry line of Bosphorus, which is operating between European side and Asian side of the city. Local people generally prefer this ferry because of traffic jam occurring on the bridges connecting the two shores. At this line, ferries don't accept big vehicles, thus serving only to motorcycles, cars, SUVs, pickups and minibuses.

4.3 The Current Situation at Ferry System

"Vehicle Recognition System" reads plaques with the sensors and determines the type of the vehicle, when vehicle arrive to the ticket office. After paying the fare according to the type of their vehicle, they queue up and start waiting to embark. Vehicles are getting placed to ferryboats by instructions of ferry flagmen. The placement process may be seen at the below Figure 4.1. The flagman that is standing in front of this queue is given the task to assign each of the cars to a location on the ferryboat so as to minimize the free area of the ship. Because the view of the flagmen is partially blocked, he can only see the first car of the queue at a time. Once this car has been assigned to a position on the ship, the cars move up and he sees the next car in the queue. Due to his years of experience, the flagman knows exactly the space required a car once he sees it. For obvious reasons, the assignment of a car cannot be changed once the decision has been made. However, ferry flagmen bring balance to the ferry subjectively using his personal eye observation experience.

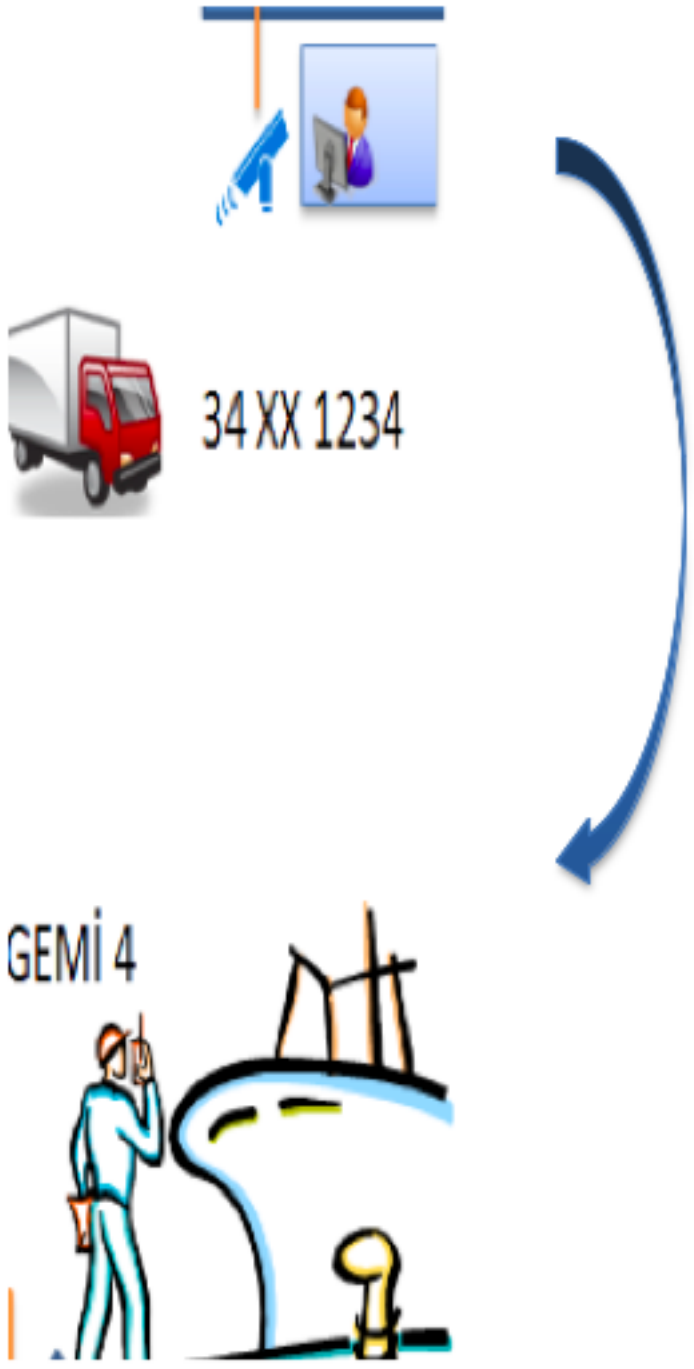

Figure 4.1: Vehicle Placement Process

4.4 Objective of the Problem

The current situation approach, which we mentioned above, may take a long time to load the ferryboat, increase waiting time for drivers and passengers. There should be empty spaces and this situation decreases the profit of the organization. Transporting more vehicles with less number of ferryboats increases the profit. The objective of the problem is the optimal placement of vehicles in the ferryboat and minimizing empty space. The other objective; which is also the starting point of our thesis; we are investigating whether there is need for an automation system to place vehicles in ferry.

4.5 Constraints of the Problem

The constraints of this problem are:

- *Capacity of the ferry:* Each ferry has different capacities it is not standard. The chosen ferry, which is mentioned at previous section "SADABAT", has capacity 80 Type 1 vehicles.
- *Balance of the ferry:* It is important for safety; heavy vehicles shouldn't be at the same side of the ferry.
- *Safety Distance between vehicles:* It is independent of the vehicle type. After parking of the vehicle passengers get off and open the doors. So there should be enough space to prevent clash of the vehicles.
- *First Come First Serve sequencing rule:* The vehicles have to be placed according to arrival sequence. There is no choice like ordering the vehicles first then placing them to ferry.

4.6 Assumptions of the Problem

The assumptions of this problem are:

- *Grid based design:* The deck of the ferryboat has divided into parallel and vertical lanes (grid based design) of equal width and equal length. Each cell is wide enough to be able to contain any of the cars. And vehicles can only be assigned to the prescribed cells according to VPA. This assumption may eliminate the "Safety distance between vehicles" constraint.

  However, new ferries has vertical park lines as we seen from "SADABAT" in Figure 3.7.

- *Vehicle types:* There is wide variety of vehicles; but two of them have been used in this thesis. The first reason of this; Sirkeci-Harem ferry line just carries car, pickup, minibus and SUV. So the vehicles have been specified as Type 1which refers to car and Type 2, which refers to pickup, minibus, and SUV.

- *Dimensions of the vehicles:* In the daily life, the dimensions of vehicles change according to their model but we assume that type 1 is one to one cell and type 2 is one to two cell .The grid based design eliminates this situations regardless of size vehicles fit to cells in our approach. as shown in Figure 4.2 ; two type 1 vehicle (pink car) equals to one type 2 vehicle (blue truck).

Figure 4.2: Vehicle Types

4.7 Solutions of the Problem

The problem was solved according to four different methods.After than the results was compared .

4.7.1 Solution of VPA: No Balance

Balance of the ferry was mentioned constraint of the problem in 4.4. The details of this algorithm are given in Chapter 3. The algorithm is solved with generated random vehicle types and arrival sequence. The percentages of vehicle types are changed and percentages of placed vehicles were analyzed. An example output is shown in Figure 4.3 Output of VPA: No Balance constraint. 120 vehicles were generated randomly and 75 % of vehicles were Type 1 and 25% of vehicles Type 2. As it can be seen from figure the heavy vehicles that are blue trucks in the figure subsided at right side of the ferry layout.

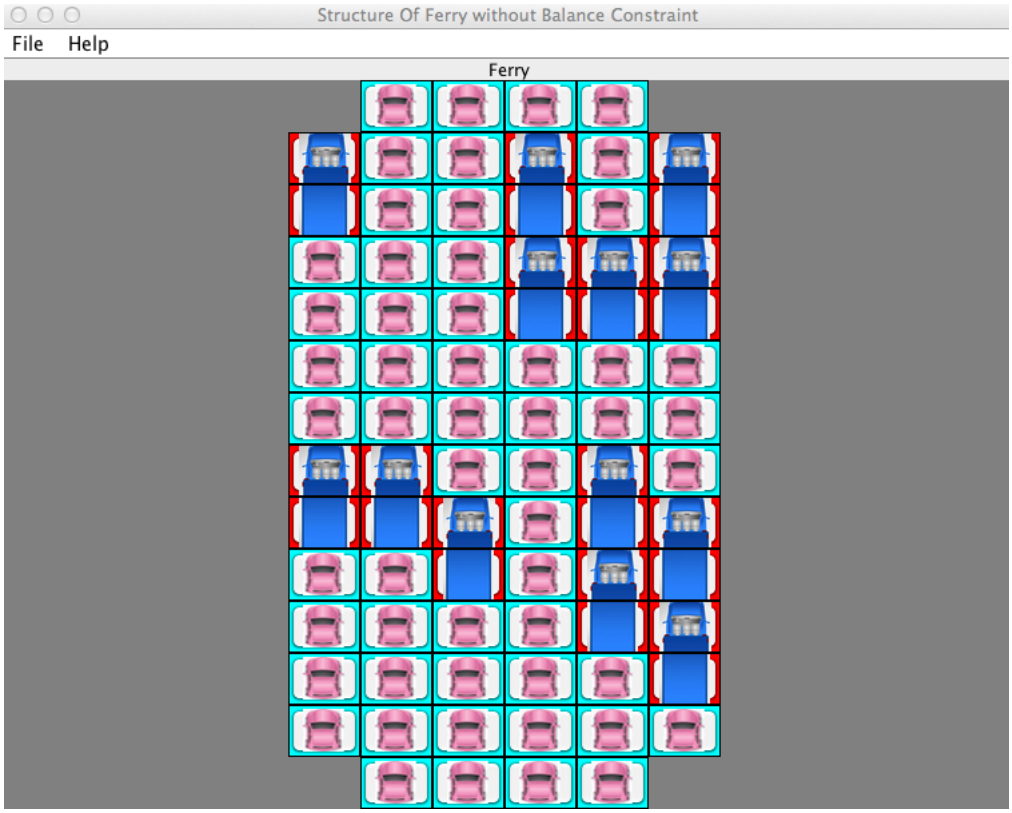

Figure 4.3: Output of VPA:No Balance

4.7.2 Solution of VPA: Balance

The second VPA is solved with balance constraint . At manual placement flagmen carries about balance of the ferry for safety of passengers. As like as VPA-without

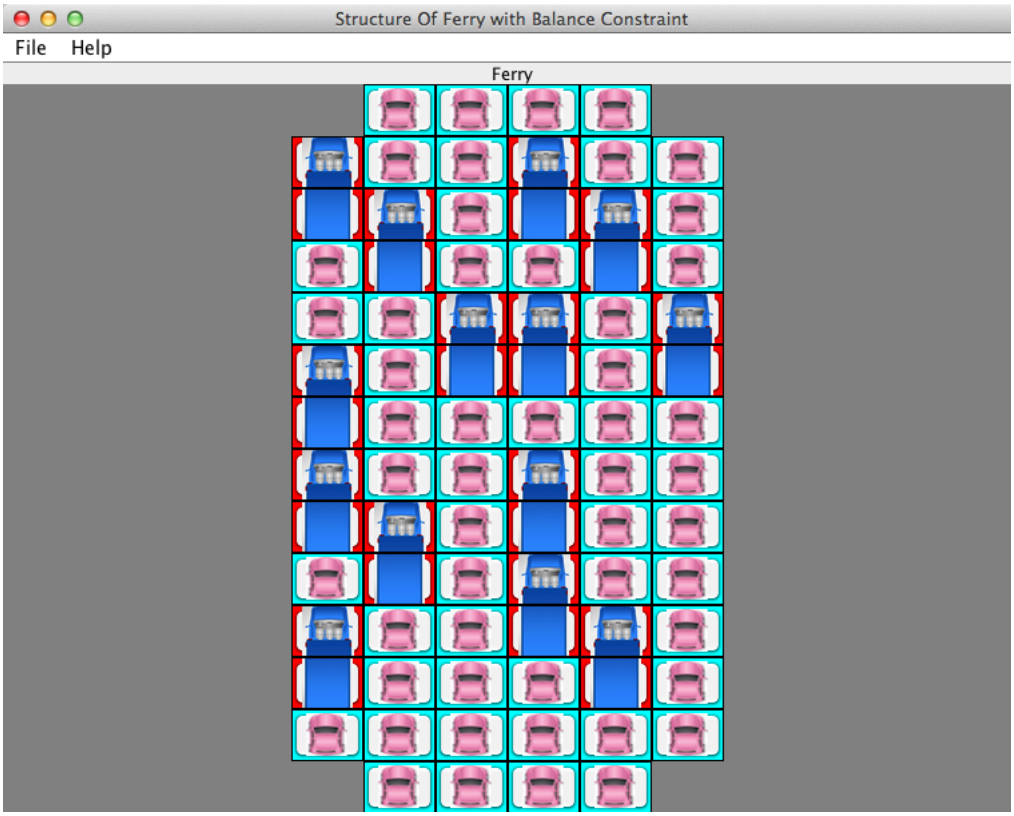

Figure 4.4: Output of VPA:Balance

balance, the details of this algorithm are given in Chapter 3.120 vehicles were generated randomly and 75% of vehicles were Type 1 and 25% of vehicles Type 2. The output may be seen at Figure 4.4 .As it is seen from figure the number of heavy vehicles at left side is equal to 7 and right side is equal to 7. The ferry is balanced according to arrival sequence. If one more type 2 vehicle had come it would been placed to right side of the ferry layout.

4.7.3 Solution of VPA: Balance at the Peripheries

At the second model condition of balance is provided according to number of vehicles at the left and right hand side,but this is not an accurate balance according to momentum laws. So the heavy vehicles placed at peripheries of the ferry layout. Firstly this grids were reserved to type 2 vehicles, after they have placed and if there is no truck at the queue type 1 vehicles are placed to reserved grids.120 vehicles were generated randomly and 75%of vehicles were Type 1 and

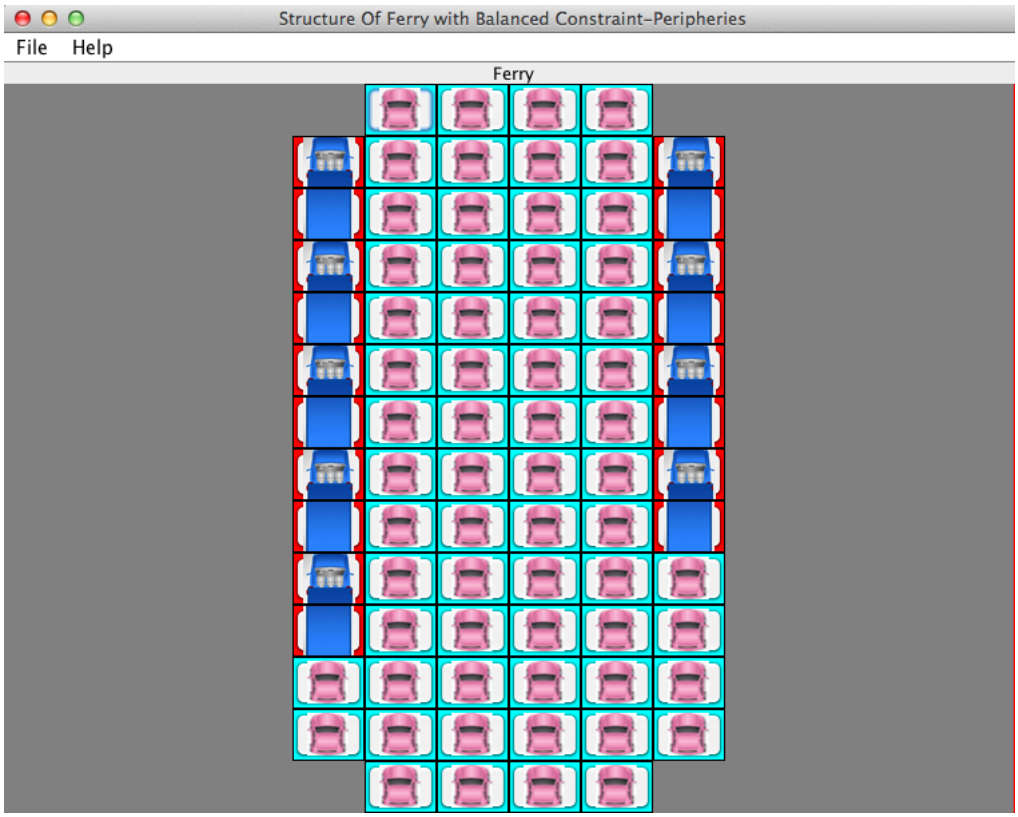

Figure 4.5: Output of VPA:Balance at the Peripheries

25% of vehicles Type 2. The output may be seen at Figure 4.5. As it is seen from figure the number of heavy vehicles at left side is equal to 5 and right side is equal to 4 and after type 2 vehicles have finished type 1 vehicles were placed to peripheries.

However, it can be said that this method secures balance more than second model because distance to the center of the ferry is equal of heavy vehicles.

4.7.4 Solution of VPA: Balance at the Center

Similar to the previous model (VPA: Balance at the Peripheries); heavy vehicles placed at the same distance to the center of the ferry. At this model heavy vehicle placed at nearest cells to center at the both left and right sides. Firstly this grids were reserved to type 2 vehicles, after they have placed and if there is no truck at the queue; type 1 vehicles are placed to reserved grids.120 vehicles

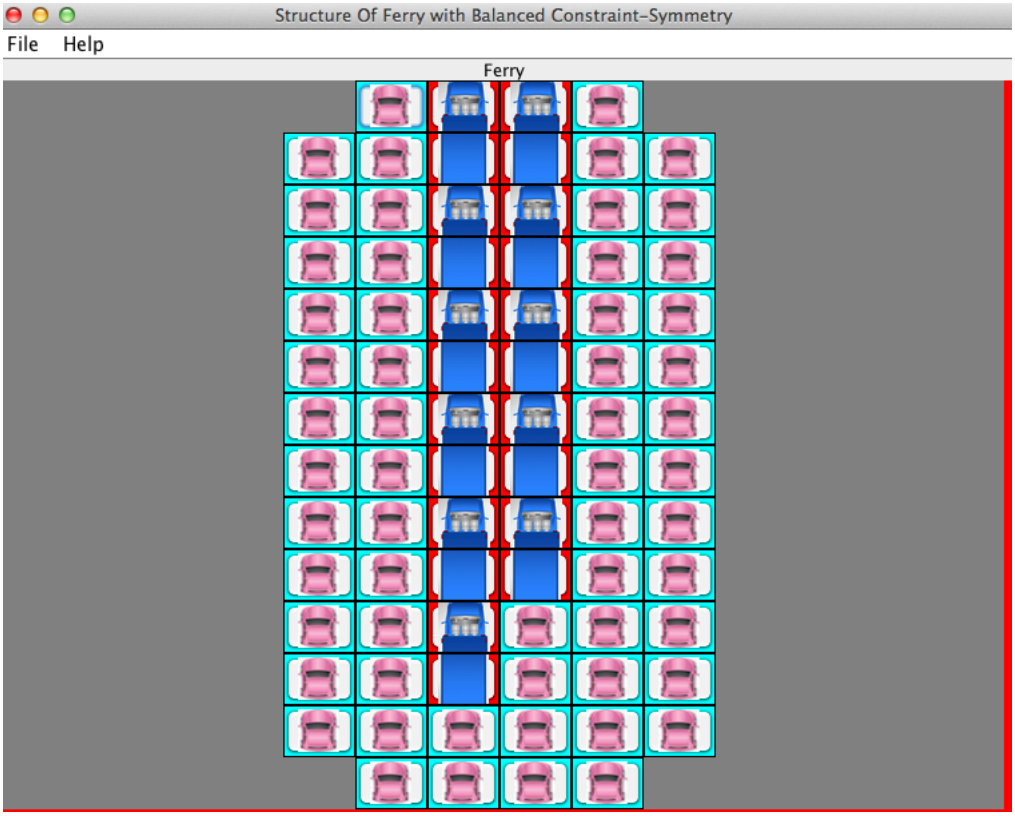

Figure 4.6: Output of VPA:Balance at the Center

were generated randomly and 75% of vehicles were Type 1 and 25% of vehicles Type 2. The output may be seen at Figure 4.6. As it is seen from figure the number of heavy vehicles at left side is equal to 5 and right side is equal to 4 and after type 2 vehicles have finished type 1 vehicles were placed to peripheries. This model secures balance as like as third model but placement may be more difficult according to arrival order of trucks.If most of them are placed before small vehicles, placement of small vehicles may be difficult.

4.7.5 Solution of VPA: Momentum Law

Solution of this algorithm shows that same type of vehicles are placed to the ferry layout equidistantly. According to algorithm; type 2 vehicles are started to placing from center line of the ferry and type 1 vehicles are started from peripheries. Firstly they are placed to the left side and then right side. Also; the number of each type of vehicles are equal at each side of the ferry. It can

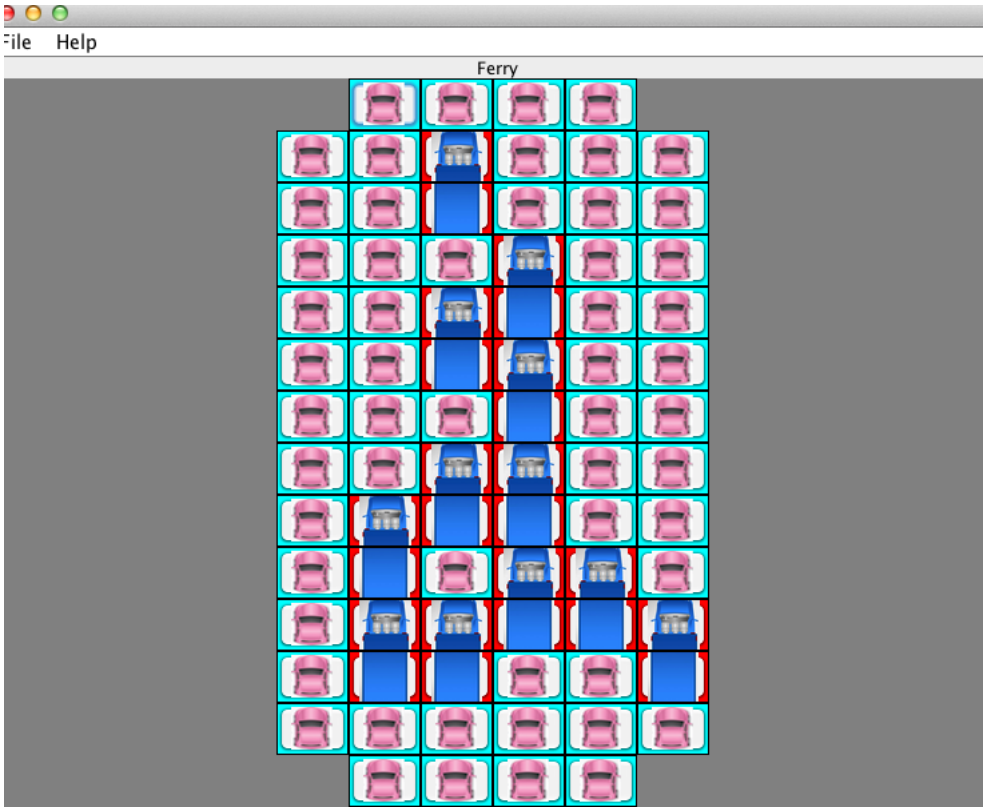

Figure 4.7: Output of VPA:Momentum Law

be said that this algorithm places vehicles more balanced. The details of this algorithm may be seen at the below pseudo-code and Figure 4.7. 120 vehicles were generated randomly and 75% of vehicles were Type 1 and 25% of vehicles Type 2. The output may be seen at Figure 4.6. As it is seen from figure the number of heavy vehicles at the columns nearest to center is equal to 4.

4.8 Results and Analysis

4.8.1 Comparing The Algorithms

All five heuristic solutions have been coded in Java an for benchmarking the solutions a series of tests were run on the program. The percentages of type 1 and type 2 were changed for different combinations. The software run 21 times with the same vehicle order and 120 vehicles was produced for query. The vehicle types 1 and 2 was produced with different percentages. For example 5 percent of

the query is type 2 and 95 percent of the query is type 2. The change of vehicle percentage was investigated by varying the rate of type of vehicles as you see in Table 4.1 Percentage of Placed Vehicles for Proposed Algorithms. Then the data in the table, was graphed for two types and algorithms at Figure 4.8 and Figure 4.9.

Figure 4.8 shows that percentage of placed vehicle vs. Percentage of type 1. As we see from the graph while percentage of type 1 is decreasing the percentage of placed vehicle of query decreases. Because the of type 2 vehicles take more place in the ferryboat, so this decreases the fill rate. We have run the software for 5 different algorithm but we have seen that there is no big difference between the algorithms. It can only be said that VPA: Balance at the Peripheries and VPA: Balance at the Center has lower fill rates. Because of the structure of the algorithms; they have reserved cells, if the percentage of type 2 is low there would be empty cells because type 2 couldn't be placed type 2 instead of type 1 for reserved cells of type 1.

In contrast to previous graph (Figure 4.8), in Figure 4.9; if the percentage of type 2 vehicles increases the fill rate of the ferry decreases because of type 2 vehicles take up large spaces in the ferry and most of the vehicle at the query have to continue to wait for next ferry.

Table 4.1: Percentage of Placed Vehicles for Proposed Algorithms

Simulation	Type 1%	Type 2%	VPA:No Balance	VPA:Balance	VPA:Balance Perip.	VPA:Balance Center	VPA:Momentum
1	0%	100%	39.17%	39.17%	25.67%	26.33%	42.50%
2	5%	95%	40.83%	40.00%	28.33%	28.67%	43.33%
3	10%	90%	40.83%	42.53%	31.67%	30.00%	44.17%
4	15%	85%	40.00%	43.33%	33.33%	30.00%	45.00%
5	20%	80%	42.50%	45.00%	36.67%	33.33%	45.83%
6	25%	75%	43.33%	45.83%	38.33%	35.00%	47.50%
7	30%	70%	44.17%	46.67%	40.00%	36.67%	47.50%
8	35%	65%	46.67%	48.33%	43.33%	41.67%	50.00%
9	40%	60%	48.33%	48.33%	43.33%	45.00%	51.67%
10	45%	55%	49.17%	48.33%	43.33%	46.67%	52.50%
11	50%	50%	50.83%	51.67%	46.67%	50.00%	54.17%
12	55%	45%	51.67%	52.50%	51.67%	51.67%	55.83%
13	60%	40%	53.33%	53.33%	53.33%	55.00%	56.67%
14	65%	35%	55.67%	55.67%	56.67%	55.00%	58.33%
15	70%	30%	55.83%	55.83%	56.67%	55.83%	59.17%
16	75%	25%	56.67%	57.50%	57.50%	56.67%	60.00%
17	80%	20%	58.33%	58.33%	58.33%	58.33%	61.67%
18	85%	15%	60.83%	60.83%	60.83%	60.83%	62.17%
19	90%	10%	61.67%	62.50%	62.50%	61.67%	63.00%
20	95%	5%	64.17%	64.17%	64.17%	64.17%	64.17%
21	100%	0%	66.67%	66.67%	66.67%	66.67%	66.67%

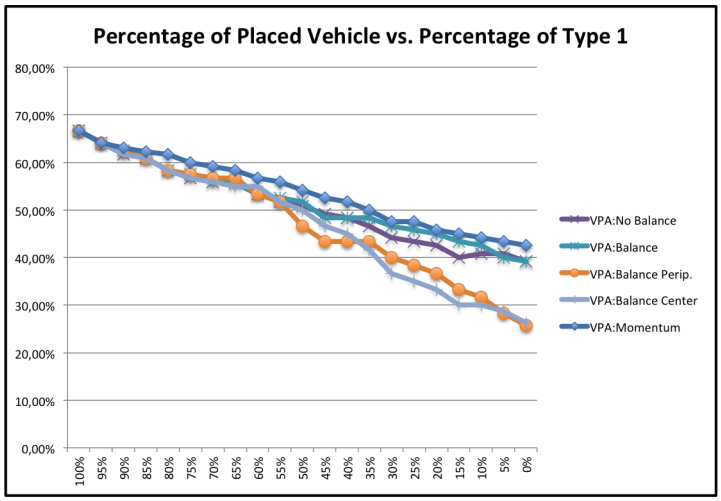

Figure 4.8: Chart of Percentage of Placed Vehicle vs. Percentage of Type 1

As we see from Figure 4.8 and Figure 4.9; while the rate of the generated vehicles is changing we change the rate of the vehicles percentage of the placed vehicles changes linearly. When we increase the rate of the type 2 vehicles which are trucks; the ferry should take less vehicle or vice versa. However, if five algorithms are compared it may be seen that there is not a significant difference. The difference just can be seen at lower volume of Type 1 and higher volume of Type 2. The first one shows that percentage of placed vehicle vs. Percentage of type 1. As we see from the graph while percentage of type 1 is decreasing the percentage of placed vehicle of query decreases. Because the of type 2 vehicles(big vehicles) take more place in the ferry, so this decreases the percentage. We have run the software for 5 different algorithm but we have seen that there is no big difference between the algorithms. But we may say that Balance at the peripheries and balance at the center has lower fill rates. Because as I have said before they have reserved cells If the percentages of type 2 is low there are empty cells because we couldn't place type 2 instead of type 1

4.8.2 Validation

To validate combined VPA, real data was collected from the ferryboat, as mentioned at previous sections; "SADABAT". It has capacity of 80 vehicles (If it is

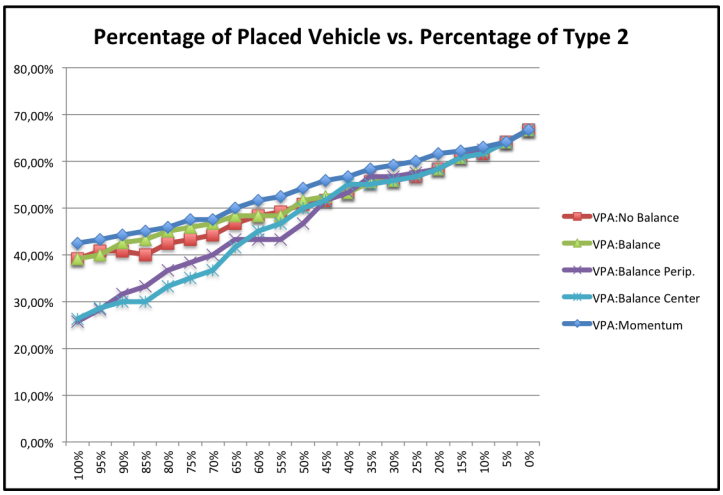

Figure 4.9: Chart of Percentage of Placed Vehicle vs. Percentage of Type 2

just filled with Type 1 vehicle). Data belong to 50 journeys from Sirkeci to Harem between the hours 11:00 and 16:30 in working day. The data shows that on the average 48 type 1 vehicle and 12 type 2 vehicle is carried on that cruise. Also flagman added that almost placement is like that except extraordinary days such as holiday days. VPA-with balance constraint model was chosen for validation because it is more realistic for ferry system. The flagmen take into account balance of the ferry by rule of thumb. flagman added that almost placement is like that except extraordinary days such as holiday days. The model has been simulated 50 times on the program according to real data statistics. Percentages of the types are like 75% Type 1 and 25% Type 2; these percentages determined according to real data averages. So random data is produced according to these percentages and also vehicles come to queue at random order. According to this simulation while 48 type 1 vehicles could be placed in average, at the automated system 61 type 1 vehicles was placed in average. If two system are compared according to revenue and waiting time at the queue, which can be said as performance criteria of the model, the automated system seems better clearly. The results are given in Table 4.2 and graph is drawn in Figure 4.10 to see the difference between revenues.

The revenue is calculated according to ticket fare of the Sirkeci-Harem ferry line, that are shown in Table 4.3. But, because of the assumptions the model has 2

Table 4.2: Revenue of AS vs. Revenue of MS

ID	AS-type 1	MS-type 1	AS-type 2	MS-type 2	AS-Revenue(TL)	MS-Revenue(TL)
1	55	54	12	9	652.20	591.74
2	58	40	11	16	655.29	575.72
3	56	50	12	11	653.45	586.82
4	59	46	10	13	656.79	582.88
5	61	46	9	13	658.89	583.02
6	66	51	7	10	664.30	588.59
7	65	52	8	10	662.84	589.37
8	55	51	12	10	652.35	588.42
9	62	47	9	12	659.99	584.08
10	60	53	10	9	657.44	590.99
11	61	43	9	14	658.75	579.77
12	64	44	8	14	661.53	580.49
13	57	43	12	14	653.80	579.47
14	62	50	9	11	659.36	587.38
15	62	52	9	10	659.62	589.05
16	59	55	10	9	656.41	592.79
17	68	39	6	17	666.34	574.52
18	64	48	8	12	662.47	585.25
19	58	55	11	9	654.87	592.63
20	68	42	6	15	666.69	578.02
21	72	52	4	10	671.37	589.77
22	64	48	8	12	662.54	585.20
23	64	48	8	12	662.51	584.80
24	62	47	9	12	660.05	584.18
25	60	41	10	15	657.79	577.20
26	59	52	10	10	656.52	589.47
27	62	44	9	14	659.90	580.68
28	57	48	12	12	654.04	585.42
29	63	45	9	13	660.73	582.06
30	61	50	10	11	658.20	587.39
31	62	44	9	14	659.96	580.49
32	67	47	6	13	665.51	583.73
33	57	53	11	10	654.20	590.07
34	65	44	8	14	662.66	580.06
35	61	48	10	12	658.18	585.47
36	61	39	10	16	658.43	575.36
37	68	45	6	14	666.45	581.24
38	61	47	10	13	658.24	583.59
39	61	48	9	12	659.16	584.79
40	56	45	12	13	653.21	581.99
41	64	42	8	15	661.59	577.93
42	59	47	10	12	656.56	584.22
43	70	48	5	12	669.11	584.91
44	63	48	9	12	660.57	584.90
45	54	42	13	15	650.43	577.97
46	62	46	9	13	659.32	582.21
47	61	55	10	8	658.10	593.15
48	59	46	10	13	656.62	582.53
49	65	46	7	13	663.37	583.12
50	51	40	15	16	647.09	575.52
AVG	**61**	**47**	**9**	**12**	**659.12**	**584.01**

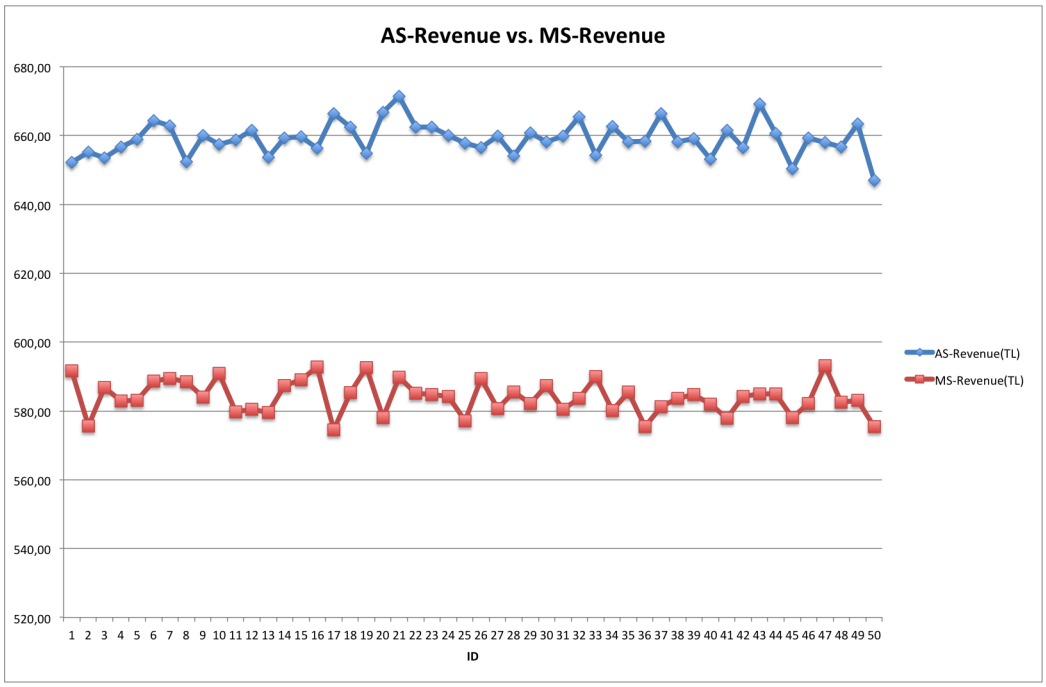

Figure 4.10: Chart of Revenues

types of vehicles. So the unit fare for type 1 is 8,5 TL and unit fare for type 2 is the average of the fares of SUV, minibus, midibus and truck that is 14,75 TL.

Table 4.3: Ticket Fares

Type Of Vehicle	**Fare (TL)**
Car	8.5
SUV	12
Minibus	12
Midibus	15
Truck	20

However, in regard to results placing more vehicles decreases waiting time at the queue. According to result table automated system places 11 vehicles more in average and if we assume average waiting time for a vehicle is 15 minutes, these 11 vehicles that couldn't be placed with manual system to first ferry, would be waiting for 15 minutes more. The number of excess placed vehicles can be seen in Table 4.4 and in Figure 4.11. As a summary automated system for VPA places

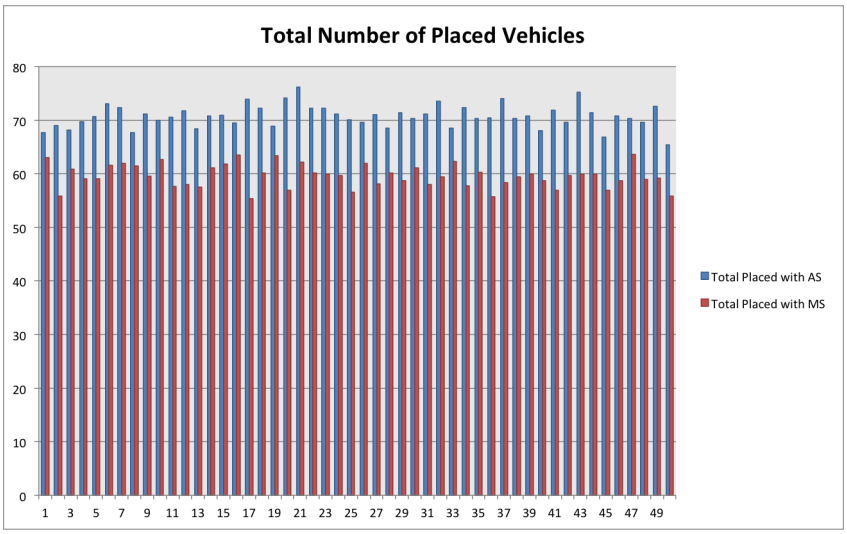

Figure 4.11: Chart of Total Placed Vehicles

vehicles more efficiently and increases the revenue. Although there are some assumptions; it may be said that VPA gives near optimal solutions because of it is a heuristic model.

Table 4.4: Number Placed Vehicles of AS and MS

ID	Total Placed with AS	Total Placed with MS	Difference
1	68	63	5
2	69	56	13
3	68	61	7
4	70	59	11
5	71	59	11
6	73	62	11
7	72	62	10
8	68	62	6
9	71	60	12
10	70	63	7
11	71	58	13
12	72	58	14
13	68	58	11
14	71	61	10
15	71	62	9
16	70	63	6
17	74	55	19
18	72	60	12
19	69	63	5
20	74	57	17
21	76	62	14
22	72	60	12
23	72	60	12
24	71	60	11
25	70	57	14
26	70	62	8
27	71	58	13
28	68	60	8
29	71	59	13
30	70	61	9
31	71	58	13
32	74	59	14
33	69	62	6
34	72	58	14
35	70	60	10
36	70	56	15
37	74	58	16
38	70	59	11
39	71	60	11
40	68	59	9
41	72	57	15
42	70	60	10
43	75	60	15
44	71	60	11
45	67	57	10
46	71	59	12
47	70	64	7
48	70	59	11
49	73	59	13
50	65	56	10
AVG	71	60	11

Conclusion

In this thesis, a combined algorithm is proposed for placement of rectangular vehicles in a ferry. The algorithm, VPA, is a combination of First Fit Bin-Packing, Bottom Left and Knapsack algorithms. VPA uses FCFS sequencing rule of First Fit Bin Packing algorithm, placing the items to the bottom left of the layout method from context of BL algorithm and filling knapsack with optimum capacity objective of Knapsack algorithm.

VPA is developed in five variant ways that are VPA-No Balance, VPA-Balance , VPA-Balance at the Peripheries, VPA-Balance at the Center and VPA-Momentum Law.Based on the results obtained in the previous chapter, the proposed models were compared to each other but there seem no remarkable difference related to number of vehicles placed to ferry floor. But balance is an important constraint for safety of passengers and vehicles in the ferry. So the VPA-with balance constraint should preferably be used in real life.

Not only the efficiency of the proposed algorithms also the algorithm was evaluated using randomly generated test cases and it has been tested on a realistic ferry line data. Numerical tests were showed that the proposed VPA can improve the placement results significantly compared to the actual data and provides quick results.However; if an automated system is used, the revenue of the organization may increase as shown in the report Also; the reasons have been shown to be NP-complete and therefore practical solution for VPP tend to be a heuristic solution that may not yield optimum results.

Although it is a heuristic method, the results show that the proposed combined algorithm performs better than the current manual placement process. If this model is used;

- Seaport efficiency will be improved,
- Queue delays will be minimized,
- Engine run times will decrease,
- Fuel cost for ferry will decrease,
- Profit of the organization will increase.

The contributions in this thesis are a new algorithm (VPA) which is a combination of First Fit Bin-Packing, Bottom Left and Knapsack algorithms and a new research topic for optimization literature which is placement of rectangular vehicles. Also as we know this research is the first demonstration of the vehicle placement in a ferry.

References

[1] HopperE. *Two Dimensional Packing utilising evolutionary algorithms and other meta-heuristic methods,*. PhD thesis, University of Wales, 2000.

[2] Garey et al. Computers and intractability: A guide to the theory of np-completeness. *W.H. Freeman Co.*, 1979.

[3] Andrea Lodi, Silvano Martello, and Michele Monaci. *Two-dimensional packing problems: A survey.* Volume 141 of Lodi et al. [8], 2002. doi: 10.1016/S0377-2217(02)00123-6. URL `http://www.sciencedirect.com/science/article/pii/S0377221702001236`.

[4] LodiS.MartelloD.Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Appl. Math.*, 123:379–396, 2002.

[5] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for np-hard problems. pages 46–93, 1997. URL `http://dl.acm.org/citation.cfm?id=241938.241940`.

[6] János Csirik and GerhardJ. Woeginger. On-line packing and covering problems. 1442:147–177, 1998. doi: 10.1007/BFb0029568. URL `http://dx.doi.org/10.1007/BFb0029568`.

[7] György Dósa and Jiří Sgall. First fit bin packing: A tight analysis. *ACM*, 1998.

[8] Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241 – 252, 2002. ISSN 0377-2217. doi: 10.1016/S0377-2217(02)

00123-6. URL http://www.sciencedirect.com/science/article/pii/S0377221702001236.

[9] B. Baker, E. Coffman, Jr., and R. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980. doi: 10.1137/0209064. URL http://epubs.siam.org/doi/abs/10.1137/0209064.

[10] E. Hopper and B.C.H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *European Journal of Operational Research*, 2001.

[11] G.Kendall E.K.Burke and G.Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52(4):655–671, 2004.

[12] Stefan Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88(1):165 – 181, 1996. ISSN 0377-2217. doi: 10.1016/0377-2217(94)00166-9. URL http://www.sciencedirect.com/science/article/pii/0377221794001669.

[13] K. K. Lai and J. W. M. Chan. A evolutionary algorithm for the rectangular cutting stock problem. 1997.

[14] Wei Shih. A branch and bound method for the multiconstraint zero-one knapsack problem. *The Journal of the Operational Research Society*, 30(4): 369–378, April 1979.

[15] Chaitr S. Hiremath. *New Heuristic And Metaheuristic Approaches Applied To The Multiple-choice Multidimensional Knapsack Problem.* PhD thesis, Wright State University, 2008.

Curriculum Vitae

Busra Pasali was born on 22 July 1988, in Istanbul. She received his B.S. degree in Industrial Engineering in 2010 from Isik University. She worked as a research assistant at the Department of Industrial Engineering of Işık University from 2010 to 2011. Then she started to working at The Scientific and Technological Research Council Of Turkey "TUBITAK" as researcher.

APPENDIX

```
package tr.edu.isikun.gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.MessageFormat;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.border.LineBorder;

import tr.edu.isikun.algorithm.FerryParameters;
import tr.edu.isikun.algorithm.PlacementServicer;
```

```
public class FerryGUI extends JFrame {

        private JPanel contentPane;
        private static JPanel pnlFerry;
        public static final int DISABLED_CELL = -1;
        public static final int EMPTY_CELL = 0;
        public static final int FILLED_CELL_T1 = 1;
        public static final int FILLED_CELL_T2 = 2;
        public static final int FILLED_CELL_T3 = 3;
        public static final int TRUCK_CELL = 4;

        private JMenuBar menuBar;
        private JMenu mnFile;
        private JMenuItem mntmExit;
        private JMenu mnHelp;
        private JMenuItem mntmAbout;

        static int type = 0;
        public static int carCount =120; //120 yap!
        public static int ferrsVolume=0;
        public static double per=0.0;
        //miliSecond
//      static long waitTime=200;
static long waitTime=0;
        /**
         * Launch the application.
         */
        public static void main(String[] args) {

                                try {
```

```
                                        FerryGUI frame = new
    FerryGUI(type);
                                        frame.setVisible(true);
                                        frame.placeVehicles(type,
    carCount);

//                                      FerryGUI frame2 = new
    FerryGUI(1);
//                                      frame2.placeVehicles(1,
    carCount);
//                                      frame2.setVisible(true);

                                } catch (Exception e) {
                                        e.printStackTrace();
                                }

//                              pnlFerry.
                //Get Panel All Component
                for(int i=0;i< pnlFerry.getComponentCount();i++){

                        JPanel asd=(JPanel) pnlFerry.getComponent(i);
                        //Get Visible State
                        if(!asd.isVisible()){

                                        try {
                                                Thread.sleep(waitTime);
                                        } catch (Exception e) {
                                                // TODO Auto-generated
    catch block
                                                e.printStackTrace();
                                        }
```

```
                                asd.setVisible(true);
                    // if Visible state Red Then this Vehicle is
Truck
                    if(asd.getBackground().equals(Color.RED)){

                            System.out.println("Type 2");
                            //if Truck State is fist Column  Then
add 13
                            if(i<4){
                                    JPanel asdd=(JPanel)
pnlFerry.getComponent(i+13);
                                    asdd.setVisible(true);

                            }else{

                                    JPanel asdd=(JPanel)
pnlFerry.getComponent(i+14);
                                    asdd.setVisible(true);
                            }


                    }else{
                            System.out.println("Type 1");

                    }

                    }

                }

            String infoMessage = "Bir sonraki feribota geciniz !";
```

```
                JOptionPane.showMessageDialog(null, infoMessage, "INFO",
    JOptionPane.INFORMATION_MESSAGE);


                        try {


                                int type = 1;
                                int carCount =90;
                                FerryGUI frame2 = new FerryGUI(type);
                                frame2.placeVehicles(type, carCount);
                                frame2.setVisible(true);
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
//              }
//      });

//                      pnlFerry.
        //Get Panel All Component
        for(int i=0;i< pnlFerry.getComponentCount();i++){

                JPanel asd=(JPanel) pnlFerry.getComponent(i);
                //Get Visible State
                if(!asd.isVisible()){

                                try {
                                        Thread.sleep(waitTime);
                                } catch (Exception e) {
                                        // TODO Auto-generated catch
    block
                                        e.printStackTrace();
```

```
                        }

                        asd.setVisible(true);
                // if Visible state Red Then this Vehicle is Truck
                if(asd.getBackground().equals(Color.RED)){

                        System.out.println("Type 2");
                        //if Truck State is fist Column  Then add 13
                        if(i<4){
                                JPanel asdd=(JPanel)
pnlFerry.getComponent(i+13);
                                asdd.setVisible(true);

                        }else{

                                JPanel asdd=(JPanel)
pnlFerry.getComponent(i+14);
                                asdd.setVisible(true);
                        }


                }else{
                        System.out.println("Type 1");

                }

                }

        }

    JOptionPane.showMessageDialog(null, infoMessage, "INFO",
JOptionPane.INFORMATION_MESSAGE);
```

```
                try {

                        int type = 2;
                        FerryGUI frame3 = new FerryGUI(type);
                        frame3.placeVehicles(type, carCount);
                        frame3.setVisible(true);
                } catch (Exception e) {
                        e.printStackTrace();
                }
//      }
//});

//              pnlFerry.
//Get Panel All Component
for(int i=0;i< pnlFerry.getComponentCount();i++){

        JPanel asd=(JPanel) pnlFerry.getComponent(i);
        //Get Visible State
        if(!asd.isVisible()){

                        try {
                                Thread.sleep(waitTime);
                        } catch (Exception e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }

                        asd.setVisible(true);
        // if Visible state Red Then this Vehicle is Truck
        if(asd.getBackground().equals(Color.RED)){
```

```
                System.out.println("Type 2");
                //if Truck State is fist Column  Then add 13
                if(i<4){
                        JPanel asdd=(JPanel) pnlFerry.getComponent(i+13);
                        asdd.setVisible(true);

                }else{

                        JPanel asdd=(JPanel) pnlFerry.getComponent(i+14);
                        asdd.setVisible(true);
                }


        }else{
                System.out.println("Type 1");

        }

        }

    }

JOptionPane.showMessageDialog(null, infoMessage, "INFO",
    JOptionPane.INFORMATION_MESSAGE);


try {

        int type = 3;
        int carCount =90;
```

```
        FerryGUI frame4 = new FerryGUI(type);
        frame4.placeVehicles(type, carCount);
        frame4.setVisible(true);
} catch (Exception e) {
        e.printStackTrace();
}

for(int i=0;i< pnlFerry.getComponentCount();i++){

JPanel asd=(JPanel) pnlFerry.getComponent(i);
//Get Visible State
if(!asd.isVisible()){

        try {
                Thread.sleep(waitTime);
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

        asd.setVisible(true);
// if Visible state Red Then this Vehicle is Truck
if(asd.getBackground().equals(Color.RED)){

System.out.println("Type 2");
//if Truck State is fist Column  Then add 13
if(i<4){
        JPanel asdd=(JPanel) pnlFerry.getComponent(i+13);
        asdd.setVisible(true);
```

```
}else{

        JPanel asdd=(JPanel) pnlFerry.getComponent(i+14);
        asdd.setVisible(true);
}


}else{
System.out.println("Type 1");

}

}

}

JOptionPane.showMessageDialog(null, infoMessage, "INFO",
    JOptionPane.INFORMATION_MESSAGE);



try {
        int type = 4;
        int carCount =90;
        FerryGUI frame4 = new FerryGUI(type);
        frame4.placeVehicles(type, carCount);
        frame4.setVisible(true);
} catch (Exception e) {
        e.printStackTrace();
}

for(int i=0;i< pnlFerry.getComponentCount();i++){
```

```
JPanel asd=(JPanel) pnlFerry.getComponent(i);
//Get Visible State
if(!asd.isVisible()){

        try {
                Thread.sleep(waitTime);
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

        asd.setVisible(true);
// if Visible state Red Then this Vehicle is Truck
if(asd.getBackground().equals(Color.RED)){

System.out.println("Type 2");
//if Truck State is fist Column  Then add 13
if(i<4){
        JPanel asdd=(JPanel) pnlFerry.getComponent(i+13);
        asdd.setVisible(true);

}else{

        JPanel asdd=(JPanel) pnlFerry.getComponent(i+14);
        asdd.setVisible(true);
}


}else{
System.out.println("Type 1");
```

```
}

}

}

JOptionPane.showMessageDialog(null, infoMessage, "INFO",
    JOptionPane.INFORMATION_MESSAGE);


}

        protected void placeVehicles(int produceType, int carCount)
        {
        if( 0 == produceType)
        {

    PlacementServicer.produceRatedDistrubitionCarType(carCount);

    PlacementServicer.runAlgorithm(FerryParameters.ferryState,FerryParameters.vehicles
                drawFerry(FerryParameters.ferryState);
        }
        if (1 == produceType)
        {

    PlacementServicer.produceRatedDistrubitionCarType(carCount);

    PlacementServicer.runAlgorithm2(FerryParameters.ferryState2,FerryParameters.vehicl
                drawFerry(FerryParameters.ferryState2);
        }

        if (2 ==produceType)
```

```
        {

    PlacementServicer.produceRatedDistrubitionCarType(carCount);

    PlacementServicer.runAlgorithm3(FerryParameters.ferryState3,FerryParameters.vehicl
                drawFerry(FerryParameters.ferryState3);
        }
        if (3 ==produceType)

        {

    PlacementServicer.produceRatedDistrubitionCarType(carCount);

    PlacementServicer.runAlgorithm4(FerryParameters.ferryState4,FerryParameters.vehicl
                drawFerry(FerryParameters.ferryState4);
        }
        else if (4 ==produceType)

        {

    PlacementServicer.produceRatedDistrubitionCarType(carCount);

    PlacementServicer.runAlgorithm5(FerryParameters.ferryState5,FerryParameters.vehicl
                drawFerry(FerryParameters.ferryState5);
        }
}

        private void drawFerry(int[][] ferryState) {
                for (int row = 0; row < ferryState.length; ++row) {
                        for (int col = 0; col < ferryState[row].length;
    ++col) {
```

```
                                putLabel(ferryState[row][col]);
                        }
                }
        }

        private void putLabel(int state) {

                JPanel currentPanel = new JPanel();
                currentPanel.setBackground(Color.RED);

                if (state == DISABLED_CELL ) {
//                      currentPanel.setBorder(new LineBorder(new
    Color(0, 0, 0)));
                        currentPanel.setBackground(Color.GRAY);
                }

                else if (state == FILLED_CELL_T3) {
                        currentPanel.setBorder(new LineBorder(new
    Color(0, 0, 0)));
                        currentPanel.setLayout(new BorderLayout(0,0));
                        currentPanel.setBackground(Color.RED);
                        JButton car3=new JButton();
                        car3.setBorder(null);
                        car3.setIcon(new
    ImageIcon(FerryGUI.class.getResource("/resources/T2.2.png")));
//                      currentPanel.add(new Label("
    "),BorderLayout.NORTH);
                        currentPanel.add(car3,BorderLayout.CENTER);
                        currentPanel.setVisible(false);
                }
                else if (state == FILLED_CELL_T2) {
```

```
                        currentPanel.setBorder(new LineBorder(new
    Color(0, 0, 0)));
                        currentPanel.setLayout(new BorderLayout());
                        currentPanel.setBackground(Color.RED);
                        JButton car3=new JButton();

                        car3.setIcon(new
    ImageIcon(FerryGUI.class.getResource("/resources/T2.1.png")));
//                      currentPanel.add(new Label("
    "),BorderLayout.NORTH);
                        currentPanel.add(car3,BorderLayout.CENTER);
                        currentPanel.setVisible(false);
                }


                else if (state == FILLED_CELL_T1) {

                        currentPanel.setBorder(new LineBorder(new
    Color(0, 0, 0)));
                        currentPanel.setLayout(new BorderLayout());
                        currentPanel.setBackground(Color.CYAN);
                        JButton car3=new JButton();
                        car3.setIcon(new
    ImageIcon(FerryGUI.class.getResource("/resources/T1.png")));
//                      currentPanel.add(new Label("
    "),BorderLayout.NORTH);
                        currentPanel.add(car3,BorderLayout.CENTER);
                        currentPanel.setVisible(false);
                }


                pnlFerry.add(currentPanel);
```

```
    }

    /**
     * Create the frame.
     */
    public FerryGUI(int type) {

setIconImage(Toolkit.getDefaultToolkit().getImage(FerryGUI.class.getResource("/res
            // title
            //setTitle("Structure Of Ferry");

            if ( 0 == type)
                    setTitle("Structure Of Ferry with Balance
Constraint");
            if ( 1 == type)
                    setTitle("Structure Of Ferry without Balance
Constraint");
            if (2==type)
                    setTitle("Structure Of Ferry with Balanced
Constraint-Peripheries");
            else if (3==type)
                    setTitle("Structure Of Ferry with Balanced
Constraint-Symmetry");


            // close method
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            // window bound
            setBounds(100, 100, 763, 616);
```

```
                setLocationRelativeTo(null);

                menuBar = new JMenuBar();
                setJMenuBar(menuBar);

                mnFile = new JMenu("File");
                menuBar.add(mnFile);

                mntmExit = new JMenuItem("Exit");
                mntmExit.setIcon(new
ImageIcon(FerryGUI.class.getResource("/resources/exit.png")));
                mntmExit.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent arg0) {
                                System.exit(0);
                        }
                });
                mnFile.add(mntmExit);

                mnHelp = new JMenu("Help");
                menuBar.add(mnHelp);

                mntmAbout = new JMenuItem("About ...");
                mntmAbout.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {

                                JOptionPane.showMessageDialog(rootPane,
"This project was made for thesis
demonstration.","About",JOptionPane.INFORMATION_MESSAGE);


                        }
```

```
                });
                mntmAbout.setIcon(new
    ImageIcon(FerryGUI.class.getResource("/resources/info.png")));
                mnHelp.add(mntmAbout);
                contentPane = new JPanel();
                contentPane.setBorder(new EmptyBorder(0, 0, 0, 0));
                setContentPane(contentPane);
                contentPane.setLayout(new BorderLayout(0, 0));


                // Explanation panel
                JPanel pnlExplanaiton = new JPanel();
                contentPane.add(pnlExplanaiton, BorderLayout.NORTH);
                pnlExplanaiton.setLayout(new
    FlowLayout(FlowLayout.CENTER, 0, 0));

                // Enter some info for user
                JLabel lblFerrry = new JLabel("Ferry");
                pnlExplanaiton.add(lblFerrry);

                // ferry panel
                pnlFerry = new JPanel();
                pnlFerry.setBackground(Color.RED);
//              pnlFerry.setBorder(new
    EtchedBorder(EtchedBorder.LOWERED, null, null));
                contentPane.add(pnlFerry, BorderLayout.CENTER);


                pnlFerry.setLayout(new
    GridLayout(FerryParameters.ferryState.length,
    FerryParameters.ferryState[0].length, 0, 0));
```

```
        }



}

//PlacementServicer

package tr.edu.isikun.algorithm;
import java.text.MessageFormat;
import java.util.Random;

import tr.edu.isikun.gui.FerryGUI;

import cern.jet.random.Poisson;
import cern.jet.random.engine.DRand;
import cern.jet.random.engine.RandomEngine;

public class PlacementServicer {


        private static int countLeft;
        private static int countRight;
        public static int carPlaced;
        public static int truckPlaced;
        public static double per=0.0;
        public static void producePoissonCarType(int count)
        {

//              int totalCount = 0;
```

```
                double lambda = 1;
                RandomEngine engine = new DRand(1);
                Poisson poisson = new Poisson(lambda, engine);
                int poissonObs = poisson.nextInt();
                for(int i= 0 ; i < count; i++)
                {

                        poissonObs = poisson.nextInt();
//                      if(totalCount<60){

                                FerryParameters.vehicles[i] = poissonObs;
//                              System.out.println("Gelen Ara[U+FFFD]
   Tipi: " + poissonObs);

//                      }else{
                                poissonObs = 1;
                                FerryParameters.vehicles[i] = poissonObs;

//                              if(totalCount==50)
//                                      break;
//                      }
//
//                      if(poissonObs == 1)
//                              totalCount++;
//                      else
//                              poissonObs += 2;

                }
        }
//
```

```
    public static void produceRatedDistrubitionCarType(int
totalCount)
    {
            int type1Count = (totalCount *
FerryParameters.RATE_OF_CAR_TYPE_1) / 100;
            int type2Count = (totalCount*
FerryParameters.RATE_OF_CAR_TYPE_2) / 100;

            // fill %x type 1
            for(int i= 0 ; i < type1Count; i++)
            {
                FerryParameters.vehicles[i] = 1;
            }

            // fill %y type 2
            for(int i= 0 ; i < type2Count; i++)
            {
                FerryParameters.vehicles[i + type1Count] = 2;
            }


            // shuffle vehicles
            shuffle(totalCount);

            calculateVolume();
    }


    private static void calculateVolume() {
```

```
                int volume = 0;
                for (int i = 0; i < FerryParameters.vehicles.length;
    i++) {

                        if (volume<65) {

                                if(FerryParameters.vehicles[i] == 1)
                                        volume++;
                                else
                                        volume +=2;

                        }else{
                                FerryParameters.vehicles[i] = 1;
                                volume++;
                                if(volume == 85){
                                        FerryGUI.ferrsVolume = i;
                                        break;
                                }
                        }
                }
//
    System.out.println(FerryGUI.ferrsVolume+"------------------------");
//      System.out.println(volume+"------------------------");

        }

        // shuffle with one-pass algorithm
           public static void shuffle(int totalCount)
           {

              Random randomNumbers = new Random(1);
```

```
            // for each Card, pick another random Card and swap them
            for ( int first = 0; first < totalCount; first++ )
            {
               // select a random number between 0 and 51
               int second =  randomNumbers.nextInt( totalCount );

               // swap current Card with randomly selected Card
               int temp = FerryParameters.vehicles[ first ];
               FerryParameters.vehicles[ first ] =
FerryParameters.vehicles[ second ];
               FerryParameters.vehicles[ second ] = temp;
            } // end for
         } // end method shuffle



    public static void runAlgorithm(int[][] ferryState, int[]
vehicles) { // VPA:Balance
               int countLeft=0;
               int countRight=0;
               int countRightTir=0;
               int countLeftTir=0;
               int x= 0, y=0;
               carPlaced=0;
               truckPlaced=0;

               for (int carIndex = 0; carIndex < FerryGUI.ferrsVolume;
++carIndex)
               {
                                        if(vehicles[carIndex]==
FerryParameters.TYPE1_FERRY_STATE )
```

```
                                        {

                                                String
    indexes=getLocation(ferryState,14);
//
    System.out.println(indexes);

    if(!indexes.equals("")){


    x=Integer.valueOf(indexes.split("-")[0]);

    y=Integer.valueOf(indexes.split("-")[1]);

    ferryState[x][y]=FerryParameters.TYPE1_FERRY_STATE;


    carPlaced++;

//
    countRight += FerryParameters.TYPE1_CELL_INCREASE;
                                                        }
                                                }
//

    if(vehicles[carIndex]==FerryParameters.TYPE2_FERRY_STATE)
                                        {

    if(countLeftTir>countRightTir)
                                                {
```

```
                                                    String
indexes=getLocationRight(ferryState,14);

if(!indexes.equals("")){


x=Integer.valueOf(indexes.split("-")[0]);

y=Integer.valueOf(indexes.split("-")[1]);

ferryState[x][y]=FerryParameters.TYPE2_FERRY_STATE;

ferryState[x+1][y]=FerryParameters.TYPE22_FERRY_STATE;

countRight+=FerryParameters.TYPE2_CELL_INCREASE;

countRightTir++;

                                                    }
                                                }
                                                else if
(countLeftTir<=countRightTir)
                                                {
                                                    String
indexes=getLocationLeft(ferryState,14);

if(!indexes.equals("")){


x=Integer.valueOf(indexes.split("-")[0]);

y=Integer.valueOf(indexes.split("-")[1]);
```

```
ferryState[x][y]=FerryParameters.TYPE2_FERRY_STATE;

ferryState[x+1][y]=FerryParameters.TYPE22_FERRY_STATE;

countLeft+= FerryParameters.TYPE2_CELL_INCREASE;

countLeftTir++;
                                                        }
                                                    }
                                                    truckPlaced++;
                                                }
                }

                per=1.0*(carPlaced+truckPlaced)/(FerryGUI.carCount);
                String actualResult3 =
MessageFormat.format("{0,number,%#.##}", per);
                System.out.println("Percentage of Placed Vehicle : "+
actualResult3);
                System.out.println("Placed Vehicle : "+(carPlaced +
truckPlaced));
                System.out.println("Waiting at the Queue :
"+(FerryGUI.carCount - (carPlaced+truckPlaced)));


                }

    public static void runAlgorithm5(int[][] ferryState, int[]
vehicles) { // VPA:Momentum Law
                int countRightTruck=0;
                int countLeftTruck=0;
```

```
            int countRightCar=0;
            int countLeftCar=0;
            int x= 0, y=0;
            carPlaced=0;
            truckPlaced=0;

            for (int carIndex = 0; carIndex < FerryGUI.ferrsVolume;
++carIndex)
            {

if(vehicles[carIndex]==FerryParameters.TYPE1_FERRY_STATE)
                    {
                            if(countLeftCar>countRightCar)
                            {
                                    String
indexes=getLocationRightCar(ferryState,14);
                                    if(!indexes.equals(""))
                                    {

x=Integer.valueOf(indexes.split("-")[0]);

y=Integer.valueOf(indexes.split("-")[1]);

ferryState[x][y]=FerryParameters.TYPE1_FERRY_STATE;
                                            countRightCar++;
                                    }
                            }
                            else if (countLeftCar<=countRightCar)
                            {
                                    String
indexes=getLocationLeftCar(ferryState,14);
                                    if(!indexes.equals("")){
```

```
x=Integer.valueOf(indexes.split("-")[0]);

y=Integer.valueOf(indexes.split("-")[1]);


ferryState[x][y]=FerryParameters.TYPE1_FERRY_STATE;
                                        countLeftCar++;
                                }
                        }
                        carPlaced++;
                }


if(vehicles[carIndex]==FerryParameters.TYPE2_FERRY_STATE)
                {
                        if(countLeftTruck>countRightTruck)
                        {
                                String
indexes=getLocationRightTruck(ferryState,14);
                                if(!indexes.equals(""))
                                {

x=Integer.valueOf(indexes.split("-")[0]);

y=Integer.valueOf(indexes.split("-")[1]);

ferryState[x][y]=FerryParameters.TYPE2_FERRY_STATE;

ferryState[x+1][y]=FerryParameters.TYPE22_FERRY_STATE;
                                        countRightTruck++;
```

```
                                        }
                                }
                                else if (countLeftTruck<=countRightTruck)
                                {
                                        String
indexes=getLocationLeftTruck(ferryState,14);
                                        if(!indexes.equals("")){


x=Integer.valueOf(indexes.split("-")[0]);


y=Integer.valueOf(indexes.split("-")[1]);


ferryState[x][y]=FerryParameters.TYPE2_FERRY_STATE;


ferryState[x+1][y]=FerryParameters.TYPE22_FERRY_STATE;
                                                countLeftTruck++;
                                        }
                                }
                                truckPlaced++;
                        }
                }

                per=1.0*(carPlaced+truckPlaced)/(FerryGUI.carCount);
                String actualResult3 =
MessageFormat.format("{0,number,%#.##}", per);
                System.out.println("Percentage of Placed Vehicle : "+
actualResult3);
                System.out.println("Placed Vehicle : "+(carPlaced +
truckPlaced));
```

```
                System.out.println("Waiting at the Queue :
    "+(FerryGUI.carCount - (carPlaced+truckPlaced)));


                }



        public static void runAlgorithm2(int[][] ferryState, int[]
    vehicles) { // VPA:No Balance

                int x= 0, y=0;
                carPlaced=0;
                truckPlaced=0;
                for (int carIndex = 0; carIndex < FerryGUI.ferrsVolume;
    ++carIndex)
                {
                                        if(vehicles[carIndex]==
    FerryParameters.TYPE1_FERRY_STATE )
                                        {


                                                        String
    indexes=getLocation(ferryState,14);
//
    System.out.println(indexes);

    if(!indexes.equals("")){


    x=Integer.valueOf(indexes.split("-")[0]);
```

```
y=Integer.valueOf(indexes.split("-")[1]);

ferryState[x][y]=FerryParameters.TYPE1_FERRY_STATE;

carPlaced++;

                                                    }

                                        }

                                        else
if(vehicles[carIndex]==FerryParameters.TYPE2_FERRY_STATE)
                                        {

                                                    String
indexes=getLocation(ferryState,14);

if(!indexes.equals("")){

x=Integer.valueOf(indexes.split("-")[0]);

y=Integer.valueOf(indexes.split("-")[1]);

ferryState[x][y]=FerryParameters.TYPE2_FERRY_STATE;

ferryState[x+1][y]=FerryParameters.TYPE22_FERRY_STATE;
```

```
truckPlaced++;
                                                        }

                                        }
                        }
                per=1.0*(carPlaced+truckPlaced)/(FerryGUI.carCount);
                String actualResult3 =
MessageFormat.format("{0,number,%#.##}", per);
                System.out.println("Percentage of Placed Vehicle : "+
actualResult3);
                System.out.println("Placed Vehicle : "+(carPlaced +
truckPlaced));
                System.out.println("Waiting at the Queue :
"+(FerryGUI.carCount - (carPlaced+truckPlaced)));
                }




    public static void runAlgorithm3(int[][] ferryState3, int[]
vehicles) { // VPA:Balance at the Peripheries

                int x= 0, y=0;
                carPlaced=0;
                truckPlaced=0;
                for (int carIndex = 0; carIndex < FerryGUI.ferrsVolume;
++carIndex)
                {
                                        if(vehicles[carIndex]==
FerryParameters.TYPE1_FERRY_STATE )
                                        {
```

```
                                                    String
    indexes=getLocation(ferryState3,14);

    if(ferryState3[13][8]==FerryParameters.TYPE1_FERRY_STATE)     {

    indexes=getLocation2(ferryState3,14);

                                            }
//
    System.out.println(indexes);

    if(!indexes.equals("")){


    x=Integer.valueOf(indexes.split("-")[0]);

    y=Integer.valueOf(indexes.split("-")[1]);
//
    if(Integer.valueOf(ferryState[x][y])==4){


    ferryState3[x][y]=FerryParameters.TYPE1_FERRY_STATE;
                                                              carPlaced++;

                                                              }
                                                }

//                          }
```

```
                                        else
    if(vehicles[carIndex]==FerryParameters.TYPE2_FERRY_STATE)
                                        {



                                                    String
    indexes=getLocation2(ferryState3,14);
//
    System.out.println(indexes);



    if(!indexes.equals("")){

                                                            x =
    Integer.valueOf(indexes.split("-")[0]);
                                                            y =
    Integer.valueOf(indexes.split("-")[1]);



    ferryState3[x][y]=FerryParameters.TYPE2_FERRY_STATE;

    ferryState3[x+1][y]=FerryParameters.TYPE22_FERRY_STATE;
                                                            truckPlaced++;

                                                            }



                                        }

                                        }
                per=1.0*(carPlaced+truckPlaced)/(FerryGUI.carCount);
```

```
            String actualResult3 =
MessageFormat.format("{0,number,%#.##}", per);
            System.out.println("Percentage of Placed Vehicle : "+
actualResult3);
            System.out.println("Placed Vehicle : "+(carPlaced +
truckPlaced));
            System.out.println("Waiting at the Queue :
"+(FerryGUI.carCount - (carPlaced+truckPlaced)));

                                    }



    public static void runAlgorithm4(int[][] ferryState4, int[]
vehicles) { // VPA:Balance at the Center
            int x= 0, y=0;
            carPlaced=0;
            truckPlaced=0;
            for (int carIndex = 0; carIndex < FerryGUI.ferrsVolume;
++carIndex)
            {
                    if(vehicles[carIndex]==
FerryParameters.TYPE1_FERRY_STATE )
                    {
                            String
indexes=getLocation(ferryState4,14);

if(ferryState4[13][8]==FerryParameters.TYPE1_FERRY_STATE)     {

indexes=getLocation2(ferryState4,14);

                    }
```

```
                                        if(!indexes.equals("")){

    x=Integer.valueOf(indexes.split("-")[0]);

    y=Integer.valueOf(indexes.split("-")[1]);



    ferryState4[x][y]=FerryParameters.TYPE1_FERRY_STATE;
                                        carPlaced++;

                                        }
                        }

//                      }
//

//

    if(vehicles[carIndex]==FerryParameters.TYPE2_FERRY_STATE)
                                        {


                                                String
    indexes=getLocation2(ferryState4,14);

    if(!indexes.equals("")){
```

```
                                                    x =
Integer.valueOf(indexes.split("-")[0]);
                                                    y =
Integer.valueOf(indexes.split("-")[1]);


ferryState4[x][y]=FerryParameters.TYPE2_FERRY_STATE;

ferryState4[x+1][y]=FerryParameters.TYPE22_FERRY_STATE;

truckPlaced++;


                                                    }}

                                        }
            per=1.0*(carPlaced+truckPlaced)/(FerryGUI.carCount);
            String actualResult3 =
MessageFormat.format("{0,number,%#.##}", per);
            System.out.println("Percentage of Placed Vehicle : "+
actualResult3);
            System.out.println("Placed Vehicle : "+(carPlaced +
truckPlaced));
            System.out.println("Waiting at the Queue :
"+(FerryGUI.carCount - (carPlaced+truckPlaced)));


        }
```

```
    public static String getLocation2(int[][] ferryState3,int
verticalCount){

            for (int horizontal = 0; horizontal <verticalCount;
horizontal++) {
                    for (int vertical = 0; vertical < 14;
vertical++) {


if(ferryState3[horizontal][vertical]==4){
                                            String
indexes=horizontal+"-"+vertical;
                                    return indexes;

                            }

                    }
            }
            return "";


            }
public static String getLocation3(int[][] ferryState3,int verticalCount){


            for (int horizontal = 0; horizontal <verticalCount;
horizontal++) {
```

```
                    for (int vertical = 0; vertical < 14;
vertical++) {


if(ferryState3[horizontal][vertical]==4){
                                        String
indexes=horizontal+"-"+vertical;
                                        return indexes;

                                }

                    }
            }
            return "";


            }

            public static String getLocation(int[][] ferryState,int
verticalCount){

                    for (int horizontal = 0; horizontal
<verticalCount; horizontal++) {
                                for (int vertical = 0; vertical < 14;
vertical++) {


if(ferryState[horizontal][vertical]==0){
                                                        String
indexes=horizontal+"-"+vertical;
                                                return indexes;
```

```
                                    }

                            }
                        }
                        return "";

                        }

    public static String getLocationRight(int[][] ferryState,int
verticalCount){

            for (int horizontal = 0; horizontal <verticalCount;
horizontal++) {
                    for (int vertical = 7; vertical < 14;
vertical++) {


if(ferryState[horizontal][vertical]==0){
                                            String
indexes=horizontal+"-"+vertical;
                                    return indexes;

                            }

                        }
            }
            return "";

            }
    public static String getLocationLeft(int[][] ferryState,int
verticalCount){
```

```
            for (int horizontal = 0; horizontal < verticalCount;
horizontal++) {
                    for (int vertical = 0; vertical < 7; vertical++)
{


if(ferryState[horizontal][vertical]==0){
                                            String
indexes=horizontal+"-"+vertical;
                                    return indexes;

                            }

                    }
            }
            return "";

            }

    public static String getLocationRightCar(int[][] ferryState,int
verticalCount){

            for (int horizontal = 0; horizontal <verticalCount;
horizontal++) {
                    for (int vertical = 13; vertical >= 7;
vertical--) {


if(ferryState[horizontal][vertical]==0){
                                            String
indexes=horizontal+"-"+vertical;
                                    return indexes;
```

```
                        }

                    }
                }
                return "";

                }
    public static String getLocationLeftCar(int[][] ferryState,int
verticalCount){

                for (int horizontal = 0; horizontal < verticalCount;
horizontal++) {
                    for (int vertical = 0; vertical < 7; vertical++)
{


if(ferryState[horizontal][vertical]==0){
                                            String
indexes=horizontal+"-"+vertical;
                                    return indexes;

                        }

                    }
                }
                return "";

                }

    public static String getLocationRightTruck(int[][]
ferryState,int verticalCount){
```

```
            for (int horizontal = 0; horizontal <verticalCount;
horizontal++) {
                    for (int vertical = 7; vertical < 14;
vertical++) {


if(ferryState[horizontal][vertical]==0){
                                            String
indexes=horizontal+"-"+vertical;
                                    return indexes;

                            }

                    }
            }
            return "";

            }
    public static String getLocationLeftTruck(int[][] ferryState,int
verticalCount){

            for (int horizontal = 0; horizontal < verticalCount;
horizontal++) {
                    for (int vertical = 6; vertical >= 0;
vertical--) {


if(ferryState[horizontal][vertical]==0){
                                            String
indexes=horizontal+"-"+vertical;
                                    return indexes;
```

```
                                }

                        }
                }
                return "";

                }


}


//FerrryParameters

package tr.edu.isikun.algorithm;

import tr.edu.isikun.gui.FerryGUI;


public class FerryParameters {

        public static int [][] ferryState = new int[][] {
                //  |  0 |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8
   |  9  |  10 |  11  |  12  |  13  |
                    { -1 , -1  , -1  , -1  , -1  , 0  ,  0  ,  0  ,
   0  , -1  ,  -1 ,  -1  ,  -1  ,  -1  },
            // line 0
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
   0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },
            // line 1
```

```
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 2
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 3
                    { -1 , -1  , -1  , -1 ,  0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 4
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 5
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1,   -1  ,  -1  ,  -1  },
        // line 6
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 7
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 8
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 9
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 10
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },
// line 11
```

```
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },                          //
line 12
                    { -1 , -1  , -1  , -1  , -1 , 0  ,  0  ,  0  ,
0  , -1  ,  -1 ,  -1  ,  -1  ,  -1  },
   // line 13



            };

    public static int [][] ferryState2 = new int[][] {
            //  |  0 |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8
|  9  |  10 |  11  |  12  |  13  |
            { -1 , -1  , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  , -1
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 0
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 1
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 2
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 3
            { -1 , -1  , -1  , -1 ,  0  ,  0  ,  0  ,  0  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 4
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 5
```

```
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,  0  ,  0
 ,  -1,   -1  ,  -1  ,  -1  },                                  //
line 6
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 7
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 8
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 9
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 10
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                              // line
11
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                          // line 12
            { -1 , -1  , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  , -1
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 13



            };
    public static int [][] ferryState3 = new int[][] {
    //  |  0 |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8  |  9  |
 10 |  11  |  12  |  13  |
```

```
            { -1 , -1  , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  , -1
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 0
            { -1 , -1  , -1  , -1  , 4  ,  0  ,  0  ,  0  ,  0  ,  4
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 1
            { -1 , -1  , -1  , -1  , 4  ,  0  ,  0  ,  0  ,  0  ,  4
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 2
            { -1 , -1  , -1  , -1  , 4  ,  0  ,  0  ,  0  ,  0  ,  4
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 3
            { -1 , -1  , -1  , -1 ,  4  ,  0  ,  0  ,  0  ,  0  ,  4
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 4
            { -1 , -1  , -1  , -1  , 4  ,  0  ,  0  ,  0  ,  0  ,  4
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 5
            { -1 , -1  , -1  , -1  , 4  ,  0  ,  0  ,  0  ,  0  ,  4
 ,  -1,   -1  ,  -1  ,  -1  },                                  //
line 6
            { -1 , -1  , -1  , -1  , 4  ,  0  ,  0  ,  0  ,  0  ,  4
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 7
            { -1 , -1  , -1  , -1  , 4  ,  0  ,  0  ,  0  ,  0  ,  4
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 8
            { -1 , -1  , -1  , -1  , 4  ,  0  ,  0  ,  0  ,  0  ,  4
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 9
```

```
            { -1 , -1  , -1  , -1  , 4  ,  0  ,  0  ,  0  ,  0  ,  4
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 10
            { -1 , -1  , -1  , -1  , 4  ,  0  ,  0  ,  0  ,  0  ,  4
 ,  -1 ,  -1  ,  -1  ,  -1  },                                // line
11
            { -1 , -1  , -1  , -1  , 4  ,  0  ,  0  ,  0  ,  0  ,  4
 ,  -1 ,  -1  ,  -1  ,  -1  },                              // line 12
            { -1 , -1  , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  , -1
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 13



            };
    public static int [][] ferryState4 = new int[][] {
    //  |  0 |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8  |  9  |
 10 |  11  |  12  |  13  |
            { -1 , -1  , -1  , -1  , -1  , 0  ,  4  ,  4  ,  0  , -1
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 0
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  4  ,  4  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 1
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  4  ,  4  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 2
            { -1 , -1  , -1  , -1  , 0  ,  0  ,  4  ,  4  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 3
```

```
            { -1 , -1  , -1  , -1 ,  0  ,  0  ,  4  ,  4  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 4
            { -1 , -1  , -1  , -1 , 0  ,  0  ,  4  ,  4  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 5
            { -1 , -1  , -1  , -1 , 0  ,  0  ,  4  ,  4  ,  0  ,  0
 ,  -1,   -1  ,  -1  ,  -1  },                                  //
line 6
            { -1 , -1  , -1  , -1 , 0  ,  0  ,  4  ,  4  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 7
            { -1 , -1  , -1  , -1 , 0  ,  0  ,  4  ,  4  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 8
            { -1 , -1  , -1  , -1 , 0  ,  0  ,  4  ,  4  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 9
            { -1 , -1  , -1  , -1 , 0  ,  0  ,  4  ,  4  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 10
            { -1 , -1  , -1  , -1 , 0  ,  0  ,  4  ,  4  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                                // line
11
            { -1 , -1  , -1  , -1 , 0  ,  0  ,  4  ,  4  ,  0  ,  0
 ,  -1 ,  -1  ,  -1  ,  -1  },                              // line 12
            { -1 , -1  , -1  , -1 , -1  , 0  ,  4  ,  4  ,  0  , -1
 ,  -1 ,  -1  ,  -1  ,  -1  },                                  //
line 13
```

```
        };

    public static int [][] ferryState5 = new int[][] {
            //  |  0 |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8
|  9  |  10 |  11  |  12  |  13  |
                    { -1 , -1  , -1  , -1  , -1  , 0  , 0  , 0  ,
0  , -1  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 0
                    { -1 , -1  , -1  , -1  , 0  , 0  , 0  , 0  ,
0  , 0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 1
                    { -1 , -1  , -1  , -1  , 0  , 0  , 0  , 0  ,
0  , 0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 2
                    { -1 , -1  , -1  , -1  , 0  , 0  , 0  , 0  ,
0  , 0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 3
                    { -1 , -1  , -1  , -1 , 0  , 0  , 0  , 0  ,
0  , 0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 4
                    { -1 , -1  , -1  , -1  , 0  , 0  , 0  , 0  ,
0  , 0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 5
                    { -1 , -1  , -1  , -1  , 0  , 0  , 0  , 0  ,
0  , 0  ,  -1,   -1  ,  -1  ,  -1  },
        // line 6
                    { -1 , -1  , -1  , -1  , 0  , 0  , 0  , 0  ,
0  , 0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 7
                    { -1 , -1  , -1  , -1  , 0  , 0  , 0  , 0  ,
0  , 0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 8
```

```
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 9
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },
        // line 10
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },
// line 11
                    { -1 , -1  , -1  , -1  , 0  ,  0  ,  0  ,  0  ,
0  ,  0  ,  -1 ,  -1  ,  -1  ,  -1  },                        //
line 12
                    { -1 , -1  , -1  , -1  , -1  , 0  ,  0  ,  0  ,
0  , -1  ,  -1 ,  -1  ,  -1  ,  -1  },
   // line 13



            };



            // type
            public static int[] vehicles = new
int[FerryGUI.carCount] ;
            public static int FERRY_MAX_LINE = 13;
    public static int TYPE1_FERRY_STATE = 1;
    public static int TYPE2_FERRY_STATE = 2;
    public static int TYPE22_FERRY_STATE = 3;


    public static int TYPE1_CELL_INCREASE = 1;
    public static int TYPE2_CELL_INCREASE = 2;
```

```
        public static int TYPE22_CELL_INCREASE = 3;


        public static int RATE_OF_CAR_TYPE_1 = 75; //write the
    percentage of Type 1
        public static int RATE_OF_CAR_TYPE_2 = 25; //write the
    percentage of Type 2

                public static int[] vehicles1=new int[FerryGUI.carCount]
    ;
}
```