



A Cost Model for Query Auto Completion in Information Retrieval.

Samantha Coll

CES Final Year Project

Supervisor: Dr Leif Azzopardi

Second Assessor: Dr David Harle

Date: 31/03/2017

# Abstract

In information retrieval, query auto completion (QAC), denotes the resulting functionality: for any given prefix of characters entered into a search box, a list of projected alternative ways of extending the prefix into a full query, must be suggested by the user interface. [1] Most approaches to this issue require query logs to produce meaningful completion proposals. These proposals are ranked in order of importance to match a certain criteria, e.g. popularity, time, location, and the top *k* proposals are offered to the user as a completion suggestion. In this report, we explore how the ranking effects these query completions by developing a cost model that will show how users interact with the search bar on a website like Booking[[1]](#footnote-1). Analysing different datasets and exploring the different ways of ranking the QACs, shows the role that different parameters have on the algorithm and indicate a high quality of completion suggestions.

The cost model developed throughout the project can be found at:   
[www.github.com/samanthacoll22/PlacesProject](http://www.github.com/samanthacoll22/PlacesProject)

# Acknowledgment

*I would like to thank Dr Leif Azzopardi for enabling me to work on this interesting topic. A big thank-you also, for your guidance and extensive knowledge of the problem. Furthermore, an extended thank you to Dr David Harle for his useful and insightful comments during the poster session.*

*Finally, my deep gratitude towards my family. Thank you for your encouragement throughout my studies, and especially this project.*

# Report Summary

Chapter 1: Introduction – A summary of aims to be discussed throughout report.

Chapter 2: Project Planning and Organisation – An analysis of the overall organisation of the project and the time management in which it was carried out.

Chapter 3: Project Context – A look into the background of the project and the different aspects it contains.

Chapter 3: Design – A technical look at the programmatic design of the application.

Chapter 4: Testing Strategies – A technical look into whether the projects outcomes were as expected.

Chapter 5: Performance Analysis & Evaluation – A study into the consistency of the project as the parameters are changed and an evaluation of the aims.

Chapter 6: Future Work – Discussion and research into ways the project could be extended and the effects this would have on the project

Chapter 7: Future Applications: A look into the different practices in which the cost model could be integrated within its current state, or with an additional extension explored in Chapter 6.

Chapter 8: Conclusions - A mirror to Chapter 1, covering which of the aims the project has met and which aims remain to be completed.

Contents

[Abstract i](#_Toc478685436)

[Acknowledgment ii](#_Toc478685437)

[Report Summary iii](#_Toc478685438)

[1. Introduction 1](#_Toc478685439)

[1.1. QAC Problem Description 1](#_Toc478685440)

[1.2. Aims and Objectives 2](#_Toc478685441)

[2. Project Planning and Organisation 3](#_Toc478685442)

[2.1. Original Project Plan 3](#_Toc478685443)

[2.1.1. Research 3](#_Toc478685444)

[2.1.2. Implementation 3](#_Toc478685445)

[2.1.3. Testing 4](#_Toc478685446)

[2.1.4. Documentation 4](#_Toc478685447)

[2.2. Final Project Plan 4](#_Toc478685448)

[2.2.1. Research 4](#_Toc478685449)

[2.2.2. Design 5](#_Toc478685450)

[2.2.3. Implementation 5](#_Toc478685451)

[2.2.4. Testing 5](#_Toc478685452)

[2.2.5. Documentation 6](#_Toc478685453)

[2.3. Organisational Tools 6](#_Toc478685454)

[2.3.2. Version Control 6](#_Toc478685455)

[2.3.3. Documentation 6](#_Toc478685456)

[3. Project Context 8](#_Toc478685457)

[3.1. Background 8](#_Toc478685458)

[3.2. Query Auto Completion Using Query Logs 10](#_Toc478685459)

[3.4. Cost Models 10](#_Toc478685460)

[3.4.1. Finding the Cost of a Place 11](#_Toc478685461)

[3.4.2. Determining the Ranking Cost. 11](#_Toc478685462)

[3.5. State Diagrams 12](#_Toc478685463)

[3.6. Probability Distribution 12](#_Toc478685464)

[3.7. Sampling 13](#_Toc478685465)

[3.7.1. Monte Carlo Simulations 14](#_Toc478685466)

[3.8. Ranking of Suggestions 16](#_Toc478685467)

[3.9. Software Engineering 17](#_Toc478685468)

[3.9.1. Architectural Style 17](#_Toc478685469)

[3.9.2. Design Patterns 18](#_Toc478685470)

[4. Design 22](#_Toc478685471)

[4.1. State Machine 22](#_Toc478685472)

[4.1.1. State Diagram 24](#_Toc478685473)

[4.1.2. Determining Probabilities 25](#_Toc478685474)

[4.1.3. Determining Costs of Each State 28](#_Toc478685475)

[4.2. Java Application 30](#_Toc478685476)

[5. Testing Strategies 35](#_Toc478685477)

[5.1. Full System Testing 35](#_Toc478685478)

[5.2. Unit Testing 36](#_Toc478685479)

[5.3. Sub-System Testing 36](#_Toc478685480)

[5.4. Parametric Simulation 36](#_Toc478685481)

[5.5. Black Box Testing 37](#_Toc478685482)

[5.6. Running Cost Model over Different Processors 37](#_Toc478685483)

[6. Performance Analysis and Evaluation 40](#_Toc478685484)

[6.1. Ranking of Suggestions 40](#_Toc478685485)

[6.2. Instance of Suggestions Displayed 43](#_Toc478685486)

[6.3. Amount of Suggestions Displayed 45](#_Toc478685487)

[6.4. Typing Cost 47](#_Toc478685488)

[6.5. Typing Cost vs Looking Cost 48](#_Toc478685489)

[7. Future Work 50](#_Toc478685490)

[7.1. Trie Structure 50](#_Toc478685491)

[7.2. Developing a Native Application 52](#_Toc478685492)

[7.2.1. Android vs iOS 52](#_Toc478685493)

[7.2.2. Location Services 53](#_Toc478685494)

[8. Future Applications 56](#_Toc478685495)

[8.1. Without Extension 56](#_Toc478685496)

[8.2. With Extension 57](#_Toc478685497)

[9. Reflection and Summary 58](#_Toc478685498)

[9.1. Overall Reflection of the Project 58](#_Toc478685499)

[9.2. Conclusion 58](#_Toc478685500)

[Appendices 59](#_Toc478685501)

[Appendix A: Original Project Plan 59](#_Toc478685502)

[Appendix B: Final Project Plan 60](#_Toc478685503)

[Appendix C: GitHub Evidence 61](#_Toc478685504)

[Appendix D: Test Results 62](#_Toc478685505)

[D.1. Full System Testing Results 62](#_Toc478685506)

[D.2. Black Box Testing Results 64](#_Toc478685507)

[References 65](#_Toc478685508)

# Introduction

Web searching is a tedious and omnipresent activity which is carried out by millions of people on a daily basis. However, with the aid of QAC, an element which tries to predict the user’s query and offers a list of proposed suggestions, this time-consuming task could be reduced by minimising the amount of input needed from the user. This is possible by keeping the amount of keystrokes to a minimum by helping form a query. The primary objective for effective QAC is to provide the user with their intended query after the fewest possible keystrokes, and at the highest possible rank in the list of completion suggestions. A brief look into the problem faced when using QAC can be seen in section 1.1, whilst section 1.2 outlines the aims and deliverables of the project.

## QAC Problem Description

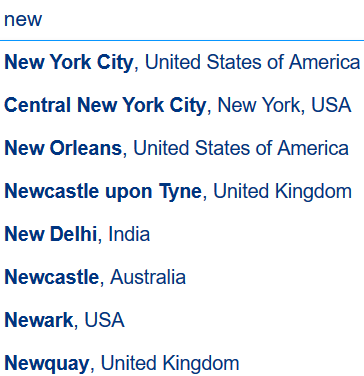
QAC is an algorithm that retrieves a list of possible query completion suggestions, Qs, and then ranks these suggestions in order of importance before showing the top *k* completions to the user. This can only be done by taking the select characters that the user has input, say qi, to form this list of proposed completions. Then the user decides whether a suggestion (preferably the highest ranked), is a match. Some approaches, like Google[[2]](#footnote-2) and other major search engines, can account for spelling mistakes and suggest what they believe the intended query was. For instance, if the string “David Attenbur” is inputted by a user as their qi, then the top query completion presented would be the intended string “David Attenborough”. Other QAC approaches support mid-string completions, where the query suggestion “Kurt Cobain” would be returned if only the string “Cobain” was entered. For this report however, spelling correction and mid-string completion are not necessary as the main aim is to build a cost model which shows how a user handles QAC. Therefore, the full query will be coded into the simulation and checked with query logs before proceeding, meaning it is compulsory that qi is a prefix of all Qs.

## 1.2. Aims and Objectives

The main aims and deliverables of the project are:

1. To create a cost model of how people interact with the search box on a website like Booking, which can be seen in Figure 1, where people start to type in letters and the system predicts where they might want to go.
2. To evaluate the impact ranking has within query autocomplete and how it influences the goal of finding the minimum cost.
3. To evaluate the number of suggestions the user is presented with and the effect it has on the cost model.
4. To evaluate the outcome of generating suggestions at different instances and how they assist to keep cost to a minimum.

An additional aim of the project is to create a Java application that will display various destinations that the user will type in at their own speed. The time taken for the user to type these destinations will be noted and an average time for each letter will be generated. This will then be implemented into the cost model for each user, giving a more realistic, personal result.



***Figure 1: Output from user typing “new” into Booking.***

# Project Planning and Organisation

To guarantee the project could be completed by the final deadline, knowledge of the status of the project was key. To guarantee success, the status of the project at all points was confirmed by the creation of a project plan. This project plan meant that if the project was destined to be incomplete for a particular reason, then this would become apparent with enough time to allow changes to rectify the problem.

## Original Project Plan

The original plan was created by splitting the project into four main sections: Research, Implementation, Testing and Documentation. Each of these sections was then sub-divided into smaller, more manageable tasks, which could be tackled individually. Throughout the entire project, there were only two strict deadlines which would have to be satisfied: Poster Creation (27th November 2016) and Report Write-up (31st March 2016), which dictated deadlines to support the sections from which they were subtasks. The creation of the poster required the research to have been exhausted, which confirmed that section complete. The report covers every aspect of the project, guaranteeing each section has been fully covered by that date. The full original plan created is located in Appendix 1: Original Project Plan.

### Research

Research was undertaken into the way query auto completion operates. A detailed look into how the user jumps between typing a letter, looking at the list of suggestions, to then picking from the list whilst exploring the probabilities of each state occurring. It was decided that seven days was a suitable amount of time to create the poster which assisted in creating a deadline for the end of December for the research section to be completed. Each subtask within the section was then split between the overall times given.

### Implementation

A careful decision was to have the Implementation of the system finalised by the end of February, thus allowing a carefully allocated suitable amount of time for the final overall system to be tested and the report to be written before the final deadline. The most important part of the implementation stage, was to have the state machine up and working. From there, a ranking system for the suggestions could be created, leaving a functioning cost model to start evaluation. The amount of time allocated to the implementation stage was divided amongst smaller tasks based on the perceived level of difficulty they may entail.

### Testing

To test the cost model, the usual four main aspects of testing were expected to be carried out throughout the project: Unit Testing, Sub-System Testing, Full Testing and User Trials. Ideally, the testing aspect of the report was to be carried out in parallel with the Implementation section. Unit Testing would be performed as each sub-section of the cost model was developed, whilst Sub-Section testing would be performed at the end of each additional feature. User Trials would then be carried out just before writing the final report, which would be centred on finding a user’s personal typing speed. This would leave Full System Testing as the only part of testing carried out independently of the other sections. It was decided that roughly two weeks would be a sufficient amount of time to satisfy this.

### Documentation

The documentation section consists solely on the poster creation and the report write-up for the project. Strategically, the timescale that is allocated to these deliverables has been greatly exaggerated to ensure the project’s completion. Creating a buffer like this allows expansion for the other sections, if in fact necessary, whilst still guaranteeing the deliverables completion before their individual deadlines.

## Final Project Plan

As the project development began, the original plan began to become difficult to shadow exactly. The more complications that arose, the need to continuously alter and update the project plan was necessary to show accurately how the project was proceeding. The final project plan is locate in Appendix 2 Final Project Plan.

### Research

The research section was largely untouched and followed the same guidelines from the original plan. The only extra research that was carried out was to source a significantly larger amount of places/cities. This makes the cost model much more accurate, as originally, there were only 125 cities being loaded into the simulation. Therefore, after the third letter was typed the user was frequently presented with their destination as the first suggestion. This resulted in them selecting it the majority of the time, when in reality they may have been forced to type again or look further down the list. There are now currently around 3000 places/cities which result in more realistic simulations.

### Design

The design section was also largely unchanged from the original plan. The state machine was derived as anticipated. The only main significant change in design was that webpage, originally to be created to find the user’s personal typing speed. This was then simplified to a Java application, reducing the need to employ a new programming language into the project. The design and research sections of the project were largely dependable on one another, meaning that at parts they were carried out simultaneously in the given time frame.

### Implementation

The implementation section was started later than initially anticipated. This prolonged start meant that the foundations for the simulation were developed later than expected, which reduced the amount of time available to evaluate in the end. The need to evaluate is such a crucial part of the aims and deliverable for the project, therefore, the time had to be taken from another section of the report. Luckily, the two-week buffer that had been created to ensure all portions of the project were completed before the write-up was used in this section to ensure all aims were met.

### 2.2.4. Testing

Testing was completely different from first anticipated. As development began it seemed logical to perform Full System testing as the implementation was undergoing. The Unit and Sub-System testing were carried out before the completion of the cost model. However, only at the point the majority of the code was complete. The addition of black box testing came as a last minute decision to test certain aspects of the JavaApp. Finally, the decision to carry out Parametric Simulation testing was made, to confirm expected outputs from changing a single parameter whilst keeping all others constant. The Parametric Simulation testing was performed as the write-up for the project was carried out, ensuring evaluation results were correct. The testing section fit within the timescale outlined in the original plan despite the re-arrangement and addition of tests.

### 2.2.5. Documentation

The poster creation and report write-up were exactly as expected from the original project plan. The only problems that arose were sharing the time allocated to the documentation with the other sections that were now being run simultaneously alongside it. These circumstances however, were foreseen whilst making the original plan, thereby being the reason so much time was allocated towards the report write-up in the first place.

## Organisational Tools

To ensure completion of the project, various organisations tools were used to keep within the timescale outlined in Chapter 2.3: 'Final Project Plan’.

### 2.3.2. Version Control

Version Control is an important aspect of any project with a heavily weighted software component. For this project specifically, Git was utilised. Git offers a variety of benefits to a project, including the ability to work on different features, or sections, in parallel whilst on separate branches. This ability allows the changes on an individual feature to have no negative consequences or effects on the other. However, the most important and most relied on feature of Git, if the key functionality breaks, is the ability to revert to a previous version of the software. This allows access to a functional version at a later date, as a form of back-up copy.

Git operates on a central server, where work can be uploaded to and downloaded from. The Git server used for this project was provided by Github[[3]](#footnote-3), which holds a copy of all work on their servers. This not only grants access to be worked on by multiple machines, but as long as the software is being pushed on a regular basis to the server, creates a safety net as large amounts of work cannot be lost if something were to happen to the work station.

### 2.3.3. Documentation

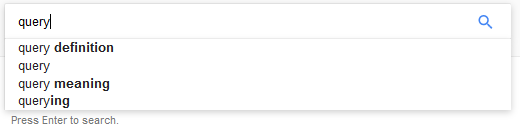
When dealing with a project of significant size, and the key aims rely highly on evaluation, it is important to maintain a detailed record of work completed throughout. As the project was carried out, design and implementation notes have been documented in a logbook, giving an insight into the thought processes throughout. Additionally, when software is pushed to Git, it is a requirement to detail any changes that have been made. These messages are known as a Git log and they show what has been altered and, more importantly, why.

# Project Context

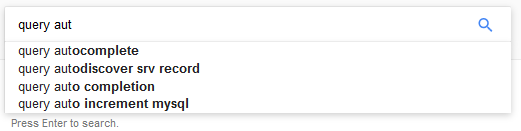
Initially, before any work could begin to take place, vital key questions had to be answered about how the project should begin to proceed. To answers these questions, research into the different approaches of QAC was initiated. Furthermore, investigations into several different aspects of the project were explored further.

## Background

The time-consuming and error-prone process of formulating and physically typing search queries is what can make query input laborious. In response to this problem, search engines have adopted the concept of query auto-complete (QAC) as a way or reducing the effort it takes to submit a query. QAC provides the user with a list of suggested queries as they begin to enter their query in the search box. The partial input of the user is often referred to as *query prefix* and the suggested queries are often called *query completions*. Figure 2, represents the users incomplete input into the search bar (Figure 2a) and how they are constantly updated as the user begins to type more letters (Figure 2b). The most common approach to QAC is to extract past queries with each prefix from a query log, and rank them by their past popularity [2]. Although this approach provides satisfactory results on average, it is hardly most favourable as it fails to take in clues such as time or user context which quite often influences the queries most likely to be typed. This report will focus on the development of a cost model that shows how a user interacts with a ‘search bar’ on a website. It will then explore these different approaches of QAC and show how each keep cost to a minimum. In addition to a minimal cost, reducing the amount of time a user spends searching denotes a lower load on the search engine, which in the end will result in savings in machine resources and maintenance [3].



**a) *A list of four query completions for the prefix “query”.***



1. **b) *An updated list of four query completions for the prefix “query aut”.  
   Figure 2: Examples of query auto completion.***

As the web continuously becomes a platform for real-time news and media, time consumption takes a vital role on the retrieval of information. Users turning to search engines for information about newly occurring or ongoing events take up the majority of daily queries [4]. However, the QAC is not restricted to search engines, such as Google or Bing. QAC has also been integrated into other common services which also have a searching function. The ability to find friends on Facebook[[4]](#footnote-4) is provided with a QAC support, this occurs by taking in mutual friends the user may share with others. QAC also assists users to formulate requests for searching through tweets on the social medium Twitter[[5]](#footnote-5). Furthermore, QAC is also used for product searches in online shopping stores, such as Amazon[[6]](#footnote-6). In the concept of searching online, research highly indicates that QAC can vastly improve user satisfaction, and assist in saving the user time [5]. The main focus for this report is how QAC is integrated into a travel metasearch engine like Booking, where user’s search for the broadest selection of accommodation in every corner or the world.

## 3.2. Query Auto Completion Using Query Logs

As previously stated, the most common approaches to QAC are based on the utilisation of data that is produced by query logs. Where the amount of times the query has been used, the more likelihood it will be chosen as a suggested query. An additional factor of context will be added to enhance the QAC result (e.g. time, location, and season). An illustrated approach to this was outlined by Whiting and Jose [6], where they solved the constraint of producing query suggestions for consistent, yet newly popular queries. They do this by providing numerous different algorithms, like the ‘sliding window’ approach where they will only calculate how popular a query had been over a limited time frame, for example, a couple of days. By varying the length of time calculated they can begin to see how popular a query is depending on how common the prefix is.

Bar-Yossef and Kraus [2] take a different approach and utilised the users query history to provide more relevant completion suggestions. Their algorithm tends to advocate queries from a query log that are similar to the users past queries. To evaluate how alike the new query is to past queries, they develop each query by adding related terms. These queries become characterised as term vectors and their resemblance is then computed as the cosine similarity.

## Cost Models

Cost estimation models are mathematical equations used to estimate the cost of a product or project [5]. These estimations, however, will never be exact as there are many parameters involved in determining a cost estimate, such as technical or human. Moreover, any process that is significantly human based will never be exactly accurate as humans are far too unpredictable and complex to be exact. Cost models function through the input of parameters that describe the attributes of the product or project in question. In the case of this report, the cost model is formed by generating the sum of the places. The equation for the cost model can be seen in “(1)”.

*Total cost = ∑ [cost (place) \* popularity (place)]* (1)

### 3.4.1. Finding the Cost of a Place

To generate the cost of a place, three main factors must be known: the cost it takes for the user to type a single letter, then the cost of a user to look at the list of suggestions, and finally, the cost for the user to pick a suggestion. However, when the user is looking at the list of suggestions there is a degree of uncertainty of what they might do next. This leads to a probability that the user might type and a probability that they may pick from the list. For the purpose of the cost model, it has been decided the user will only be presented with a list of suggestions after typing in a minimum of three letters.

Therefore, if the example “Paris” is taken, it is known that it is necessary for the user to type “P”, “A”, “R”, which each carry a cost to type the letter. So, right now the total cost for the destination is (3 \* typeCost). However, when the user is presented with the retrieved list of suggestions, will the user look at the list and pick? Or will they simply type another letter? This is where the system will have to prepare for all outcomes. The process for finding the fourth letter can be seen in “(2)”.

*(3 \* typeCost) + (probabilityType \* typeCost)   
 + ((1 – probabilityType) \* lookCost)* (2) *+ (lookCost + (probabilityPick \* pickCost))*

If we think about the process in simplistic terms, the total cost for the place is: the typing cost + the looking cost + picking cost. Therefore if “(2)”is broken down into these three categories, after typing in the 3rd letter the user may not look at the list of suggestions at all and just type another letter. Then the cost for the 4th letter would simply be: the original typing cost + the new additional letter cost, where the process will start again for the 5th letter. However, if the probability was high that the user looked at the list (1-probType), then this would result in a looking cost. So, as the user is looking at the list, if the probability is high that they will pick a query, then this will result in a picking cost as well. So in this case the total cost would be: the original typing cost + looking cost + picking cost, which will then complete the query. Although, if we go back a step to the user looking at the list of suggestions and the probability of them picking is low, then they will have to type in another letter. In this case, the total cost for the letter would be: the original typing cost + the looking cost + the new additional letter cost, where the same process would begin to find the 5th letter. This process will be explained more in detail throughout the rest of the report.

### 3.4.2. Determining the Ranking Cost.

On the other hand, the popularity of a place can be determined as the United Nations Statistical Division (UNSD), annually publishes a book on world statistics, which helps us construct a more precise idea of a country’s popularity based on its geographical size. A simple representation of the cost model can be seen in Figure 3.

Cost (Place)

Looking Cost

Typing Cost

Picking Cost

Ranking (Place)

Total Cost

Probability Pick

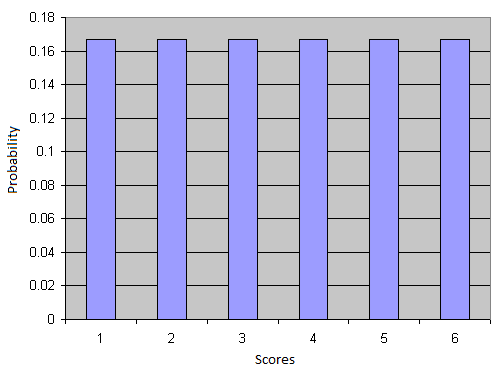
Probability Type

***Figure 3: A Simple Diagram of Key Parameters for Cost Model.***

## State Diagrams

A state diagram is used in software engineering and fellow related fields to give an abstract description of the behaviour of a system. Specifically, a state diagram analyses the behaviour of a sole object in response to a series of events occurring and takes into consideration all possible states the object could find itself in. The cost model in this report will generate four states: a type state, a look state, a pick state and a search/end state. These will each be explored in detail in the design section of the report.

## Probability Distribution

Probability distribution in probability theory and statistics is a mathematical function which provides the probability of occurrence of various possible outcomes in an experiment. If the notion of a coin toss is taken, the probability distribution for the outcome would be, 0.5 for ‘heads’, and 0.5 for ‘tails’. Probability distribution is normally separated in two types; continuous probability distribution and discrete probability distribution. The simulation developed invokes the latter, discrete probability distribution, where a random variable was exactly equal to some encoded value. An example of the discrete probability of a dice roll can be seen in Figure 4, where there are six possible outcomes then the chances of each individual, unique, outcome happening would be.

***Figure 4. Probability Distribution of a Dice Roll***

Thus, relating to the problem at hand where the user is searching a destination, values have to be set to the likelihood that they may type another letter into the search bar and the likelihood of them picking from the list of suggestions. A detailed explanation of how the value of these probabilities is elected can be seen in the design section of the report.

## Sampling

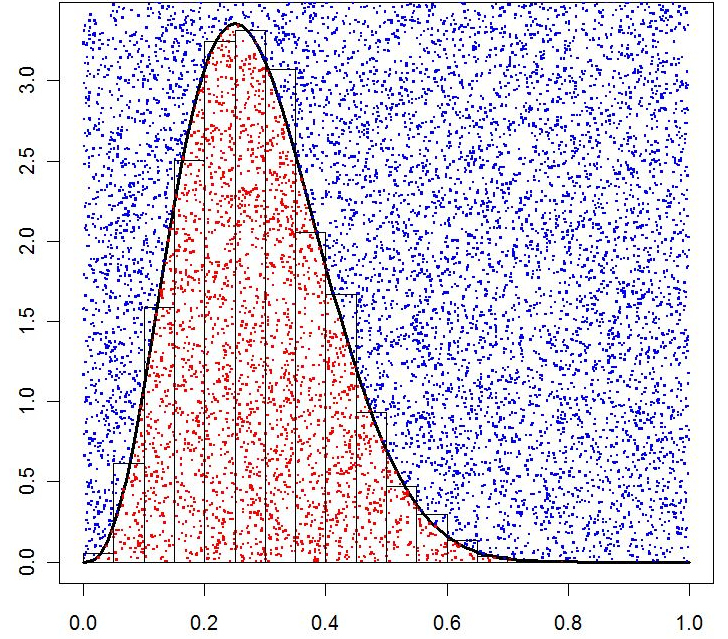
The process of sampling takes a predetermined number of observations from a larger population and carries out analysis on the smaller group. To take every city/place in the world and run the cost model could take days, or maybe even weeks to carry out. It is then far more time-efficient and economical to take a sample of the data. The simulation to run the cost model has around 3000 cities/places loaded into it. These are the cities in the world that have a population of over 100,000 people. Therefore, sampling is already being carried out before generating any total cost. Originally, 319,000 cities/places were used for the simulations which would give more accurate results, but the waiting time to load was around 10 minutes before the simulator proceeded. For that reason, a smaller amount of cities was managed. The advantages and disadvantages of sampling can be seen in Table 1.

***Table 1: The Advantages and Disadvantages of Sampling***

|  |  |
| --- | --- |
| Advantages of Sampling | Disadvantages of Sampling |
| * Low cost of sampling. If data was being collected for the every place in the world, the cost would be extremely high. | * **Chance of bias.** Since specific data is predetermined, this data could favour some places more than others. This could draw erroneous conclusions, e.g. more well-known places are perhaps more likely to get selected. |
| * Less time consuming. Could take days to run the simulation for all places. Sampling gives us an understanding, in a healthier amount of time. | * **Difficult to select truly representative sample.** For instance, favouring higher ranked places. This will give a lower cost than lower ranked places. Meaning the lower ranked destinations are not represented truly and the total cost may be lower than it should be. |
| * Organisation of convenience. By taking a selection of places, it makes it easier to handle and less chance of error along the way. |  |

### 3.7.1. Monte Carlo Simulations

Monte Carlo methods are an extensive class of computational procedures that varies exclusively on recurring random sampling to obtain an assortment of potential numerical results. Monte Carlo simulations take an action choice and display the user with a wide range of results and the probability of them occurring. It shows the extreme possibilities (a user typing every letter of the desired place), whilst highlighting all middle-of-the-road decisions (the user picking their destination once presented in the suggestions). Monte Carlo simulation of a variety of places will allow a mean and standard deviation to be calculated which will be compared to simulation with different parameters. An example of a Monte Carlo simulation can be seen in Figure 5.



***Figure 5: A Monte Carlo Simulation***

## Ranking of Suggestions

In essence, query completions are ranked according to predetermined criterion. Therefore, when a prefix is provided, some of this criterion is then returned to the user. Normally, a precomputed auto-completion system is necessary for determining QAC which correspond to each individual prefix in advance; storing these associations in data structures, such as prefix trees. Figure 6 shows a simple QAC framework.



***Figure 6: Simple QAC Framework***

When a user enters a prefix in the search box, it is shown in Figure 6, that a basic list of suggestions is retrieved. Based on further re-ranking using signals determined at query time, e.g. location, season, time, the user will then receive a finalised list of query completions. Mainly these completions that are returned have a regulated length, for instance, 5 for Booking and 4 for Google. For this project a list of cities/places will be searched and the ones that match the prefix of the desired place will be placed into a separate list. This list will have no form of structure and will simply be outputting the first suggestion it comes across. One of the parameters for the cost model will allow the user to decide how they would like their suggestions to be ranked, either alphabetically, reverse-alphabetically or by population, depending on their desired destination.

## Software Engineering

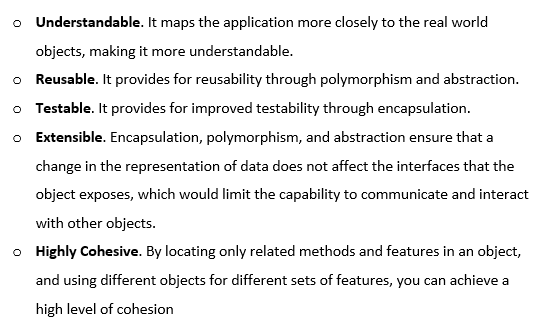
To develop and ensure an easy maintainable project it is vital to try and conduct good software engineering practices throughout. By using well-structured software engineering practices, they should assist in making testing easier as the software should be more flexible and subject to change. The CS409 Software Architecture and Design at Strathclyde University helped pin point various designs and patterns used in the project.

### Architectural Style

An architectural style improves partitioning and promotes design reuse by providing solutions to frequently recurring problems, making this the highest level programming choice to be made in a project. They are thought of as sets of principles that shape an application and they determine how much data and functionality should be separated. Since the decision was made for the project is to be developed using Java, as this was the most familiar programming language, the architectural design that best suits would be Object-Orientated architectural style. Therefore, this is the primary style used throughout the project.

#### 3.9.1.1 Object-Oriented

Object-oriented architecture style is based on dividing up tasks a system may have into individual, independent, reusable objects, which hold the data and behaviour relevant to that object. This style interprets the system as a series of banding objects, instead of a set of routines or procedural calls. This approach evokes a high level of abstraction, which means if one section of code is altered then it would not dramatically affect the programs other sections. Objects are discrete, self-sufficient and lightly coupled. They communicate with one another through interfaces, by calling on methods or by retrieving properties within other objects. Creating an interface gives access to publically available method calls that additional classes may use to interconnect with any class implementing that interface. The key advantages of the object-oriented architectural style are:

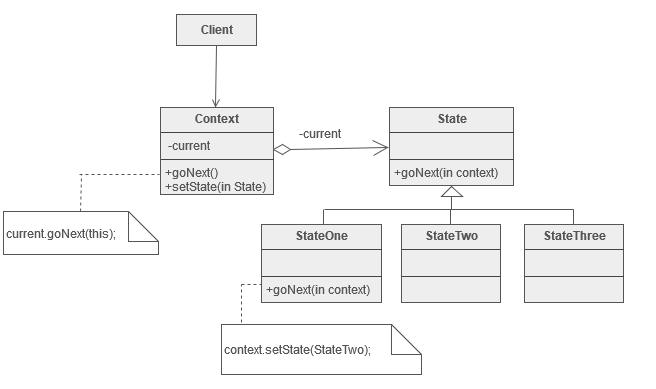


### 3.9.2. Design Patterns

Design Patterns, demonstrate a template of the best design solutions that developers have come across, and allow the same notion be applied to your own design. Design Patterns are effectively recognised solutions to problems in software engineering that are repetitive. By using a design pattern, it allows these problems to be solved in an efficient manner and knowing design pattern sets a familiar vocabulary between software developers. This sub-section of the report will outline the different design patterns that were used throughout this project.

#### 3.9.2.1. State Pattern

The state pattern is a behavioural software design which implements a state machine in an object-orientated approach. Within the state pattern, each distinct state is implemented through its own class from a shared state pattern interface. The transitions are implemented by calling methods from the pattern superclass. This design pattern was an obvious choice for the project as the pattern is used to capture an objects variety of behaviour which is grounded from its internal state. State pattern design keeps code maintainable by creating a smoother transaction between states as it removes the need for conditional statements. The UML diagram of a state pattern can be seen in Figure 7.



***Figure 7: UML Diagram of State Pattern***

Since an object’s behaviour is the outcome function of its state, the crucial advantage of state pattern design is that the behaviour then gets transformed depending on the current state. This thereby, eliminates the need for as many conditional statements. The layout of the design also allows additional states to be easily implemented, whilst improving cohesion as these state-specific behaviours are grouped within ConcreteState classes, which can only be found at one specific location within the code. This can be seen in Figure 7, where the ConcreteState classes are ‘StateOne’, ‘StateTwo’ and ‘StateThree’. With the state pattern design making it so effortless to add additional object behaviour, this enhances the flexibility of the code, improving the overall maintenance of the system.

#### 3.7.2.2. Observer Pattern

The Observer Design Pattern is effectively, where one section of code permits others implementing an interface to register for updates. At the point where an object notices a change occurring, it will then inform the others listening for updates. The Observer Pattern permits two separate sections, which hold similar data, to stay consistent while also staying lightly coupled. A disadvantage to using the Observer Pattern is that debugging problems begin to arise as the movement of code begins to become more subtle. On the other hand, an advantage of using the Observer Pattern is that it unequivocally decouples the concrete observer from the concrete subject, permitting changes to the observer. An Observer Pattern can be seen in Figure 8.

Observer

*Update()***Update(State)**

Concrete Subject

*getState()*

Subject

addObserver(Observer)  
notify()

Concrete Observer

***Figure 8: Observer Design Pattern UML***

The Observer pattern is used in the creation of the JavaApp that allows the user to login and find their typing speed. This application depends greatly on the use of listeners, which the Observer pattern is based on. The event listener was used a great deal, where an ActionListener was registered to a button UI control. In this case the ActionListener is the observer and the button is the subject. As the button changes state, the next method would be called.

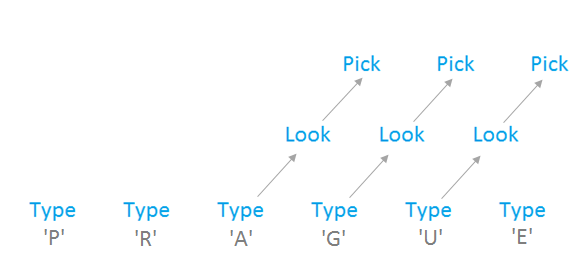
# Design

Whilst developing a cost model that has interchangeable parameters, a high quality software design is crucial to allow the user to swap out these parameters, whilst ensuring an easily maintainable program. This chapter will look at the algorithms developed and how they were broken down to solve the problem at hand. It will also discuss the User Interface for the JavaApp, whilst outlining other notable features. The formation of a strong design will result in the software being easily understood by an outside party experienced in that programming field. If the design can be understood simply, the option of extending the project at a later time becomes far more probable.

## State Machine

The fundamental algorithm necessary for developing the cost model was to design and build a state machine that could handle all possible scenarios of a user searching a destination. A state machine is an abstract machine that can be in precisely one of a finite number of states at any given point in time. There are four states in the development of the state machine for the cost model, these include: when a user is typing a letter into the search bar, this is known as ‘type’ state. When the user is looking at the list of suggestions to check if their desired destination is there, this is known as the ‘look’ state. When the user has selected their destination from the list then this is known as the ‘pick’ state. Finally when a user is finished their query, then there will be a final end state to show that the state machine has come to a halt. This is known throughout the program as the ‘search’ state.

However, these four different states can only jump between each other depending on their response to some external inputs; the process of changing from one state to another is called a *transition*. These transitions will rely solely on a list of probabilities, which will determine what the next state will become. Time was taken to analyse all possible ways a user may reach their end result to help aid in the designing of the state machine. Figure 9 shows the though process behind searching ‘Prague’.



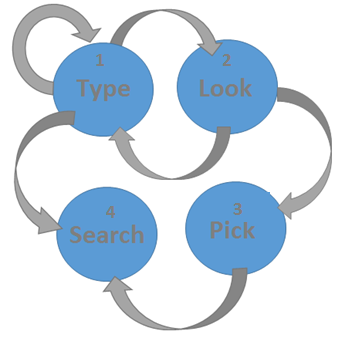
***Figure 9: Thought Process behind ‘Prague’***

In Figure 9, it is shown that whilst the user will remain in the ‘type’ state until the length of the prefix if greater than three, the user then has the option to jump to the ‘look’ state or remain in the ‘type’ state. Remaining in the ‘type’ state is still a viable option as in this example, ‘Prague’ is a city made up by a small amount of letters. Therefore, by the time the list of suggestions will appear to the user, they will have already inputted half the query. Meaning they could potentially just type in the last three letters to complete the search without jumping state. However, whilst in the ‘type’ state, they can transition to the ‘look’ as this can be a valid option if the user is unsure of spelling or know there is a high likelihood that the desired destination will be on the list. Whilst in the ‘look’ state, the user now has they are presented with two options: ‘type’ and ‘pick’. By entering the ‘type’ state it confirms that the destination was not presented in the list of suggestions. Resulting in them returning to the initial state of typing a letter, where the process will begin again whether or not to enter the ‘look’ state for a second time or just type another letter. However, at any point the user is in the ‘look’ state (first time, second time, third time etc.), if there desired place is visible on the list of suggestions they will find themselves transitioning to the ‘pick’ state. Although it is not visible in Figure 9, once the user reaches this ‘pick’ state, the only next state possible to transition to will be the end state, ‘Search’.

When typing ‘Par’ into Booking, ‘Prague’ is there number one listed suggestion, so the order of states for this query would be: *Type 🡪 Type 🡪 Type 🡪 Look 🡪 Pick 🡪 Search*¸ unless the user remained in the ‘type’ state throughout or switched to the ‘look’ state at a later time.

### 4.1.1. State Diagram

The thought process model representing the user’s search, which is found in Figure 9, was then transformed into a visual representation of the system that could be used for all queries, not just specific to ‘Prague’. A state diagram is one of the five UML diagrams which are used to model dynamic nature of a system. The state diagram created for the reactive system being developed can be found in Figure 10.

******

***Figure 10: State Diagram for cost model.***

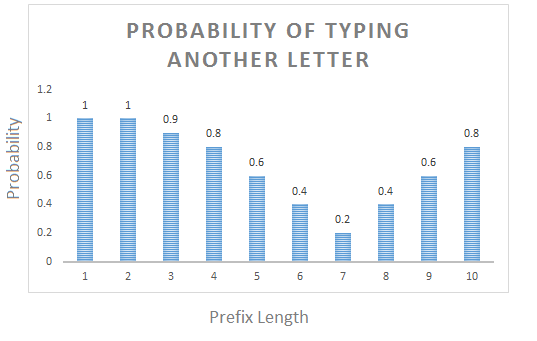
In Figure 10, the flow of control from one to state to the next is clearly demonstrated. The states can be shown clearly in blue circles, whilst the transitions can be seen in grey. The states are numbered in order of which state is most likely to be visited first, with exception to the transition from ‘type’ to ‘search’ state which could occur without the other states being visited at all. Normally in state machines, the first state is an idle one where the object waits on an event to trigger whether or not the object transitions into the next state. However, with the program being developed, the user must already be in the ‘type’ state for the state transition to proceed. Therefore, this represents the initial state, (State 1), whist the end state remains the ‘search’ state, (State 4). Figure 10 satisfies that the most important purpose of a state diagram is to model the transition of an object from creation to termination.

State diagrams are also used for forward and reverse engineering. By doing this a basic state model was created for the system. At this point, the user had to physically tell the system which state they wanted to transition to because there was no way of determining what state to transition to next.

### 4.1.2. Determining Probabilities

It is already known that the transition from one state to another, is dependent on its inputs from the original state. In the development of the cost model, it is a necessity for the state machine to know what state to enter next to be able to satisfy the user’s query. It is at this point that setting probabilities take form.

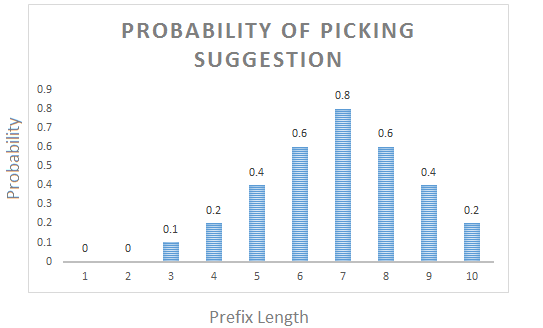
When the user finds themselves in either the ‘type’ state or the ‘look’ state they find themselves answering the question of whether or not to type another letter. In the ‘look’ state the answer is simple, if their chosen place is not part of the list of suggestions, then it makes it impossible to select it, so they must revert to the ‘type’ state. However, whilst in the ‘type’ state the choice of staying there or moving to the ‘look’ state are both feasible options. It is in this case that the probability for each option must be encoded into the state machine to show which state should be visited next. Figure 11 shows the probability of typing another letter, depending on the prefix length.



***Figure 11: Probability of Typing another Letter***

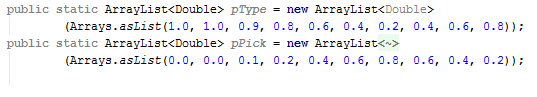
Since the list of suggestions does not appear until the users input consists of at least three letters, Figure 11, shows the accurate assumption that the user will definitely be obligated to type another letter. However, once the prefix becomes three letters long, the probability of the user typing another letter begins to slowly decrease. This happens because the greater the amount of letters that is inserted by the user, the higher chance that the suggestions list will display their desired place, allowing the user to select it. Nevertheless, Figure 11 shows that after the user has typed in seven letters of their desired place, then the likeliness of them typing another begins to increase fairly rapidly. This is due to high probability that if a user’s anticipated suggestion is not yet revealed then the user may get jaded and finish off the query without the aid of QAC.

The same principle works for when the user is in the ‘look’ state and they find themselves answering the question of whether or not to pick an option from the list. Ideally, if the anticipated destination is uncovered in the list of suggestions, then the chances of the user selecting it is extremely high. However, this may not always be the case. The user may accidently fail to spot their destination and begin typing again but the chances of this happening are slim. The probabilities that a user will pick from the suggestions are shown in Figure 12.



***Figure 12: Probability of Picking Suggestion***

The probability that the user will pick from the suggestions made, works in reverse of the probability that a user will type another letter. Figure 12, shows that whilst the prefix is less than three that it is impossible to pick a place from a list of suggestions which does not yet exist. Once the user has typed in their third letter then the chances of them picking begins to increase quickly. This is because the higher the prefix then the more likely that the looked-for place will become visible. After a seven-lettered prefix, the reason for the decrease in probability is the increase in typing probability that the user may just finish typing the destination without the aid of QAC. Logically, as the probability of picking decreased, then the probability of typing must increase, and vice-versa. The probabilities for picking and typing are then stored in ArrayLists within the program, which can be seen in Figure 13. Therefore, the prefix length can be used to find the position number in the ArrayList and the probability can be determined.



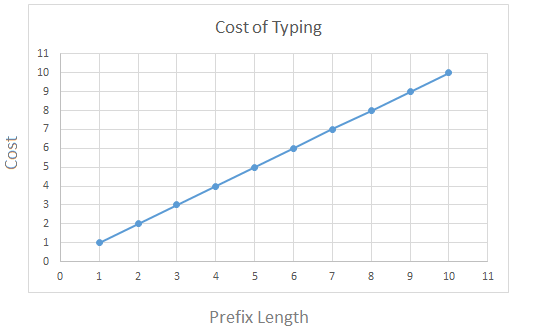
***Figure 13: Code Snippet of Pick and Type ArrayLists***

For instance, if a user was searching the destination ‘Edinburgh’ and, for this example, it is assumed they have already typed in ‘EDINBU’ and that they are currently in the ‘look’ state then by taking the length of the prefix, Figure 13, determines that the probability they will type is 0.4, and the probability they will pick is 0.6. Therefore, given that ‘Edinburgh’ has been suggested, then the user will pick. This feature was then integrated into the state machine, which now carried the capability of running a place from start to finish without any physical input from the user.

### 4.1.3. Determining Costs of Each State

Once the probabilities were determined and the state machine was in fully working order. The next protocol was to generate the cost associated with changing from one state to another. It is the sum of these costs that calculates the overall cost associated with each individual city/place in the model.

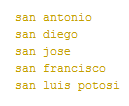
Whilst the user finds themselves in the ‘type’ state, a fixed cost will be placed in addition to the total cost of that time. This cost will never be changed throughout the simulation and the linear graph can be seen in Figure 14, where the typing speed was set to 1 second per letter.



***Figure 14: Cost of Typing***

Originally the cost associated with typing a letter was hard coded within the program at 1 second per letter typed. However, this was presumptuous by believing the speed of all human interaction to be the same. For instance, the time taken for a computer programmer who spends most of their day, with years of experience using a keyboard, will be vastly faster that an elderly person who may not have had much interaction with a keyboard at all. Therefore, it was at this point that the idea of creating the JavaApp took form. Regardless, it is plausible to run the simulation without determining their speed but then a typing speed of 1 second will be implemented, which could be far-fetched or an injustice to their actual speed.

Whilst the user finds themselves in the ‘look’ state, there are two possible costs that can be the outcome: the cost of looking at the list of suggesting whilst picking one, and the cost of looking at the list of suggestions without picking one. Determining how long a person takes to read a list of suggestions is a hard task to pursue. One way of determining this would be to give a user a sentence and record how long it takes them to read that sentence, then calculate an average speed for each word, or better, give a group of users the same sentence and then calculate an overall average. Instead, with some research, the average adult in the United Kingdom reads roughly 3 words per second [5]. However, all the adults this figure represents are educated. The query log stored is made up of places ranging between 1 and 5 words, with the majority being 1, we make the assumption it will take any user on average 1 second per suggestion presented. A list of possible suggestion can be seen in Figure 4.1.3



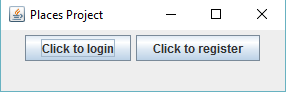
***Figure 15: List of Suggestions for prefix “san”.***

As previously stated, to read every suggestion in Figure 15 should take 5 seconds (1 second per suggestion), to complete. Pursuing this logic, if the user’s intended destination was “san antonio”, then it is assumed that they would not carry on reading the rest of the suggestions, and that they would just select their desired place. Therefore, the cost of looking, whilst picking, in this instance would be 1 second, because of the position it appears at in the list. Likewise, if the user was searching for “san francisco”, then the cost of looking, whilst picking, would be 4 seconds, and so on. The cost for the user to look at the list of completions and not select from it, is simply the average number of the amount of suggestions being shown. For instance, if 10 suggestions were shown, it would cost the user 5.5s to glance at the list before going on to type another letter.

The only state an object could be in that does not already have a cost assigned to it is the search cost, this is because this is being treated as an end/terminated state. Therefore, the object will stop here and simply calculate the different costs it’s gathered from the fellow states. Now the developed cost model had all necessary parameters to generate the total cost for each place visited.

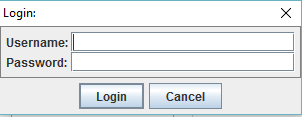
## Java Application

As the JavaApp was an extended aim, no unnecessary time was spent making a highly detailed user interface. Instead, a basic login system was created using Swing, which is a GUI widget toolkit for Java. The JavaApp consisted of a main interface called Login which called each new class if an event was triggered to say that the current class was succeeded. The initial look of the Login page can be seen in Figure 16.



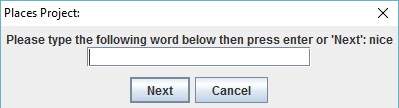
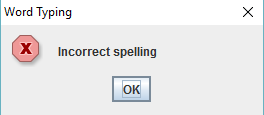
***Figure 16: GUI Login Home Page***

Figure 16, represents what can be described as the JavaApp home page. This is the first window that will appear to the user when they run the application. From here, a previously registered user can press the “Login” button to calculate a new typing speed, or a new user can sign up by pressing the “Register” button.



***Figure 17: Login Page GUI for JavaApp***

For either the “Login” button or the “Register” button, the next window the user will be presented with will look like the representation in Figure 17. However, this figure is specifically for the “Login” button. Here is where the user will enter their pre-registered username or password. At this point if the user has realised that their details are not in the system, then there is a “Cancel” button that will return them to the home page. Nevertheless, there is check on the system that ensures if the details are not already registered, an alert notifies the user giving them the option to register them. Likewise, on the Register GUI page, if a user enters already known credentials, this alerts the user and asks them if they would like to login with these details. If the user is able to log in successfully, then the next window is the Input class, which can be seen in Figure 18a.

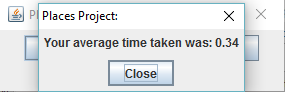


1. ***User Input of Several Places b) Error Handling Message***

***Figure 18: User Input GUI for JavaApp***

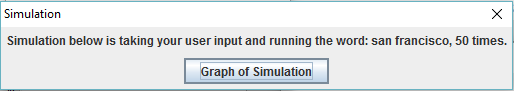
Figure 18a, represents where the user will be presented with a variety of different places, which they will type into the given input field. The dictionary used to generate random places was composed of the 125 most popular cities in the world. This happens because the original dictionary of all 3000 places was generating places that the user may not have heard of or knew how to spell. This resulted in inaccurate results for the average typing speed at the user was constantly having to double check the place for confirmation of the spelling. The amount of places presented to the user is one of the parameters that can be changed in the simulation, where the higher the amount, the more accurate the result.

A timer will start when the user presses their first key, with exception to the Enter key, which is a shortcut to submit their word and move onto the next awaiting word. So, pressing the enter key would cause an alert to tell the user they have not entered a place. Then again, when the first accepted key is pressed the timer will begin and will only finish when the last letter of the same word is typed. This time is then saved in an ArrayList and once all they words have been attempted, the average for each letter is then taken. If the user happens to make a spelling mistake whilst submitting their input, then an error message like the one in Figure 18b will be displayed. At this point the timer will be stopped and once the user presses “Ok” they will be able to have another attempt. This feature was made simpler by calculating the time of each word separately, meaning the user could have as many attempts on the one word without hindering the results from the previous words. Once all the words were completed, the calculated average for each letter was displayed to the user. This can be seen in Figure 19.

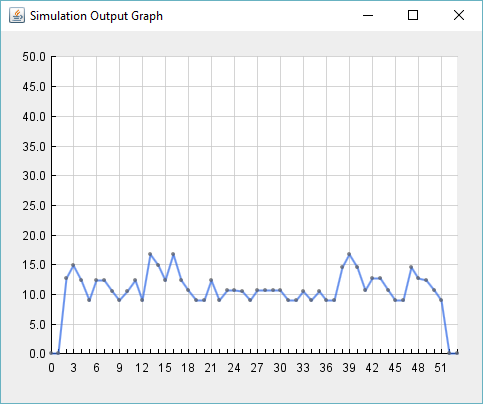


***Figure 19: Pop-up of Average Letter Type***

The pop-up window in Figure 19, shows the recently generated average time to type a single letter for that specific user. At this point their user information will be updated to include this new result. However, a drawback is that if the users previous personal time was faster, it will be automatically overwritten at this point, which may not reflect an accurate result. This could be easily adapted in the code but was found to be as it is expected that most users using the JavaApp will be newcomers. Once the user presses the “Close” button, the average time to type a letter will be taken and run in the simulation to find their personal cost of searching a specific place. The simulation will continue to run in the console of the development environment, but the results will be outputted in a graph made for the purpose of the JavaApp. This can be seen in Figure 20.



1. ***Option for user to generate the graph.***

******

1. ***Graph for running ‘san francisco’ 50 times.***

***Figure 20: Graph of Simulation***

In Figure 20a, we can see that the user is made aware of what place is being simulated and by how many times. Then, they have the option to generate the graph which can be seen in Figure 20b, this was made an option to represent a visual representation of the simulation as leaving the user jumping between the JavaApp and the console is not very user friendly. For a user wanting to see how each individual place was found and the cost for each, this JavaApp is inadequate. Nevertheless, for the purpose this JavaApp was made, finding the users average letter speed, it works perfectly and is simple enough to navigate.

# 5. Testing Strategies

Testing a simulation based greatly on probability can sustain a difficult task, as there is an array of acceptable outcomes. Therefore, there can be an expected output through the assumptions made, through the parameters inputted.

Once the initial state transition model was created, it allowed basic testing with a text file containing 125 places to be tested. Whilst output was expected, the amount of places did not deem sufficient to test a simulation of its complexity. After the user had completed their prefix of three, the desired destination was often top of the suggestions list, causing the user to automatically pick if they had glanced the suggestions. Where this is ideal, realistically it would not happen unless it was the top of the popularity ranking. Therefore, after the addition of 3000 places, results became far more accurate and realistic. Once the JavaApp was created, it allowed the state transition functionality to be tested with a changing input using Full System testing. Sub-System testing was then used as a means of providing a deeper assurance that each segment of the system was working as expected. Finally, Parametric Simulation would be carried to ensure each parameter was in full working order, and they were not getting altered throughout the simulation. The cost model was also run on different processors, to analyse if the results differed and the time taken to compute them was compared.

The overall system was tested using a database of 300,000 places. However, in order to operate in an adequate amount of time, all the places could not be loaded into the system at the beginning of the simulation. Instead, the first letter of the desired destination was taken and all the places beginning with that same letter were then fed into the simulation, where it where it proceeded as normal.

## 5.1. Full System Testing

The Full System tests that were performed evolved the more the project began to progress. During the final stages of the project, several tests would be carried out if any part of the system was altered. The tests were to ensure that any changes made to a section had not affected any other areas of the program. The tests carried out after each alteration were:

1. Generate the cost with user just typing letters.
2. Test a String (place), so ranker will find test String in list.
3. Never picking a suggestion, just look and type until end.

A further look into the Full System tests and the results they produced can be found in Appendix 5 Test Results.

## Unit Testing

To confirm that each individual component worked as expected, Unit testing was employed. To perform the Unit testing, the Junit framework was implemented. The Junit framework operates by creating instances of the class that is being tested, where methods can be called on the newly formed class and the results are tested to confirm they meet expectations. The routinely ran tests, which return either pass or fail, are a confirmation that no changes in code cause the functionality to break. However, to create the instances necessary to test the program, other classes need to be made available. Thus, if one of the available classes does not work as expected, then it can cause the test to fail although the class being tested functions correctly.

## Sub-System Testing

Sub-System Testing is fairly similar to unit testing. The main difference is that with Sub-System Testing, instead of test being performed on one class, it is performed on a cluster of classes that are tested simultaneously. The Sub-System that was tested within this project was the cost model on its own, followed by the JavaApp, to show that both entities could operate independently. The cost model was also split into separate sections, the state transition and ranking system, where Sub-Testing was also carried out.

## Parametric Simulation

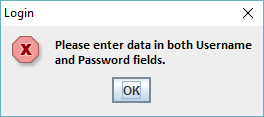
Parametric simulation is used to improve the overall performance of a system. This method will change a single variable being inputted to the system and hold the others constant and as a result, the effect on the design will be closely monitored. Depending on the scale of the project this process can be time-consuming whilst partially improving the performance. The Parametric Simulation carried out within the project were:

1. Changing the cost of typing to see there is a continuous increase/decrease in total cost.
2. Changing when the user was able to see the list of suggestions. For example, after typing the first letter and after typing the third. Therefore, showing if users were able to select their destination fast, the difference this will make on the total cost.

A deeper look into the results of these tests can be found in the Performance Analysis and Evaluation section of the report.

## 5.5. Black Box Testing

Black-box testing is a method of software testing that takes the functionality and examines it without peering into its internal structures or workings. Black Box Testing was used when developing the JavaApp for alerting the user when their intended action did not meet the criteria needed to carry it out. For example, if the user attempted to register or login with a blank username or password. Then an alert would pop up saying that there has been no data entered. An example of one of these alerts can be seen in Figure 21.



***Figure 21: Example of Black Box Testing***

## 5.6. Running Cost Model over Different Processors

We know that the total cost for the model is the sum of all places and that the need for sampling is crucial in a simulation this vast. This theorem is proved when the simulation was executed over two very different processors; an AMD A10 quad core processor @ 2Hz and a faster, more efficient i7-6700 CPU @ 4GHz. The results for running the cost model were recorded and can be seen in Table 2.

|  |  |  |
| --- | --- | --- |
| Type of Processor | AMD A10 Quad Core @ 2GHz | i7-6700 CPU @ 4GHz |
| Mean (s) | 17.3 | 17.28 |
| Standard Deviation (s) | 0.28 | 0.34 |
| Total Cost (s) | 8,591.25 | 8,637.93 |
| Time Taken to Execute (s) | 7,529.54 | 1,513.44 |

***Table 2. : Running the cost model over different processors.  
  
a) Running 500 places, 500 times.***

***b) Running 1000 places, 1000 times.***

|  |  |  |
| --- | --- | --- |
| Type of Processor | AMD A10 Quad Core @ 2GHz | i7-6700 CPU @ 4GHz |
| Mean (s) | 17.75 | 17.48 |
| Standard Deviation (s) | 0.42 | 0.26 |
| Total Cost (s) | 17,754.25 | 17,988.73 |
| Time Taken to Execute (s) | 64,527.08 | 13,005.41 |

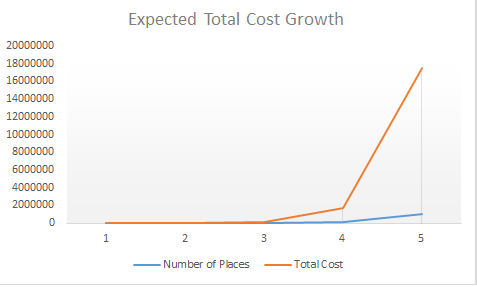
As can be seen in Table 2a, when the same 500 randomly generated places, each place being simulated 500 times, were executed on each processor they produced similar means, standard deviations and total costs. This was to be expected. The real shock was how dramatically the time to execute, on each of them, differed. These results stay consistent when the cost model executed another randomly generated 1000 places, with each place being simulated 1000 times, onto each processor. These results can be seen in Table 2b. It was expected that the i7-6700 processor would produce quicker results but operating just under 5 times quicker, truly shows how important the need for sampling unless being executed in a data centre.

For instance, the cost model was developed and tested on the slower AMD A10 processor, therefore, running the 1000 places took just under eighteen hours to generate a total cost. This time-consuming task made it difficult to test the boundaries of what the highest number of places could be executed. Therefore, by sampling the data already gathered, the cost model was executed 100 of the previous 1000 places, which were each simulated 1000 times. The results are shown in Table 3.

***Table 3: Running the cost model 1000 times for 100 places.***

|  |  |
| --- | --- |
| Type of Processor | AMD A10 Quad Core @ 2GHz |
| Mean (s) | 17.74 |
| Standard Deviation (s) | 0.87 |
| Total Cost (s) | 1,746.51 |
| Time Taken to Execute (s) | 638.8 |

As can be seen in Table 3, the mean and standard deviation for 100 places is similar to the mean and standard deviation for 1000 places, although the simulation only took ten minutes to execute. Therefore, we can make an assumption that for running 10,000 places the mean and standard deviation should also stay similar. It is through this assumption that we can predict the total cost for 10,000 places being simulated 1000 time each to be roughly 175,000s. The expected total cost growth for up to 1 million places can be seen in Figure 22.

***Figure 22: Expected total cost growth for 1 million places***

# Performance Analysis and Evaluation

This section of the report focuses on taking the cost model developed and logging the different effects that changing the parameters, which formulate the simulation, have on the cost model. There are four main parameters that make up the simulation; ranking of suggestions, when suggestions are shown to the user, the amount of suggestions shown and the typing cost. Each of these factors will be evaluated and the sum of all places is generated. It can be assumed that unless the parameter is being evaluated, the following figures are constant:

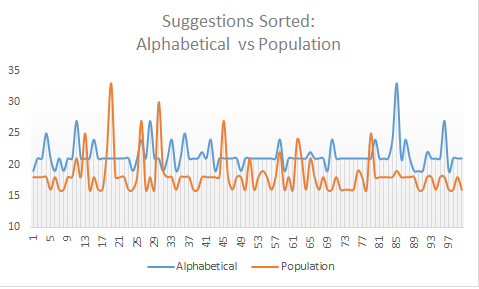
*Ranking of Suggestions: Population  
Instance of Suggestions to User: After prefix of 3*

*Amount of Suggestions Displayed: 5*

*Typing Cost: 2s*

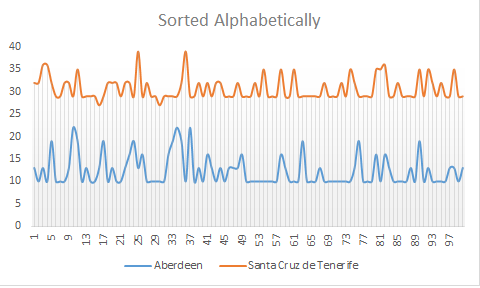
## 6.1. Ranking of Suggestions

As previously stated numerous times, the ranking of suggestions is one of most important factors in QAC. How accurate the ranking is in reference to the anticipated query, plays a major role on how quickly the query can be suggested and therefore, completed. For instance, if the ranking of suggestions was set to alphabetical, and the users prefix was ‘Saint‘, then ‘Saint Denis’ would appear before ‘Saint Petersburg’. This ranking would be ideal if the user intended query was in fact ‘Saint Denis’, however, it was not. Therefore, by changing the ranking to population instead, the intended ‘Saint Petersburg’, would be found in a shorter amount of time. Figure 23, show’s a representation of how these rankings affected the total cost over the city ‘San Francisco’.



***Figure 23: Alphabetical Vs Population***

Figure 23, represents the city ‘San Francisco’ being simulated 100 times sorting alphabetically, and 100 times sorting by population. Where the total cost of each simulation is plotted on the graph, the blue line represents alphabetical, and the orange line shows the population simulation. As can be seen, the alphabetical line seems to have a higher cost associated, with an estimated cost of roughly 21s, whereas, by population, most results seem to be found at roughly 18s. These results were completely as expected, as within the dictionary of places, if the prefix entered was ‘San’, ‘San Francisco’ was 14th in the dictionary alphabetically, whereas it as the 4th by population. Therefore, even after a prefix of only ‘San’, the destination could potentially be selected for population. However, alphabetically the user would have to type more letters before it made the top five suggestions, therefore, increasing the cost. This difference of 3s is expected to increase if a wider range of places were being held in the dictionary. The population, orange line, should stay roughly the same, whereas the alphabetical blue line, would increase with the more potential places. Figure 24, also shows this concept by running a place ‘Aberdeen’ and ‘Santa Cruz de Tenerife’, alphabetically sorted.

******

***Figure 24: ‘Aberdeen’ Vs ‘Santa Cruz de Tenerife, Alphabetically sorted.***

Figure 24, shows how ‘Aberdeen’ is being found at roughly half the cost of ‘Santa Cruz de Tenerife”. Again, this happens because of the sheer lack of places with the prefix ‘Abe’ and the vast about that carry ‘San’. Therefore, when user had typed ‘Abe’, if they had looked at the list of suggestions, it would be their guaranteed top choice.

Having the ability to change how the user would like to have the suggestions ranked played a vital role at keeping the cost to minimum. If the user knew they were searching a well-known, highly populated, destination (which most of the time they are) then the obvious choice is population. By choosing an individual city to evaluate, certain places are guaranteed to output expected results. Therefore, by running the simulation over a selection of randomly generated places, removes bias choices and gives a realistic result. Moreover, these simulation tests carried out resulted in finding the minimum cost whilst the sorting of suggestions was by population overall. This can be seen in Table 4, where 10-100 places were executed and the mean cost for each was found.

***Table 4: Alphabetically Ranked Vs Population Ranked, Overall Mean***

|  |  |  |
| --- | --- | --- |
| **No of places, simulated 100 times.** | **Mean Cost (Alphabetical Ranking)** | **Mean Cost (Population Ranking)** |
| **10** | ***19.73*** | ***17.55*** |
| **20** | ***18.09*** | ***17.89*** |
| **50** | ***18.90*** | ***17.43*** |
| **100** | ***19.28*** | ***17.74*** |

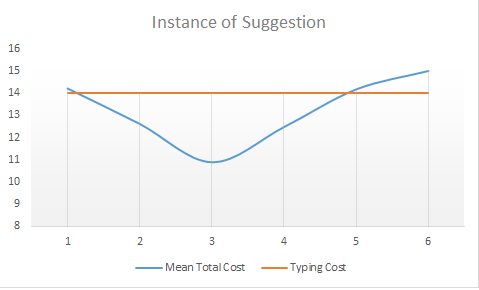
As can be seen in Table 4, when a random selection of places are simulated, ranking the suggestions by population gives a consistently lower result. Therefore, to keep cost to a minimum this is the most-efficient way of sorting and will be utilised throughout the rest of the project. Ideally, the most efficient way to rank the list of suggestions would be the popularity. If there was access to this form of dataset then the cost model created would be much more accurate as it would be expected to reduce the cost by finding the user’s query much quicker.

## 6.2. Instance of Suggestions Displayed

To date, it has been a user requirement to have a prefix of at least three letters before the list of retrieved suggestions is displayed to the user. However, how can it be determined that this is the most cost-efficient option for the user? Table 6.2, shows five different simulations for the city ‘Jackson’, each of these simulations are executed 100 times to find the mean cost. Each simulation will be identical, except for the first simulation, the list of retrieved suggestions will be generated after a prefix of 1 letter. Then in the second simulation, the list of completions will appear after a prefix of 2, and so on. A visual representation for Table 5, can also be seen in Figure 25.

***Table 5: ‘Jackson’, total cost at instance of suggestions.***

|  |  |
| --- | --- |
| **Instance of Suggestion: Prefix Length** | **Mean Total Cost** |
| **1** | **14.22** |
| **2** | **12.62** |
| **3** | **10.89** |
| **4** | **12.48** |
| **5** | **14.18** |
| **6** | **15.01** |



***Figure 25: Suggestion Prefix Length***

In Figure 25, the average cost for each simulation is represented by the blue line whereas, the orange line represents the cost ‘Jackson’ would consume if the user only typed (7 letters at 2s per letter). It can be seen in Figure 25, that it not beneficial for the user to display the list of suggestions until prefix of at least two letters is inputted, with the optimal time being after a prefix of 3. After the 4th letter, the suggestions become irrelevant in this case, as the total cost taken is greater than simply typing each individual letter, therefore, rendering QAC unnecessary.

As can be seen in Table 5, displaying the suggestions after a prefix length of three letters gives the lowest total cost for ‘Jackson’. Therefore, to keep cost to a minimum this is the most-efficient way of displaying the completions and will therefore, be utilised throughout the rest of the project.

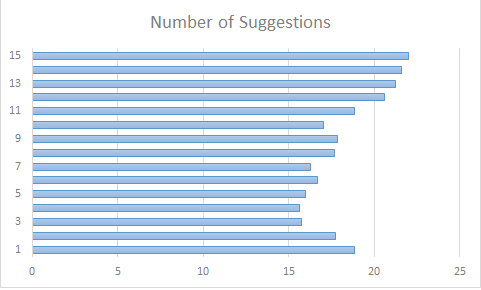
## 6.3. Amount of Suggestions Displayed

Whilst using any search bar that employs QAC functionality, it can be seen that the amount of suggested queries is minimised to between roughly four and eight. This section of the report discusses how these figures are calculated and at what amount of suggestions keeps the cost model to a minimum throughout this specific project.

To evaluate this theory, it would be bias to execute only one place into the simulation. As this specific destination may be the top populated place for their three letter prefix, causing this place to be the top ranked on a list of 1 suggestion, or 20 suggestions. Therefore, by generating 100 randomly selected places throughout the world and simulating them 100 times each to get an average cost for each place, will show a true reflection of the importance the number of suggested queries carries. This is shown in Table 6, with a visual representation shown in Figure 26.

***Table 6: Number of suggestions over 100 randomly selected places.***

|  |  |
| --- | --- |
| **Number of Suggestions** | **Mean Total Cost** |
| **1** | **18.82** |
| **2** | **17.72** |
| **3** | **15.73** |
| **4** | **15.62** |
| **5** | **15.99** |
| **6** | **16.65** |
| **7** | **16.27** |
| **8** | **17.66** |
| **9** | **17.87** |
| **10** | **17.03** |
| **11** | **18.83** |
| **12** | **20.59** |
| **13** | **21.26** |
| **14** | **21.62** |
| **15** | **22.01** |



***Figure 6: Number of Suggestions Over 100 Places***

In Table 6, and Figure 6, the total cost for displaying only one suggestion was expected to be much higher, however, with closer inspection this figure is completely accurate. This is because the cost of looking at the list without picking from the list of suggestions is the average of the amount suggested. So, in this case for 1 suggestion, the looking cost would be 0.5s. This cost of looking without picking can be expected frequently when the amount of suggestion is smaller as the QAC wants to kick in and help, it just does not have the resources to do so. Also, with this cost being so miniscule, the amount was not noticed as much as if there was say, six suggestions, which would add 3.5s each time they looked without selecting. Where the higher cost for only displaying one suggestion comes from is when the user finds themselves in a state of looking without picking, as they are forced back into typing another letter. Thereby, costing another 2s each time until their intended query becomes visible.

The cost for looking at the list of suggestions, with or without picking, will begin to cancel out the typing cost. As with more available suggestions, the time spent on typing letters will be redirected into looking at the list. This can be seen in Figure 6, where the amount of suggestions becomes higher than eleven, the total cost begins to increase to a maximum. Thereby, with only a select amount of letters typed, they user will have to scroll down the list where they may be successful and pick, which will cost 1s for every suggestion supplied until that point. Otherwise, it will cost an average of the amount of shown suggestions, where they will then have to type another letter in hope their query becomes visible in the next batch of completions.

The reason for choosing five suggestions to be presented to the user was determined by the fact Booking displayed this amount, and their process was to be replicated to give the user the same interaction. However, Figure 6, shows clearly that the most cost efficient way was displaying three or four suggestions, like Google, therefore, this could be integrated into the system to reduce costs. Companies that use QAC and list a high number of completions are normally retail companies. By listing several options to the user, it is seen as a form of window shopping, where the user may not see their intended query, but instead find themselves browsing through products the company has to offer.

## 6.4. Typing Cost

To show the impact that the cost of typing a letter has on the cost model, the speed at which they are pressing each key has to be evaluated. Therefore, if we take a user who is fast at typing, say 0.2s per letter, and a user who is slower at typing, say 2s per later, we can assess the effect this has on the total cost. Figure 27, shows the box plot of both users searching ‘San Francisco’ 100 times.

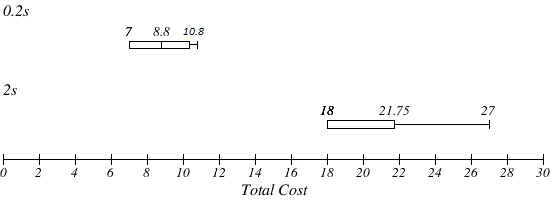
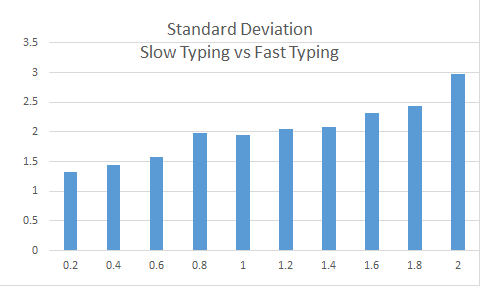
***Figure 27: Box Plot of Typing Cost Distribution***

Figure 27, shows clearly the expected result that the slower the typing speed then the higher cost to the simulation by almost double the cost. By showing this effect on a box plot, the distribution can be clearly seen. It is to be expected that the higher the cost of typing, then the greater distribution of values could be seen. The standard deviation for each speed between the slow and fast letter typing speed can be seen in Figure 28.



***Figure 28: Standard Deviation, Fast Typing vs Slow Typing***

A high standard deviation shows that the data from the simulation is widely spread and therefore, less reliable. Whereas, a low standard deviation shows that the data are clustered closely around the mean, therefore making it more dependable. As can be seen in Figure 28, the longer the user takes to type each individual letter, the more diverse the results are going to be. Therefore, the faster the typing speed then it not only produces a smaller cost but also a more normal distribution resulting in more accurate results.

## 6.5. Typing Cost vs Looking Cost

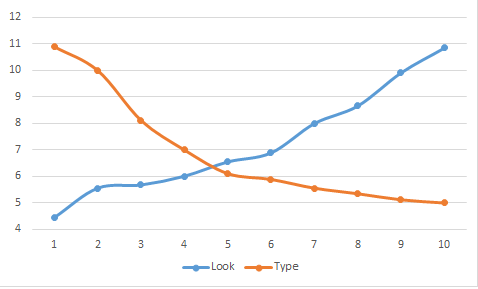
As the amount of suggestions increase, it has been proven that the cost of looking will increase. This is because the larger amount of suggestions available to the user will create a larger cost to scroll down before selecting their intended query. Furthermore, the cost will also be greater if they find themselves at the end of the list without their query, than it would be if the amount of suggestions was less. From this proven theory, it can be determined that the cost of typing will decrease as the cost of looking increases. This is determined by the logic that if they are looking, then they are not typing. Table 7, shows the data for a simulation executing 100 places, 100 times which reduces/increase the typing and looking cost, respectively, by the same amount each time the amount of suggestions increases and Figure 29 shows the results.

***Table 7: Looking Cost Vs Typing Cost***

***a) Typing cost against no. of Suggestions b) Looking cost vs no. of suggestions***

|  |  |  |
| --- | --- | --- |
| **Number of Suggestions** | **Typing Cost** | **Mean Total Cost** |
| **1** | **2.0** | **10.89** |
| **2** | **1.8** | **9.98** |
| **3** | **1.6** | **8.11** |
| **4** | **1.4** | **6.99** |
| **5** | **1.2** | **6.12** |
| **6** | **1.0** | **5.88** |
| **7** | **0.8** | **5.54** |
| **8** | **0.6** | **5.34** |
| **9** | **0.4** | **5.12** |
| **10** | **0.2** | **4.97** |

|  |  |  |
| --- | --- | --- |
| **Number of Suggestions** | **Looking Cost** | **Mean Total Cost** |
| **1** | **0.2** | **4.48** |
| **2** | **0.4** | **5.59** |
| **3** | **0.6** | **5.67** |
| **4** | **0.8** | **6.12** |
| **5** | **1.0** | **6.54** |
| **6** | **1.2** | **6.88** |
| **7** | **1.4** | **7.98** |
| **8** | **1.6** | **8.45** |
| **9** | **1.8** | **9.78** |
| **10** | **2.0** | **10.11** |

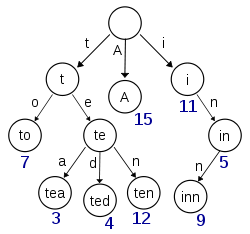


***Figure 29: The effect the amount of suggestion has on the look and type cost***

# Future Work

Although the current structure of the system will generate a cost model for finding places that would satisfy the user, there are certain features which could be implemented into the system to improve the finished product.

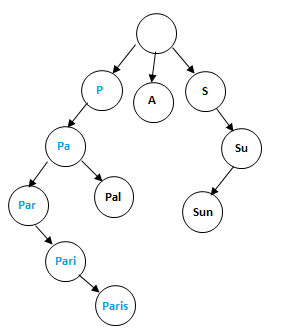
## 7.1. Trie Structure

 A search tree is an ordered tree data structure used to store dynamic sets or associative arrays, where the key is normally a String [6]. In computer science, a trie, which is also known as a prefix tree, is a form of these search trees. On the contrary to a binary search tree, it is the position in the tree that defines the key to which it associates, instead of the node in the tree storing the key. An example of a Trie structure can be found in Figure 30.

***Figure 30: Trie Structure [12].***

As can be seen in Figure 30, the successor of each node share a mutual prefix of the String associated with that node, where the root itself associates with an empty string. Values are not assigned to each node they are instead associated with the resulting leaves. For example, in Figure 30 the keys are listed within the nodes and the values are presented below them. Therefore, each English word has a subjective integer value to which it associates. Using this trie structure for the implementation of the project, would exploit the trie's ability to quickly search for, insert, and delete entries. However, since storing words is the only necessary requirement, then a minimal deterministic acyclic finite state automaton (DAFSA) could be implemented instead, as this would use less space than a trie. The reason for this is that a DAFSA can condense branches that are the same from the trie which correspond to the same suffixes of different words being stored [6].

The main reason the implementation of a trie structure would be greatly beneficial for the simulation developed, is the amount of time it would take to simulate each city/place would dramatically decrease. This would allow the user to reach their destination in a well-structured, orderly fashion. Currently, as stands, the program takes in a list of places/cites and stores it in an ArrayList. Once the first letter is typed, the program then runs through each String in that ArrayList that begins with the same prefix, whilst extracting them into another ArrayList of suggestions. For example, if the user types “PAR”, the program will remove each String beginning “PAR” and store it elsewhere to soon be displayed to the user. With each additional input from the user the list of suggestions will not be stored and the program will return to the beginning by clearing the ArrayList of suggestions. Therefore, when the prefix becomes “PARI” it will search the original ArrayList of places and extract all the places beginning with this new prefix. This backwards notion of clearing and starting again would continue until the full destination of “PARIS” was resulted. If the program was adjusted to implement a trie structure, the same search can be seen in Figure 31.



***Figure 31: Trie Structure for “PARIS”***

## Developing a Native Application

In today’s society, an obvious advancement for any project would be the ability for software to work on a smartphone or tablet. The ability to use the simulation as a mobile application, offers great flexibility for the user. The development of an app allows the user’s location to be determined, resulting in discovering the places that surround them if they find themselves in an unknown area. Therefore a study into what type of application to develop was utilised and how this location feature would operate. Adding a location service, would be a difficult but worthwhile addition for the end user. These location services would calculate the user’s current location, and constantly update the nearer a user got to their desired location.

### Android vs iOS

When developing a Native Applications, the two choices to be considered are: Android and iOS. Table 8 shows the comparison of these platform and any possible issues that might hinder the project.

***Table 8: Android vs iOS***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Issue | iOS | Android | Choosen platform | Reason |
| % of users worldwide | 17.7%[7] | 80.7%[7] | Android | Android offers a larger market. |
| Integration with mapping services | Supports many mapping services including Google and Bing [8] | Supports many mapping services including Google and Bing [8] | Undecided | Undecided as both offer the same mapping facilities. Therefore, both would be feasible. |

As Table 8 shows, since both Android and iOS have very similar mapping services, then the sensible decision would be to choose Android as they offer the largest market place of the application. Also, since the original cost model was developed in Java, this would be easily transferrable. If iOS was chosen, the cost model would have to be converted to a compiled programming language that was developed for Apple, e.g. Swift, that could prove time consuming to learn a new programming language.

### 7.2.2. Location Services

The necessity for a location service is unnecessary for a company like Booking, as their popularity ranking will be constantly updated to produce the most popular places and the likelihood of the user wanting to visit the city closest to them is fairly unlikely. For example, if the location is “Glasgow” and the user types “GRE” into the search box, then they would expect “Greece” and the “Greek Islands” to be suggested before “Greenock”, despite the distance between them. Therefore, the only time a location service would be necessary would be if the user was still located in “Glasgow” and they were searching the destination “Newcastle”. At this point the simulation would understand the likelihood that they meant “Newcastle, England” instead of “Newcastle, Australia”. This however, does not require an accurate location to be taken.

However, where there is great need for a location service application is when a user is searching for a place with a map service. Whilst in a precisely located destinations, a user may want to find the nearest cinema. At this point, ranking means nothing. The user would simply want to know the nearest cinemas location and how exactly to get there. In order to implement this feature into an app, it must be decided what location service is most feasible. There are three main technologies that are used to pinpoint a user’s locations: Global Navigation Satellite System, Cellular Positioning and Wi-Fi.

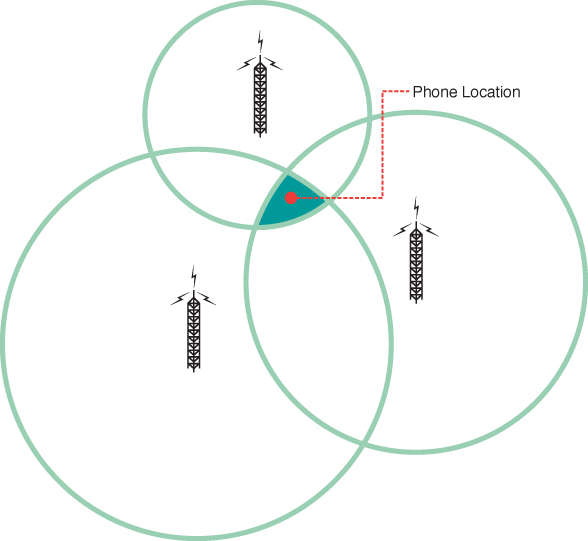
#### 7.2.2.1 Global Navigation Satellite System (GNSS)

The most well-known GNSS is the Global Positioning Service (GPS), which is funded by the United States, but there are several currently in operation. The Russian Federation have GLONASS, and Europe is soon to have Galileo (which they plan to have in operation by 2020). The main aim for a GNSS is to allow a user to determine their location. The American GPS system is made up of twenty-four satellites, which continuously generate signals. These signals inform the user of the location of the satellite and they deliver precise information about when the signal was generated. The user’s location is predetermined by listening to the signals produced by four or more satellites, and determining their exact location. The result of the time taken to receive the signal and the distance from each of the satellites shared with location of the satellite is what determines the user’s locations [9].

The accuracy of GPS is approximately 3.5m [10], which although for the purpose of this project would suffice, a limitation would be the GPS receiver on a mobile device. A mobile phone’s GPS receiver can range from 2 - 10 meters, depending on the multipath conditions [11]. Multipath is when the signal takes a variety of paths to reach the receiver, this could be due to the reflection on buildings. Therefore, since the cost model is likely to be used in the surroundings of an urban landscape, the GNSS location will be of lower accuracy than desired. With the signal being low outside, mixed with the background noise of an indoor environment, the signal could be as low as 10-100000 times weaker when indoors [12]. This signal to noise ratio, may cause severe problems when trying to gather the information from the four different satellites and without the data from all the satellites, calculating the location whilst indoors could prove very difficult. To add in a location service would be a challenging task, for the user to only be guaranteed an accurate result whilst outdoors, therefore GPS is not a logical fit for the future development of the cost model.

#### 7.2.2.2 Cell Tower Based Location

Determining a user’s location using cell towers is similar to the technique used in GNSS. The mobile devices sends a signal to the nearest cluster of cell towers and it will determine how long to get a response. Then by taking the location of the cell towers, the phones location can be generated using the distances for each individual cell tower. The workings of this can be seen in Figure 32.



***Figure 32: An Example of Cell Towers***

It is known that the user must be within the circle with the response time they are receiving and the greatest distance they could be from the cell tower is a straight line, aka the radius of the circle. Although, there is no confirmation that the signal is following a straight path, as similar to GPS, it may be being reflected off buildings. Taking these precautions ensures the user is definitely at some point in the circle. Using more than one cell tower allows the user’s location to be accurately calculated, which can be seen in Figure 32, where the circles are overlapping one another.

As long as the user maintained a mobile phone signal throughout, then the location could still be calculated whilst indoors. Additionally with the increase of Cell Towers in urbanised areas, Cell Tower location could be a viable method for the project. However, a major flaw is that cell tower locations are not made public knowledge, causing the use of Cell Tower Triangulation problematic [12]. Therefore, if location of the cell tower is unknown then the current location of the user cannot be found.

#### 7.2.2.3 Wi-Fi Based Location

Similarly, the user’s location can be discovered, through Wi-Fi based stations. The distance these stations are from the user can be easily determined thought criteria like the strength of the signal. If numerous base stations are received and their location is identified, then the user’s location can be computed by triangulation. Therefore, this would seem the most desired location service for the project. The only drawback for a user would be if there was no Wi-Fi connection made available, but nowadays, especially in urbanised areas, this is very unlikely.

# 8. Future Applications

Creating a cost model for a concept as universally employed as QAC, allows the cost model developed to be integrated into various different entities that provide a search bar whilst in its current condition, provided the additional data can be obtained. The list or possible future applications expand if some of the progress alluded to in the future work section is completed.

## 8.1. Without Extension

This cost model would be ideal for a digital music service that has access to millions of songs, like Spotify[[7]](#footnote-7). The cost for searching through various artists and their songs could be generated by finding the cost of the individual artist or song using the same process outlined throughout the report. However, the suggestions could then be ranked by popularity, as this could be determined by the amount of online streams associated with that artist or song. Therefore, the more online streams, the higher the popularity ranking. As Spotify has a feature that allows the user to follow their favourite artists, these artists could then be checked first to see if they have a related song that fits the prefix before searching the rest of the song database. Furthermore, the songs on Spotify that have had a dramatic increase in streams within a specific time frame (e.g. a day, week etc.), could be stored separately and also search alongside their favourite artists. These three categories will present the user with a combination of personal favourites, high trending and classic query completions.

Spotify is only one of many examples. The cost model created is idyllic for a search bar which sorts through a large database of data. Further examples could be searching the names or street names of a countries population, searching through different animal species, or even searching significant historical events throughout the world.

## 8.2. With Extension

With the additional feature of a location service, this allows the user to search through data nearby their current location. For instance, whilst on the mapping function of an Apple Mac computer, if the user searches for a place, say ‘Perth’, then currently it is assumed the user meant ‘Perth’ in Australia. However, in reality they meant ‘Perth’ in Scotland, which is nearest to them. To get the desired destination the prefix entered would be ‘Perth S’, causing more time and effort for the user. With this additional extension, the cost model would be perfect for searching a database of food establishments. By generating the queries within a chosen square mile radius, this solution improves user’s satisfaction and accuracy by removing the improbable, naive options.

# 9. Reflection and Summary

A cost model capable of showing how a user interacts with a search bar operating a query autocomplete functionality has been developed throughout the project. The cost model has been proven to be able to handle this task in an efficient and strategic way. This section of the report will reflect on the project as a whole and summaries key aims and deliverables achieved throughout.

## 9.1. Overall Reflection of the Project

The development of the QAC cost model has been generally wholesome. The majority of different aspects were implemented without many significant problems. As expected there was sections in building the state machine that were challenging to incorporate and to get working correctly. Having sufficient time to resolve any errors or complications that arose, allowed the cost model to be developed to a higher standard than having to be developed last minute. Writing chapters of the report at each section of the model would have saved crucial time in the weeks leading up to the deadline, however, not drifting from the original project plan and creating buffers helped the overall progress of the report.

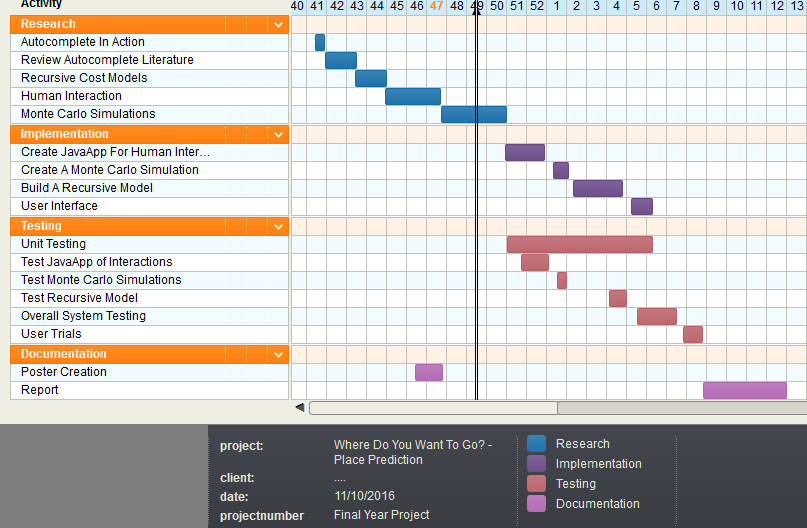
Features that were not implemented within the current version of the cost model could be integrated at a later date. Formulating the query suggestions using a trie structure, would decrease the cost as queries are found much faster and require less storage than the approach taken in the project. Implementing the cost model into Android devices in the form on an app, will vastly improve the scope of the cost model’s capabilities in terms of features and functionality.

## 9.2. Conclusion

The development of a cost model for QAC in information retrieval to which all possible outcomes are explored for a given query, was constructed throughout the project. The approach is evaluated on a large dataset, which gives a genuine result to the nature of the problem. The importance that different parameters play on the algorithm were illustrated, such as the ranking of suggestions, the instance they are generated and the amount displayed by the user interface. The effects of these parameters were investigated in the evaluation section of the report, satisfying the aims of the project whilst keeping the total cost of the model to a minimum.

# Appendices

## Appendix A: Original Project Plan

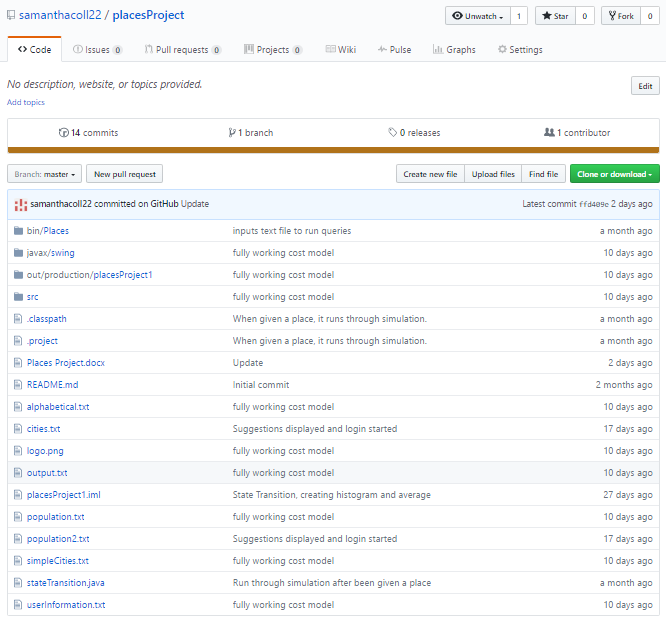


This Gantt Chart was created using www.tomsplanner.com

## Appendix B: Final Project Plan

This Gantt Chart was created using www.tomsplanner.com

## Appendix C: GitHub Evidence

The GitHub repository [www.github.com/samanthacoll22/placesProject](http://www.github.com/samanthacoll22/placesProject), where the full project can be found.

## Appendix D: Test Results

### D.1. Full System Testing Results

The following table shows the Full System test cases that were carried out on the cost model each time there were changes to the code. The steps taken to create these tests are outlined along with the expected results and actual results.

#### D.2.1 Checking Typing Cost.

|  |  |
| --- | --- |
| Description | * To generate the total cost of a system if the user remains in the ‘type’ state and doesn’t invoke QAC. |
| Preconditions | * User is searching for ‘Paris’. * Cost of typing is 2s. |
| Test Steps | * Set the state machine to remain in the typing state until at least a prefix of five. |
| Expected Result | * 10s. |
| Actual Result | * Result as Expected |

#### D.2.1 Checking System Locates Correct Place.

|  |  |
| --- | --- |
| Description | * To ensure the cost model if locating the correct string within the list of retrieved suggestions. |
| Preconditions | * User is searching for ‘San Francisco’. * Simulation is fed in text file consisting of: San antonio   San Diego  San Jose  San Francisco  San Luis Potosi   * Suggestions are shown after a prefix of 4. |
| Test Steps | * Once suggestions are shown, output position number of intended query. |
| Expected Result | * 4 |
| Actual Result | * Result as Expected |

#### D.2.1 Checking Looking Without Picking.

|  |  |
| --- | --- |
| Description | * Never pick from list of suggestions, just always type and look until the end of the query. |
| Preconditions | * User is searching for ‘Prague’. * Cost of typing is 2s. * The amount of suggestions shown are 5, as the average of this is the looking at the list but not picking cost. Therefore, looking cost is 3s. * The user still type’s three letters before suggestions appear. |
| Test Steps | * Set next state to ‘type’ state from ‘look’ state. |
| Expected Result | * 21s |
| Actual Result | * Result as Expected |

### D.2. Black Box Testing Results

The following table shows the black box test cases carried out on the JavaApp for the project, along with the test ID, expected output and the results.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Input** | **Expected Output** | **Pass/Fail** |
| **1** | Trying to login with unknown credentials | Alert telling the user this is an invalid username/password. | *Pass* |
| **2** | Trying to register with already known credentials | Alert telling user that these credentials are already stored in the database and if they would like to login with them. | *Pass* |
| **3** | Leaving the username input field blank and trying to register/login. | Alert pops up telling the user to enter a valid password. | *Pass* |
| **4** | Leaving the password input field blank and trying to register/login. | Alert pops up telling the user to enter a valid password. | *Pass* |
| **5** | Spelling a place incorrectly. | Alert to say the place is spelt incorrectly, timer stops. | *Pass* |
| **6** | Pressing graph button after simulation has been run. | A visual representation of the simulation results from the console will pop up for the user. | *Pass* |
| **7** | Trying to login with a known username and password but not the corresponding ones. | Alert to say an invalid username and password. | *Pass* |
| **9** | Saving a newly registered users credentials | Credentials added to database, allowing user to login with them at a later date. | *Pass* |
| **10** | Only saving a user fastest typing speed | Typing speed remains the same even if the most recent attempt is slower. | *Fail* |

# References

[1] Fei Cai, Maarten de Rijke. [Online]  
“A Survey of Query Auto Complete in Information Retrieval”  
https://staff.fnwi.uva.nl/m.derijke/wp-content/papercite-data/pdf/cai-survey-2016.pdf

[2] Z. Bar-Yossef and N. Kraus.  
Context-sensitive query auto-completion.   
In WWW ’11, pages 107–116, 2011

[3] S. R. Kairam, M. R. Morris, J. Teevan, D. J. Liebling, and S. T. Dumais.   
Towards supporting search over trending events with social media.   
In ICWSM ’13, 2013.

[4] Yang Song, Dengyong Zhou, and Li-wei He.

Post-ranking query suggestion by diversifying search results Research and Development in Information Retrieval,

SIGIR 11, pages 815–824, New York, 2011b. ACM.  
  
[5] Bret Nelson  
https://www.forbes.com/sites/brettnelson/2012/06/04/do-you-read-fast-enough-to-be-successful/#571fe7f6462e [Accessed March, 20017].

[6] Whiting, S., and Jose, J. M.  
Recent and robust query auto-completion. (2014).  
ACM, pp. 971–982.

[7] Wikipedia – Trie Structure [Online]  
https://en.wikipedia.org/wiki/Trie, [Accessed March, 2017].

[8] Joe Rossignol [Online]  
https://www.macrumors.com/2016/02/18/ios-android-market-share-q4-15-gartner/ [Accessed March, 2017]

[9] Google [Online]  
 “Google Maps SDK for iOS and Android,”  
 https://developers.google.com/maps/documentation/ios-sdk/. [Accessed March 2017].

[11] E. K. a. C. Hegarty,   
“GPS Overview,”  
Understanding GPS: Principles and Applications, Artech House, 2005, pp. 3-4.

[12] K. M. Pesyna, Jr, R. W. Heath, Jr and T. E. Humphreys [Online].  
 “GPS World”  
http://gpsworld.com/accuracy-in-the-palm-of-your-hand/. [Accessed March 2017].

[13] George Deded [Online]  
Indoor GPS Positioning, Challenges and Oppurtunities.  
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.367.4023&rep=rep1&type=pdf

[14] Google [Online]  
“Location Strategies,”  
http://developer.android.com/topics/location/strategies.html. [Accessed March 2017].

1. www.booking.com [↑](#footnote-ref-1)
2. www.google.com [↑](#footnote-ref-2)
3. www.github.com [↑](#footnote-ref-3)
4. https://www.facebook.com [↑](#footnote-ref-4)
5. https://www.twitter.com [↑](#footnote-ref-5)
6. https://www.twitter.com [↑](#footnote-ref-6)
7. www.spotify.com [↑](#footnote-ref-7)