

Adam Philip Lewis

***Internet Scale Clustered Web Search with Pre-Query
Cluster Formation
MSci Computer Science Innovations
May 2013***

*All working documents available at:
URL HERE*

*For a copy of the index, please contact Dr. Paul Rayson:
p.rayson@lancaster.ac.uk*

Table of Contents

1	Introduction.....	1
2	Alternatives To Standard Web Search, Why?.....	1
2.1	Alternatives To Standard Search.....	2
3	Cluster Based Search.....	4
3.1	Cluster Based Web Search In Practice.....	5
	Figure 1: Screenshot of Clusty's homepage.....	5
	Figure 2: Screenshot of a search for “Apple”.....	6
	Figure 3: Query “Clouds” for “Apple”.....	7
3.2	Anatomy, and flaws of, Cluster-Based Search Engines.....	7
	Figure 4: Query Execution Stages.....	8
4	Lime, Design & Related Work.....	9
	Figure 5: Basic Search Engine Structure.....	9
4.1	Related Work.....	10
5	Data.....	10
5.1	Data Pre-processing, Filtering.....	11
	Figure 6: Simplified representation of the XML data after filtering.....	12
5.2	Data Pre-processing, Formatting.....	12
6	Lime, Implementation.....	13
6.1	Lime – Crawling.....	13
6.2	Lime - Clustering.....	14
6.3	Output & Summary.....	15
	Figure 7: Indexer Diagram.....	15
6.4	Lime – Search Interface.....	16
	Figure 8: Lime's Homepage.....	16
	Figure 9: Lime's Results page.....	16
	Figure 10: Results page after clicking result in Lime.....	17
	Figure 11: Lime's result page with a cluster selected.....	17
	Figure 12: Structure of a page record.....	18
	Figure 13: Structure of a cluster record.....	18
7	Evaluation.....	18
	Figure 14: Distribution of similarity values between page pairs.....	19
	Figure 15: Distribution of similarity values between page pairs for the reduced data set.....	20
7.1	User Evaluation.....	20
	Figure 16: “Looking at the results section after entering your query, how would you rate the results?”.....	21
	Figure 17: “Looking at the clusters section, how would you rate the labels of the clusters with respect to your query?”.....	22
	Figure 18: “Select one of these clusters and click it. Then using the newly filtered results, rate how well the cluster's label describes the results it contains”.....	22
	Figure 19: “How well do these filtered results relate to your query?”.....	23
8	Future Work.....	24
9	Conclusion.....	24
10	References.....	24
11	Appendices.....	26

Abstract

Information Retrieval on the World Wide Web is a problem that many consider to be solved, yet there are those who develop alternatives to improve on and disrupt the status quo. One particular area of improvement that has been identified is the level of assistance search engines provide to users. In response, a new paradigm of search based on clustering has arisen; such technology remains in its infancy however and is flawed by design. Through the movement of clustering from post query to pre-query execution, the research within explores a new approach to indexing that is believed to be the next step in enabling Web-scale, cluster based search engines. From the integration of clustering and indexing technologies the prototype search engine “Lime” emerged, a new search engine capable of operation on the website Wikipedia.

1 Introduction

The problem of information retrieval is one that is as old as the concept of a library itself. “Scientia potentia est”, or “Knowledge is power”, holds as true today as it ever has, and quite recently an ever-growing library of knowledge has exploded into being. This collection is of course, the World Wide Web; a vast system that plays host to nearly the entirety of humanity's knowledge and has driven several innovations in the field of Information Retrieval.

For without the mechanisms we employ to retrieve knowledge, we could not enact upon it nor share the knowledge we possess. A DVD without a player can not be viewed, a book written in a foreign tongue can not be read and a simple word can not be recalled when experiencing lethologica. A World Wide Web without these mechanisms would be much less useful.

As such, the growth of the World Wide Web has pushed Information Retrieval into new frontiers and towards exciting new discoveries. It should not however, be believed that we have reached the pinnacle of technological innovations when discussing search on the Web. Or perhaps it is better to say that we have not explored the alternatives in sufficient depth to rule out the idea of a web with multiple facets to search.

Fortunately, there are those who possess a desire to transform web search and have already created several notable alternatives to the current model. Each of these defying convention and exploring its own interpretation of search on the web. One such example, cluster based search, is a search paradigm that attempts to enrich a user's experience by clustering and tagging similar results using extracted meta-data.

When using such engines however there is a common weakness to be found across all of the known examples – returned set size. No matter which engine you choose, the result set is orders of magnitude smaller when compared to the input. With such an obvious pattern, it is only sensible to ask why and to contemplate how it could be done differently. Surely if we are to build cluster-based web-scale search engines, the clustering algorithms used must also scale to a web sized dataset? It was this curiosity that drove this research forward and brought Lime into being.

This document is the result of this research and is as follows. First we discuss the motivations behind the project in detail, covering why web based search may in the future change and how cluster based search fits into this future. Second we discuss the basic concepts behind Lime, similar work and the process of obtaining its dataset. Third is a discussion of Lime's design in detail and followed by an evaluation of the prototype system. Finally, we conclude the results of the research and propose potential investigations for the future.

2 Alternatives To Standard Web Search, Why?

We are all familiar with the concept of and use of modern search engines. Today's search engines are

enable us to search a global database of over 15 billion pages [1] in less than a second, for any arbitrary query, with impressive recall and precision rates [27]. And while their internal workings are well within the territory of experts, it can be accepted without question that the majority of any search engine's user base is not from a related background of expertise.

A typical child can use these tools as proficiently as most adults and this accessible nature is largely owed to the fact that the typical user interface is the total antithesis of complex - type your query into a text field, click search, view your results. If a suitable result can not be found in the first two pages [2], reform your query and try again. This is an extremely easy system to use and appears to suit the general populace quite well; Google alone accumulated more than 4.5 billion searches on an average day in 2011 [3]. In fact, these tools only begin to venture towards the realms of complexity once we as users start learning how to use operators and construct much more complicated queries.

As mentioned in the introduction however there are alternatives available to the standard model of search, and considering how popular modern tools are we are naturally inclined to ask "Why, what's wrong with the current tools?". This very question in fact was often the typical challenge raised during the duration of the project, with those asking it often requiring convincing that search will have to change to meet the needs of the future.

In response, consider how rare it is to find a perfect "one size fits all" solution in any area of design and then how it is even rarer to find a design that can not be improved upon given enough time. The field of information retrieval itself is not exempt from such change and historically speaking has already undergone significant reworking.

"The oldest known library was created in Elba, in the 'Fertile Crescent', currently northern Syria, some time between 3,000 and 2,500 BC" [28]. Such collections would have been organised by hand but since then things have drastically changed. Libraries have grown dramatically in size and continue to grow, gradually rendering human efforts to sort them ever more redundant.

Fortunately however the introduction of computer technologies enabled us to keep up with the influx of information, gradually replacing human components of information retrieval for digital ones over the last 60 years. This core remodelling was then magnified by the growth of the World Wide Web, a collection so large that the concept of sorting by hand defies comprehension.

And when factoring in that historically speaking the Web is still quite young, who is to say that information retrieval will look the same in the Web's future as it does today? In fact given how much the Web has changed in the last twenty years alone, moving from research to commerce, social & entertainment based focuses, it is highly unlikely that search will remain static.

2.1 Alternatives To Standard Search

If unconvinced that how we search on the World Wide Web is susceptible to change from Information Retrieval's past alone, consider some of the ways that it is changing right now. There are three areas that we as designers can use to modify the standard model and several ongoing projects highlight these changes in action. The first of these areas covers modifications to the typical engine's output, i.e. how results are presented to the user, not what form of data is given.

By altering the design of an engine's output, it is possible to set a completely different tone in comparison to the standard model. Simple changes such as creating more search windows [4], simultaneously searching over several mediums [5] or displaying thumbnail results [6] can have a profound effect on how it feels as a user to experience search on the Web. If one of these designs proved itself to be popular and found favour with users, it could be enough to establish a new standard.

The second form of alteration is a matter of input; as how we enter queries should be considered as much a foundation stone to search as the shape the output takes. To represent this form of change, consider a recent and substantial deviation to how we as users interact with search engines: smart phones. These devices have, without doubt or question, revolutionised mobile computing; They continue to grow in popularity, grow in terms of processing power and provide Internet connectivity on high speed mobile networks. For the first time in human history we are

capable of accessing the entire global information store of the World Wide Web anywhere provided there is coverage. Just this advancement of mobility alone has already begun to usher us into a potential new era of search. An era where engines are aware of both the queries we make and of the location they are made in [10].

This is not all however as mobile devices may yet distort search in another way. With this new area of human computer interaction there are also new challenges designers face; precisely speaking the task of text entry. This is an area of HCI that has long been known as troublesome and has been the subject of an extensive amount of research. “Although the ubiquitous Qwerty keyboard reigns supreme as the primary text entry device on desktop systems, mobile and handheld systems lack an equivalent dominant technology or technique for the same task. And so, the challenge of text entry for mobile computing presents itself” [7].

Knowing that the standard method of search relies heavily on text based input, is it reasonable to believe that search will remain the same on mobile devices given enough time? Fundamentally text entry differs greatly between static and mobile computers, and this may be enough to fragment web based search.

The difference after-all has already shown itself as a driving force towards the utilisation of a mobile device's alternative hardware offerings as input tools. Computer vision based tools such as Google Goggles [8] and TinEye [9] for example, exploit the practically ubiquitous presence of cameras on mobile devices for the purpose of search. Furthermore, voice based alternatives do much the same by using microphones.

Alternative input tools such as these that may one day find a place, amongst today's modern search engines, of equal importance on the web. In particular, there is a strong case that voice based alternatives will find such success. So much so, that two of the world's largest technology companies, Apple & Google, have both placed large amounts of research into advancing this field of computer science. The Google Glass project [11] promises a future of totally hands free search and Siri [12] goes even further than this with the promise of search that *understands us*.

This brings us neatly onto the third form of change we can employ: how we search. Not how in the sense of what we use as input or how in the sense of viewing our results, but how in the form of logic used by the search engine. It is this change in “how” that has lead to the creation of some truly groundbreaking systems; systems such as Wolfram Alpha [13].

When using Wolfram, not does it provide a set of results but it does so by understanding our queries in their natural language format. For the first time in history we have engines that we can understand questions, process questions and give us answers. To illustrate, when querying the engine with “Who won the Olympics?” a table of medals from the 2012 London Olympics is presented as a part of the results. The same query on Google however (at the time of writing) simply returns a list of pages it believes hold the answer. That is Google does not *know* what we hope to gain from the query, it can only return the pages it believes to be the best match.

For the most part Google's approach to search works but if these “intelligent” search engines are developed further, they will no doubt be powerful tools for future use. Such advancements would bring us a step closer to the widespread use of natural user interfaces and could even jump-start a whole new industry catered towards electronic personal assistant provision.

And in another completely different vein of search, we have the up-coming search engine Facebook Graph [14]. Currently in Beta for US English speaking users only, Facebook Graph advertises itself as trying to imagine search fuelled by people not pages. Want to find a new restaurant? A new band? Parties this weekend etcetera? Use the data generated by your social circles to answer these queries and more.

Such an approach to search simply would not have been plausible before the days of social networks; Just as the concept of instant price/review checking products in physical stores could not be realised before the advent of e-commerce on the Internet. Without question or doubt it can be seen that the web has changed in many ways over the last twenty or so years, and search is definitely going to change with it.

3 Cluster Based Search

Having discussed some of the broader motivations for change in search and their real world manifestations, we at last come to the topic of cluster based search on the World Wide Web. Again, we must consider the motivations behind such engines before we can truly understand them.

First and most strikingly, a cluster based engine differs greatly from high concept engines such as intelligent & social search. Cluster based search does not try to completely re-invent how we search, rather it attempts to improve upon the status quo in several ways. The first of these improvements are potential solutions to problems caused by ambiguity.

To emphasise consider the example query “Tiger” and then give thought to what we as a user may want this search term to find. Upon first reading the query, you may conclude that we are searching for the animal or perhaps in a few cases the beverage. But then again, perhaps we are looking for the place Tiger, Georgia. Or it could alternatively be that we are looking for Apple's OS X Tiger, the Chinese zodiac symbol, the tank or even the golf player. The list of potential meanings for tiger continues and is quite extensive [15].

It is, after all, not difficult to find homonyms in any given language and this can make matters complicated for users. “Web search is difficult because it is hard for users to construct queries that are both sufficiently descriptive and sufficiently discriminating to find just the web pages that are relevant to the user’s search goal. Queries are often ambiguous: words and phrases are frequently polysemantic and user search goals are often narrower in scope than the queries used to express them. This ambiguity leads to search result sets containing distinct page groups that meet different user search goals. Often users must refine their search by modifying the query to filter out the irrelevant results” [17].

So, faced with failure, we think nothing of taking the time to reword ourselves, putting extra effort into the phrasing of a query for the hope of better results. This may not however be the best approach to employ for two reasons. First, there is the mental strain that composing a query places upon us as users.

When forming a query we explicitly use conscious thought processes and semantic memory to complete our task, and with failure comes further expenditure. To handle failure, we do not only exert mental energy forming a new query, but also on identifying the cause of failure and the necessary correction(s). It is then only logical to conclude that having to reform a query at least doubles strain upon us as users. Such strain represent wasted energy and should be reduced to the lowest amount possible. Heavy usage users, such as researchers for example, would no doubt benefit from as little unnecessary mental strain as possible when performing search.

Secondly, there is the time required to re-query a search engine. Upon realising there is a problem with results we must first attempt to mentally correct the fault, then enter the new query and finally wait for its execution. And although small, such time wastage is of genuine concern to large business. The culture of “counting seconds” is necessity/fruitless depending on who you ask, but when considering that an average information worker spends 9 hours per week just on search [16], arguments in favour of counting seconds could be well founded.

The hope is that cluster based search engines will lead us towards the remedy of both these problems, and if effective, they could improve the experience and efficiency of search at a global scale. How valid such hopes are and the impact they will have is an unknown at this point in time however, the field of cluster based search is still new and not as popular, making it impossible to draw any concrete conclusions.

Finally, cluster based search simply tries to just provide a better service to users. One way this can be done for example is to use the clusters as a form of additional level of “filtering”, improving the quality and relevance of results by ignoring all pages that can not be clustered. Another is promoting the co-location of similar documents for information discovery, a useful feature for knowledge workers.

3.1 Cluster Based Web Search In Practice

With the motivations of cluster based search covered, it is time to see what such an engine actually looks like. The necessary components required to build a cluster based search engine are not new concepts, and several examples of cluster based search engines exist on the World Wide Web already, each attempting to provide a viable solution to its users. For the purpose of this document however, one example must be chosen above the others to fairly represent clustered search from a user's point of view. For this, the reasonably well known search engine “Clusty” was selected.



Figure 1: Screenshot of Clusty's homepage

Starting with Clusty's homepage, design wise it is typical of any search engine on the World Wide Web. There are several search options available, links to company information/services and most importantly: a query entry field. The page itself is clean, promotes ease of use and follows the typical paradigm that we as users are all highly familiar with.

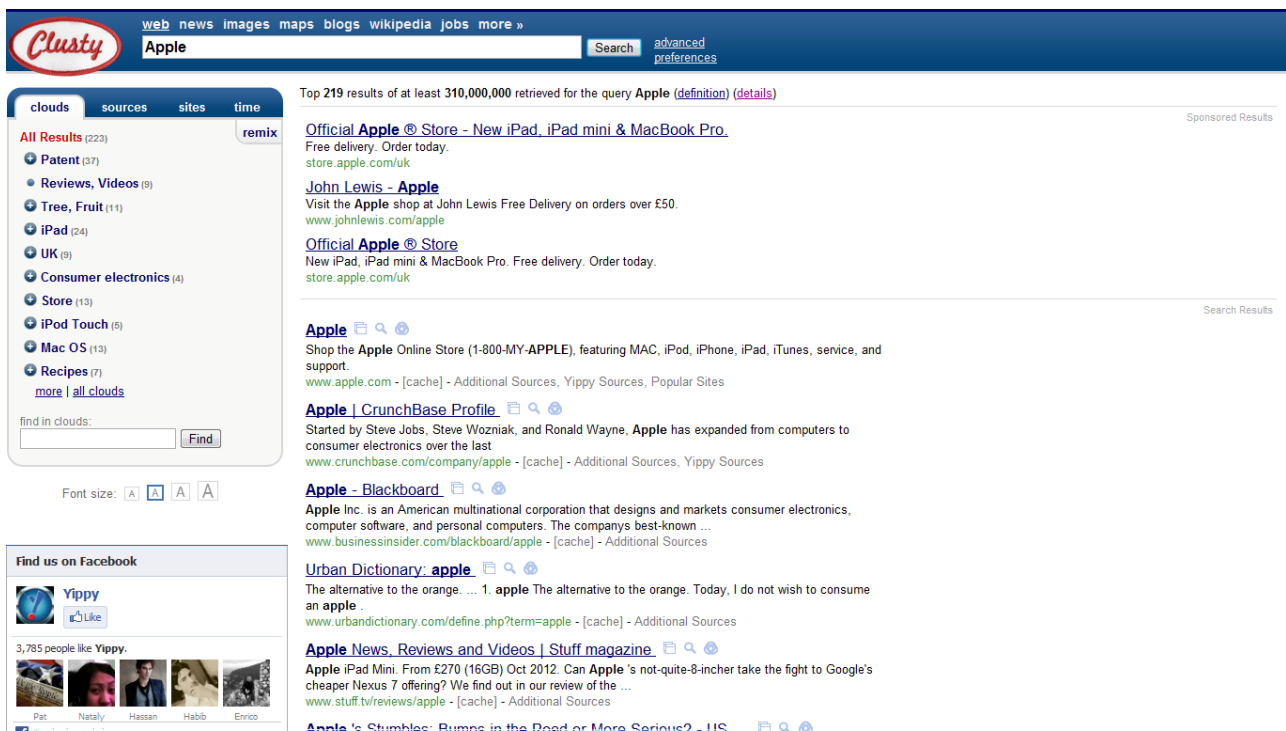


Figure 2: Screenshot of a search for “Apple”

Upon executing a query, a page similar to figure 2 above is displayed. Again this is very similar to the standard engine design, displaying a list of results to view in a list. The list takes up the majority of the web page, displays page information and has the option to load subsequent pages of further results.

The list includes both genuine results and advertisements, as ranked by their relevance, and everything thus far looks and acts the same as any typical web search engine. It is only once we begin to explore the panels to the left does Clusty's fundamental difference become apparent to a user.

Clusty, a product of Vivísimo and acquired by Yippy Inc. in 2010, is a search engine that “groups similar results together into "clouds." Clouds help you see your search results by topic so you can zero in on exactly what you’re looking for or discover unexpected relationships between items” [18].

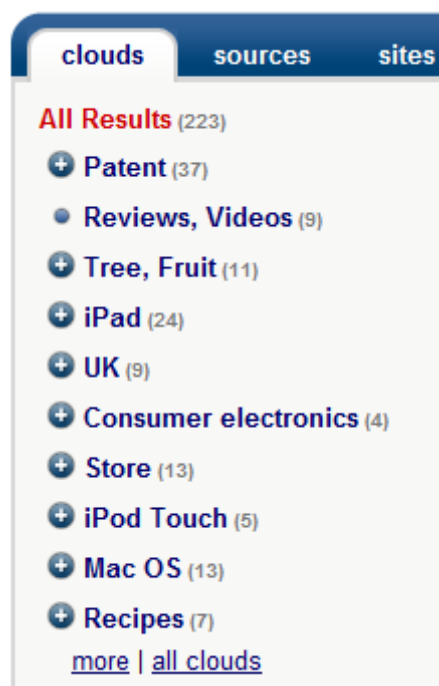


Figure 3: Query “Clouds” for “Apple”

These “Clouds”, essentially, are filters that can be clicked to reduce the size and scope of the results set. The search for “Apple” in figure 3 above for example has provided the clusters: “Store”, “Patent” and “Consumer Electronics”. Clicking any of these “Clouds” triggers a change to the results page, selecting only those that fit within the scope of a cluster's topic to remain on display and removing those that do not.

3.2 Anatomy, and flaws of, Cluster-Based Search Engines

As the name implies, a cluster based search engine combines both clustering algorithms and traditional search as one. The result of which is a search engine that, like Clusty, is capable of grouping results into clusters. These clusters can then be used to navigate subsets of returned query results, helping to highlight the pages that are desired by the user and excluding from view those that are not.

With the exception of Yahoo's Directory (which is human generated) all modern and past cluster based search engines operate using the same design. The only distinguishing features that separate one engine from the other, aside from naming, is the choice of algorithm used for clustering and data sources. Carpineto, Osiński, Romano and Weiss discuss this in great detail [19], revealing the designs of thirteen completely separate search engines to be nearly identical.

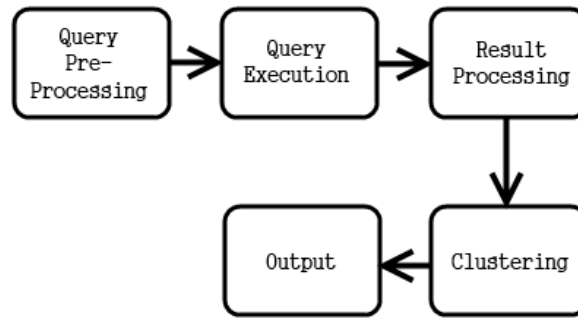


Figure 4: Query Execution Stages

This common design, as shown above, follows five basic steps when queried: query pre-processing, query execution, result processing, clustering and output. The first of these steps, query pre-processing, describes the act of making the search query suitable for use by the engine.

At its simplest this step covers tokenising the string into individual non-repeating search terms. At its more complex, this step is responsible for any special functionality such as handling logical operators, correcting spelling errors, etc. Once the query is ready for use, the search engine executes it and obtains a list of ranked results ready for use in the next stage: result processing.

Like query pre-processing before it, result processing is merely concerned with changing the format of an input into another for use by the next stage. Precisely, it is concerned with taking the results from the query and processing the data into a format suitable for use by the clustering stage. Once done, the clusterer uses an algorithm (such as STC, AHC, Lingo, SHOC, k-Means) to form clusters, and then finally labels the newly created clusters to give them meaning. The clusters and results are then sent to output for ranking and the search is now complete. The whole process from beginning to end is done and the user is able to view the conclusion of their search.

Within this design is a weakness however, a subtle flaw but most certainly a concrete one. A flaw that is present across all modern cluster based search engines; the effects of which visible on any results page. That is, you as a user are subjected to result set reduction by orders of magnitude when using any of the modern cluster based search engines. The previous query “Apple” for example returns 219 results of a potential 310,000,000. That is for this given query, approximately only 0.00007% of the data has been retained, the rest is thrown away.

Users rarely make it past the first ten results [2] let alone browse to the two hundred and nineteenth however, so how can this be an issue? The answer to this is two-fold. Firstly there is the issue of obscurity. When performing an ambiguous search, one meaning can often override the other – for example the programming language interpretation of “Java” against the drink or island.

This creates searches that ignore lesser ranked pages and as such, creates searches do not include less topical pages in clusters. In effect this results in search engines that tend towards only providing clusters based on top ranking results, requiring users to still re-enter queries to negate ambiguity issues.

Secondly, and theoretically, result set reduction should lead to a degradation in cluster quality. Intelligent/learning algorithms tend to get better as their experience widens, with more data meaning better results. It makes it only logical then to conclude that as the amount of data available to a clustering algorithm is reduced, the certainty with which clusters can be formed also drops and with it, their validity/quality.

So what causes such a large reduction to take place? First there is the fact that the data source used by most cluster based engines is not usually their own. As most of the engines are research based, a majority choose to piggy back other popular/reliable engines through publicly available APIs. This allows them to generate results for use in clustering without undertaking any web crawling/data retention processes.

This requires less investment on their part but does have drawbacks. The use of 3rd party

APIs is often restricted by quotas and overheads, preventing any piggy-back services from querying too much data and too often. Request too much data per search and you'll soon exceed your quota. Request too often per query and you'll soon find a noticeable delay in your engine's performance. The art here is to strike a balance between providing a quality service to your users and maintaining connections to third party providers. This would not be an issue for the API providers however, but even they can not circumvent the second problem modern cluster-based search engines face: time.

Clustering takes time to perform and, depending on the choice of algorithm, can grow at non-linear rates as the input's size increases. Under current system designs this is simply an overhead that must be contended with for each and every search. As the use of clustering in query execution forces clustering to be done in real time, it places very strict requirements on execution time. Take too long and you will lose traffic [29].

So, even with access to a database of the entire World Wide Web, if too many results are chosen for use in clustering there will be a severe negative impact on your engine's response times. You will lose traffic and you will lose potential revenue.

It was these small observations that really inspired the creation of Lime and pushed the project forwards. Here was an observable trend that was simply not being addressed by researchers. Perhaps it had not been noticed before, perhaps it had been eclipsed by a desire to create better algorithms first. Regardless, it was now the sole focus of this project to find potential solutions to the scalability problem.

4 Lime, Design & Related Work

Finally, we can now begin to discuss the project's implementation over its motivation. As shown by the prior section, "Cluster Based Search", all modern cluster based engines perform clustering during query execution. This in turn has a large impact on the scalability of such engines and theoretically degrades the quality of service they can provide. The challenge we now faced was how to provide similar service, all while avoiding this design.

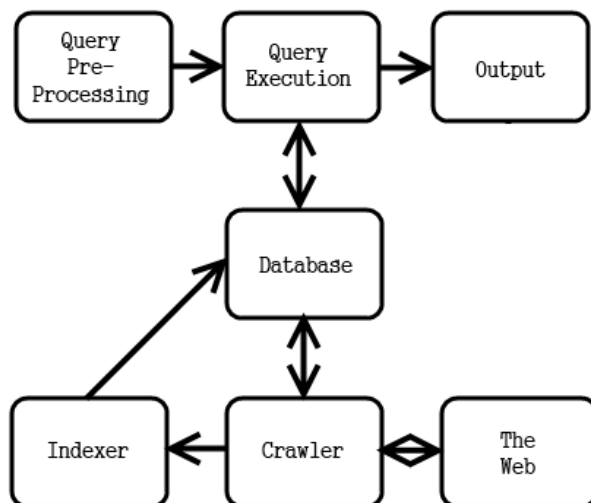


Figure 5: Basic Search Engine Structure

To discover how, let us first consider the design of a typical search engine without any clustering technologies built in, and then how we might apply them. As shown by Figure 5 above, we can see that without any clustering, to search is simply a case of query in, query pre-process, query execute and query complete. And as we have already seen how clustering can be applied here and its effects (Figure 4), we must look elsewhere for answers. To explore potential applications of clustering, we must look deeper into an engine's internals, how we get from query to results.

We could use brute force; load every single known web page on the Internet at once, judge if it is suitable and return true or false as needed. Alternatively, we could even design a protocol to multicast over the entire Internet and let individual domains reply to queries.

Were either of these options actually used in practice however the Internet would more than likely grind to a complete halt due to excessive congestion. And the latter protocol based example would probably do even worse, with spammers worldwide trying to advertise sites to anyone who made the mistake of asking. In answer to such issues, search providers choose to utilise programs known as “crawlers”.

A crawler is simple program designed to archive data for search engines. Given a search engine's database, it will select a suitable URL to load – an uncrawled website or one that may have expired. Once downloaded, that page can be sent to the indexer if necessary for archiving and the cycle repeats. In effect this slowly creates an always “just out of date” copy of the World Wide Web, ready for use whenever a query arrives.

Overall however, this particular component of search engines barely even begins touching a web page's content/context. These programs are solely concerned with maintaining an up to date database and finding new content. Due to this, they are not suitable for merging with clustering techniques. That is, for clarity, not to say clustering techniques can not be applied to crawling as shown by Tan and Mitra [26], rather they can not be used in a manner suitable for enabling cluster-based search.

To provide a true cluster-based search engine experience, we instead looked towards modifying the only remaining component: the indexer. How this component is modified fundamentally underpins the operation of Lime and provides the system with one key benefit.

To understand, begin by considering how the act of indexing can be generalised to an iterative, mutually exclusive process. Under such a generalisation, each newly crawled page is added one at a time to the database and can only be accessed by search once indexed.

This already uncovers the largest benefit to clustering in the indexer: time. Unlike the standard method of clustering in query time, clustering during indexing is not constrained by real time requirements. A clustering process can now take several times longer, making algorithms with higher time complexities and data volumes feasible, all with no detriment of user experience.

Any clustering algorithms employed by Lime would however by necessity need to match the above generalisation, growing a single page at a time. They would also need to be able to co-operate with a database that has the potential to grow to the size of the World Wide Web. To meet these needs (as discussed in section 6) we introduced our own basic clustering algorithm based on TF-IDF and hyperlink data.

4.1 Related Work

Brin & Page's “The Anatomy of a Large-Scale Hypertextual Web Search Engine” is generally considered to be the birth of modern search, revealing PageRank & Google to the wider public for the first time. Many since have scrutinised the field of search, contributing research and critique wherever it is needed.

In the field of cluster based search for example, there have been several attempts to create innovative approaches to enabling clustered search [30, 31, 32, 33]. In addition to this there has also been research on how clustered search could be used, from search personalisation [34] to images [35].

There is however a disappointing amount of work published on the topic of scaling cluster based search to the size of the Web. If such work does exist, it is heavily buried underneath work on scalable web server technologies, making it almost poetic that our own research into cluster-based search would highlight a scenario when it would truly be useful. Given this, we knew that many of the choices to be made during the project would be educated guesses based upon the information available at the time.

5 Data

Having found that our efforts needed aiming towards the indexing process, the next step was of course to create ideas and try to form potential solutions. Before any designs could even be

considered however a data source would be required first; no data meant no input for the indexer and in turn no search engine.

The data would of course need to possess properties similar to web pages, containing both hyperlinks and text based information. It would also need to be large. Large enough to cover a wide range of differing topics, and large enough to repeatedly cover topics with sufficient frequency to promote the formation of clusters.

The first instinct, of course, was to use traditional methods and employ a web crawler to obtain the data set. The data after all is publicly available on the Internet and simply required downloading to be of use.

Unfortunately however we did not have a crawler ready and waiting to be used. And as constructing a good crawler is no doubt a highly time consuming process, one that would subtract substantial amounts of time from the project, a hunt for publicly available crawlers began.

Several free public use crawlers were found, but one distinct example proved to be of particular interest: Heritrix [20]. This “is the Internet Archive's open-source, extensible, web-scale, archival-quality web crawler project. Heritrix (sometimes spelled heretrix, or misspelled or mis-said as heratrix/heritix/ heretix/heratix) is an archaic word for heiress (woman who inherits)” [21].

This very crawler is responsible for helping the Internet Archive create the Wayback Machine; a database that houses “240,000,000,000 URLs, a total of about 5 petabytes of data” [22]. And a service that can be accessed by anyone globally for free, provided they have an Internet connection and a browser to make use of it.

In conjunction with this discovery came an epiphany. Perhaps it would be possible to download an archived copy of the World Wide Web as opposed to crawling it. Doing so would no doubt have several benefits.

First, there would be the time saved compared to attempting a crawl due differences in overheads. Using an FTP download for example would immediately prevent all of the overheads and problems that could be encountered by a crawler. Problems such as traffic congestion, multiple TCP connection set-up delays and HTTP overheads.

Second, an archive would most likely have been filtered before the host would agree to host it. Any “junk” would likely have been removed from the dataset to one: improve the dataset's overall quality, and two: reduce disk space and bandwidth wastage.

Third, there are security benefits from this approach. By downloading websites via proxy, it stops the crawling of malicious web pages. This in turn helps prevent our systems from becoming compromised and infected by malevolent third parties.

Finally, it removes the pressure of choosing good crawling seeds. Had a crawler been used during the project, it would somehow have to be decided where to start the crawling process and then whether the resultant dataset is of good quality after the crawl had completed. Such assurances could never be correctly made however, as how could it ever be truthfully known that the crawled pages fairly represented the World Wide Web and its breadth of subjects evenly?

Each of these benefits motivated the next wave of research, research hoping to find a large archive of data that would be suitable for the project's needs. Our search soon began to feel futile however as no archives could be readily found. Thankfully, all concerns were soon extinguished upon making the discovery that English Wikipedia's entire database is freely available for download [23].

The Wikipedia dataset is quick and easy to obtain, covers a massive range of topics, is composed of mostly verified content and is most self contained, hosting a large amount of self-referential links. Most importantly however this dataset would be perfect for use in evaluation later on in the project, as some of the cluster-based engines available online can be focused upon a single domain. This would allow for a direct comparison between search engines later on in the project's timeline.

5.1 Data Pre-processing, Filtering

With January 2013's revision of the Wikipedia database archive downloaded and unzipped, the

result was one single XML file approximately 40 gigabytes in size (13167739 pages). This, if anything, was simply too much data for the project and needed to be reduced. Eventually the dataset was brought down to a total of 15 gigabytes (4212492 pages) and optimised for use by Lime's indexer. How this reduction was achieved is the focus of this section's remaining text¹.

To begin the cutbacks, all redirection pages were removed from the data set. These pages exist to serve no purpose other than directing a user from one article to another, and when used correctly this helps guide users towards desired content even if it is titled differently. For example redirecting to the article “Snake Charming” from “Snake Charmer”. As these pages contain next to no usable data for indexing, their removal was the first logical cut to make. This brought the data set down to approximately 36 gigabytes in size (7163921 pages). Already things were beginning to look more reasonable so further reductions were sought out. The next cut was a simple one to make and works by exploiting a page's content type in order to make substantial savings.

Within the XML file, as shown by Appendix B, is a range of content types: history pages, talk pages, regular pages and even files. These name-space values are referenced by all page nodes in the tree, giving every page's data context and meaning. Appendix C for example contains two types of page from the original data set: a typical Wikipedia article (Name-space 0) and a Wikipedia specific article (Name-space 4).

Immediately it was obvious that not all of these content types would be needed and should be filtered off. Eventually, only name-spaces 0 and 15 remained creating a dataset of Wikipedia articles and category pages. This data was then split to create two separate files: the main data set of Wikipedia articles and another secondary data set with the potential for later use in evaluation.

This step alone led to a removal of approximately 15 gigabytes of data, bringing the total to roughly 26 gigabytes. This was much better than the original 40, but the stripping was still not yet complete. Upon inspecting the contents of page nodes it can be seen that yet more data has the potential to be stripped away.

Superfluous information tags such as page IDs, contributor details, timestamps, sha1 hashes etc would all be unnecessary for this project. In the end, only the text content and the titles of each page was preserved, bringing the main data set down to around 23 gigabytes in size or 4212492 unique Wikipedia articles. In terms of immediately disposable data, this was as far as the data could be reasonably pushed.

```
<page>
  <title>.....</title>
  <text>.....</text>
</page>
<page>
  <title>.....</title>
  <text>.....</text>
</page>
```

Figure 6: Simplified representation of the XML data after filtering

Yet even after the data had been filtered to such a reduced point, it was felt that it was still not just yet ready for use in indexing. The database download of Wikipedia had provided the project with an incredible amount of information but all of it unfortunately is given as wiki formatted text. This is an extensive format, containing various syntax rules and requires a great deal of effort to learn in full. Again faced with time constraints third party code was once more sought to simplify the situation, as attempting to construct our own parser would have taken too long to be a worthwhile venture.

5.2 Data Pre-processing, Formatting

Wiki mark-up is a fairly simple mark up language that can be usually parsed to render full HTML pages as they are required. This creates a low learning curve for new users (a crucial part to attracting editors) while simultaneously giving access to powerful tools for article creation. And fortunately due to its widespread popularity, multiple parsers are available online in several

programming languages; each free for use by anyone [24].

The basic plan was to first use one of these parsers to convert the Wiki format text into HTML. This text could then be processed using comparatively less effort due to a strong familiarity with HTML and the stricter tag formatting rules it employs.

This goal in the end was simple to realise and required around two days to fully execute. This was an unusually long time in comparison to filtering stage executables. They usually required only two hours each, nowhere near the forty eight that now presented itself.

The reason for this is all due to the parser and coding language used. Instead of coding in C as all previous code had been, the choice was made to use Java and the parser Mediawiki2HTML [25]. This meant that the converter was very quick to code but also required a long time to execute in the JVM due to the data set's size.

This was a problem that was not adequately foreseen and overall cost about 5 days of project time due to automatic computer restarts. Once finally finished however, the HTML version of the dataset was now ready and totalled approximately forty gigabytes in size. Of course this was again much too large and to handle this issue, further code was executed to strip all HTML tags from the document.

This in turn created a version of Wikipedia containing nothing but titles and plain-text. You may however be wondering that if all HTML was stripped, what happened to all of the links between pages?

It is true that once the HTML tags were removed the dataset had lost all linking data. This could still yet be recovered however and to achieve this, the filtered Wiki-format version of the data was again parsed to instead form a second dataset of all the links. This was then heavily filtered to remove any externally targeted links and dead links that no longer pointed to an existing article within the reduced data-set.

This file of links was then merged together with the plain-text only file to form the penultimate form of the dataset. This data only required a few minor cosmetic changes, changes such as replacing HTML entities with actual characters and stripping blank space. Finally, the data-set was ready for use. Each page containing a title², a list of its links and a plain-text version of the article.

¹ The Wikipedia data set is human generated and as such contains several forms of error within it. For brevity the specifics of these and their corrections have been neglected, but if you wish to repeat the process as described be prepared to face issues created by bad syntax.

² The titles within the final file are not in fact strings, they are a pair of four byte file pointers that reference the title's beginning and end in a separate file.

6 Lime, Implementation

Finally, everything required was in place and ready from an identified design space to the required dataset. All that remained now was the task of constructing three vital components: the engine's user interface, the indexer and the crawler. This section of text covers the implementation of each component, introducing the finer details of Lime's internal workings.

6.1 Lime – Crawling

As shown by figure 5 there are two sides to any search engine, the data generation back end and the query handling front end. To discuss the front end without first detailing the back would be pointless, hence we will first explore how the crawler and indexer of Lime operates. To begin, some basic details and context.

The back end of Lime exists as a single thread 64 bit executable, is written in C and is capable of indexing an entire processed Wikipedia file in one pass. In terms of code the crawler and indexer are merged as one, joined to enable data sharing between the components and reap the resulting speed benefits. This merging amongst other tweaks was vital, allowing for greatly reduced

execution/debugging times on fairly limited resources.

When running the executable it goes through six individual stages: file parsing, page linking, link filtering, clustering, output filtering and finally output. The first two steps, file parsing and page linking, are handled by what could be considered the crawler.

The “crawler” Lime uses is very simple and basic. This code at its simplest merely needs to read the entire XML file and parse it into pages. For each page it finds it stores three things related to that page: a reference pointer to that page's title, the page's links and the page's word profile (every word in the page and how often it occurs). Making matters slightly more complex however is the page linking stage which is vital to helping the indexer form clusters. During this stage, the crawler will look up the page to see if it has any “call backs” that need handling.

A call back, put simply, is a notification that this page has an inbound link from another previously indexed page. It is possible that the calling page and the new one can form clusters (As explained in section 6.2), and the new page must add the calling page to its list of links. The purpose of this is to create “eventual” clusters that can be formed once all the necessary information is available.

The page is now parsed with all required call backs added to the links list, it is now ready for indexing and the first part, link filtering, is used to create the data necessary for previous stage, page linking. When performing link filtering the crawler simply looks at the links of the page, separating those that have been indexed from those that have not. If a page has been indexed it is used for clustering and if not, it is given a call back message to ensure it eventually contacts us. With nothing more than a title reference, a list of links and a word profile the page is now ready for indexing/clustering.

6.2 Lime - Clustering

Not much has been said yet about the clustering algorithm used by Lime other than it is based on TF-IDF and linked data. Here we discuss the algorithm in detail to underline its inner workings. To begin, the algorithm relies on a very simple concept: people power. Or to be precise, the fact that humans have a natural talent for categorising and grouping information.

When placing information onto the web, we often link it to other pages with relevant contents. This is information joined and organised by links based upon human reasoning. In time this has created a network of semantically connected data on the World Wide Web, a network that Lime attempts to exploit. By assuming that content creators make relevant connections, Lime works with the data to form clusters using a page's immediate links only, creating clusters without comparing one page against them all.

For each page that is crawled, it has been said that a list of its links is made. For each link, the two pages are compared to calculate how similar they are and based on the results, potentially clustered. The metric for doing so is as follows:

$$C = A \cup B$$

$$If(w, P) = \frac{freq(w, P)}{total(P)}$$

Let page C represent the union of all terms in pages A & B, and the individual frequency of a term be defined as the frequency of a term in page P, divided by the total of terms within the page.

$$Similarity = 1 - \frac{\sum_{w \in C} |If(w, A) - If(w, B)|}{2}$$

Then for every term in C, calculate the sum of absolute difference between each individual frequency value of term w for both pages. Divide this value by two to obtain a value of dissimilarity, between zero and one. Finally, subtract this value from one to obtain the overall similarity value between pages A and B.

Now based on a threshold, the similarity value if large enough will put the two pages forward as a new cluster. The name of this cluster is selected as the word that caused the largest disparity, i.e. the word that differed the most between the two documents. If a cluster exists under the same name already the two documents are added into the cluster, else the new cluster is formed.

6.3 Output & Summary

As Lime's indexer was coded with speed in mind, output was not allowed until the whole file had been crawled & indexed. This had a major implication, there wasn't enough RAM available to give each word more than 100 pages and 10 clusters to be used as “best fit” results. Instead of reflecting reality and gradually outputting results as data came in, there was no choice but to filter results and accept that lesser ranked data would be lost.

Choosing the best results however is at least simple. To rank either a page or cluster for a term merely requires calculating that term's inverse frequency for the page/cluster. Then once filtered, the data can be sent to disk and stored in a modified version of an inverted index.

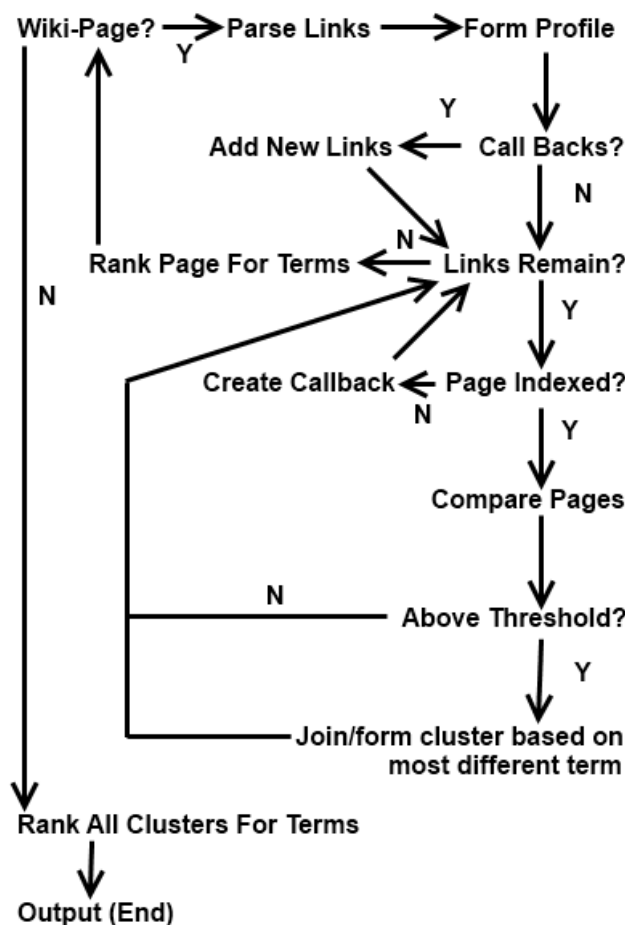


Figure 7: Indexer Diagram

In summary, the diagram in figure 7 above shows the basic logic of the entire crawling/indexing system as it executes. For each page in the input, the links and word profile of that page are extracted. Once done, any call backs are added to the page's links list to be later used in clustering. Then for each link, if that link can be targeted, it is compared content wise against the newly crawled page. If the pages are similar enough, a new cluster of two documents forms and is named by the word that differs most between them. If a cluster under the same name already exists, the new cluster merges with the older one, else the new cluster is added to the database. Once all the links have been processed, the page is ranked against all of the terms within to judge if it is worthy of a top 100 spot for that term. Finally, once all of the pages have been indexed, all of the clusters undergo the same ranking judgement, deciding how worthy they are of the ten spaces available for

each term they contain. Finally, all of the generated data is output for each term and the indexing process is complete.

6.4 Lime – Search Interface

With the dataset processed, indexed and output to disk, all that remains is to build a system capable of executing queries and displaying the results. To achieve this, a PHP enabled Apache web server was set up and a web based interface built upon it. The resulting web site can be viewed in the figures below.



Figure 8: Lime's Homepage

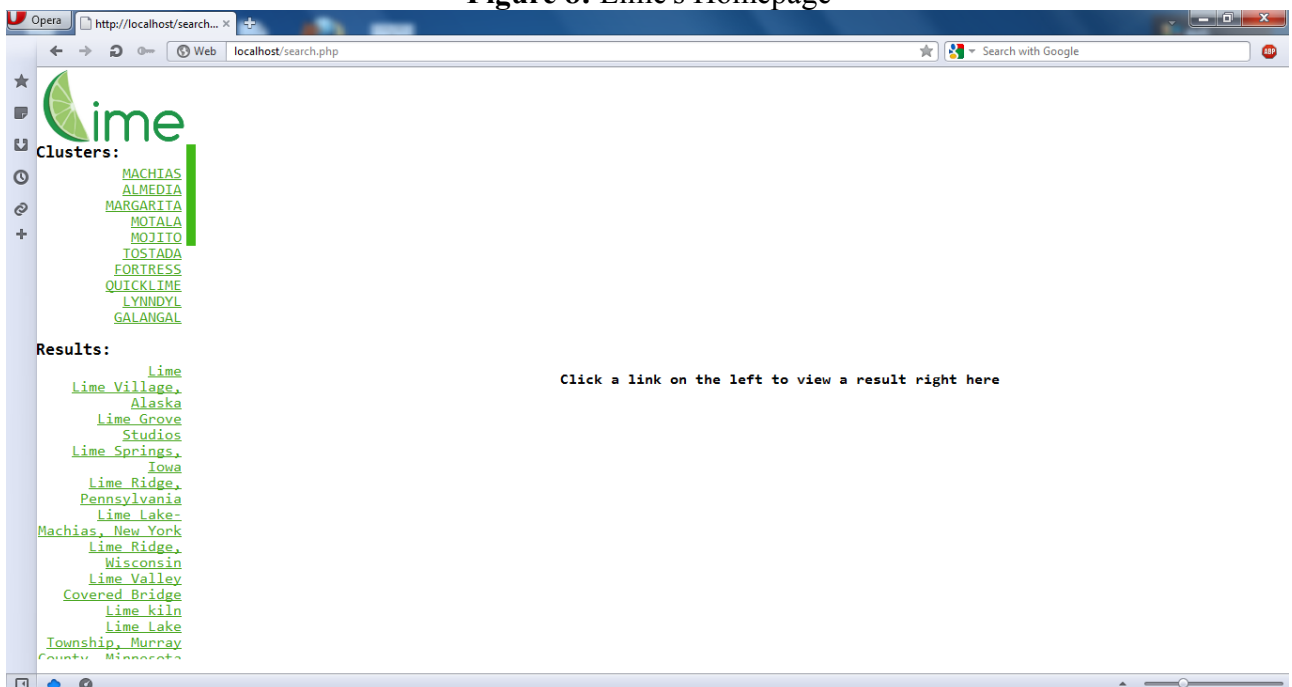


Figure 9: Lime's Results page

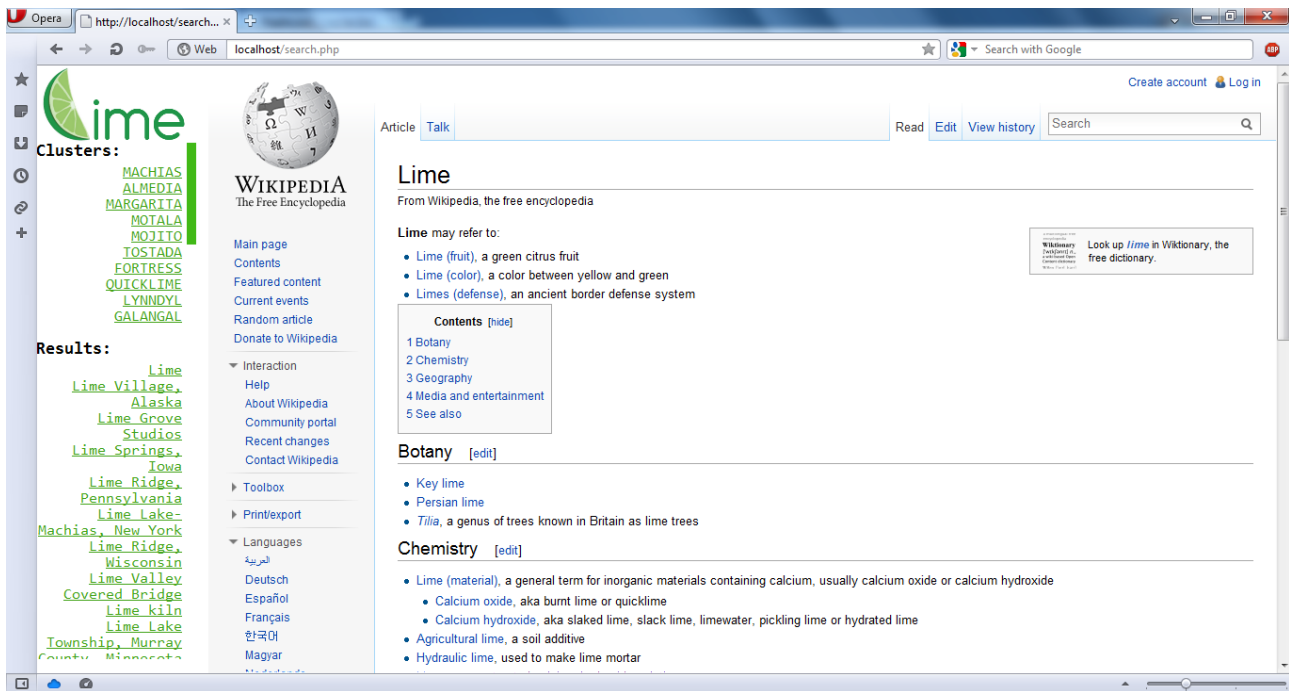


Figure 10: Results page after clicking result in Lime

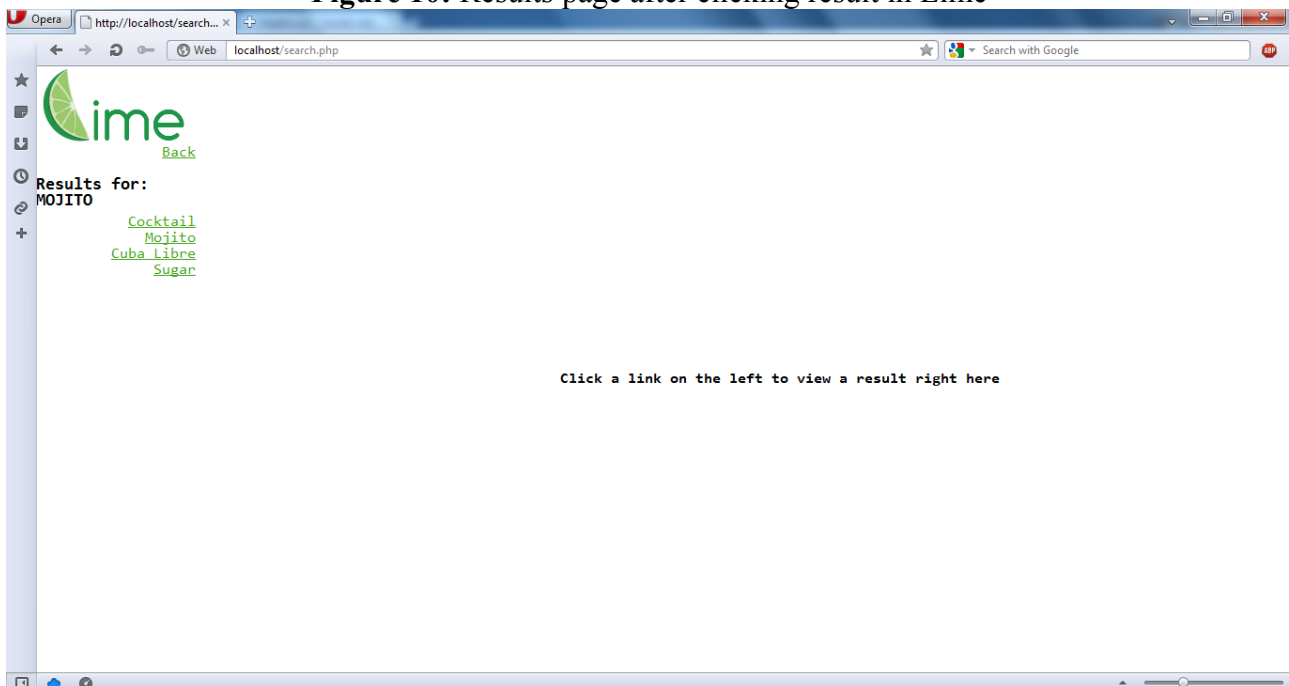


Figure 11: Lime's result page with a cluster selected

As seen by Figure 8 above, the homepage of Lime is quite unremarkable, containing nothing more than a logo and space for query entry. Once a query is executed however a page similar to Figure 9 is shown and clicking a result displays a page similar to Figure 10. Taking inspiration from Windows 8's snap mode, Lime attempts to differ from the typical engine user interface, keeping results within easy reach and the user's sight at all times. Finally Figure 11 shows an example of what the user sees upon clicking a cluster, in this case “Mojito”, and the results that are displayed: Cocktail, Mojito, Cuba Libre and Sugar.

To get the results as shown by these figures requires the use of the index generated by Lime's back end. Upon receiving a query, the engine first takes the user's input and tokenises the string into non-repeating search terms. Each individual term is then used to load results contained by that term's inverted index file. For example when searching for the term “Apple”, the engine's opens the file “A/P/P/Apple.txt”.

Once open, Lime processes the index to extract results. How this is achieved first requires an understanding of the structure of records held within index files. Starting with the first byte of any index file, there is either a value of 0 or 1 flagging whether the first record is a page or a cluster respectively. Based on the value, the records can be processed in sequence as such:

# of Bytes	Purpose
1	Byte set to 0, page record flag
4	Four byte pointer to the start of the page's title string
4	Four byte pointer to the end of the page's title string
8	Double representation of how important this term is to this result (Page's rank)

Figure 12: Structure of a page record

# of Bytes	Purpose
1	Byte set to 1, cluster record flag
∞	Null terminated string of the cluster's name
4	The number of pages within the cluster
8	Double representation of how important this term is to this cluster (Cluster's rank)
At least 1	Byte used as a record type flag: • If set to zero the next 16 bytes are a page record. • If set to one, the next ∞ bytes are a cluster within this cluster (No hierarchical clusters are available in this version of Lime however). • If set to two, this cluster has ended and the next record (page/cluster) is to begin.

Figure 13: Structure of a cluster record

In practice, page records are concrete pointers to a page that can be displayed as a result, while a cluster record can either be displayed by label as a result or processed as a collection of records. Once done processing index files, results for each search term are collected and ranked by total relevance to the query on frequency alone. The results are then output to the user and the execution of the query is complete.

7 Evaluation

With a prototype search engine built, it was now time to evaluate the system. To achieve this, the system would first need to be primed with index data and a clustering threshold. To find a threshold however can rely heavily on guesswork/intuition unless more information is available to make an informed choice. Hence, the clusterer was first run as a debug pass through to export all similarity values between page pairs.

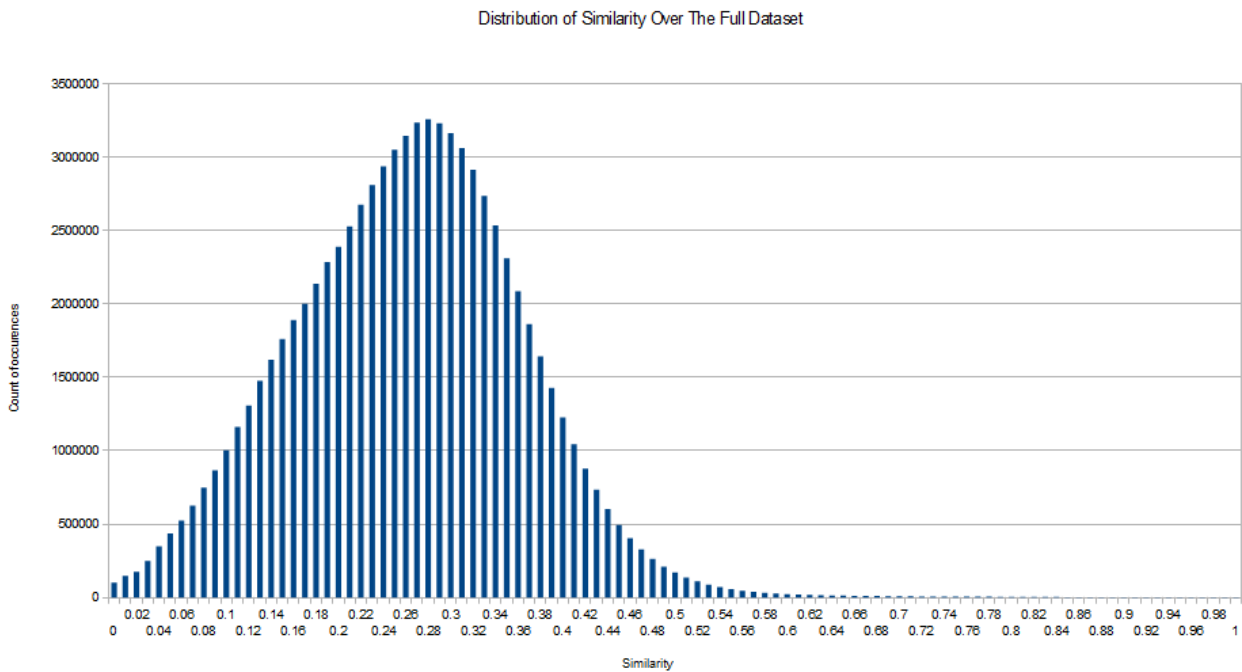


Figure 14: Distribution of similarity values between page pairs

The results of this exportation process can be seen in Figure 14 above. Along the y-axis (ranging from 0 to 3,500,000) is the count of occurrences for each similarity value. The x-axis represents these values, ranging from 0 to 1 in steps of 0.01. This can be seen to peak at 0.29 with a total 3,200,000 occurrences. Using this distribution a threshold of 0.35 was selected for the first incarnation of Lime. This value would allow for the formation of clusters from only the top 20% of pages, a value we believed to be suitable for evaluation.

The resulting execution of the indexer using this threshold value over all 4212492 pages (containing 6404644 unique words), generates 334600 clusters from 117464903 non self-referential links. To get these particular statistics however took three weeks (mainly due to RAM shortage and page faults) and the process is still yet to finish six weeks after it began.

Due to time constraints, action had to be taken to ensure that the system could undergo some form of evaluation, even if limited. There was no choice but to limit the size of the indexer's input by roughly 1/40th, taking the data from 4,212,492 pages down to 100,000.

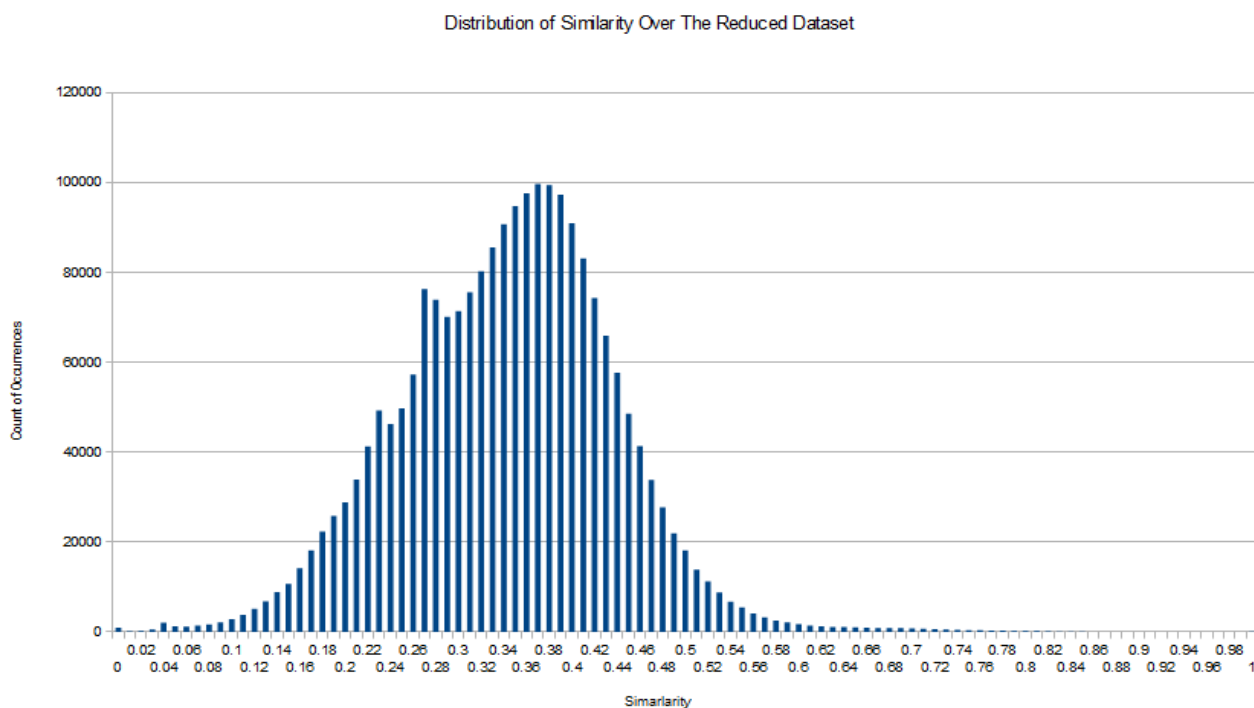


Figure 15: Distribution of similarity values between page pairs for the reduced data set

This in contrast took around three hours to fully execute, generating approximately 112 gigabytes of index data. WRITE STUFF HERE TO DO WITH STATS OF THE RUN AND HOW LONG IT TOOK!

Again, the distribution of similarity within this dataset was generated for inspection and can be seen above. It becomes immediately clear that the new curve is less smoothed out and skewed to the right, with a peak value of 0.38. Regardless, the original chosen threshold of 0.35 was kept to ensure that the data clustering would occur just as if it were a part of the full dataset.

7.1 User Evaluation

With clusters and an index generated it is now possible to evaluate the system's performance. Standard methods to verify clustering in research usually involve internal and/or external validation, but neither would be possible for Lime. Internal validation methods rely on the cross comparison of cluster properties between results of differing algorithms, and external validation requires an expert formed “gold standard”.

To handle this, efforts were made to try and recreate external validation quality evaluation results through the distribution of user surveys. These surveys were given to 30 individuals (mostly students) who were asked to enter a query of their choosing and to rate their experience around four key areas. For each question and rank assigned, 1 represents poor performance and 10 perfect.

To create context/relevance for our results, users were also asked to answer the same set of questions for the search engines Clusty and Carrot². These engines could be focused upon the Wikipedia domain & this allowed us to draw a reasonably fair comparison between all three.

All efforts were made to ensure that no bias existed, with participants told that the survey was to review existing search engines and not told that Lime was our own. Every user was allowed to choose their own query and allowed to explore the website as they wished.

Unfortunately however, this was not enough as once surveying was complete a suspiciously large discrepancy in ratings began to emerge, leading to the discovery of a problem. The search engine Carrot² had been using sources outside of Wikipedia. This led us to believe that the results obtained from surveys, and shown in this document, are not truly reflective of this search engine's quality.

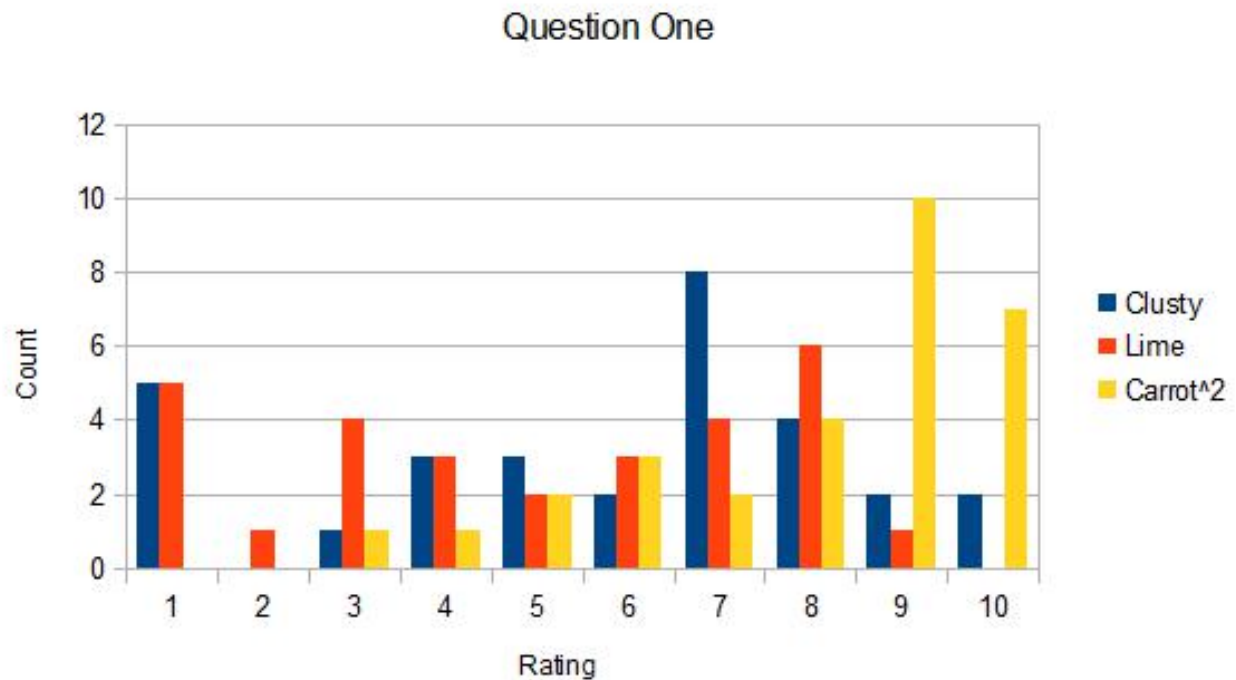


Figure 16: “Looking at the results section after entering your query, how would you rate the results?”

First, we wanted to assess how well all three engines perform when simply running search and discovering results. Beginning with Clusty, we can see that there appears to be a distribution forming around the 7 mark, with a few complete failures to find anything, hinting towards a fairly good recall rate. For Lime however, there appears to be a bi-modal distribution forming with peaks either side of 5. This result, if anything, is expected as only 1/40th of Wikipedia is available in the index, greatly reducing Lime's potential recall rates. Finally, Carrot² however can be seen to be highly successful with most ratings in the higher 9 to 10 region. Again, this is mainly believed to reflect the engine's use of external sources, giving it an unfair advantage.

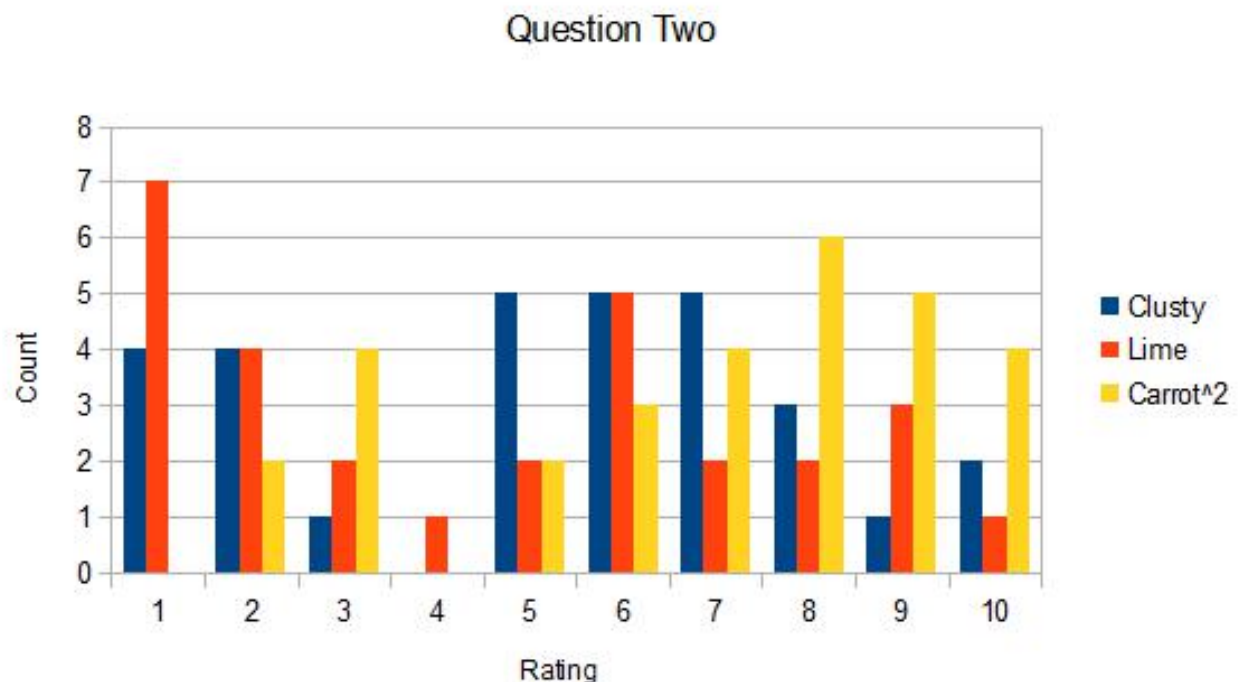


Figure 17: “Looking at the clusters section, how would you rate the labels of the clusters with respect to your query?”

Next, we wanted to assess how suitable generated clusters were to a query. Again, Carrot² can be seen performing better than both of the other engines and Lime slightly worse. This suggests that Lime's label generation method is perhaps not up to par and is a point for future work.

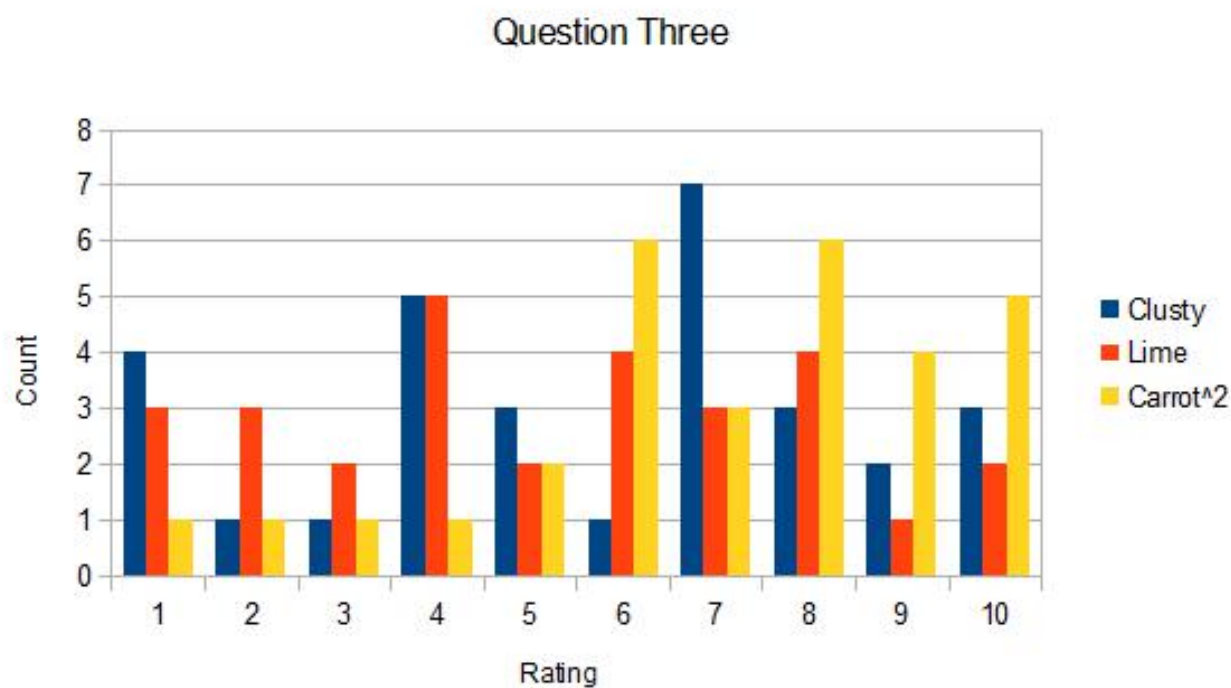


Figure 18: “Select one of these clusters and click it. Then using the newly filtered results, rate how well the cluster's label describes the results it contains”

For the next question, we wanted to assess how well a cluster's label matched with its contents. Here Clusty & Lime's distributions are too spread to draw any valid conclusions, but Carrot² can again be seen to perform well in this area. If anything, this suggests that further surveying of users may be required to discover if any patterns emerge and that experimentation with the employed clustering methods is required.

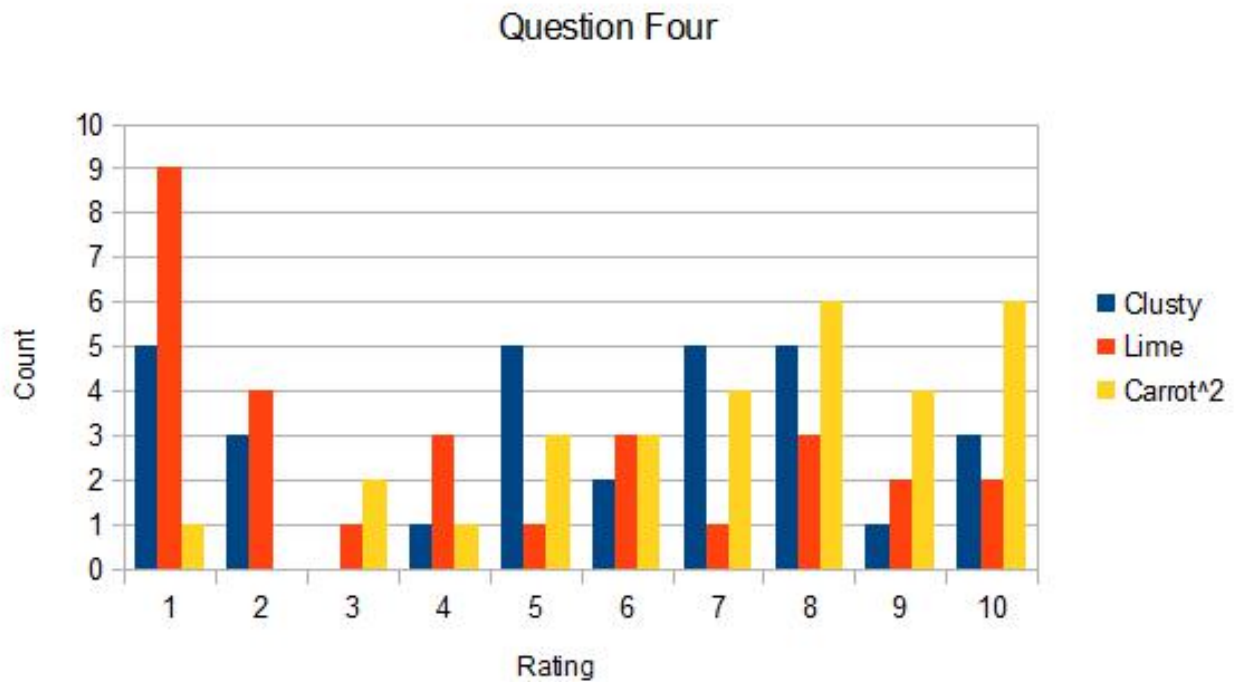


Figure 19: “How well do these filtered results relate to your query?”

Finally, we wanted to find out how related the contents of a cluster were to the user's original query. Here Clusty and Carrot² can be seen to achieve much the same in terms of ratings, showing a strong tendency to provide clusters that contain results immediately relevant to the user's query. Lime's rankings however tell another story.

Here we can see a clear spike towards the lower end of the scale, which we believe, is an effect of Lime's design. When using either Clusty or Carrot², their designs limit cluster formation solely to results found by the initial query, around 250 results on average. For Lime however, this is not the case as clusters are generated from related pages, meaning that results related to the original query by proxy are shown.

As an example, searching Lime for “Alice” generates the cluster “Liddell”, referring to Alice Liddell of Alice In Wonderland fame. Selecting this cluster reveals a collection of related pages that are not however present in the main results set but are still yet relevant, pages such as “Christ Church, Oxford” and “John Ruskin”.

Further examples could be the search for “Fantasy”, which brings up the cluster “Wonderswan” and games related to it (including the franchise “Final Fantasy”); the search for “Apple” containing the cluster “Macromedia”, with pages related to web technologies; and the search “Towel” which generates a cluster of the same title, containing pages ranging from boxing to barbers.

This brings us back to a quote used earlier in the text, “Clouds help you see your search results by topic so you can zero in on exactly what you’re looking for or discover unexpected relationships between items”. Based on these early findings, it would appear that our system somewhat amplifies the ability to discover, creating links that would otherwise be missed.

This could be particularly useful to knowledge workers who are interested in learning about a topic and its surrounding areas. Participant 22 even went as far as to single Lime out for this purpose, stating that they “Might consider using Lime for searching, as this would allow easy of access between multiple pages relating to the same concept. Might be beneficial in research and learning based environments”. Overall, we feel that Lime performed reasonably well and the average scores obtained for each engine are summarised below.

	Clusty	Lime	Carrot ²
Question 1	4.95	4.33	6.61
Question 2	4.59	4.03	5.73
Question 3	4.9	4.5	5.88
Question 4	4.78	3.89	6
Overall	4.8	4.17	6.05

Figure 20: Table of average scores

8 Future Work

There is a lot of scope for future work in this area. First and foremost is a need to test that this approach to search really can scale to a dataset of substantial size and that it can be applied to the Web, a dataset which will most certainly be more unstructured than Wikipedia's archive. Second, there is a need to improve upon the clustering approach used. As a prototype Lime's clustering and labelling algorithms are exceptionally basic, there is likely a great deal to be gained from the application of more advanced concepts to this approach to search. Finally, there is a need to follow research both into how users will use such tools and how or not our prototype's design behaves in comparison to the de facto methodology. Without such information, it will be impossible to know the effects cluster-based search will generate with relation to time consumption and enabling information discovery, and ultimately whether our approach once fine-tuned is actually capable of providing a higher quality of service.

9 Conclusion

To conclude, it has been shown that the World Wide Web and how we use it has experienced incredible amounts of change. In order to meet this change, search engines have by necessity had to adapt and improve. One of these potential improvements uses clustering technologies in an attempt to improve upon current standards. All efforts so far however have been limited by design and their ability to scale is greatly hindered, leading to the simple questions of "Why?" and "Can it not be done differently?". To answer this, the prototype search engine Lime was created. This proof of concept is capable of scaling many times over and has the potential to enable a new era of information discovery. There is still a great deal of work yet to be done in this area however and only time will tell of the results.

10 References

All websites last known as active 16th November 2012

- [1] <http://www.worldwidewebsite.com/>
Website that tracks the size of the web
- [2] <http://www.optify.net/inbound-marketing-resources/new-study-how-the-new-face-of-serps-has-altered-the-ctr-curve>
Study by Optify analysing the click through rates by a page's ranked position
- [3] <http://www.statisticbrain.com/google-searches/>
Google search statistics
- [4] <http://web.archive.org/web/20091206045041/http://www.googlegooglegooglegoogle.co.uk/>
Google x 4 as an alternate interface
- [5] <http://www.collarity.com/>
Search engine that combines multiple search mediums
- [6] <http://www.managedq.com/>
Search engine that returns thumbnails of the pages found
- [7] *Text Entry for Mobile Computing: Models and Methods, Theory and Practice - I. Scott*

- MacKenzie and R. William Soukoreff, 2002, page 3 3rd paragraph
- [8] <http://www.google.co.uk/mobile/goggles/#text>
Google Goggles
 - [9] <http://www.tineye.com/>
Tineye search engine
 - [10] Blumberg, Brad W., and Eric M. Blumberg. "Mobile location aware search engine and method of providing content for same." U.S. Patent No. 7,599,795. 6 Oct. 2009.
 - [11] <http://www.google.com/glass/start/>
Homepage of Google Glass
 - [12] <http://www.apple.com/uk/ios/siri/>
Homepage of Siri
 - [13] <http://www.wolframalpha.com/>
Wolfram Alpha's homepage
 - [14] <https://www.facebook.com/about/graphsearch>
Homepage of Facebook Graph
 - [15] [http://en.wikipedia.org/wiki/Tiger_\(disambiguation\)](http://en.wikipedia.org/wiki/Tiger_(disambiguation))
Examples of just how many types of "Tiger" there can be
 - [16] WHITE PAPER, The Hidden Costs of Information Work, Sponsored by: Content Technologies Multiclient Study, Susan Feldman, Joshua Duhl, Julie Rahal Marobella, Alison Crawford March 2005
 - [17] Cluster Based Web Search Using Support Vector Machine, Rita. S. Shelke, Devendra Singh Thakore
 - [18] <http://search.yippy.com/about-yippy-search>
Information taken from Clusty's About Page
 - [19] A Survey of Web Clustering Engines, Claudio Carpineto, Stanislaw Osinski, Giovanni Romano, Dawid Weiss
 - [20] <http://sourceforge.net/projects/archive-crawler/develop>
Heretrix Download Page
 - [21] <https://webarchive.jira.com/wiki/display/Heritrix/Heritrix>
Heretrix Quote from their own Wikipage
 - [22] <http://blog.archive.org/2013/01/09/updated-wayback/>
Internet Archive Statistics
 - [23] http://en.wikipedia.org/wiki/Wikipedia:Database_download
Wikipedia Download Page
 - [24] http://www.mediawiki.org/wiki/Alternative_parsers
Wiki Format Parsers
 - [25] <http://code.google.com/p/gwtwiki/wiki/Mediawiki2HTML>
Mediawiki2HTML
 - [26] Clustering-based Incremental Web Crawling, Qingzhao Tan and Prasenjit Mitra, The Pennsylvania State University
 - [27] Precision and Relative Recall of Search Engines: A Comparative Study of Google and Yahoo, B.T. Sampath Kumar, J.N. Prakash
 - [28] Modern Information Retrieval, the concepts and technology behind search, Second edition, Ricardo Baeza Yates & Berthier Ribeiro-Neto, page 2
 - [29] <http://blog.kissmetrics.com/wp-content/uploads/2011/04/loading-time-lrg.jpg> Infographic of "second counting" and its importance on the web
 - [30] Cluster Based Web Search Using Support Vector Machine, Rita. S. Shelke, Devendra Singh Thakore
 - [31] Inducing Word Senses to Improve Web Search Result Clustering, Roberto Navigli and Giuseppe Crisafulli
 - [32] Clustering Web Search Results with Maximum Spanning Trees, Antonio Di Marco and Roberto Navigli

- [33] *A Concept-Driven Algorithm for Clustering Search Results*, Stanislaw Osinski and Dawid Weiss
- [34] *Criteria for Cluster Based Personalized Search*, Hyun Chul Lee and Allan Borodin
- [35] *Improving Web-based Image Search via Content Based Clustering*, Nadav Ben-Haim, Boris Babenko and Serge Belongie

11 Appendices

[A] List of found working cluster based search engines. This list is by no means exhaustive and more than likely has changed since writing:

- <http://search.yahoo.com/> Select “Directory” option, provides list based results with cluster based filters
- <http://clusty.com/> provides list based results with cluster based filters
- <http://www.mnemo.org/> provides a graph of clusters for filtering
- <http://www.polymeta.com/> provides list based results with cluster based filters
- <http://search.carrot2.org/stable/search> provides list based results with cluster based filters
- <http://search.yippy.com/> pseudonym for Clusty

[B] Generalised Wikipedia XML file:

```
<mediawiki xmlns="http://www.mediawiki.org/xml/export-0.8/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.mediawiki.org/xml/export-0.8/
http://www.mediawiki.org/xml/export-0.8.xsd" version="0.8" xml:lang="en">
<siteinfo>
<sitename>Wikipedia</sitename>
<base>http://en.wikipedia.org/wiki/Main_Page</base>
<generator>MediaWiki 1.21wmf8</generator>
<case>first-letter</case>
<namespaces>
<namespace key="-2" case="first-letter">Media</namespace>
<namespace key="-1" case="first-letter">Special</namespace>
<namespace key="0" case="first-letter" />
<namespace key="1" case="first-letter">Talk</namespace>
<namespace key="2" case="first-letter">User</namespace>
<namespace key="3" case="first-letter">User talk</namespace>
<namespace key="4" case="first-letter">Wikipedia</namespace>
<namespace key="5" case="first-letter">Wikipedia talk</namespace>
<namespace key="6" case="first-letter">File</namespace>
<namespace key="7" case="first-letter">File talk</namespace>
<namespace key="8" case="first-letter">MediaWiki</namespace>
<namespace key="9" case="first-letter">MediaWiki talk</namespace>
<namespace key="10" case="first-letter">Template</namespace>
<namespace key="11" case="first-letter">Template talk</namespace>
<namespace key="12" case="first-letter">Help</namespace>
<namespace key="13" case="first-letter">Help talk</namespace>
<namespace key="14" case="first-letter">Category</namespace>
<namespace key="15" case="first-letter">Category talk</namespace>
<namespace key="100" case="first-letter">Portal</namespace>
<namespace key="101" case="first-letter">Portal talk</namespace>
<namespace key="108" case="first-letter">Book</namespace>
<namespace key="109" case="first-letter">Book talk</namespace>
```

```

    <namespace key="446" case="first-letter">Education Program</namespace>
    <namespace key="447" case="first-letter">Education Program talk</namespace>
    <namespace key="710" case="first-letter">TimedText</namespace>
    <namespace key="711" case="first-letter">TimedText talk</namespace>
  </namespaces>
</siteinfo>
<page>.....</page>
<page>.....</page>
<page>.....</page>
<page>.....</page>
.
.
.
<page>.....</page>
<page>.....</page>
</mediawiki>

```

[C] Generalised Wikipedia Page:

```

<page>
  <title>Anarchism</title>
  <ns>0</ns>
  <id>12</id>
  <revision>
    <id>536066461</id>
    <parentid>536058767</parentid>
    <timestamp>2013-02-01T19:07:40Z</timestamp>
    <contributor>
      <username>Eduen</username>
      <id>7527773</id>
    </contributor>
    <comment>issues in discussion. see talk section</comment>
    <text xml:space="preserve">.....</text>
    <sha1>f0ry35mqvsk6mbfq2d30rkfl ewixkx7</sha1>
    <model>wikitext</model>
    <format>text/x-wiki</format>
  </revision>
</page>
<page>
  <title>Wikipedia:Building Wikipedia membership</title>
  <ns>4</ns>
  <id>4135</id>
  <revision>
    <id>429291515</id>
    <parentid>428200536</parentid>
    <timestamp>2011-05-15T21:00:33Z</timestamp>
    <contributor>
      <username>Hawkseraph</username>
      <id>12367887</id>
    </contributor>
    <comment>Undid revision 428200536 by [[Special:Contributions/Wasami007|Wasami007]]
    ([[User talk:Wasami007|talk]])</comment>
  </revision>
</page>

```

```
<text xml:space="preserve">.....</text>
<sha1>oxxkq6ogncaoscb3kgfplwclfj2982o</sha1>
<model>wikitext</model>
<format>text/x-wiki</format>
</revision>
</page>
```