Spring 5-25-2015

Optimization of Scheduling and Dispatching Cars on Demand

Vu Tran
San Jose State University

Writing Project

Optimization of Scheduling and Dispatching Cars on Demand

Final Report

By

Vu Tran

CS 298

12/20/2014

Guided by

Professor

Dr. Chris Tseng

A Writing Project Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment of the Requirements for the Degree: Master of Science

The Designated Committee Approves the Project Titled

Optimization of Scheduling and Dispatching

Cars on Demand

By

Vu Tran

Approved for the Department of Computer Science

San Jose State University

December 2014

Dr. Chris Tseng	Department of Computer Science
Dr. Thanh Tran	Department of Computer Science
Mr. Peter Tran	Employee of NASA Ames Research Center

Acknowledgement

I would like to express my sincere gratitude to Dr. Chris Tseng for his guidance, time and the knowledge he shared with me. Without his direction and support, my report might not have shaped to completion. I would like to thank my committee members, Dr. Thanh Tran and Mr. Peter Tran, for their inputs and suggestions. At last, I would like to thank my family and friends for their encouraging support through the project.

ABSTRACT

Taxicab is the most common type of on-demand transportation service in the city because its dispatching system offers better services in terms of shorter wait time. However, the shorter wait time and travel time for multiple passengers and destinations are very considerable. There are recent companies implemented the real-time ridesharing model that expects to reduce the riding cost when passengers are willing to share their rides with the others. This model does not solve the shorter wait time and travel time when there are multiple passengers and destinations. This paper investigates how the ridesharing can be improved by using the genetic algorithm that gives the optimal solution in terms of passengers wait time and routes duration among passengers' start and end locations. The simulator uses the Google digital maps and direction services that allow the simulator to fetch the real-time data based on the current traffic conditions such as accident, peak hours, and weather. The simulation results that are sub-optimal routes are computed using the advanced genetic algorithm and real-time data availability.

TABLE OF CONTENTS

TABLE OF FIGURES

1. Project Description

1.1 Introduction and Problem Statement

Uber is one of the most popular on-demand ridesharing car services in the United States and expanding internationally to at least 45 countries. It dominates the entire transportation network marketplace because many passengers realized the benefits of the service, such as conveniently using their mobile phone application to request the service without the need to carry any cash or provide tips at the end of ride. The Uber services provide an estimate of the cost on-the-fly, allows rating the driver, and having payment transaction via emails. The ridesharing service also provides an Online Dispatch System (ODS) that uses communication technologies and geo-location services, such as Global Positioning System (GPS) and digital maps. When a passenger requests for a ride, the system collects passenger's location and finds matching drivers within that location. ODS simply computes the distance between passengers and drivers, and then it shows the passengers the closest proximity of available drivers on the map. This concept may not be ideal for many cars, passengers and destinations in terms of minimizing the passenger wait time and travel time.

In order to optimize process, the ODS first needs software algorithms and data structures to define various scheduling and dispatching policies, *such as First-In-First-Out (FIFO) known as a queue, Last-In-First-Out (LIFO) also known as a stack, processor sharing, priority, shortest service first, preemptive shortest service first, and Shortest Remaining Time First (SRTF)* that can be used when passengers request a ride and drivers provide the service. Secondly, the ODS needs to define model constraints and objectives. The two types of objectives to be considered are: (1) reducing passenger's wait time and (2) travel time. The "cost" function will be used to optimize the scheduling and dispatching algorithms, which can be very complex and interesting.

1.2 Project Goal

The objective of this project is to minimize passengers wait time and travel time using an advanced Genetic Algorithm with Constraints.

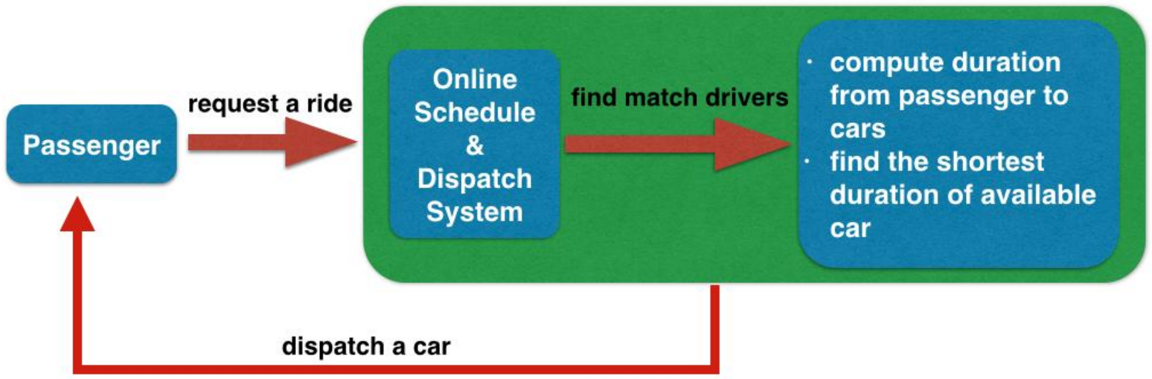

Passenger
request a ride
Online Schedule & Dispatch System
find match drivers
compute duration from passenger to cars
find the shortest duration of available car
dispatch a car

Figure 1: Online Schedule and Dispatch System

1.3 Real-Time Ridesharing Schedule And Dispatch Problem

The first scenario **(scenario 1a),** a passenger requests a ride, and multiple drivers are nearby. It's not problematic to locate the closest driver to the passenger. Because the ODS detects passenger and drivers' locations, it can measure the distance or travel time from the passenger to every nearby driver using Google direction services. If there are 5 nearby drivers, then OSD performs 5 computations. We can solve this scenario using Brute Force algorithm, and it is guaranteed to find the short duration of available driver.

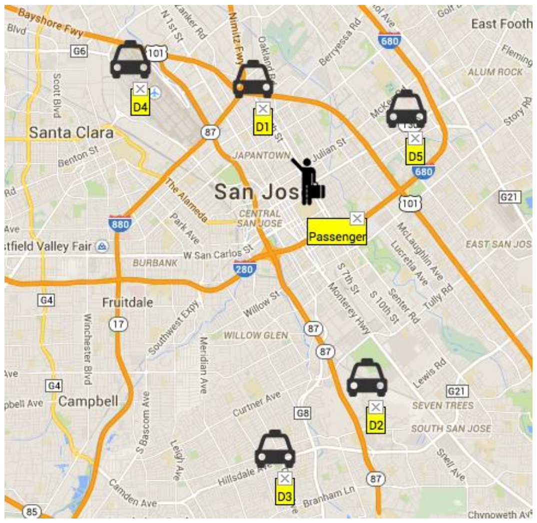


Figure 2: Multiple cars and one passenger

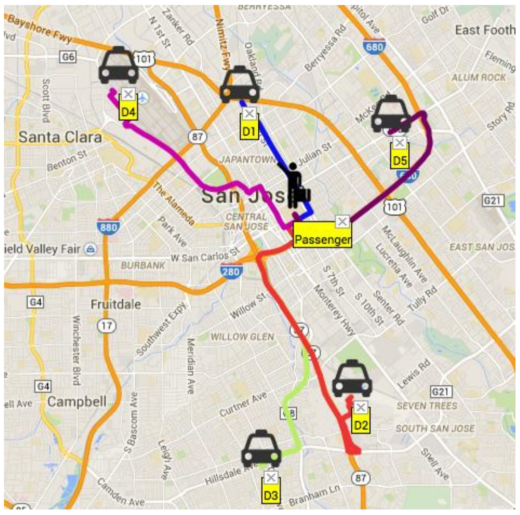


Figure 3: Multiple cars and 1 passenger with simulated routes

Another scenario **(scenario 1b),** a passenger requests a ride but multiple drivers are going to different directions. This scenario is more complicated than the previous one because ODS cannot quickly measure the travel time from the passenger to the driver when the driver is moving. For instance, the worst case is that five drivers are delivering 5 passengers to 5 different destinations when a new passenger requests a ride. The task is to predict which driver completes his/her route before heading to pickup the new passenger in the shortest duration. The ODS can solve this case in two steps. First, calculating the travel time from the new passenger to 5 driver's destinations that are 5 computations. Second, calculating the travel time from 5 driver's sources to 5 driver's destinations that are also 5 computations. The total cost for this case is 10 computations. The Brute Force algorithm can be used for this scenario as well.

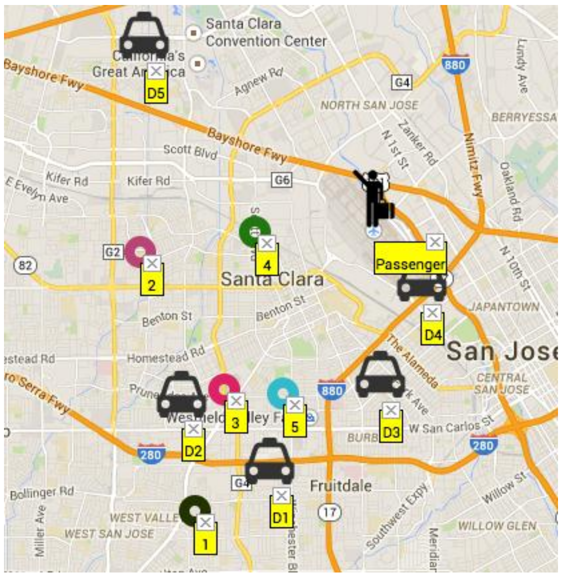


Figure 4: Multiple cars with multiple directions, and one passenger

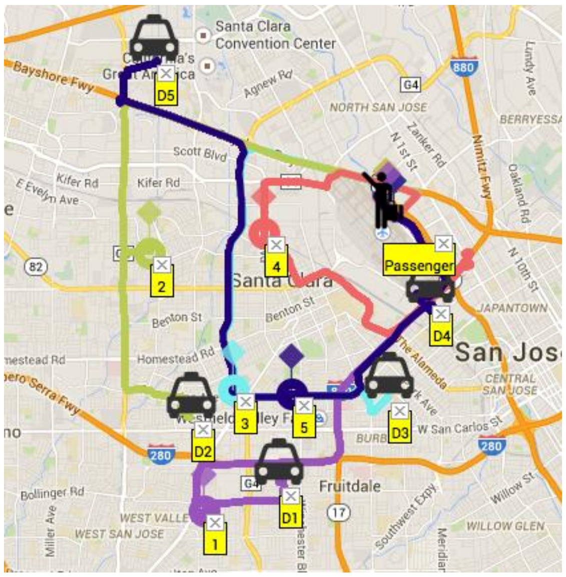


Figure 5: Multiple cars with multiple directions, and one passenger with simulated routes

The second scenario **(scenario 2),** when multiple passengers are willing to share one car to a destination, they can split the rate. Realistically, passengers share a ride to the airport when the wait time and travel time are important due to the flight's schedule. For instance, 4 passengers are going to the same airport. The ODS locates a driver that is near to 4 passengers. The task is to provide the driver the optimal routes in terms of the passenger wait time and travel time. The ODS has to decide which passenger is pickup first and so on. The first passenger requests a ride is the first one to be picked up may be is not an ideal solution (FCFS). If there are 4 passengers, then the distance or duration of 24 possible routes will be computed. The number of routes gets increased when the car has more capacities. If a van has 7 seats, it can hold 7 passengers. Hence, 5040 possible routes are measured. To solve the problem with 4 passengers, Brute Force algorithms can quickly find the best optimal route of 24 possible routes. To solve this problem with 7 passengers, besides Brute Force algorithm, Nearby Neighbor algorithm (NNA) can be applied. NNA may not give the best optimal route, but its computation is inexpensive as needing only 1 computation, but the data structure is required. This problem can be also solved by Standard Genetic Algorithm.

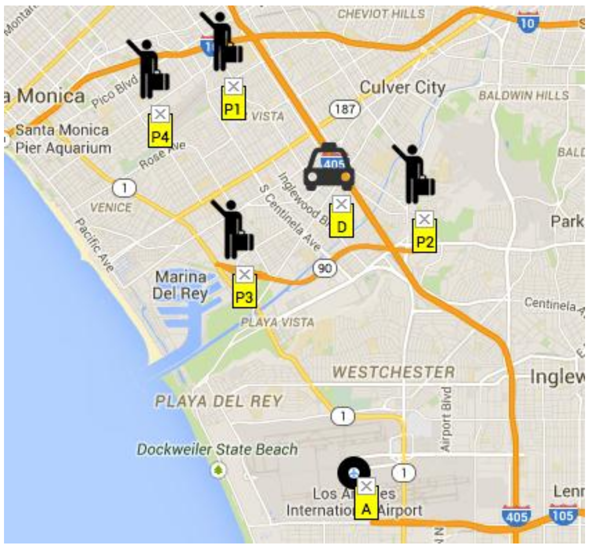


Figure 6: One car, multiple passengers and one destination

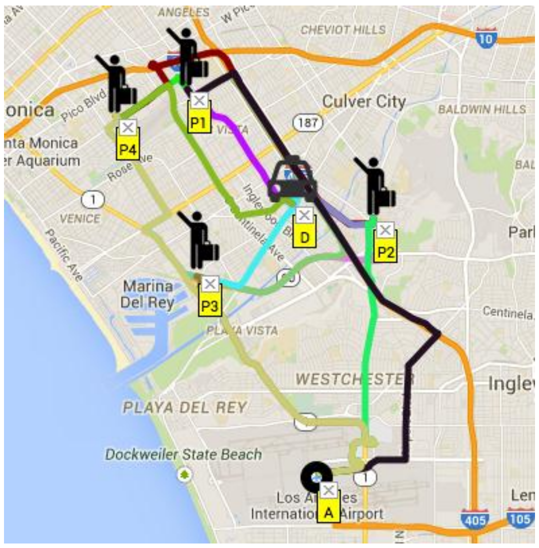


Figure 7: One car, multiple passengers and one destination with simulated routes

The third scenario **(scenario 3)**, multiple passengers are sharing one car to multiple destinations. This problem requires 3 constraints. First, driver must visit all passengers P1, P2, P3, P4 (Pi, 1 <= i <= n), but only once. Second, driver must visit all destinations D1, D2, D3, D4 (Di, 1 <= i <= n), but only once. Third, driver cannot visit Dj before visiting Pj, (1 <= j <=n. There are 40320 possible routes with 4 passengers and 4 destinations that are permuted, but 2520 possible routes satisfy the third constraint. This scenario is more complicated if it deals with multiple cars when there are multiple passengers and multiple destinations **(scenario 4).** For instance, 7 passengers with 7 destinations require 2 cars. A car can serve up to 4 passengers. In this case, car 1 can serve 4 passengers and car 2 can serve 3 passengers. Furthermore, there are many possible routes for 2 cars with 7 passengers and 7 destinations. Because this problem involves with a lot of computations and arrangements, it is complicated and complex when using Brute Force or Nearby Neighbor algorithms. However, Genetic algorithm is an ideal solution to deal with a problem of great complexity that will be explained in the next section.

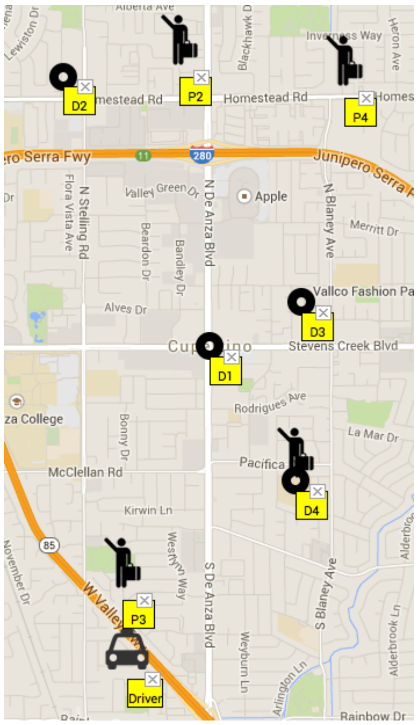


Figure 8: One car, multiple passengers and multiple destinations

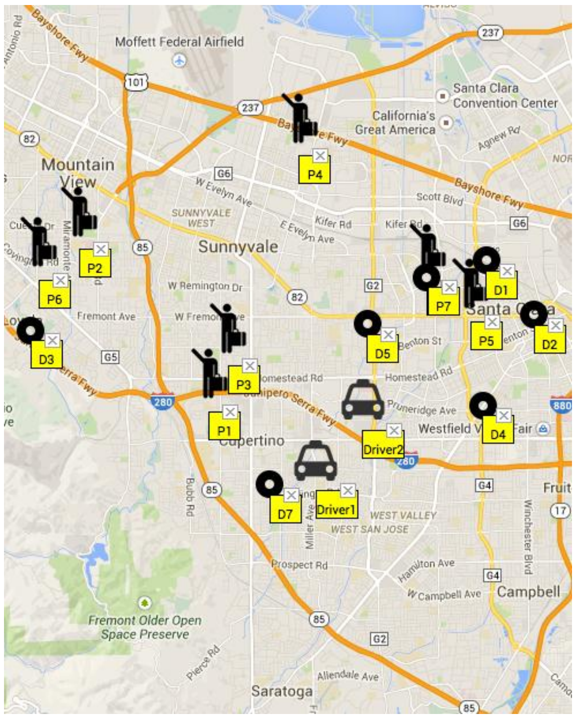


Figure 9: multiple cars, multiple passengers and multiple destinations

2. Project Design

2.1 Simulation Design

First, we need a lot of data point to simulate the drivers whether are driving or idling in certain locations and passengers are waiting for drivers to pickup. Passengers need to provide their locations whether they can enter or share their current location using GPS or enabling Web Share Location. Based o the passenger's location, the system will randomly generate the drivers' locations with given amount of drivers we want to simulate. System utilizes Google Maps API to produce the longitude and latitude as coordinates of passengers and drivers. The system then uses these coordinates to visually draw locations on Google Maps.

Second, system needs to capture the en-routes of drivers using Google Direction Service. This service will give us real-time data such as distance, duration and each step of drivers taking from point A to point B. The algorithms will use this data to analyze and formulate the model.

Third, we will use the Amazon Cloud Web Service EC2 to host the system and database. We will also use NoSQL database (MongoDB) to store the directions and analyzed data. Due to number of requests that are allowed by Google Services, the database becomes very useful in terms of improving the smoothness of simulation.

2.2 Modeling Ridesharing Schedule And Dispatch Algorithms Design

Brute Force search algorithm for the scenarios 1a, 1b, and 2 performs as follow:

1. Make a list of all possible routes
2. Calculate the duration of each route by adding up the duration of its edges
3. Choose the route with the smallest total duration

Nearest Neighbor search algorithm for the scenario 2 performs as follow:

1. Stand on an arbitrary vertex as current vertex.
2. Find out the shortest duration connecting current vertex and an unvisited vertex V.
3. Set current vertex to V.
4. Mark V as visited.
5. If all the vertices in domain are visited, then terminate.
6. Go to step 2.

$$Cost = \frac{1}{d\left(C,\ P_f\right) + \sum_{i=1}^{n} d(P_i,\ P_{i+1}) + d(P_l,\ D)}$$

Notation:

d: Duration from the current coordinate to the next coordinate

C: Car

D: Destination

P_f: The first Passenger in the vehicle's route, where 1 <= f <= n

P_l: The last Passenger in the vehicle's route, where 1 <= l <= n

P_i: Passenger, where 1 <= i <=n

n: Number of passengers

Comparison

Brute-Force Algorithm	**Nearest-Neighbor Algorithm**
• Optimal (guaranteed to find the shortest duration) • Inefficient (long time)	• Non-optimal (not always find the shortest duration) • Efficient (quick and easy)

Standard Genetic Algorithm for scenario 2 performs as follow:

1. **[Start]** Generate random population of n chromosomes (suitable solutions for the problem)
2. **[Fitness]** Evaluate the fitness f(x) of each chromosome x in the population
3. **[New population]** Create a new population by repeating following steps until the new population is complete
    a. **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
    b. **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
    c. **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).
    d. **[Accepting]** Place new offspring in a new population
4. **[Replace]** Use new generated population for a further run of algorithm
5. **[Test]** If the end condition is satisfied, stop, and return the best solution in current population
6. **[Loop]** Go to step 2

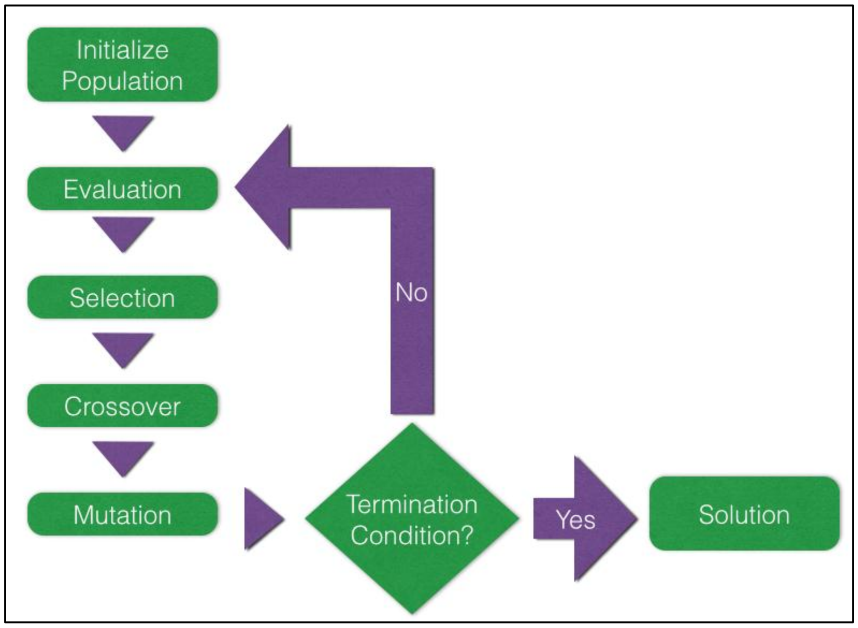


Figure 10: Standard Genetic Algorithm Process

Genetic algorithm with constraints for scenario 3 performs as follow:

Constraints:

1. Driver must visit all passengers P1, P2, P3, P4 (Pi, 1 <= i <= n), but only once
2. Driver must visit all destinations D1, D2, D3, D4 (Di, 1 <= i <= n), but only once
3. Driver cannot visit Dj before visiting Pj, (1 <= j <=n)

Algorithm

1. **[Start]**
    a. Generate random population of n passengers and random of n destinations.
    b. Combine population of n passengers with population of n destinations
    c. Compute population of 2n passengers and destinations named as routes
    d. Organize routes using multiple points crossover algorithm **(algorithm 1d)** that meet above constraints.
2. **[Fitness]** Evaluate the fitness f(x) function of each route x in the population
3. **[New population**] Create a new population by repeating following steps until the new population is complete
4. **[Selection]** Select two parent routes from a population according to their fitness (the better fitness, the bigger chance to be selected)
5. **[Crossover]**

   a. Apply single point crossover algorithm **(algorithm 5a**) over the parent routes to form a new offspring (children) route.
   b. Validate the new offspring route with constraints above. If the offspring route is not valid, then mutate its DNAs in step 6. Otherwise, go to step 7.

6. **[Mutation]** Change positions in a offspring route that meet constraints above
7. **[Accepting]** Place new offspring route in a new population
8. **[Replace]** Use new generated population for a further run of algorithm
9. **[Test]** If the end condition is satisfied, stop, and return the best solution in current population
10. **[Loop]** Go to step 2

Fitness Function (scenario 3: one car, many passengers, and many destinations)

$$fitness_{travelTime} = \frac{1}{d\left(C,\ P_f\right) + \sum_{i=1}^{2n} d(G_i,\ G_{i+1})}$$

Notation

d: Duration from current coordinate to next coordinate

C: Car

P_f: The first Passenger in the vehicle's route, where 1 <= f <= n

P_i: Passenger i, where 1 <= i <= n

D_i: Destination i, where 1 <= i <= m

$\{P_i, D_i\} \in G_i$, where 1 <= i <= 2n

n: Number of passengers, where n = m

m: Number of destinations, where m = n

Fitness Function (scenario 4b: many cars, many passengers, and many destinations)

a. Travel Time

$$fitness_{travelTime} = \frac{1}{\sum_{j=1}^{k} d\left(C_j,\ P_f^j\right) + \sum_{i=1}^{2n} d\left(G_i^j,\ G_{i+1}^j\right)}$$

Notation

d: Duration from current coordinate to next coordinate

C: Car, where 1 <= j <= k

P_f: The first Passenger in the vehicle's route, where 1 <= f <= n

P_i: Passenger i, where 1 <= i <= n

D_i: Destination i, where 1 <= i <= m

$\{P_i, D_i\} \in G_i$, where 1 <= i <= 2n

k: Number of cars

n: Number of passengers, where n = m

m: Number of destinations, where m = n

b. Average Wait Time

$$fitness_{avgWaitTime} = \frac{\mathrm{n}}{\sum_{i=1}^{n} WaitTime_i}$$

Notation

WaitTime: Wait Time of a passenger from Car to P_i and/or D_i, where 1 <= i <= n, but do not include durations within the passenger's destination and unvisited passengers and their destinations

n: Number of passengers

Multiple Points Crossover Algorithm in 1d

Gene 1	P2	P3	P1	P4					1 <= i <= n
Gene 2	D3	D2	D4	D1					1 <= i <= n
Parent	**P2**	**P3**	**D3**	**D2**	**P1**	**P4**	**D4**	**D1**	**1 <= i <= 2n**

Algorithm

1. Set the index I to 0
2. Process the first D in Gene 2
3. Find the corresponding P in Gene 1.
4. If the index of corresponding P less than I, then insert current D after the previous D in Gene 2 and mark current D as visited. Go to step 8
5. Mark current D as visited
6. Insert current D after the corresponding P in Gene 1
7. Set I to index of P
8. If all Ds in Gene 2 are visited, then terminate.
9. Process the next D in Gene 2, go to step 3

Single point crossover algorithm in 5a

Parent 1	**P2**	**P3**	**D3**	**D2**	P1	P4	D4	D1
Parent 2	P2	D2	P3	P1	P4	D3	D1	D4
Parent 2	*X*	*X*	*X*	*P1*	*P4*	*X*	*D1*	*D4*
Offspring	**P2**	**P3**	**D3**	**D2**	**P1**	**P4**	**D1**	**D4**

Algorithm

1. Copy the first half gene of Parent 1 and form a new offspring
2. Mark all DNAs in Parent 2 that are matching with DNAs in the offspring
3. Copy all unmarked DNAs in Parent 2 and keep their orders.
4. Add DNAs to the offspring to form a complete gene.

Mutation Operation

Gene	**P2**	**D3**	**D2**	**P3**	**P1**	**P4**	**D1**	**D4**
New Gene	**P2**	**P3**	**D2**	**D3**	**P1**	**P4**	**D1**	**D4**

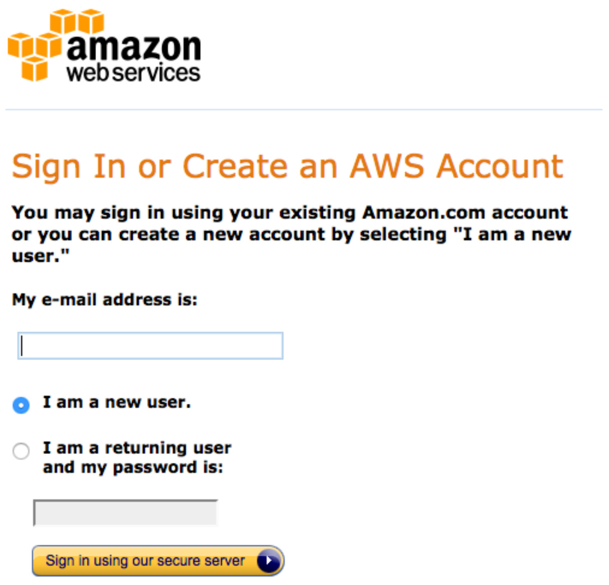
Algorithm

1. Find D in the gene and record its index
2. Find corresponding P in the gene
3. [Test] If the index of corresponding P is greater than index of D, then swap their position.
4. [Test] If visit all D in the gene, then terminate
5. [Loop] Go to step 1

3. Project Implementation

3.1 Server Setup with Amazon Cloud Infrastructure EC2

To setup an Amazon Elastic Compute Cloud (EC2), first we need to create Amazon Web Service (AWS) account by following step by step as shown in figure below.


amazon web services

Sign In or Create an AWS Account

You may sign in using your existing Amazon.com account or you can create a new account by selecting "I am a new user."

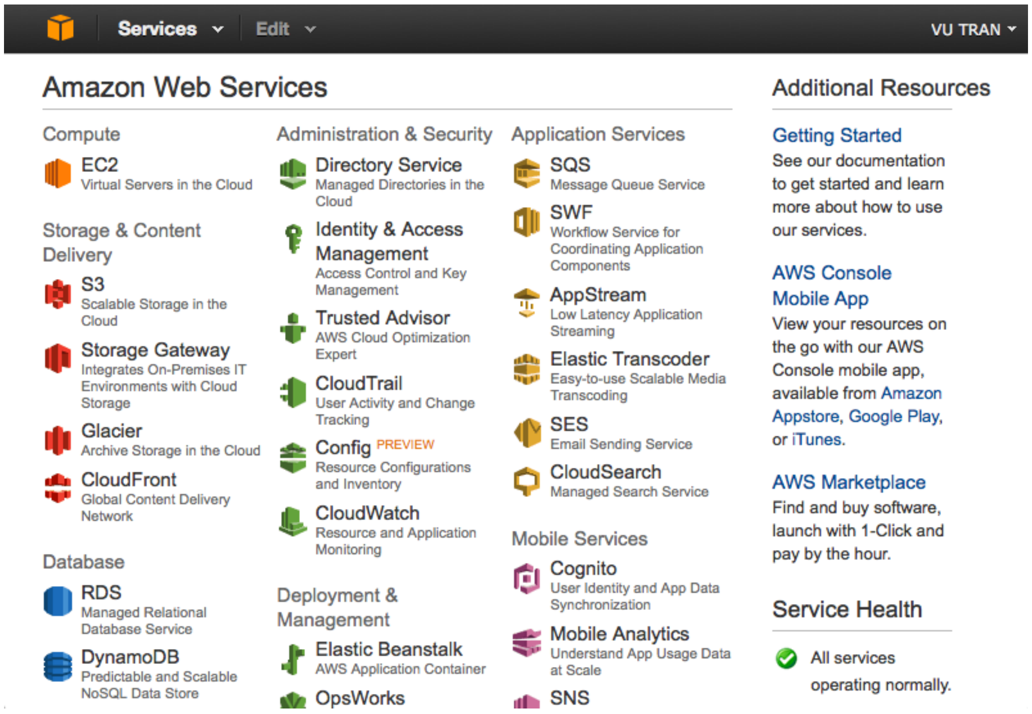
My e-mail address is:

[✓ I am a new user.]

I am a returning user and my password is:

Figure 11: Creating AWS account

After signing into AWS console, we see a lot of services. We will only need EC2 to host our application.



Amazon Web Services

Compute

EC2
Virtual Servers in the Cloud

Storage & Content Delivery

S3
Scalable Storage in the Cloud

Storage Gateway
Integrates On-Premises IT Environments with Cloud Storage

Glacier
Archive Storage in the Cloud

CloudFront
Global Content Delivery Network

Database

RDS
Managed Relational Database Service

DynamoDB
Predictable and Scalable NoSQL Data Store

Administration & Security

Directory Service
Managed Directories in the Cloud

Identity & Access Management
Access Control and Key Management

Trusted Advisor
AWS Cloud Optimization Expert

CloudTrail
User Activity and Change Tracking

Config PREVIEW
Resource Configurations and Inventory

CloudWatch
Resource and Application Monitoring

Deployment & Management

Elastic Beanstalk
AWS Application Container

OpsWorks

Application Services

SQS
Message Queue Service

SWF
Workflow Service for Coordinating Application Components

AppStream
Low Latency Application Streaming

Elastic Transcoder
Easy-to-use Scalable Media Transcoding

SES
Email Sending Service

CloudSearch
Managed Search Service

Mobile Services

Cognito
User Identity and App Data Synchronization

Mobile Analytics
Understand App Usage Data at Scale

SNS

Additional Resources

Getting Started
See our documentation to get started and learn more about how to use our services.

AWS Console Mobile App
View your resources on the go with our AWS Console mobile app, available from Amazon Appstore, Google Play, or iTunes.

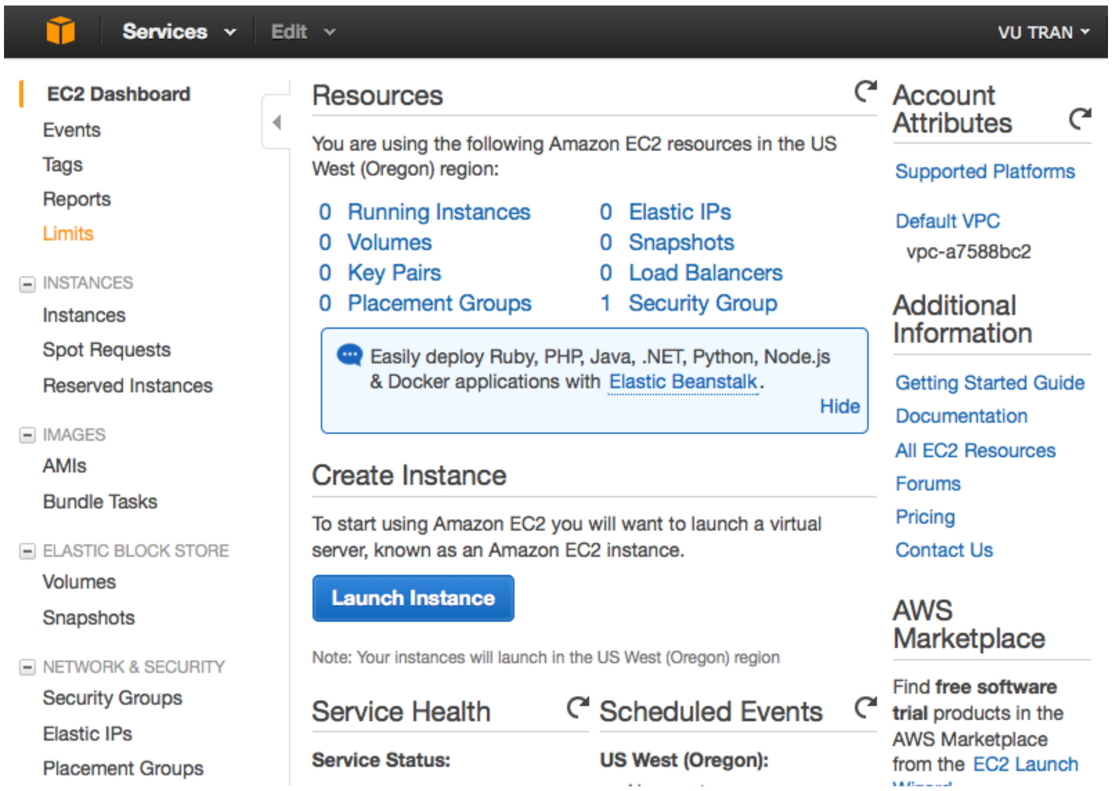
AWS Marketplace
Find and buy software, launch with 1-Click and pay by the hour.

Service Health

All services operating normally.

Figure 12: Amazon Web Services

Next, choose EC2 service to begin launching an instance



Services Edit VU TRAN

EC2 Dashboard
Events
Tags
Reports
Limits

INSTANCES
Instances
Spot Requests
Reserved Instances

IMAGES
AMIs
Bundle Tasks

ELASTIC BLOCK STORE
Volumes
Snapshots

NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups

Resources

You are using the following Amazon EC2 resources in the US West (Oregon) region:

0 Running Instances	0 Elastic IPs
0 Volumes	0 Snapshots
0 Key Pairs	0 Load Balancers
0 Placement Groups	1 Security Group

Easily deploy Ruby, PHP, Java, .NET, Python, Node.js & Docker applications with Elastic Beanstalk.
Hide

Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Launch Instance

Note: Your instances will launch in the US West (Oregon) region

Service Health
Service Status:

Scheduled Events
US West (Oregon):

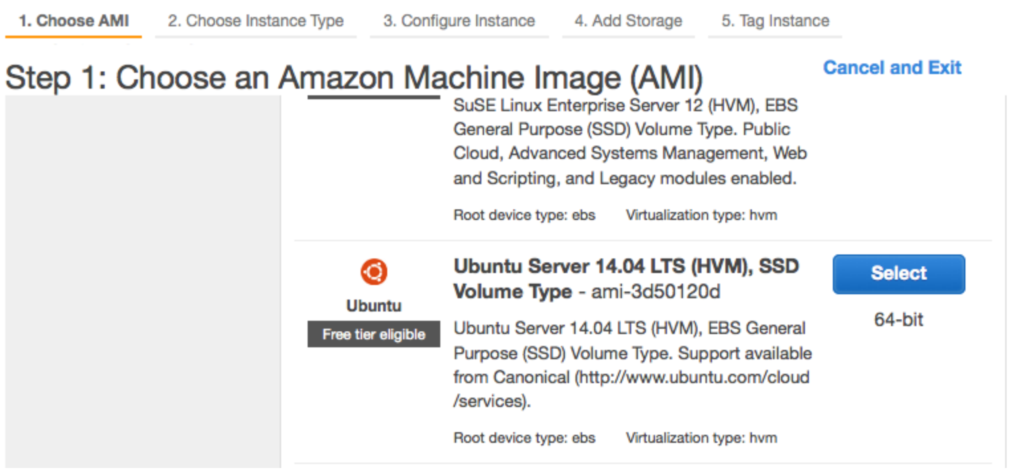
Account Attributes
Supported Platforms
Default VPC
vpc-a7588bc2

Additional Information
Getting Started Guide
Documentation
All EC2 Resources
Forums
Pricing
Contact Us

AWS Marketplace
Find **free software trial** products in the AWS Marketplace from the EC2 Launch

Figure 13: Launching EC2 Instance

Step 1: Choose Ubuntu Server



1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Tag Instance

Step 1: Choose an Amazon Machine Image (AMI)
Cancel and Exit

SuSE Linux Enterprise Server 12 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled.

Root device type: ebs Virtualization type: hvm

Ubuntu
Free tier eligible

Ubuntu Server 14.04 LTS (HVM), SSD Volume Type - ami-3d50120d

Select
64-bit

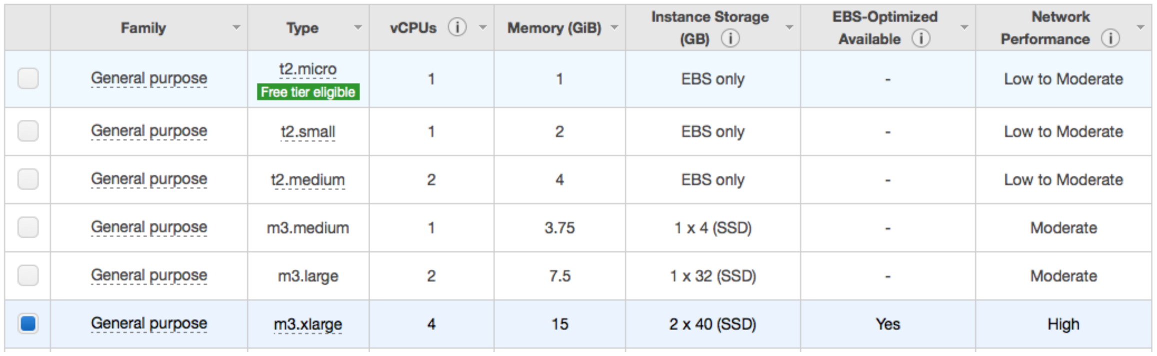
Ubuntu Server 14.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).

Root device type: ebs Virtualization type: hvm

Figure 14: Amazon Machine Image, Ubuntu Server

Step 2: Choose an Instance Type

Because our application requires a lot of computations, we need a high performance instance such as: m3.xlarge (13 ECUs, 4 vCPUs, 2.5 GHz, Intel Xeon E5-2670v2, 15 GiB memory, 2 x 40 GiB Storage Capacity)

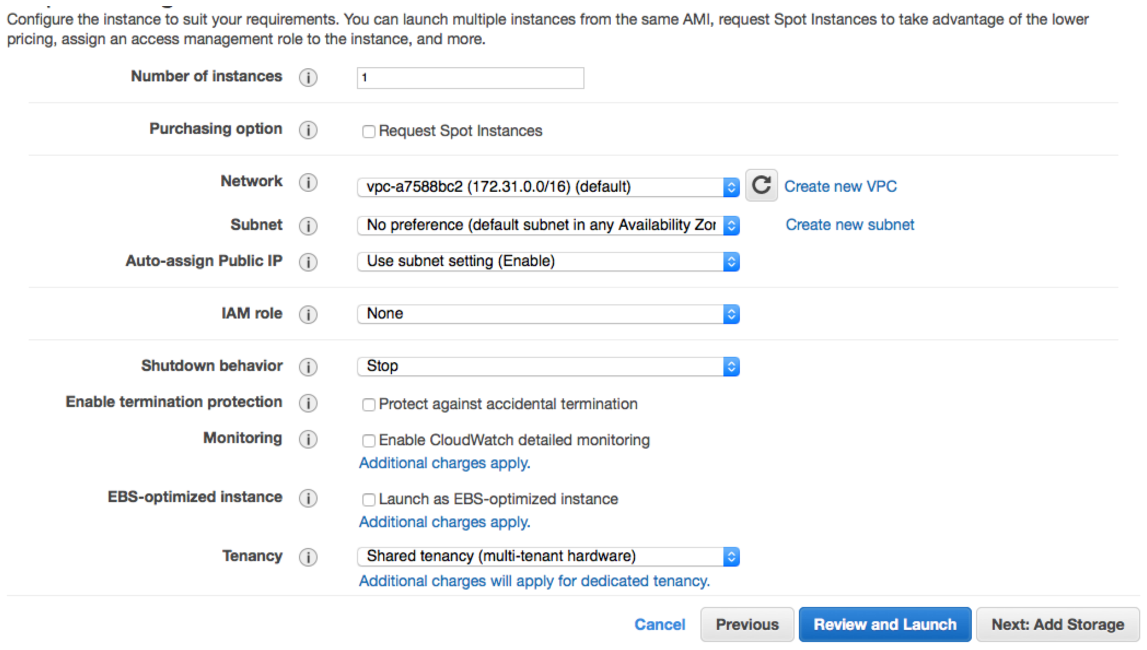


	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
☐	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate
☐	General purpose	t2.small	1	2	EBS only	-	Low to Moderate
☐	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate
☐	General purpose	m3.medium	1	3.75	1 x 4 (SSD)	-	Moderate
☐	General purpose	m3.large	2	7.5	1 x 32 (SSD)	-	Moderate
■	General purpose	m3.xlarge	4	15	2 x 40 (SSD)	Yes	High

Figure 15: EC2 Instance Type

Step 3: Configure Instance Details

We use default configurations and finally launch the instance



Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot Instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances	1
Purchasing option	☐ Request Spot Instances
Network	vpc-a7588bc2 (172.31.0.0/16) (default) — Create new VPC
Subnet	No preference (default subnet in any Availability Zor — Create new subnet
Auto-assign Public IP	Use subnet setting (Enable)
IAM role	None
Shutdown behavior	Stop
Enable termination protection	☐ Protect against accidental termination
Monitoring	☐ Enable CloudWatch detailed monitoring Additional charges apply.
EBS-optimized instance	☐ Launch as EBS-optimized instance Additional charges apply.
Tenancy	Shared tenancy (multi-tenant hardware) Additional charges will apply for dedicated tenancy.

Figure 16: EC2 Configuration

3.2 Simulation Implementation

a. Randomizing Passengers and Drivers Coordinates

In order to calculate the random coordinates, we will use the geometry and math functions such as *sin*, *cos*, *asin*, *atan2* in the snippet code below.

```
        var lat = deg(asin(sinstartlat * cos(dist) + cosstartlat * sin(dist) * cos(brg[0])));

        var lon = deg(normalizeLongitude(startlon * 1 + atan2(sin(brg[0]) * sin(dist) *
cosstartlat, cos(dist) - sinstartlat * sin(lat))));
```

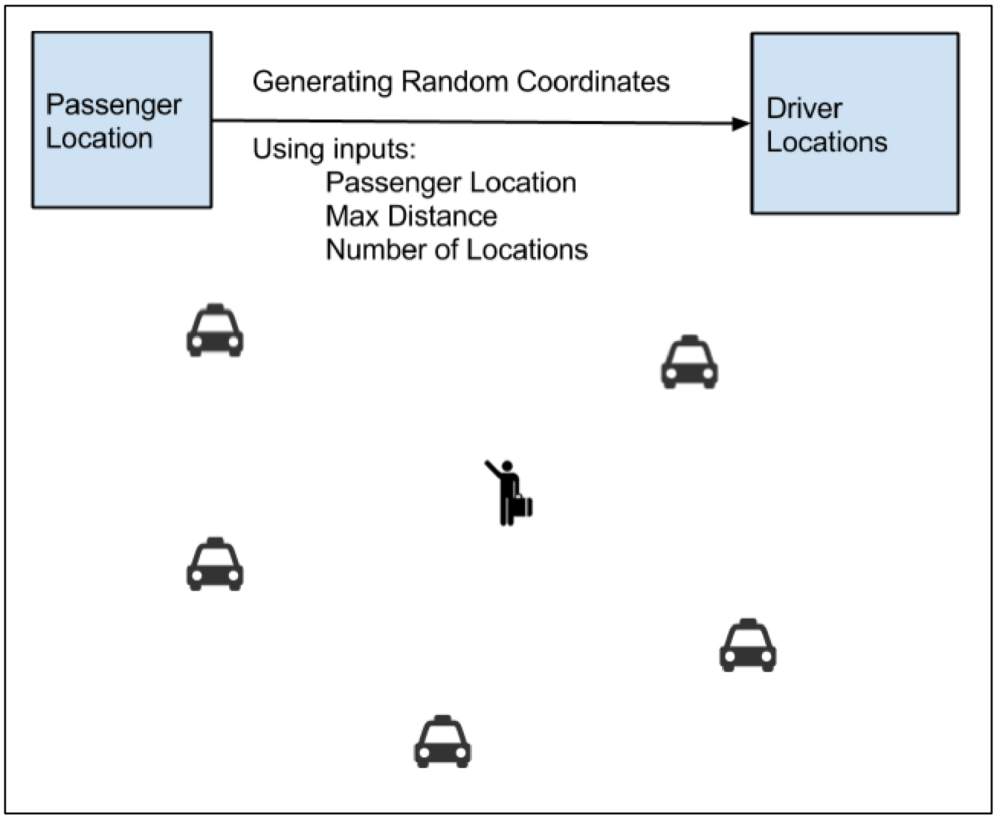


Figure 17: Driver's Locations based on the Passenger Location

b. Randomizing Passenger's Destination and Driver's Destinations

For each driver's start coordinate, generating the driver's end coordinate with the direction that driver is heading to.

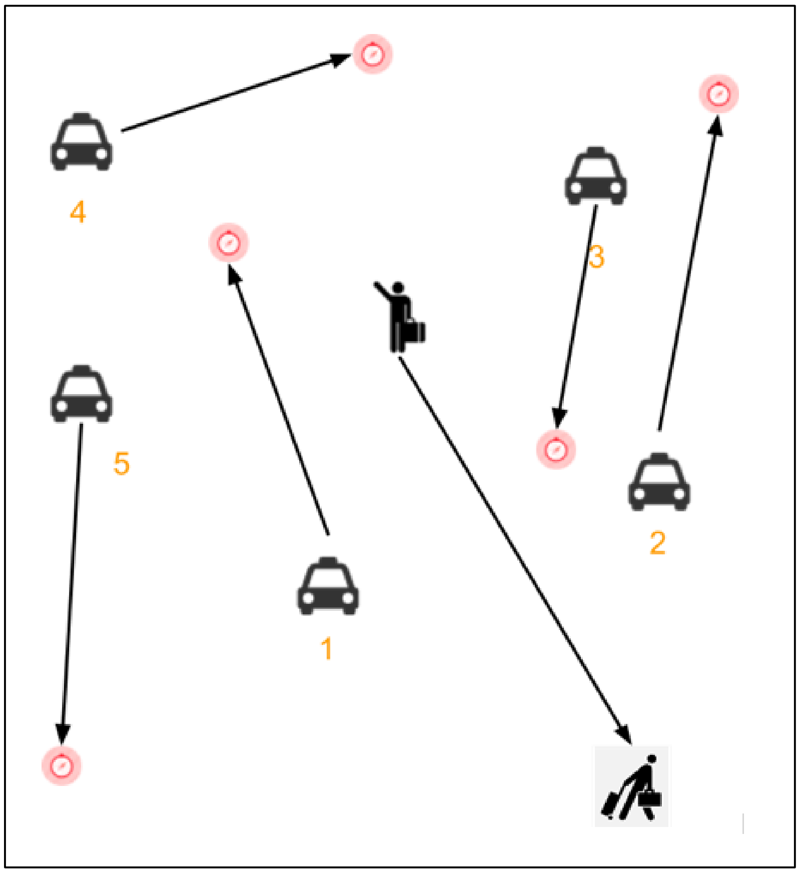


Figure 18: Many Cars and One Passenger

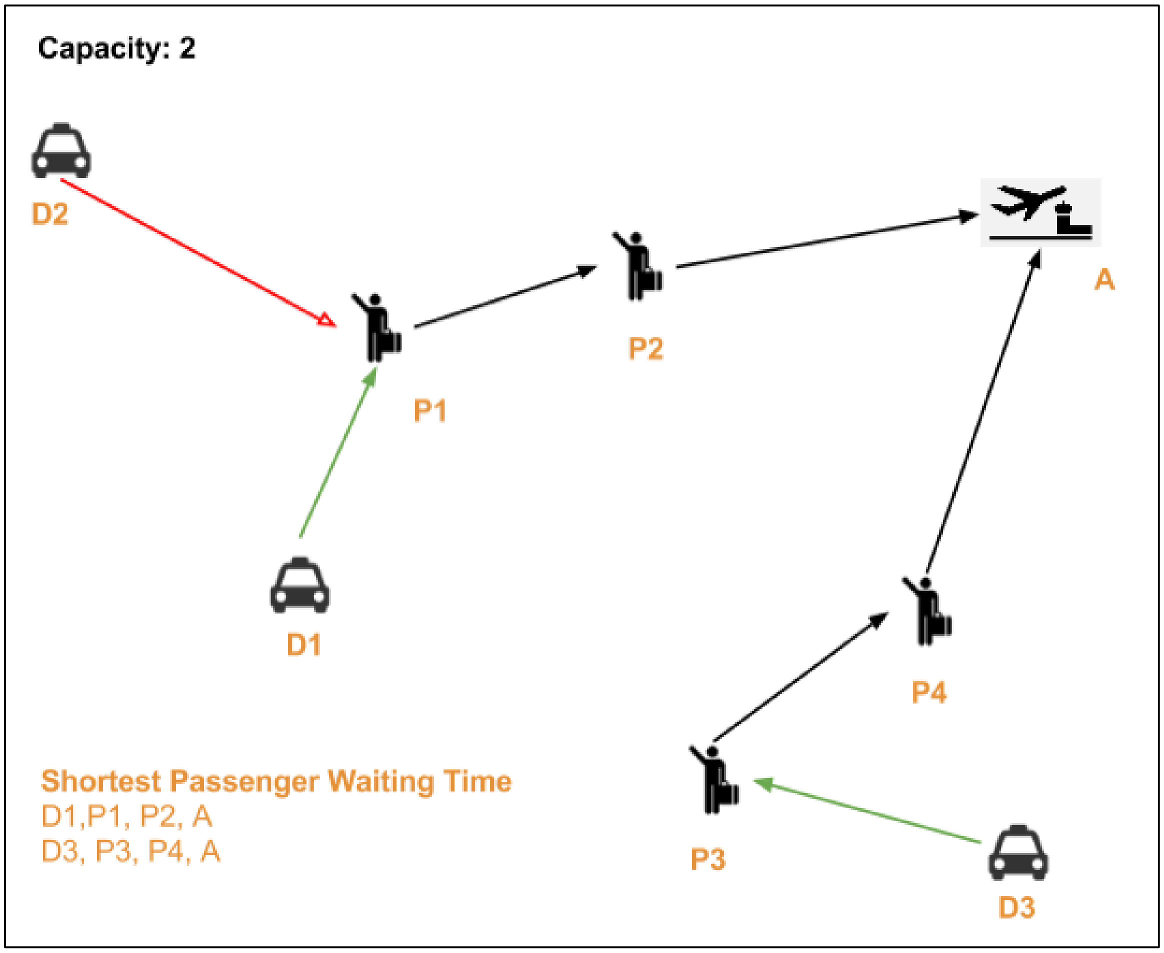
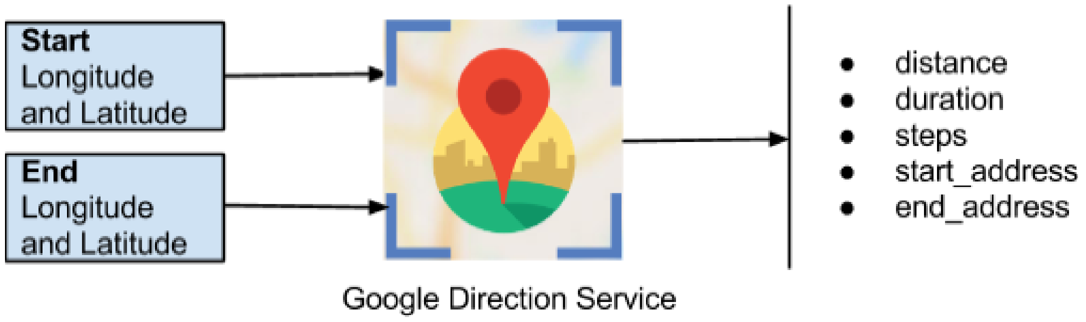


Figure 19: Many Cars, Many Passengers and One Destination

c. Integrating Google Direction Service



3.3 Genetic Algorithm Implementation

We attempt to solve a problem with 2 cars, 4 passengers, and 4 destinations

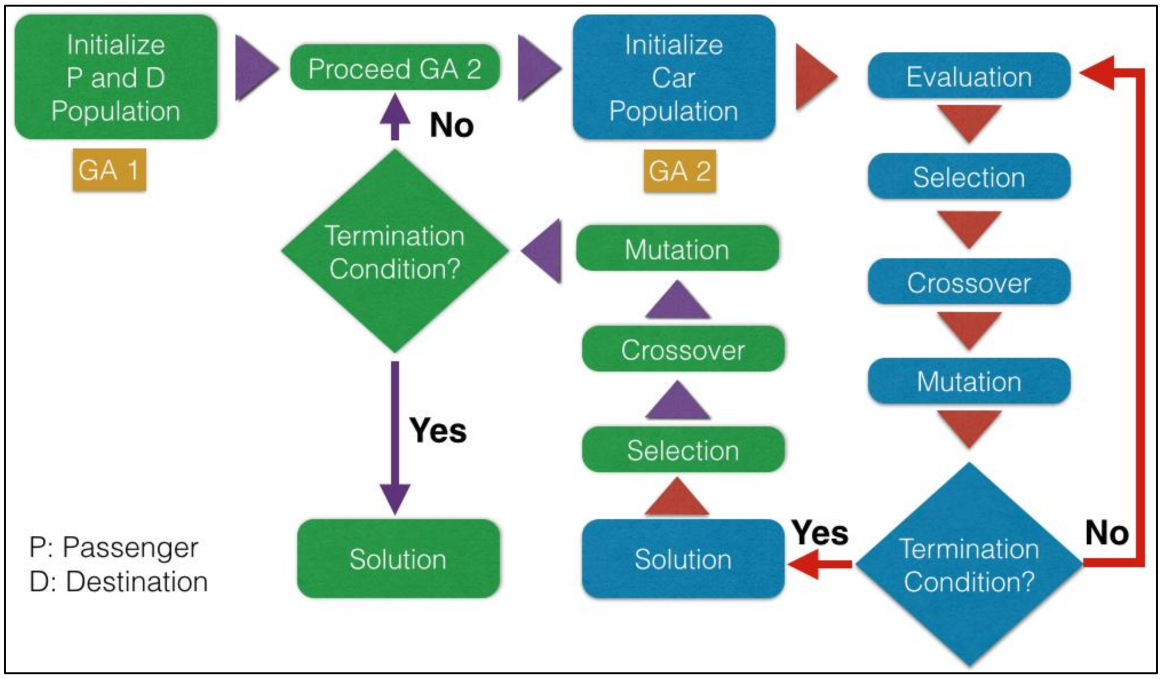
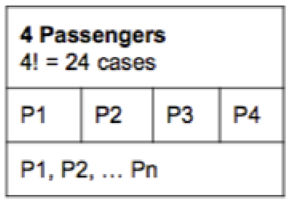


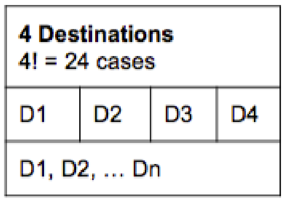
Figure 20: Advanced Genetic Algorithm Process with Constraints

GA 1 Process

a. Initialize Passenger and Destination Population



4 Passengers 4! = 24 cases			
P1	P2	P3	P4
P1, P2, ... Pn			



4 Destinations 4! = 24 cases			
D1	D2	D3	D4
D1, D2, ... Dn			

First, we randomly generate 5 passenger's genes (P genes), and 5 destination's genes (D genes). Second, we combine P genes with D genes to make 25 genes. Third, we re-arrange among P genes and D genes. So, they satisfy 3 constraints that we defined in the design section.

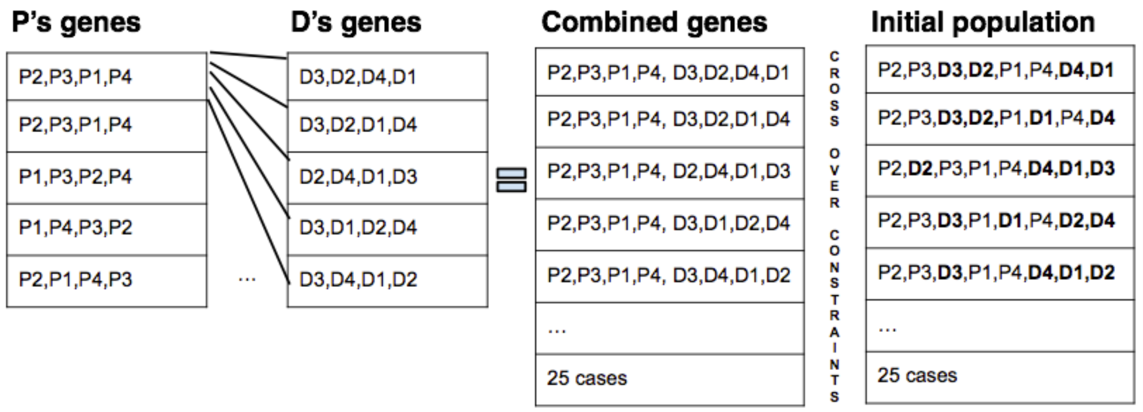


Figure 21: Genes Preparation

b. Proceed GA 2 Processes

For each combined passengers and destinations gene (P1,P3,D1,D3,P2,D2,P4,D4), proceed the second GA process.

1. **Initialize Car Population**

Initial Population	**P1**	**P2**	**P3**	**P4**	**CAR1**	**CAR2**
1,1,2,1	1	1	2	1	P1,P2,P4	P3
2,1,2,1	2	1	2	1	P2,P4	P1,P3
2,2,1,1	2	2	1	1	P3,P4	P1,P2

2. **Evaluation**

$$fitness = \sum_{j=1}^{m} \frac{1}{d\left(C_j,\ P_f\right) + \sum_{i=1}^{2n} d(E_i,\ E_{i+1})}$$

Example

P1,P3,D1,D3,P2,D2,P4,D4	
CAR1	**CAR2**
P1,P2,P4	P3
C1,P1,D1,P2,D2,P4,D4	C2,P3,D3

$$\mathrm{CAR1} = \mathrm{d(C1,P1)} + \mathrm{d(P1,D1)} + \mathrm{d(D1,P2)} + \mathrm{d(P2,D2)} + \mathrm{d(D2,P4)} + \mathrm{d(P4,D4)}$$
$$\mathrm{CAR2} = \mathrm{d(C2,P3)} + \mathrm{d(P3,D3)}$$
$$\mathrm{fitness} = \frac{1}{\mathrm{CAR1} + \mathrm{CAR2}}$$

3. **Selection**

   Select two genes from the population in which the shorter durations have more chances to get selected. For instance:

   PARENT 1: **1,2,**1,1
   PARENT 2: 2,1,**1,2**

4. **Crossover**

   Apply One Point Crossover to PARENT 1

   Child: 1,2,1,2

5. **Mutation**

   Because this GA 2 has a small population, do not need to apply mutation.

6. **Termination:** If No, go to step 2.

c. **Selection**

Select two genes from the population in which the shorter durations have more chances to get selected. For instance:

Parent 1: P1,P3,D1,D3,P2,D2,P4,D4
Parent 2: P1,P3,D3,D4,P4,D1,P2,D2

d. **Crossover**

Apply One Point Crossover to PARENT 1

Child: P1,P3,D1,D3,D4,P4,P2,D2

e. **Mutation**

[The Child is not valid because D2 and D3 are assigned before P3 and P2. The car should not visit a destination before visiting a passenger. To fix this, we need to change the position of the DNA.]*
Child: P1,P3,D1,D3,D4,P4,P2,D2
Mutated CHILD: P1,P3,P2,D3,D4,P4, D1,D2

f. **Termination:** If No, go to step 2b.

g. **Solution**

4. Results

Simulation Result for Scenario 1a:

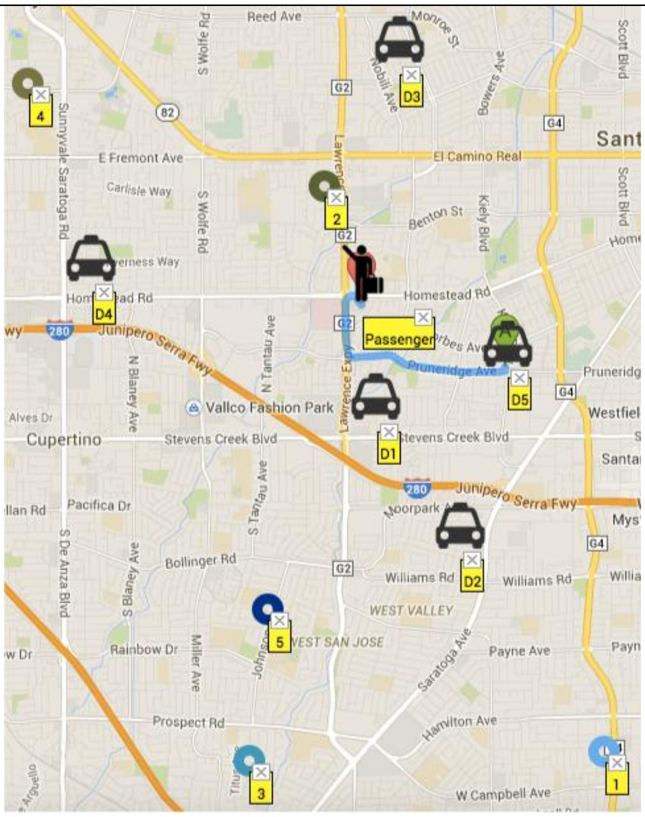
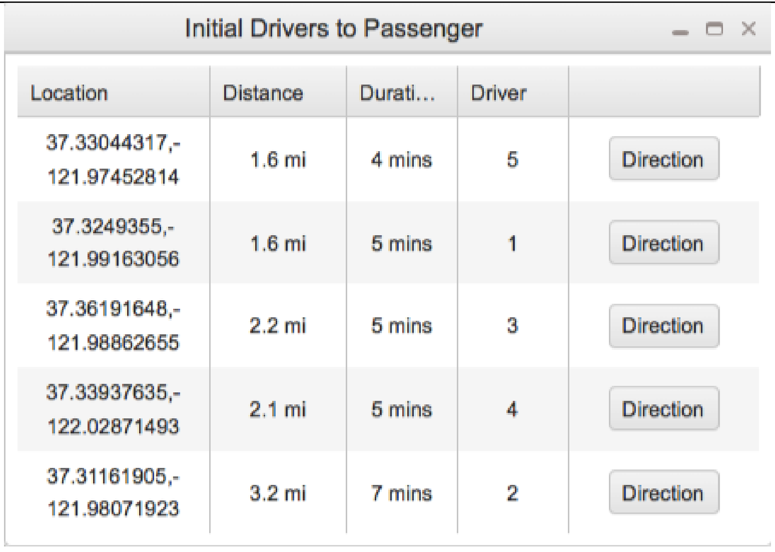


Figure 22: 4 Cars and 1 Passenger



Initial Drivers to Passenger

Location	Distance	Durati...	Driver	
37.33044317,-121.97452814	1.6 mi	4 mins	5	Direction
37.3249355,-121.99163056	1.6 mi	5 mins	1	Direction
37.36191648,-121.98862655	2.2 mi	5 mins	3	Direction
37.33937635,-122.02871493	2.1 mi	5 mins	4	Direction
37.31161905,-121.98071923	3.2 mi	7 mins	2	Direction

Driver 5 has the shortest duration

Simulation Result for Scenario 1b:

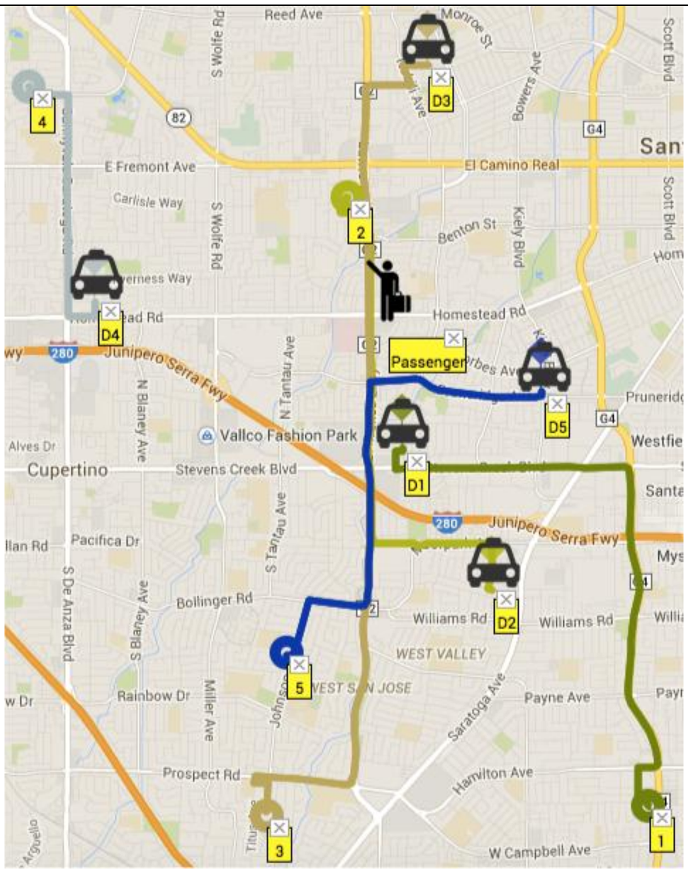
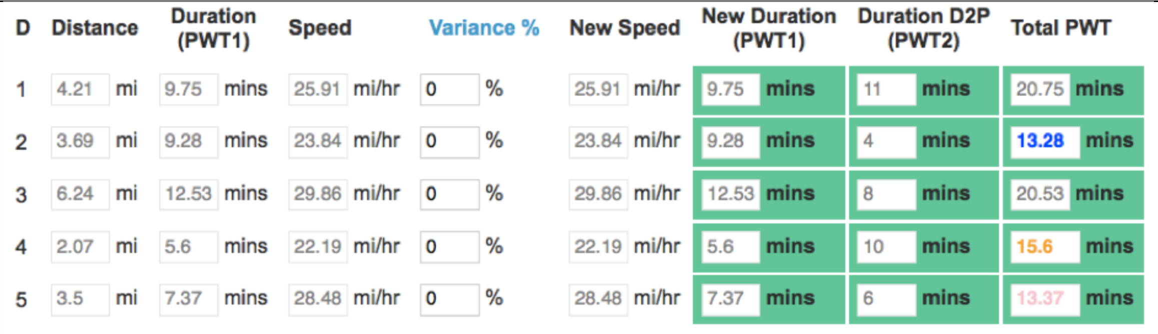


Figure 23: 4 Cars, 4 Destinations and One Passenger



D	Distance		Duration (PWT1)		Speed		Variance %		New Speed		New Duration (PWT1)		Duration D2P (PWT2)		Total PWT	
1	4.21	mi	9.75	mins	25.91	mi/hr	0	%	25.91	mi/hr	9.75	mins	11	mins	20.75	mins
2	3.69	mi	9.28	mins	23.84	mi/hr	0	%	23.84	mi/hr	9.28	mins	4	mins	13.28	mins
3	6.24	mi	12.53	mins	29.86	mi/hr	0	%	29.86	mi/hr	12.53	mins	8	mins	20.53	mins
4	2.07	mi	5.6	mins	22.19	mi/hr	0	%	22.19	mi/hr	5.6	mins	10	mins	15.6	mins
5	3.5	mi	7.37	mins	28.48	mi/hr	0	%	28.48	mi/hr	7.37	mins	6	mins	13.37	mins

Driver 2 has the shortest duration

Simulation Result for Scenario 2:

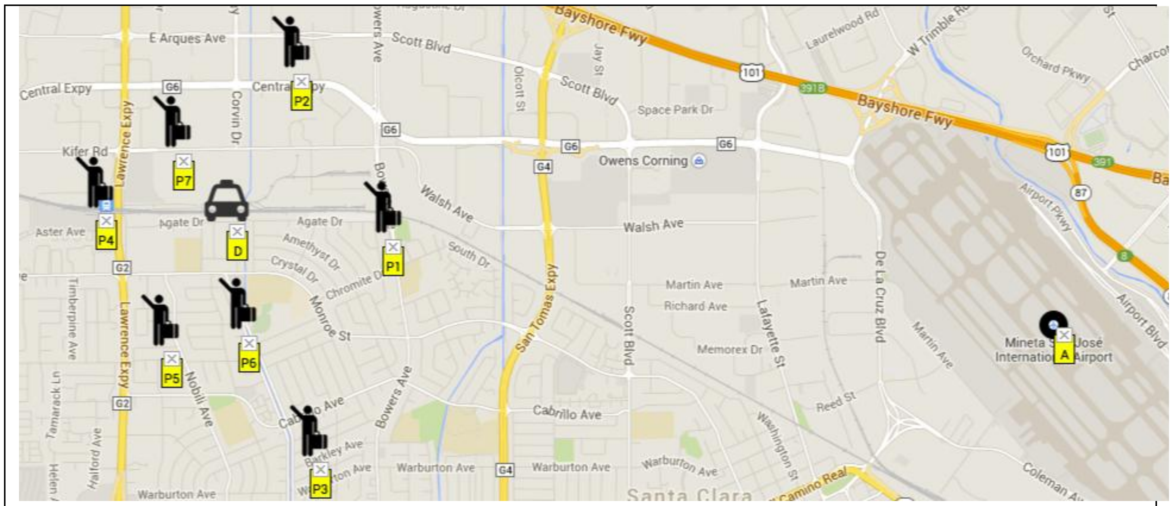
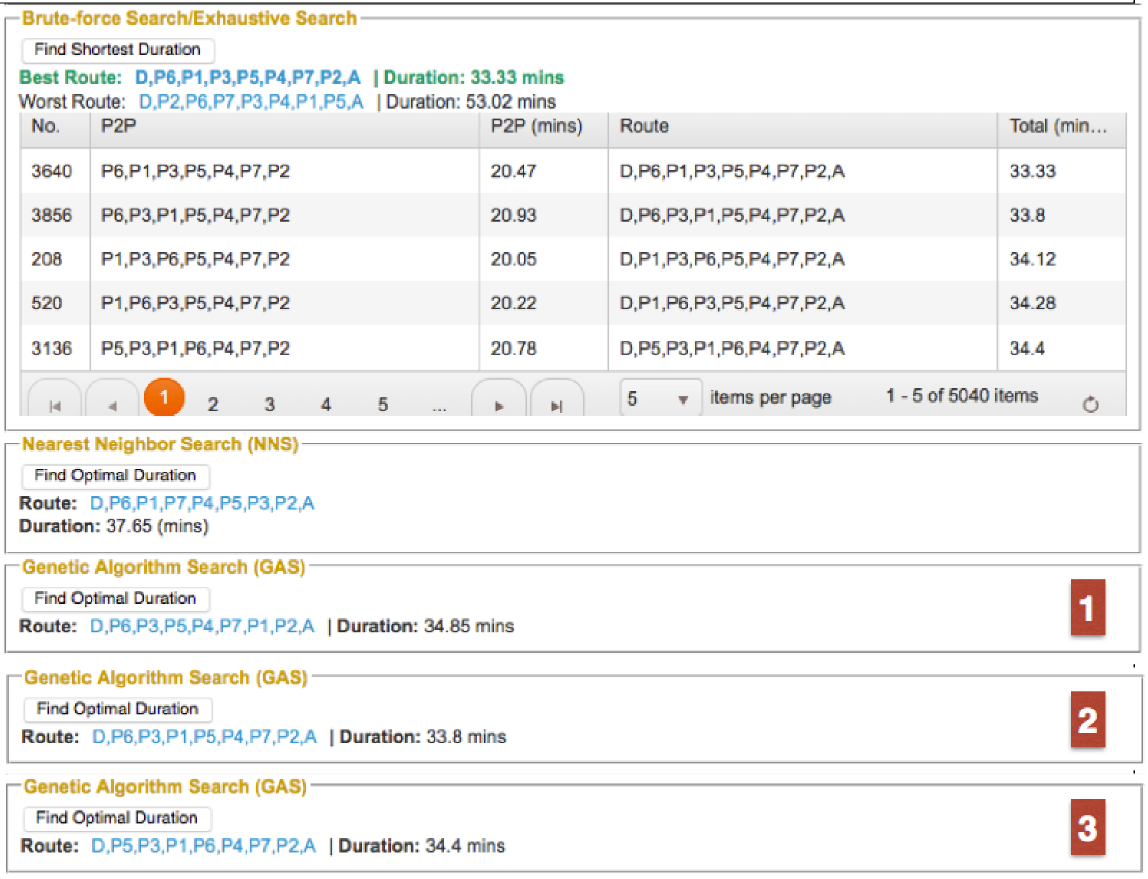


Figure 24: 1 Car, 5 Passengers and One Destination



Brute-force Search/Exhaustive Search

Find Shortest Duration

Best Route: D,P6,P1,P3,P5,P4,P7,P2,A | Duration: 33.33 mins

Worst Route: D,P2,P6,P7,P3,P4,P1,P5,A | Duration: 53.02 mins

No.	P2P	P2P (mins)	Route	Total (min…
3640	P6,P1,P3,P5,P4,P7,P2	20.47	D,P6,P1,P3,P5,P4,P7,P2,A	33.33
3856	P6,P3,P1,P5,P4,P7,P2	20.93	D,P6,P3,P1,P5,P4,P7,P2,A	33.8
208	P1,P3,P6,P5,P4,P7,P2	20.05	D,P1,P3,P6,P5,P4,P7,P2,A	34.12
520	P1,P6,P3,P5,P4,P7,P2	20.22	D,P1,P6,P3,P5,P4,P7,P2,A	34.28
3136	P5,P3,P1,P6,P4,P7,P2	20.78	D,P5,P3,P1,P6,P4,P7,P2,A	34.4

1 2 3 4 5 ... 5 items per page 1 - 5 of 5040 items

Nearest Neighbor Search (NNS)

Find Optimal Duration

Route: D,P6,P1,P7,P4,P5,P3,P2,A

Duration: 37.65 (mins)

Genetic Algorithm Search (GAS)

Find Optimal Duration

Route: D,P6,P3,P5,P4,P7,P1,P2,A | **Duration:** 34.85 mins

1

Genetic Algorithm Search (GAS)

Find Optimal Duration

Route: D,P6,P3,P1,P5,P4,P7,P2,A | **Duration:** 33.8 mins

2

Genetic Algorithm Search (GAS)

Find Optimal Duration

Route: D,P5,P3,P1,P6,P4,P7,P2,A | **Duration:** 34.4 mins

3

Simulation Result for Scenario 3:

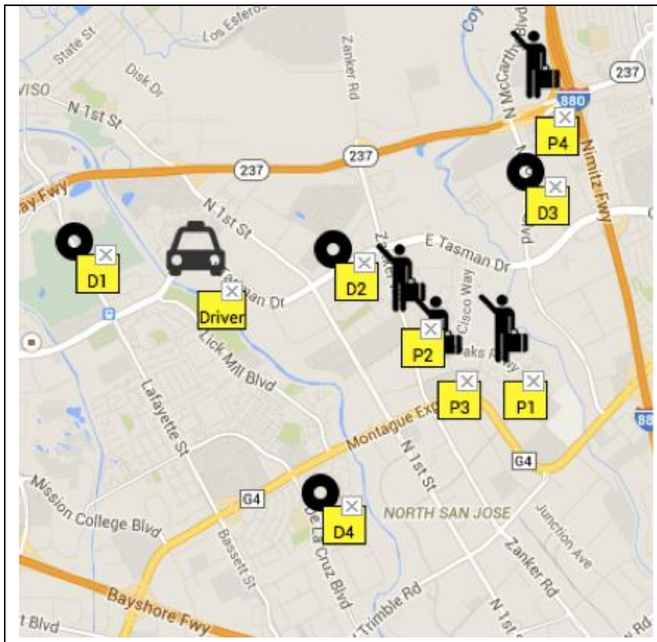


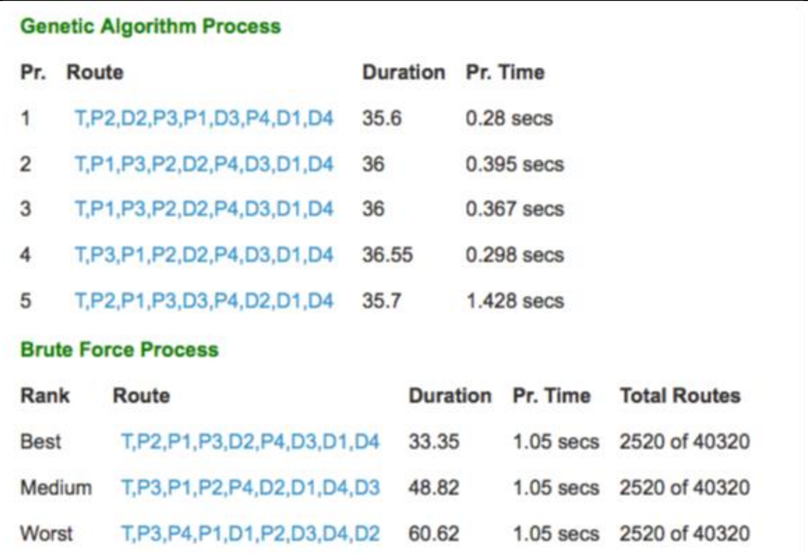
Figure 25: 1 Car, 4 Passengers and 4 Destinations

Schedule:

T,P2,D2,P3,P1,D3,P4,D1,D4

T:	**Driver/Car**
P:	**Passenger**
D:	**Destination**

2,520 routes satisfy 3 constraints for 1 car, 4 passengers, 4 destinations problem.

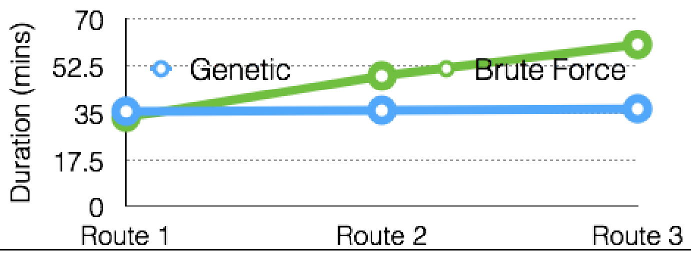


Genetic Algorithm Process

Pr.	Route	Duration	Pr. Time
1	T,P2,D2,P3,P1,D3,P4,D1,D4	35.6	0.28 secs
2	T,P1,P3,P2,D2,P4,D3,D1,D4	36	0.395 secs
3	T,P1,P3,P2,D2,P4,D3,D1,D4	36	0.367 secs
4	T,P3,P1,P2,D2,P4,D3,D1,D4	36.55	0.298 secs
5	T,P2,P1,P3,D3,P4,D2,D1,D4	35.7	1.428 secs

Brute Force Process

Rank	Route	Duration	Pr. Time	Total Routes
Best	T,P2,P1,P3,D2,P4,D3,D1,D4	33.35	1.05 secs	2520 of 40320
Medium	T,P3,P1,P2,P4,D2,D1,D4,D3	48.82	1.05 secs	2520 of 40320
Worst	T,P3,P4,P1,D1,P2,D3,D4,D2	60.62	1.05 secs	2520 of 40320



Simulation Result for Scenario 4:

Travel Time

Genetic Algorithm Process - Total Travel Time

Type	Route	Total Duration	Car 1	Duration	Car 2	Duration	Pr. Time
1	P2,P3,P1,P4,D4,D3,D1,D2	37.35 m	T1,P2,P3,P1,D3,D1,D2	26.08 m	T2,P4,D4	11.27 m	6.01 s

Brute Force Process - Total Travel Time

Rank	Route	Total Duration	Car1 Route	Duration	Car2 Route	Duration	Pr. Time	Total Routes
Best	P4,D4,P2,P1,P3,D3,D1,D2	35.52 m	T1,P4,D4,P2,P1,P3,D3,D1,D2	35.52 m	T2	0 m	26.646 s	2520 of 40320
Medium	P1,P2,P3,P4,D3,D4,D1,D2	49.87 m	T1,P1,P2,D1,D2	22.73 m	T2,P3,P4,D3,D4	27.13 m	26.646 s	2520 of 40320
Worst	P1,P4,P2,D2,P3,D4,D3,D1	65.67 m	T1,P3,D3	10.72 m	T2,P1,P4,P2,D2,D4,D1	54.95 m	26.646 s	2520 of 40320

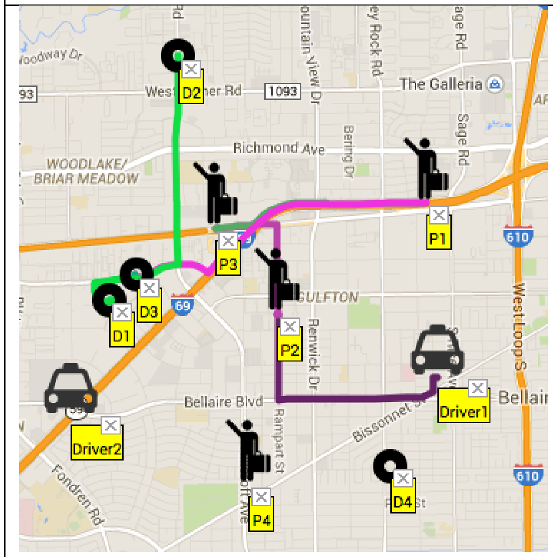


Figure 26: Car 1, 3 Passengers and 3 Destinations

GA for Car 1: T1, P4,D4,P2,P1,P3,D3,D1,D2

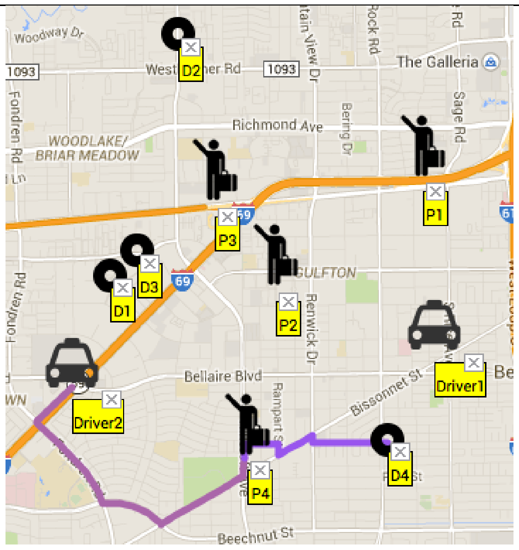


Figure 27: Car 2, 1 Passengers and 1 Destinations

GA for Car 2: T2,P4,D4

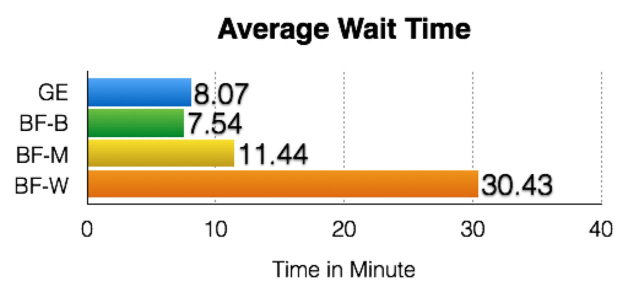
Average Wait Time

Genetic Algorithm Process - Average Passengers Wait Time

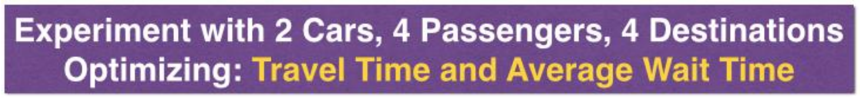
Type	Route	Total AvgWT	Car 1	AvgWT	Car 2	AvgWT	Pr. Time
1	P4,P3,P1,P2,D3,D4,D2,D1	8.07 m	T1,P4,D4	5.28 m	T2,P3,P1,P2,D3,D2,D1	9 m	0.711 s

Brute Force Process - Average Passengers Wait Time

Rank	Route	Total AvgWT	Car1 Route	AvgWT	Car2 Route	AvgWT	Pr. Time	Total Routes
Best	P3,P4,P2,P1,D3,D4,D1,D2	7.54 m	T1,P4,P2,D4,D2	7.97 m	T2,P3,P1,D3,D1	7.12 m	11.182 s	2520 of 40320
Medium	P3,P4,P2,D2,D3,D4,P1,D1	11.44 m	T1,P2,D2,P1,D1	13.63 m	T2,P3,P4,D3,D4	9.25 m	11.182 s	2520 of 40320
Worst	P1,P4,D1,D4,P2,D2,P3,D3	30.43 m	T1	0 m	T2,P1,P4,D1,D4,P2,D2,P3,D3	30.43 m	11.182 s	2520 of 40320



5. Analysis and fine-tuning of planned system



Experiment with 2 Cars, 4 Passengers, 4 Destinations
Optimizing: Travel Time and Average Wait Time

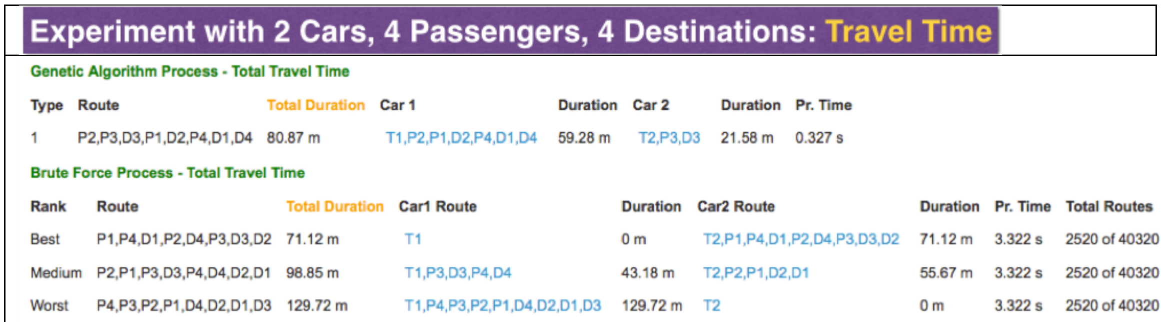
A. Compare the Performance Brute Force Algorithm with Genetic Algorithm

B. Proceed 10 GA processes with 2 configurations to form the initial population

   1. Random Initial Passengers and Random Initial Destinations

   2. Fixed Initial Passengers and Fixed Initial Destinations

In order to benchmark the algorithm, we need to use a faster server in terms of faster CPUs and larger memories. For this experiment, we use Amazon Cloud EC2 (m3.xlarge) with 4 CPUs and 15 GiB memories.



Experiment with 2 Cars, 4 Passengers, 4 Destinations: Travel Time

Genetic Algorithm Process - Total Travel Time

Type	Route	Total Duration	Car 1	Duration	Car 2	Duration	Pr. Time
1	P2,P3,D3,P1,D2,P4,D1,D4	80.87 m	T1,P2,P1,D2,P4,D1,D4	59.28 m	T2,P3,D3	21.58 m	0.327 s

Brute Force Process - Total Travel Time

Rank	Route	Total Duration	Car1 Route	Duration	Car2 Route	Duration	Pr. Time	Total Routes
Best	P1,P4,D1,P2,D4,P3,D3,D2	71.12 m	T1	0 m	T2,P1,P4,D1,P2,D4,P3,D3,D2	71.12 m	3.322 s	2520 of 40320
Medium	P2,P1,P3,D3,P4,D4,D2,D1	98.85 m	T1,P3,D3,P4,D4	43.18 m	T2,P2,P1,D2,D1	55.67 m	3.322 s	2520 of 40320
Worst	P4,P3,P2,P1,D4,D2,D1,D3	129.72 m	T1,P4,P3,P2,P1,D4,D2,D1,D3	129.72 m	T2	0 m	3.322 s	2520 of 40320

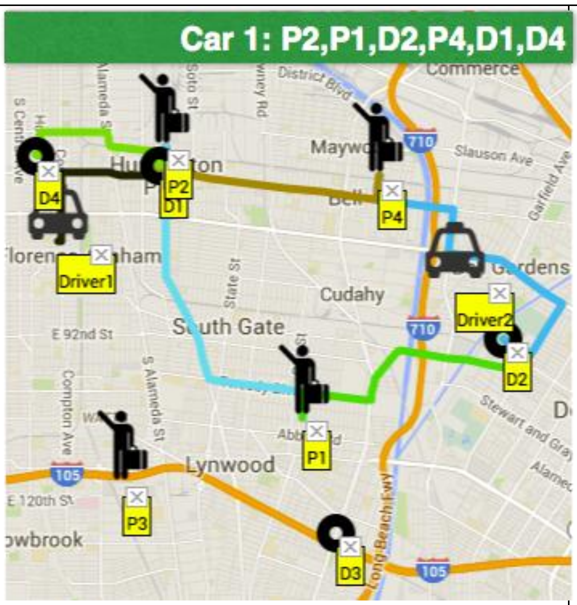


Figure 28: Car 1, 3 Passengers and 3 Destinations for Travel Time

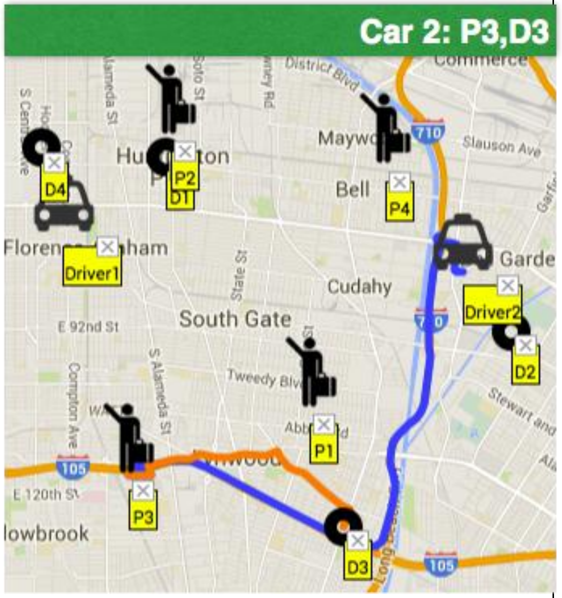
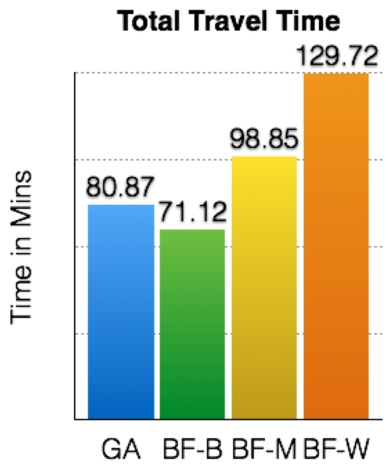


Figure 29: Car 2, 1 Passenger and 1 Destination for Travel Time

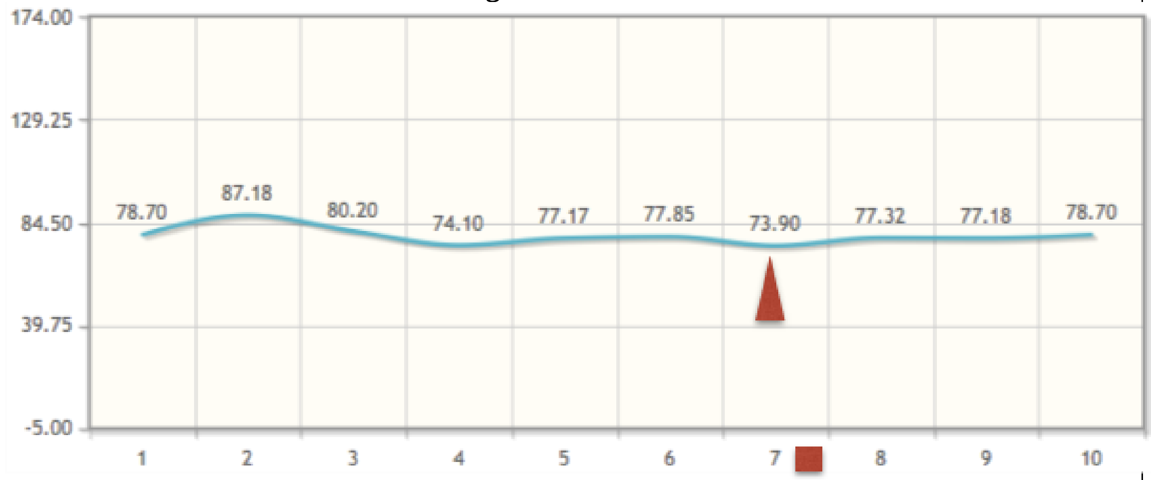


Experiment with 2 Cars, 4 Passengers, 4 Destinations: Travel Time

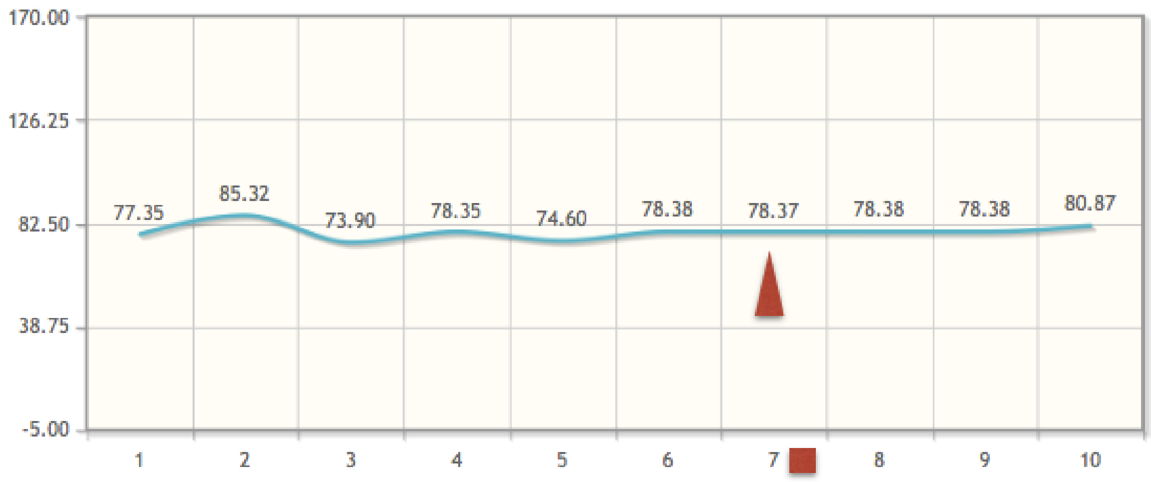
10 GA Processes

x axis: Number of Processes
y axis: Time in Minutes

Random Initial Passengers & Random Initial Destinations



Fixed Initial Passengers & Fixed Initial Destinations

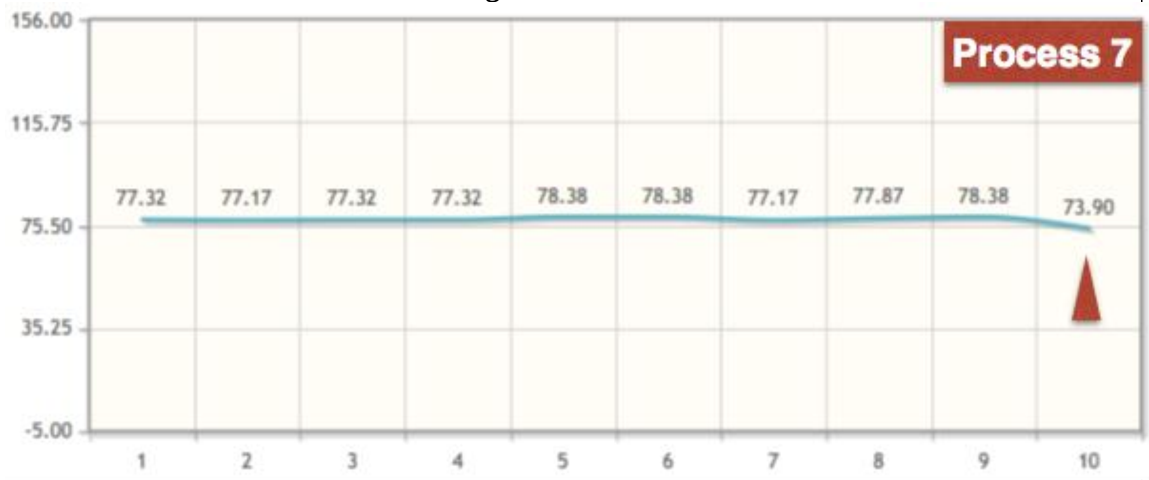


Experiment with 2 Cars, 4 Passengers, 4 Destinations: Travel Time

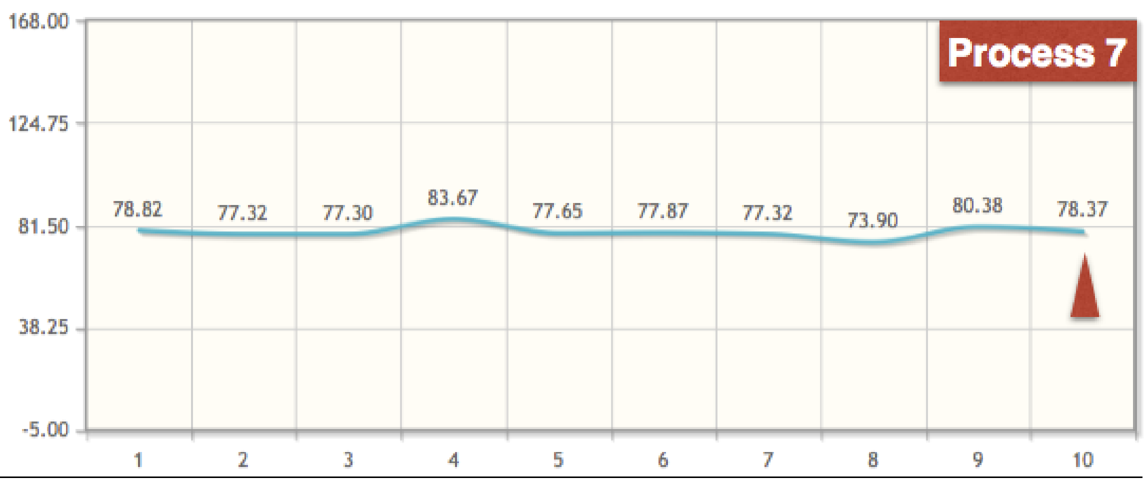
10 GA Generations

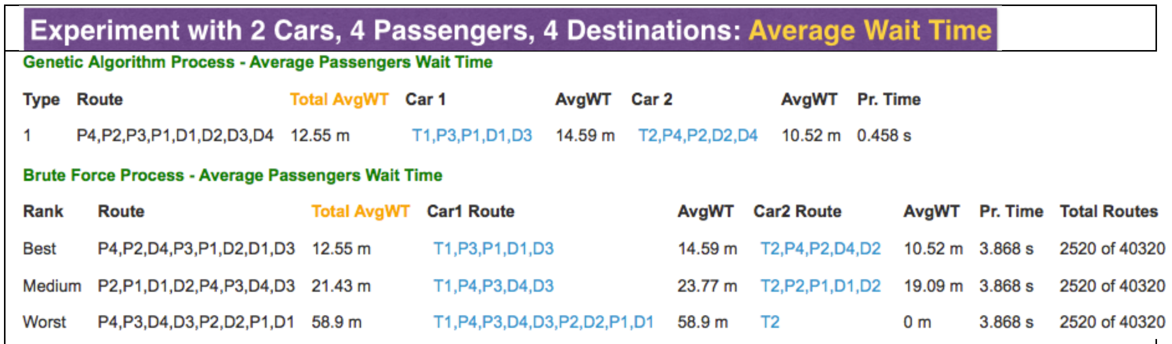
x axis: Number of Generations
y axis: Time in Minutes

Random Initial Passengers & Random Initial Destinations



Fixed Initial Passengers & Fixed Initial Destinations





Experiment with 2 Cars, 4 Passengers, 4 Destinations: Average Wait Time

Genetic Algorithm Process - Average Passengers Wait Time

Type	Route	Total AvgWT	Car 1	AvgWT	Car 2	AvgWT	Pr. Time
1	P4,P2,P3,P1,D1,D2,D3,D4	12.55 m	T1,P3,P1,D1,D3	14.59 m	T2,P4,P2,D2,D4	10.52 m	0.458 s

Brute Force Process - Average Passengers Wait Time

Rank	Route	Total AvgWT	Car1 Route	AvgWT	Car2 Route	AvgWT	Pr. Time	Total Routes
Best	P4,P2,D4,P3,P1,D2,D1,D3	12.55 m	T1,P3,P1,D1,D3	14.59 m	T2,P4,P2,D4,D2	10.52 m	3.868 s	2520 of 40320
Medium	P2,P1,D1,D2,P4,P3,D4,D3	21.43 m	T1,P4,P3,D4,D3	23.77 m	T2,P2,P1,D1,D2	19.09 m	3.868 s	2520 of 40320
Worst	P4,P3,D4,D3,P2,D2,P1,D1	58.9 m	T1,P4,P3,D4,D3,P2,D2,P1,D1	58.9 m	T2	0 m	3.868 s	2520 of 40320

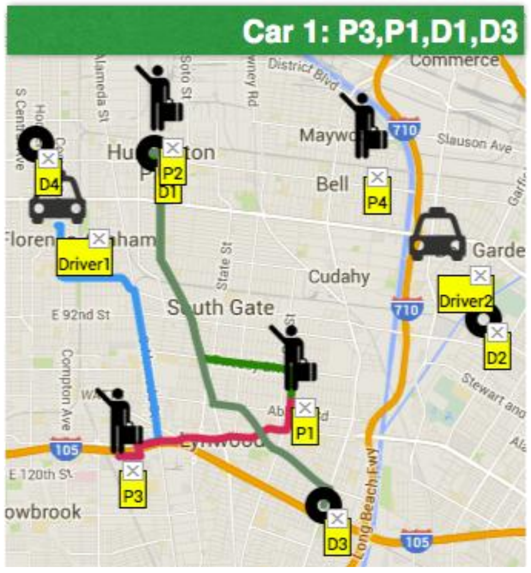


Figure 30: Car 1, 2 Passengers and 2 Destinations for Average Wait Time

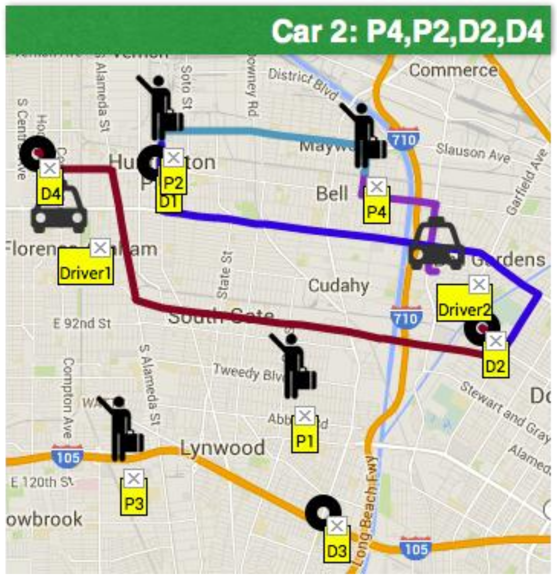
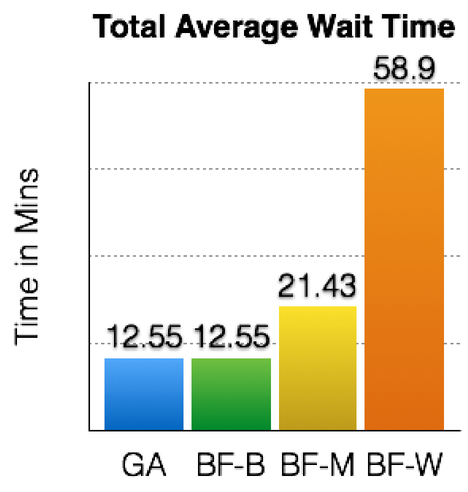


Figure 31: Car 2, 2 Passengers and 2 Destinations for Average Wait Time

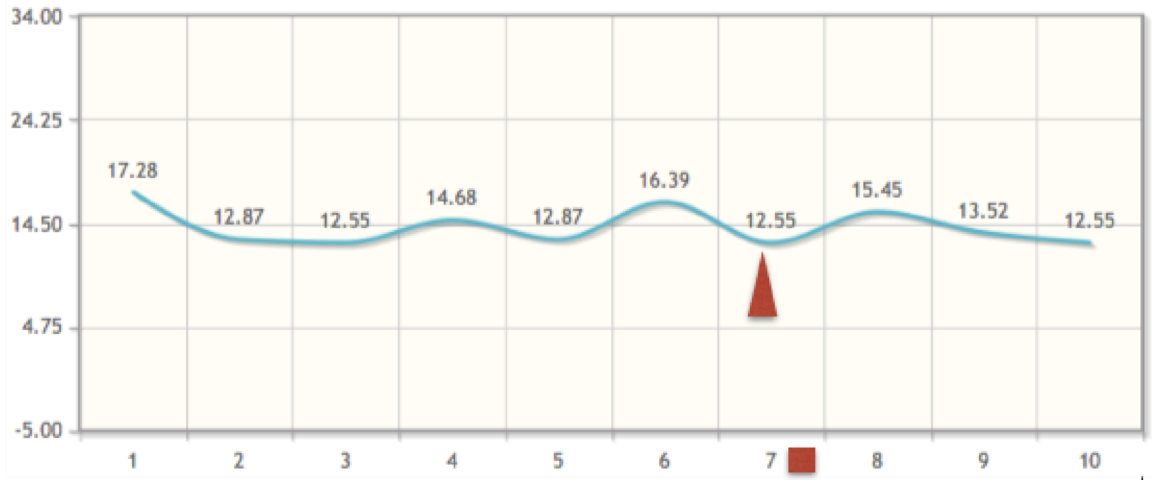


Experiment with 2 Cars, 4 Passengers, 4 Destinations: Average Wait Time

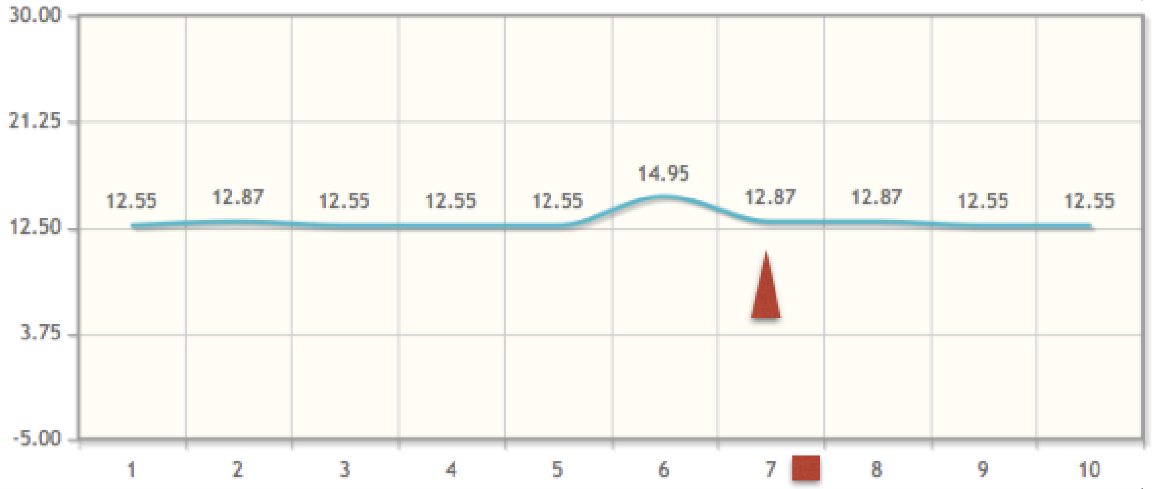
10 GA Processes

x axis: Number of Processes
y axis: Time in Minutes

Random Initial Passengers & Random Initial Destinations



Fixed Initial Passengers & Fixed Initial Destinations

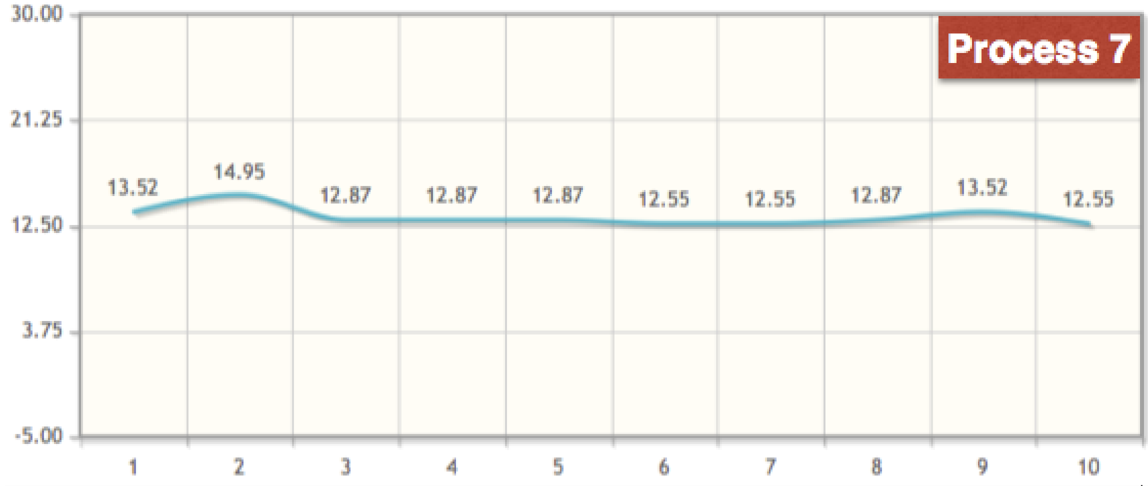


Experiment with 2 Cars, 4 Passengers, 4 Destinations: Average Wait Time

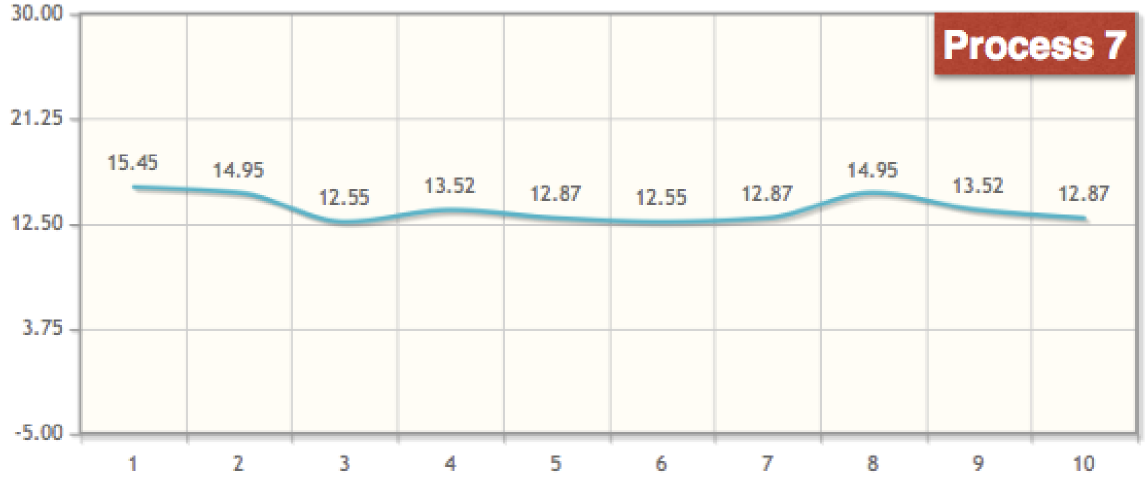
10 GA Generations

x axis: Number of Generations
y axis: Time in Minutes

Random Initial Passengers & Random Initial Destinations



Fixed Initial Passengers & Fixed Initial Destinations



Experiment with 2 Cars, 5 Passengers, 5 Destinations: Travel Time

Within PHP program, we first attempted to allocate 4GB memory to process the Brute Force algorithm for 5 passengers, 5 destinations and 2 cars. Due to this complex problem, it required a lot of computations. There are 3,628,800 candidate solutions. When it reached 4GB memory, the program was stopped and printed out the first error.

Next, we attempted to allocate 8GB memory and the execution time was 600 seconds (10 minutes). It required more than 10 minutes to process this problem. When it reached 10 minutes, the problem was stopped and printed out the second error.

Last, we attempted to allocate 12GB memory and set the execution time to infinity. The program was finally finished and printed out the result at 894.006 seconds or ~15 minutes. It also took 97% CPU resources.

Brute Force - Allocate 4GB Memory for the process

```
[Sat Dec 06 09:13:14.695016 2014] [:error] [pid 1628]
PHP Fatal error:  Allowed memory size of 4294967296
bytes exhausted (tried to allocate 233 bytes)
```

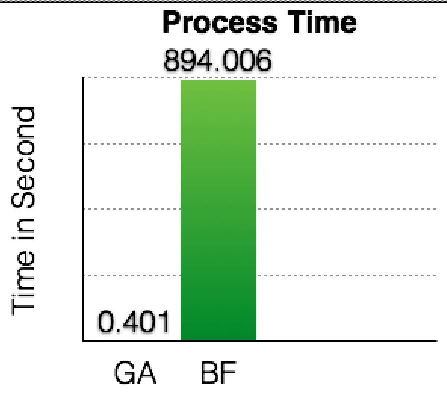
Brute Force - Allocate 8GB Memory for the process

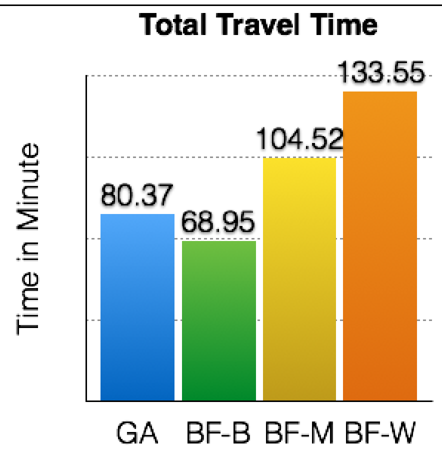
```
[Sat Dec 06 09:31:42.940121 2014] [:error] [pid 1667]
PHP Fatal error:  Maximum execution time of 600
seconds exceeded
```

Brute Force - Allocate 12GB Memory for the process

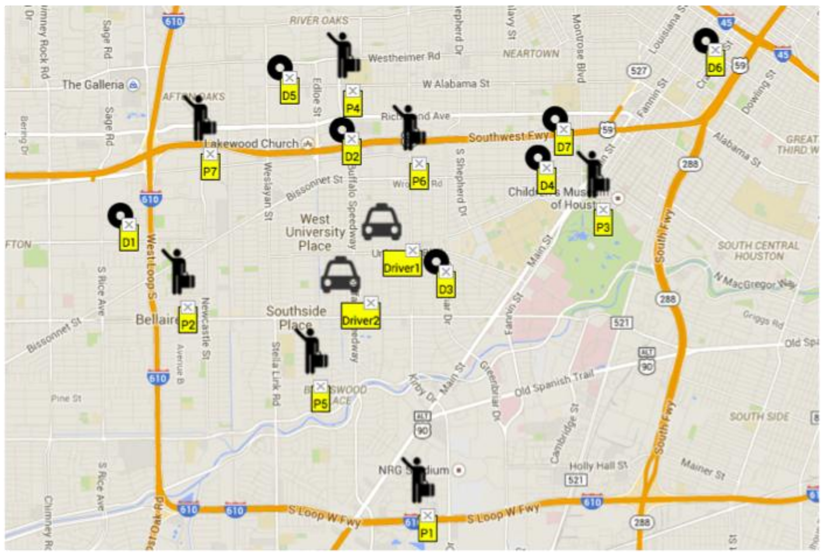
```
{"result":[
{“duration":68.95},{"duration":104.52},{"duration":133.55}
],"time":894.006,"total":"113400 of 3628800"}

real ~15 minutes              elapsed 97% CPU
```





Experiment with NP-Hard Problem: 2 cars, 7 passengers, 7 destinations



Driver: Car
P: Passenger
D: Destination

Proceed 10 GA Processes with 2 Configurations for Initial Population

1. Random Initial Passengers and Random Initial Destinations
2. Fixed Initial Passengers and Fixed Initial Destinations

Experiment with NP-Hard Problem: 2 cars, 7 passengers, 7 destinations: Travel Time

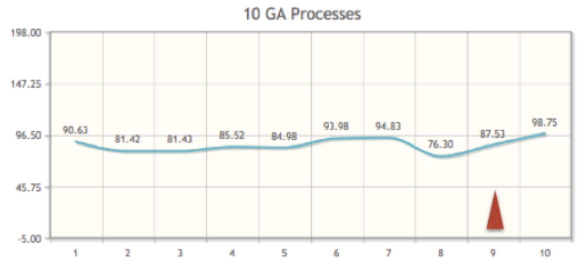
x axis: Number of Processes

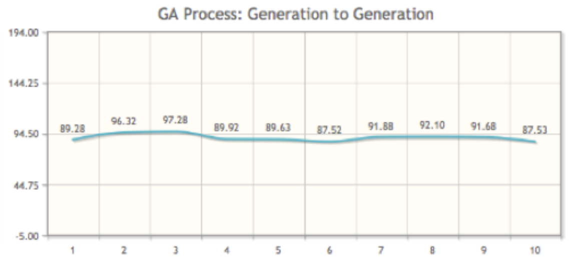
y axis: Time in Minutes

x axis: Number of Generations

Random Initial Passengers & Random Initial Destinations

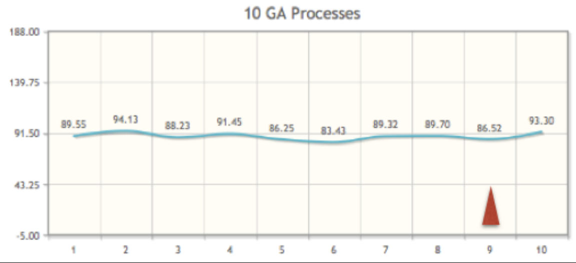
Process 9

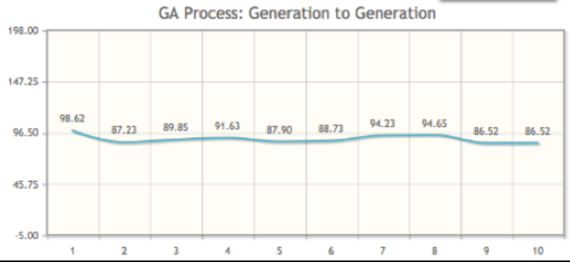




Fixed Initial Passengers and Fixed Initial Destinations

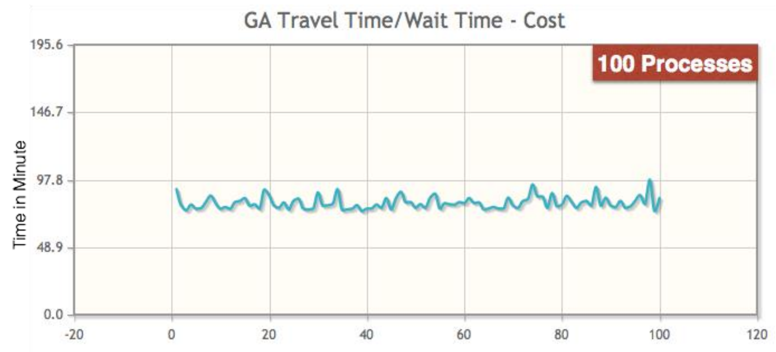
Process 9



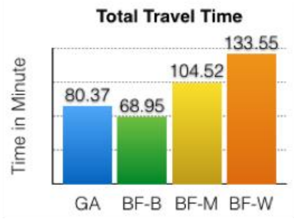


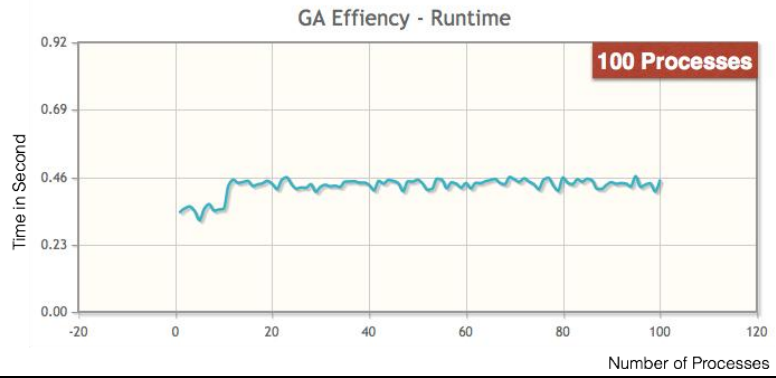
DISCUSSION - Experiment with 2 Cars, 5 Passengers, 5 Destinations: Total Travel Time

Fixed Initial Passengers & Fixed Initial Destinations

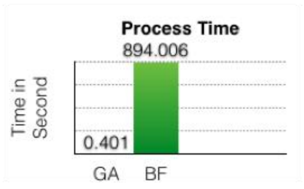


Total Travel Time
High: 97.8 mins
Low: 75 mins





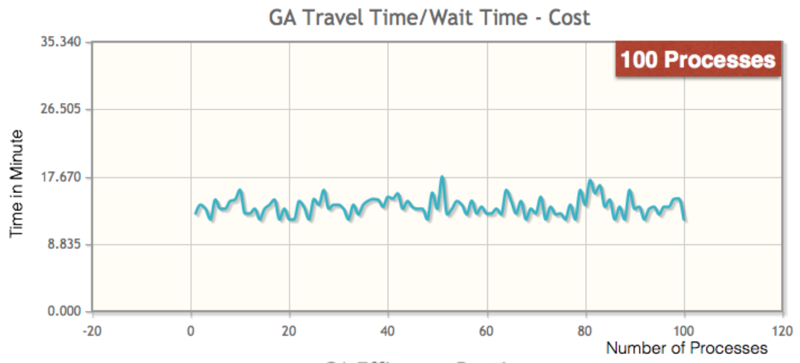
Process Time
High: 0.34 secs
Low: 0.463 secs



In this experiment, we run 100 GA processes for 2 cars, 5 passengers, and 5 destinations to find the optimal duration in terms of total travel time. The longest duration for this case is 133.55 minutes and the shortest duration is 68.95 minutes. The GA results for 100 processes are ranking from 75 minutes to 97.8 minutes. The average duration is 80 minutes. With Brute Force, it takes 15 minutes to finish the process but GA takes 0.4 second. In the reality, we do not want to wait 15 minutes to get the most optimal routes rather sub-optimal routes with the fast response from the system.

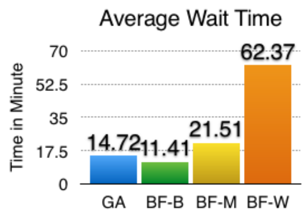
DISCUSSION - Experiment with 2 Cars, 5 Passengers, 5 Destinations: Average Wait Time

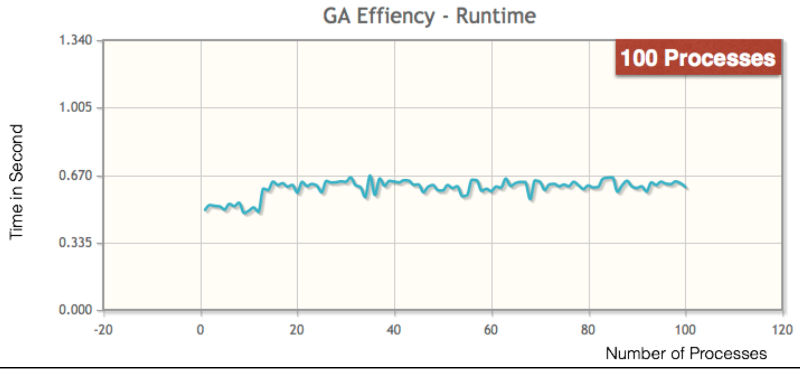
Fixed Initial Passengers & Fixed Initial Destinations



Avg Passenger Wait Time

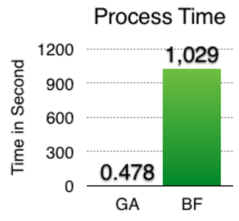
High:	17.67 mins
Low:	12.05 mins





Process Time

High:	0.481 secs
Low:	0.667 secs



The GA process of Average Wait Time (GA-AWT) takes longer than the GA process of Total Travel Time. The GA-AWT calculates all passengers' wait time, then it computes the average wait time for each routes. It takes approximate 0.478 second to complete each GA process. With 100 GA processes, the longest AWT is 17. 67 minutes, and the shortest AWT is 12.05 minutes. The average of 100 processes of AWT is 14 minutes. When we deal with NP-Hard problem, the runtime of the process is very expensive. Without this advanced GA process, we can quickly find the sub-optimal solution in a very short time. It is guaranteed to get a sub-optimal result that is better than the medium result.

6. Conclusion

The Brute Force Algorithm takes years to process a complex problem such as 2 cars, 7 passengers and 7 destinations. The factorial of combination of 7 passenger and 7 destinations (14!) is 87,178,291,200 solutions (passengers and destinations orders). The exponential of 2 cars with 7 passengers (2^7) is 128 (cars orders). The total candidate solutions for 2 car, 7 passengers and 7 destinations (87,178,291,200 * 128) are 11,158,821,273,600.

The online dispatch system is improved when using the Genetic Algorithm for NP-Hard problem. With a small initial population of the dataset, the system can quickly produce a sub-optimal result in less than a second. However, it's not guaranteed to get a global optimal solution rather than a local optimal solution.

The Genetic Algorithm with Constraints is an advanced implementation that proceeds 2 GA processes for each gene from generation to generation using revolutionary concepts. Hence, the ODS can approximately provide sub-optimal solutions in terms of minimizing the passenger wait time and travel time.

7. Future Work

In order to improve the result of Genetic Algorithm process, first we need to form a better initial population. In theory, better parents should produce better children. This work will take a lot of time to research a heuristic methodology. Second, we need to improve the fitness function by combining the travel time with average wait time.

When we deal with more complex problem such as 5 cars, 20 passengers, and 20 destinations, the small population and less number of generations will not produce optimal results. However, the large population and many generations will require a lot of resources in terms of CPUs and Memories. To improve the performance of GA procedure for this problem, we will need distributed and advanced computers or use parallel computing method to speed up the process.

8. References

Craenen B. G. W., Eiben A. E. and Marchiori E. How to handle constraints with evolutionary algorithms. In L. Chambers, The Practical Handbook of Genetic Algorithms: Applications, 2nd edition , volume 1, pages 341-361. Chapman & Hall/CRC, 2001

Feltman, Rachel. "The War between Uber and Lyft Will Be Won with an Algorithm." Quartz. N.p., 4 Apr. 2014. Web. 02 Dec. 2014. <http://qz.com/195348/the-war-between-uber-and-lyft-will-be-won-with-an-algorithm/>.

Meng, Q.B., Mabu, S., Yu, L., and Hirasawa, K. 2010. A Novel Taxi Dispatch System Integrating a Multi-Customer Strategy and Genetic Network Programming. Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol. 14, No. 5, pp. 442-452.

Monschke, Jan. "Genetic Algorithms." *An Introduction of Genetic Algorithms*. N.p., n.d. Web. 02 Dec. 2014. <http://janmonschke.com/Genetic-Algorithms/presentation/#/>.

Jung, J., Jayakrishnan, R., & Park, J.Y. (2013). Design and modeling of real-time shared-taxi dispatch algorithms. In: Proc. Transportation Research Board 92nd Annual Meeting, Washington, DC.

Tao, C.C. 2007. Dynamic Taxi-sharing Service Using Intelligent Transportation System Technologies. International Conference on Wireless Communications, Networking and Mobile Computing, pp. 3209-3212.

Uber. http://www.uber.com. Accessed Septemer 01, 2014.