Contents

[Abstract 2](#_Toc477007137)

[Report Summary 3](#_Toc477007138)

[1. Introduction 4](#_Toc477007139)

[1.1. Background 6](#_Toc477007140)

[1.2. Aims and Objectives 8](#_Toc477007141)

[2. Project Planning and Organisation 9](#_Toc477007142)

[2.1. Original Project Plan 9](#_Toc477007143)

[2.1.1 Research 9](#_Toc477007144)

[2.1.2 Implementation 9](#_Toc477007145)

[2.1.3 Testing 10](#_Toc477007146)

[2.1.5 Write Up 10](#_Toc477007147)

[2.2 Final Project Plan 10](#_Toc477007148)

[2.2.1 Research 10](#_Toc477007149)

[2.2.2 Design 11](#_Toc477007150)

[2.2.3 Development 11](#_Toc477007151)

[2.2.4 Testing 11](#_Toc477007152)

[2.2.5 Write-Up 11](#_Toc477007153)

[2.3 Organisational Tools 11](#_Toc477007154)

[2.3.2 Version Control 11](#_Toc477007155)

[2.3.3 Documentation 11](#_Toc477007156)

[Appendices 13](#_Toc477007157)

[Appendix 1: Original Project Plan 13](#_Toc477007158)

[References 14](#_Toc477007159)

# Abstract

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

# Report Summary

Chapter 1: Introduction – A look into the history of query autocomplete and an outline of the project aims.

Chapter 2: Project Planning and Organisation – A discussion of the time management and general organisation of the project.

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

Chapter 4: Testing Strategies – A technical look at ensuring the application works as expected.

Chapter 5: Performance Analysis & Predictions – A study into the continued reliability of the application as the simulation increases.

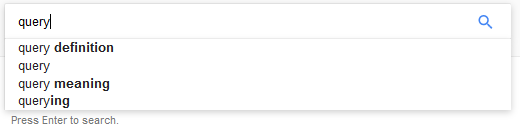
Chapter 6: Future Work – A reflection and discussion on what actions could be taken to further improve the simulation.

Chapter 7: Conclusions - A mirror to Chapter 1, covering which of the aims the project has met and which aims remain to be completed.

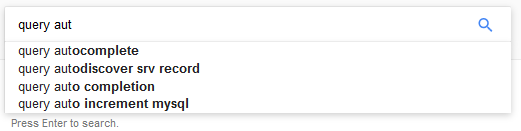
For an idea on how the application is used Appendix \*BLAH\* User Guide should be consulted.

# Introduction

Web searching is a tedious and omnipresent activity which is carried out by millions of people on a daily basis. However, this 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 incomplete input of the user is often referred to as *query prefix* and the suggested queries are often called *query completions*. As shown in Figure 1, query completions often start with the characters that the user has entered into the search box (see Figure 1a) and are updated as the user continues to type (see Figure 1b). 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. QAC helps the user to formulate their query when they have an intent in mind but not a clear way of expressing it in a query. It helps to avoid possible spelling mistakes, especially on devices which have smaller screens. 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 [1]. Although this approach provides satisfactory result 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, cutting down the search time duration of users implies a lower load on the search engine, which results in savings in machine resources and maintenance [2].



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



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

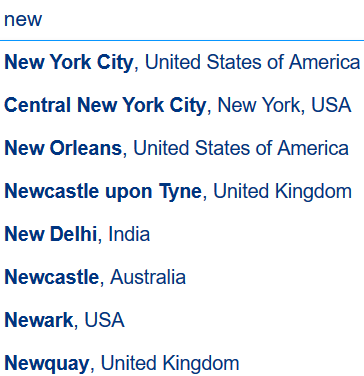
As the web continuously becomes a platform for real-time news and media, time consumption takes a vital role on the retrieval of information. Users turning to search engines for information about new or ongoing events, takes up the majority of daily queries [2]. However, importantly the use of QAC is not limited to the use or search engines, such as Bing, Google or Yahoo!. QAC has been integrated into other common services for search tasks. Facebook[[1]](#footnote-1) provides a QAC for finding friends by taking in mutual friends the user may share with others; it also assists users to formulate requests for searching through tweets on the social medium Twitter[[2]](#footnote-2); QAC is also used for product searches in online shopping stores, such as Amazon[[3]](#footnote-3) which will often provide up to 10 queries. In the concept of web search, research indicates that using the suggested queries can vastly improve user satisfaction, especially for informational queries [3]. The main focus for this report is how QAC is integrated into a travel metasearch engine like Booking[[4]](#footnote-4), where user’s search for the broadest selection of accommodation in every corner or the world.

## Aims and Objectives

The main aims and deliverable of the project are:

* To create a cost model of how people interact with the search box on a website like Booking, which can be seen in Figure 2, where people start to type in letters and the system predicts where they might want to go.
* To evaluate the impact of ranking on query autocomplete and how it influences the goal of finding the minimum cost.
* To evaluate the number of suggestions the user is presented with and the effect it has on the cost model.
* 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 the 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 2: Output from user typing “new” into Booking.***

# Related Work

Initially, before any work could begin to take place, several key questions had to be answered about how the project should proceed. To answers these questions, research into the history of query autocomplete and it’s advancements over the last few decades was initiated. Furthermore, investigations into several different aspects of the project were carried out.

## Background

The concept of Word Predictions is the process in which a writer writes the first letter or letters of a word and a word predictor generates a list of possible words or choices and if the desired word is listed, then this can be selected. It is from this concept of word prediction, where the most likely next words are listed, that the concept of query auto completion was adopted. The first instance dates back around half a century, where Longuent-Higgins and Ortony outline a method, where commands and identifiers were entered by users to decrease the amount of keystrokes necessary to complete a word [4]. Additionally, a true motivator for auto completion was to help individuals with physical disabilities increase their typing speed. As algorithms were developed, the deeper the research into the overall benefits were explored. Early studies found that a reduction in key strokes through word prediction was often over shadowed by having to scan the list of predictions for the desired word [5]. Therefore, it was discovered that displaying five suggested items to the user was found to provide a reasonable balance between keystroke saving and scanning the array of predicted words [6].

Throughout the 1970s and 1980s, the algorithmic work for word predictions was split up into three classes: character predictors, word completers and systems with a combination of both [7]. As assumed, character predictors make likely letters quicker and simpler to select. Whereas, word completers take a supported text input and return words based on the initial prefix of one or more letters entered by the user. Combined systems can carry out both these tasks. The first instance of a combined approach was The Reactive Keyboard, which was introduced by Darragh in 1990 [8]. This approach included many familiar aspects of today’s autocomplete, incorporating the use of ‘most popular’ and ‘task relevant’ words.

At its stage in time, The Reactive Keyboard is a prototypical example as it portrays the concept of word completion and vastly accelerated typewritten communication with a computer. However, its vocabulary of words was based on a standard and not extensive variety of words, therefore, later advancements of the model showed word completion based on previously entered text. In a similar approach, a user’s search history over a long-period of time can be used for query auto complete [9], but this has yet to be integrated into a word completion system. Word completion is reliant on a specific tree structure which will take the input and match it with its completions, very similar to the data structure which is used in query auto completion.

Query auto completion, also sometimes referred to as incremental search or real-time suggestions, goes back to the Emacs first text editor [10], which output a single line of feedback compared to the list of suggestions familiar to us. It wasn’t until the millennium that Jef Raskin, introduced the use of query auto completion in which the user will get instantaneous feedback throughout the process of entering a query [11]. This process was called delimited search, which is a traditional search interface where the user will search in three steps: a user submits a query, the system will compute a result, to which the user will receive said result. Since this moment, QAC has been used by websites, desktop searches, operating systems, email clients, internet browsers and search engines. Leading to 2004, when Google Suggest was launched, which was the first instance of query auto completion in the setting of a search engine.

## Cost Models

## State Diagrams

## Sorting of Suggestions

# 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 current status of the project at all points was confirmed by the creation of a project plan. The 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 were then sub-divided into smaller, more manageable tasks, which could be tackled individually. Two of these sub-tasks came with strict deadlines: Poster Creation (27th November 2016) and Report Write-up (31st March 2016), which dictated deadlines to support the sections. 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 is located in Appendix 1: Original Project Plan.

### 3.1.1 Research

Research was undertaken into the way query auto completion works. A detailed look into how the user jumps between the state of typing a letter, looking at the list of suggestions, picking from the list and the probability of each state occurring. A decision was made that ten 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 competed. The time available was then split amongst the sub tasks within research.

### 3.1.2 Implementation

The Implementation of the system was to be completed 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 time available between creating the poster and the deadline set for Implementation gave the overall time available. This time was taken and divided amongst these main tasks based on the perceived level of difficulty they carry.

### 3.1.3 Testing

Four main aspects of testing were anticipated to be employed throughout the project: Unit Testing, Sub-System Testing, Full Testing and User Trials. Ideally, the testing portion of the report was to be carried out in parallel with the Implementation section. Unit Testing would be performed as each small section of the cost model was created, with Sub-Section testing being carried out at the end of each new feature and User Trials carried out just before writing the final report. Leaving only Full System Testing as the only part of testing carried out independently of the other sections. It was decided that two weeks would be a sufficient amount of time to satisfy this.

### 3.1.4 Documentation

The documentation section consists solely on two set deliverables for the project. Strategically, the timescales that are allocated to these deliverables have been greatly exaggerated to ensure the project’s completion before the deadline. Creating a buffer like this allows expansion for the other sections, if necessary, whilst still ensuring the deliverables completion before their respective deadlines.

## 3.2 Final Project Plan

As the project progressed the initial project plan became impossible to follow precisely. As problems occurred, and the aims of the project evolved, the project plan was continually updated to reflect the current state of the project. The final project plan is available in Appendix 2 Final Project Plan.

### 3.2.1 Research

The research section was largely unchanged from the original plan.

### 3.2.2 Design

### 3.2.3 Development

### 3.2.4 Testing

### 3.2.5 Write-Up

## 3.3 Organisational Tools

In order to complete the project, various organisations tools were employed to keep within the timescale outlined in Chapter 2.2 *Final Project Plan*.

### 3.3.2 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. Allowing access to a functional version at a later date, as a form of back-up copy.

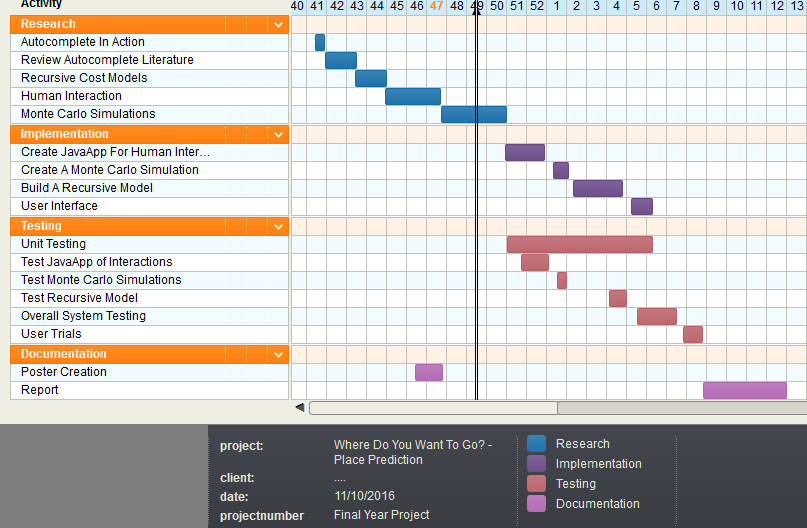
Git operates on a central server, where work can be uploaded to and downloaded from. The Git server used for this project was provided by Github[[5]](#footnote-5), which holds a copy of all work on their servers. This not only grants access to be worked on by multiple machines, but as long as the software 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.

### 3.3.3 Documentation

When dealing with a project of significant size, 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 to 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.

# Appendices

## Appendix 1: Original Project Plan



Design…. The figure shows like user types – types again – looks etc

State diagram. The cost changes the further down they look.

Graphs of probability.

If someone is a slower typer but fast looker and vice versa

At what point does the suggestions become irrelevant?

Latitude, cities that are closer are more likely to be recommended… etc

Sampling, take 1000 cities. Fix them. Only going to take 1000s for the cities, take the average and that what it is to do. Make sure cities don’t change.

# References

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

[2] 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.

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

Post-ranking query suggestion by diversifying search results. In Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval,

SIGIR ’11, pages 815–824, New York, NY, USA, 2011b. ACM.

[4] H. Christopher Longuet-Higgins and Andrew Ortony.

The adaptive memorization of sequences.

In Proceedings of the 3rd Annual Machine Intelligence Workshop, Machine Intelligence 3, pages 311–322. Edinburgh University Press, 1968.

[5] Gregg Vanderheiden and David Kelso.   
Comparative analysis of ﬁxed-vocabulary communication acceleration techniques. Augmentative and Alternative Communication, 3:196–206, 1987.

[6] Andrew Swifﬁn, John Arnott, J. Adrian Pickering, and Alan Newell.   
Adaptive and predictive techniques in a communication prosthesis.   
Augmentative and Alternative Communication, 3:181–191, 1987.

[7] John J. Darragh.  
Adaptive predictive text generation and there reactive keyboard.   
Master’s thesis, Computer Science Department, University of Calgary, 1988.

[8] John J. Darragh, Ian H. Witten, and Mark L. James.  
The reactive keyboard: A predictive typing aid.   
Computer, 23(11):41–49, 1990.

[9] Fei Cai, Shangsong Liang, and Maarten de Rijke.  
Preﬁx-adaptive and time-sensitive personalized query auto completion.   
IEEE Transactions on Knowledge and Data Engineering, 28(9):2452–2466, September 2016.

[10] Eugene Ciccarelli.   
An introduction to the Emacs editor.   
TechnicalReportAIM-447, MIT Artiﬁcial Intelligence Laboratory, Cambridge, Massachusetts, 1978.

[11] Jef Raskin.   
The Humane Interface.   
Addison-Wesley Professional, 2000.

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