**CSE 454 – University of Washington – Autumn, 2009**

**The Paperazzi**

**A Visual Search Engine for Research Papers**

*By Toshinao Iwaki, Tim Hao Li, Martin Pettersson, Patrick Tsao*

# Table of Contents

[Table of Contents 2](#_Toc248682180)

[Background 3](#_Toc248682181)

[Project Goals 3](#_Toc248682182)

[The Concept 3](#_Toc248682183)

[The Paperazzi 4](#_Toc248682184)

[Usage Scenarios 4](#_Toc248682185)

[Pan Around and Explore 4](#_Toc248682186)

[Search for Relevant Papers 4](#_Toc248682187)

[Get Detailed Information on a Paper 5](#_Toc248682188)

[See Which Papers Cited This Paper 5](#_Toc248682189)

[System Design 6](#_Toc248682190)

[Algorithmic Choices 7](#_Toc248682191)

[Discrete Spatial Partitioning 7](#_Toc248682192)

[Determining Paper Relevance and Importance 10](#_Toc248682193)

[Experiments 11](#_Toc248682194)

[Performance Testing 11](#_Toc248682195)

[Usability Testing 14](#_Toc248682196)

[Feedback: General 14](#_Toc248682197)

[Feedback: Specific to Prototype #1 15](#_Toc248682198)

[Feedback: Specific to Prototype #2 15](#_Toc248682199)

[Feature Suggestions 15](#_Toc248682200)

[Conclusion 16](#_Toc248682201)

[Lessons Learned 16](#_Toc248682202)

[Ideas for Future Work 16](#_Toc248682203)

[Appendices 17](#_Toc248682204)

[Work Division 17](#_Toc248682205)

[Group Dynamics 17](#_Toc248682206)

[External Code Used 18](#_Toc248682207)

[Instructions for Access 18](#_Toc248682208)

# Background

## Project Goals

Our goal is to revolutionize the user interaction model behind searching for research papers. Despite the market dominance Google has achieved in the online advertising and search business, they have shown limited innovation on the human-computer interaction aspect of search. Users still interact with Google.com and its various renditions largely the same way – type in a query and receive a list of links on a white page back. We seek to transcend this model by providing users with a visual search.

However, given the limited timeframe and resources we have available for this quarter, we have refined our to-dos to the following list:

1. Acquire a database of publications
2. Create an index out of this database
3. Design an innovative interaction model for search and discovery of papers
4. Perform usability studies to elicit feedback on interaction
5. Implement and deploy a web application using the new interaction model

We believe that successful implementation of all of the features above will enable us to deploy a barebones version of a visual search engine for research papers. The Paperazzi will demonstrate an innovative way to not only search for relevant papers given a query, but also explore and discover relevant papers beyond the single line of text that the user types in.

## The Concept

In essence, the visual search interaction model that we have designed is a giant graph. Each node in the graph represents a single paper and each edge represents a citation from one paper to another.

This new UI paradigm is interesting in several ways:

* **It can be useful without requiring the user to enter a query.** Upon initial load of The Paperazzi, papers with the most inbound citations are immediately displayed. Users can use this to quickly identify “important” papers around the map and zoom in or out to adjust the number of inbound citations threshold to display more or less papers.
* **It allows for a large number of papers to be displayed at once**. By default, Google shows about 10 results for any given query above the fold. The Paperazzi has the capacity to show users hundreds of results at once while taking up a relatively small amount of screen real estate.
* **The positions of papers are related to each other beyond “relevance.”** When users examine papers on The Paperazzi, they are able to easily navigate between relevant papers because they are typically located near each other on the map.
* **It enables a new scenario beyond search: discovery.** In addition to typing a query and getting a list of results back, users can now look at any single result and find papers relevant to that result instead of the query. This helps if the user does not know exactly what he or she is looking for.

# The Paperazzi

## Usage Scenarios

The Paperazzi currently supports the following usage scenarios:

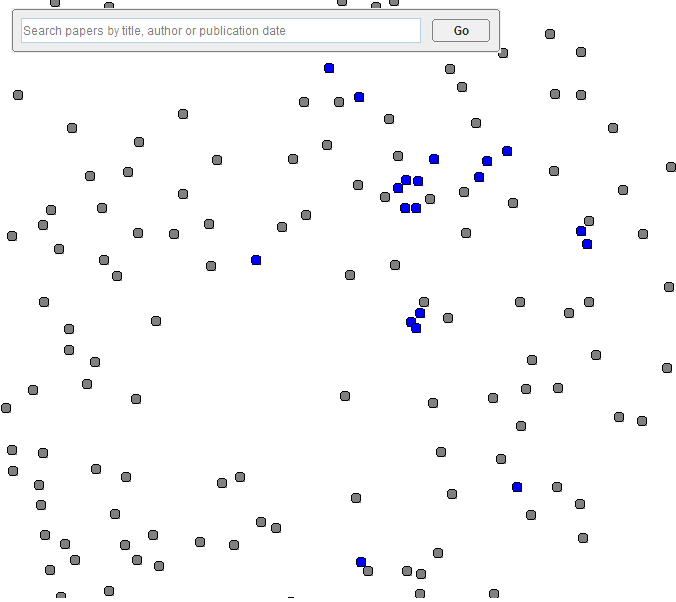
### Pan Around and Explore

**Screenshot 1**

Upon the initial load of The Paperazzi, the applet is filled with gray bubbles along with a text box and button in the top left corner for search (see Screenshot 1). Each bubble on this map-like interface represents a paper. The map is split into many sections and each section has a set of X, Y, and Z coordinates. The Z coordinate determines the threshold of “importance” in order for a paper to show up in that section. “Importance” is defined as the number of inbound citations on any given paper, where more citations make the paper more “important”. For example, the default zoom level has a Z coordinate of 1, so only the most important papers are shown when the applet loads. At this stage, the user can pan around the map and zoom in or out to adjust importance. On hover, each bubble will show its title, and on click, more detailed information about that paper will appear. This is a very free-form, discovery-oriented scenario that The Paperazzi allows the users to perform.

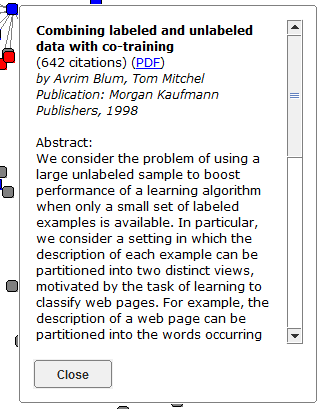
### Search for Relevant Papers

**Screenshot 2**

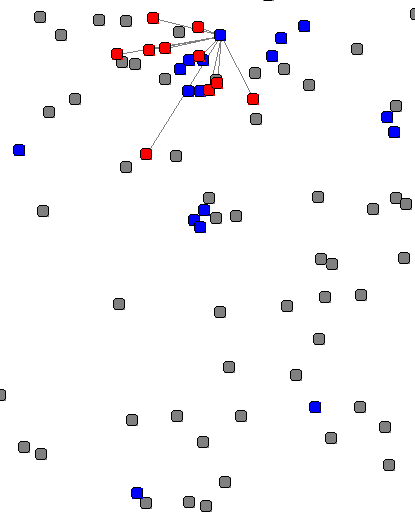
After all, The Paperazzi is a visual search engine for research papers. Users can type a search query into the text box on the top left corner, and then hit “Enter” or click the “Go” button. This sends the query back to the server and looks up relevant paper within the index. Then, a set of bubbles on the screen will turn blue. These represent the search results that are returned from the index given the query that the user typed in. Similar to before, the user can choose to hover or click on any one of the results (or any of the non-results) to get more information on each paper. As a result of the way X-Y positions of each paper is calculated, search results tend to be relatively close to each other. See Screenshot 2 to the right for a demonstration of what this looks like.

### Get Detailed Information on a Paper

**Screenshot 3**

If the user is interested in getting detailed information on a paper, he or she can click on the bubble, and a dialog will appear (see Screenshot 3). The dialog includes information such as the paper title, number of citations, authors, publisher, publishing date, the abstract, as well as a link to the PDF of that paper. The user can choose to click on the PDF link if he or she finds the paper interesting after reading some or the entire abstract, opening a new tab to the detailed CiteSeerX page for that paper. If this paper is not of interest, he or she can click on the “Close” button and move on to clicking or hovering on other bubbles.

### See Which Papers Cited This Paper

The last but certainly not least scenario allows users to see which papers cited any given paper. When a user clicks on a bubble, the dialog shown in the previous scenario will appear. At same time, a set of bubbles will appear and turn red. A line will also be drawn from all of these red papers to the paper that the user clicked on (see Screenshot 4). The red bubbles represent papers that formally cited the paper that was clicked on and tend to be located very close to each other on the map. This feature allows users to easily navigate between relevant papers, as citation is a good indicator for relatedness.

**Screenshot 4**

The above mentioned scenarios are not meant to be performed in any specific order. In fact, someone using The Paperazzi to search for papers can utilize any of these scenarios at any given time during his or her search to effectively browse through, search for, and discover papers.

## System Design

The design of The Paperazzi can be effectively divided into two parts: index creation and search. In order to create the index of research papers, we needed the research paper data. To obtain the data, we used the OAI Harvester provided by OCLC to harvest the CiteSeerX database and save it in XML format. The data provides a variety of information about each paper, including title, authors, the abstract, publisher, publishing date, a link to the paper in PDF format, and IDs of other papers citing this one. Then, we used Lucene to create the index, enabling searches on all of the fields mentioned above.

Once we have the index, we moved forward with building the web application that will allow users to search for research papers visually. We decided that a zoom-able canvas interface will be best for presenting users with a large number of papers clustered by relevance, so we chose to use a library called ZVTM (Zoom-able Visual Transformation Machine.) Martin has done prior work with this library and is familiar with the APIs available. It integrates well with Java Applets and the Swing UI library, so we built a Java Servlet as the backend to communicate with the index. Lucene is also written completely in Java, so the integration between the index and our servlet was reasonably straight forward.

The diagram below is a summary of how The Paperazzi was designed:



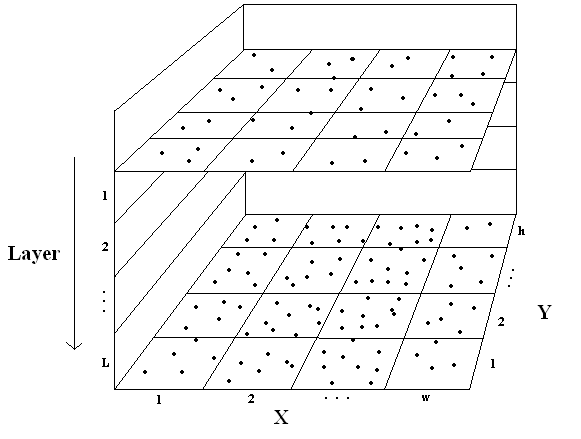
## Algorithmic Choices

### Discrete Spatial Partitioning

Having several hundred thousand papers in our data, we do not want the user to have to download all this data each time it loads. Keeping all the papers in memory and displaying them on the surface would also make it incredibly slow and cluttered. We resolve this problem by:

1. Loading only the papers that are visible within the view space of the camera
2. Load more papers as the user zooms in, displaying the most important papers first, and adding detail further down
3. Controlling approximately how many papers are visible at once

Our solution to this is to partition all the papers by using a grid cube, where the horizontal grid represents the spatial partitioning and the depth of the zoom level. Since zooming in will make the viewing area smaller, we intuitively add more papers as the zoom level increases, while keeping the same papers that were present at smaller zoom levels. Our algorithm works by first calculating the needed width, height and depth of the grid cube along with the needed paper densities for each zoom level to achieve an approximately constant amount of visible papers at one time. With this partitioning, we then have an algorithm for assigning papers to the right section based on importance.

*An important assumption in our calculations is that papers are uniformly distributed over the 2D space, which is definitely not true. However, the assumption should give a good approximation and makes the calculations easier.*

Input:

w = width of map

h = height of map

T = total number of papers

P = maximum number of papers to be displayed at one time

L= desired number of zoom levels

Output:

= width of grid

= height of grid

= list of densities for each layer (papers/square)

The number of papers displayed at a layer k is the sum over every layer above it. For layer k, we define the density as, the area as  and the number of papers visible as. We end up with:



We want the number of papers displayed at every layer to be a constant P



With this we can calculate the required density for every layer as



=> 

=> 

...

=> 

We also have the restriction that the sum of the density multiplied by the area of every layer equals T, the total number of papers we have:

(1)



We define the area shown at each layer recursively with a constant 0 < R < 1 that makes the area smaller every layer:



For this equation we have the edge cases:

and 

We then have that

=> 

Given the camera view space width and the view space aspect ratio we also have that => which is useful when we want to know the width and height of the camera view space.

To find a good number and densities  so that we find a good partitioning, we take the restriction in (1) and minimize the distance between it and T. We can’t solve it directly since an exact solution might not be available. When we've minimized it, we can be sure that the densities we get will provide a good partitioning.

Minimize forN

This will give ,and , the densities for each layer.

Now that we have the size of the partitioning space, as well as the densities per layer, we can start assigning papers to each section. We do that with the following algorithm:

For every square (x, y) in grid:

Find all papers within that square

Sort the papers by importance

For every layer k:

Select the  topmost papers and assign them to the section (x, y, k)

With probability  assign one additional paper to the section (x, y, k)

Remove the added papers from the list

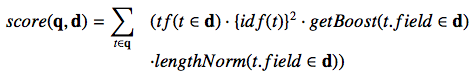
We implemented this algorithm in Matlab and it ran fairly quickly. However, this algorithm will only be run once to calculate the positions for each dataset. Thus, speed is less of an issue here.

### Determining Paper Relevance and Importance

This section includes significant content from Apache’s documentation of Lucene:

<http://lucene.apache.org/java/2_4_0/api/org/apache/lucene/search/Similarity.html>

Lucene uses the following formula to calculate the relevance (score) of each document in the data:



Where:

**tf** = term frequency in a document. Terms and phrases repeated in a document indicate the topic of the document, so implementations of this method usually return larger values when freq is large, and smaller values when freq is small..

**idf** = inverse document frequency, which is the number of documents that contain the term. Terms that occur in fewer documents are better indicators of the relevant topic; this method should return a larger value for a rare term and smaller value for common term.

**getBoost (t in q)** = the boost for the term in query. The scores of the documents matching this term will be multiplied by b, which has a default value of 1.0.

**lengthNorm(t.field in d)** = normalization value for a field given the total number of terms within a field. The function should return a smaller value when the matches occur in longer fields, and return a larger value when the matches occur in shorter fields. These values are stored in the indices.

When Paperazzi creates indices of the data, it stores the number of inbound citations for each document.  For our purposes, Paperazzi first sorts the search results using the score formula that lucene provides.  Paperazzi will then rank the results by the number of inbound citations: the more inbound citations a paper has, the higher its rank will be.

# Experiments

## Performance Testing

For our load time benchmark, six trials were run on every category and the average is presented on the graph. However, the first trial usually takes a long time because of storage accessing. Therefore, we do not take the first trial into account in every category; instead, we take the average of trials two to seven.

Below, Figure 1 shows the load time for an average use of different features of our Paperazzi search engine. Notice that initial homepage loading time takes the longest because it often has to start up the Java Virtual Machine in the browser if it has not done so.



Figure 1

We then proceeded to compare the load time between having to start up JVM versus not having to start up JVM. The comparison in Figure 2 shows that the load time doubles when JVM start time is counted.



Figure 2

It makes sense that the longer the query, the longer it takes to return the results since we have to search on every term in a query so the search time accumulates. As predicted, Figure 3 shows a pattern of search time based on the number of terms in the query. It appears that the graph follows a linear relation, which follows our intuitions. Since we have a constant search time given the index, the number of terms decides the coefficient of this constant search time.

Now we examine the load time for individual papers with different numbers of citations. We would like to make this comparison because we actually do not load all the papers that reference the paper we are looking at because of a potential heavy load on the applet by displaying all citing papers. Instead, we only show the top 20 citing papers sorted on the number of citations they each have. We use merge sort in our code and are guaranteed to have nlog(n) sorting time. But it is still less efficient than constant time, as shown by the jump in Figure 4 from below 200 milliseconds to more than 1000 milliseconds between papers with 11 and 1493 citations. Figure 4 shows three papers that have 1493, 11, and 0 citations, respectively. The choice of 1493 and 0 is to show extreme cases of papers and see their performances while looking at the benchmark of a somewhat normal paper with about 10 citations.

Figure 3



Figure 4

The last loading scenario occurs when we switch views either by zooming in and out or scrolling. This is achieved by loading different sections of the whole graph. Since we also index on the section number for each paper, section retrieval is constant time and there is little to no difference in retrieving various sections. We can observe this phenomenon by looking at Figure 5.

Figure 5

## Usability Testing

We created two different UI prototypes of The Paperazzi. Prototype #1 behaves similar to most existing search engines. When the page loads, the applet contains only the search box and button for users to conduct a search. Once the user submits a query, papers that are returned as results will show up in the graph. Prototype #2, the design we ultimately decided to implement, begins with the graph showing all papers with the most number of inbound citations and highlights search results when queries are submitted. The prototypes can be found at this location:

<http://students.washington.edu/pattsao/paperazzi>

The table below summarizes the unique characteristics of each of the three users we tested the two different prototypes with.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User # | Gender | Age | Field of Study | Level of Experience with Usability |
| 1 | Male | 22 | CS | HCI concentration, work experience with UI design |
| 2 | Female | 22 | Finance / IS | Little to none |
| 3 | Female | 24 | HCDE | User-centered design concentration |

### Feedback: General

* The graph-based model is pretty intuitive
  + Reminds users of Google Maps
* UI is simple and clean
* Hesitated at first, not sure what to do
  + Would like to see some quick help or FAQ
* The faceted UI using sliders would be a very useful feature
* “Trim Leaves” is confusing, no one knew what that was supposed to mean
* People lose track of which bubble’s they’ve previously clicked on
  + Some feedback is needed so users know
* It is annoying to have to close one description box to open another
* Being able to see related papers is useful
  + However, this should not come at the expense of seeing the title and keywords related to any given search result
  + A list of ranked results on the side of the graph seems necessary
* It would be nice if one could drag description boxes around
  + User wants to open several of these boxes and then drag them into some place
  + Perhaps go through them one by one later to see if the paper is interesting
* It is nice to be able to quickly see the abstract of the paper
* The spatial relationship between papers is very unclear
* The query terms should stay in the search box after clicking “Go”

### Feedback: Specific to Prototype #1

* It is very clear that the bubbles mean search results, since it appears only after querying
* This design does not allow users to pan and explore as much
  + Users must query to do anything interesting with the graph
  + This somewhat defeats the purpose of having a graph
* The page feels a little bit empty

### Feedback: Specific to Prototype #2

* A bit overwhelming to see all of those bubbles right away
* It is clear that the highlighted bubbles mean search results
* However, all those other nodes add visual noise
  + The UI feels “muddy” because there are so many bubbles that I don’t care about

### Feature Suggestions

* Add a “recently viewed” panel to show you which papers you’ve clicked on before
* Enable selection of multiple papers
* Add a “download PDFs of selected” to download a collection of papers at once
* “Recommended Papers” feature similar to Amazon’s product recommendations
* Hovering on an author’s name should provide information on him or her
  + Could be author background, credentials, influential papers, etc
* Clicking on an author’s name should generate a new query returning all papers by this author
* Add a Quick-help or FAQ page
* Implement drag-able description boxes
* Use thumbnails of papers instead of just bubble

In the end, we decided to implement the interaction model represented by Prototype #2. We wanted to build a system that allows usage scenarios beyond the traditional query-and-get-results model. Prototype #1 has a simpler interaction model, but it does not allow users to discover papers on their own as much. It is much closer to the way existing search engines work. We are very interested in seeing how users will react to the new discovery usage scenario.

# Conclusion

## Lessons Learned

* **Data harvesting is surprisingly hard.** While working with the CiteSeerX database, we experienced many difficulties in trying to harvest their data. The link to a harvester they supplied was dead, and we were unable to get our hands on a good harvester until Patrick called OCLC. The harvester we got worked well, as long as the connection stayed up. The CiteSeerX database often goes down or refuses our connections, forcing us to restart.
* **There are a lot of useful and well-documented open sources projects and libraries.** It is amazing what one can do by simply aggregating a set of open source code. We are really glad we chose to work with Lucene; it was highly educational for us.
* **Lucene has a new release every 2 weeks.** We were surprised to find that several weeks after we began changing the Lucene source code, we were already 2 releases behind the most recent version. We were careful to keep track of which version of each library we were using.
* **Don't always trust open source code.** Developers in the open source community are no different than we are. They make mistakes just like we do, meaning that debugging should include the code that came with open source packages and not limited to our own code.
* **Don't be afraid to fix bugs on open source software.** If there is a found bug in the open source code, don’t be afraid to fix it. In fact, we encourage anyone to take this a step further and contact the developers in the open source community so that they can fix the bug for the next release of the software.
* **Use sample code!** The sample code that comes with open source software often serves as a great demonstration and offers insight into how the code works as well as how to integrate it into our own code. This was something that helped us tremendously throughout the quarter.

## Ideas for Future Work

Due to time and resource constraints, our team was only able to implement the graph consisting of papers as nodes. However, one could also imagine a more author-centric model, where there are “author nodes” and “paper nodes” in the graph. Each paper is the child of all of its authors, and each author is linked to all of the papers by him or her. In this case, links between papers still represent a citation, but links between authors can represent collaboration or mentorship. Here are some ideas for what other kinds of nodes can be added to The Paperazzi to extend its functionalities:

* Authors
* Area of study (subject)
* Research Organization (corporate or academic)
* Conference
* Publisher

Obviously, there is a lot of room for further innovative work here. We believe that although we have only scratched the surface of the possibilities of visual search with The Paperazzi, this project represents a good first step in the direction of shying away from receiving only lists of links as search results.

# Appendices

## Work Division

**Toshinao Iwaki**

* Parsing of initial Cora data into MySQL (not used in the end)
* Relevance algorithm and implementation
* Expanding servlet APIs to support queries

**Tim (Hao) Li**

* Harvesting XML data
* Build index out of XML data
* Performance testing
* Index –servlet integration

**Martin Pettersson**

* UI (applet) implementation
* Basic implementation of servlet
* All integration features
* High-level architecture design

**Patrick Tsao**

* Initial setup of environment (Apache Tomcat, Lucene, etc)
* High-level architecture design
* UI design and mock-up
* Usability tests

## Group Dynamics

The team got along and worked very well together. The work was broken down fairly effectively, and everyone contributed a significant amount. We set deadlines for ourselves throughout the quarter, and most of them were met. Regular meetings were held at least once or twice a week to discuss any new problems or issues that arose. Major decisions were made in a democratic manner, involving in-depth discussions at group meetings. Every group member was given the opportunity to voice his opinion and/or propose a solution. A consensus had to be reached before any major decision was made.

We faced some difficulties as a group as well. It was often hard to schedule a time for everyone to show up and do work together because each member has a different set of schedules. Thus, we divided the work in such a way that coupling between different pieces of the system was minimized. At times, it also seems like the group gets a long a little bit too well. Maintaining focus during meetings and not going off topic was a challenge. Regardless, we feel like we did an excellent job as a group and were able to complete and deploy an interesting visual search engine as our project.

## External Code Used

**Apache Lucene**

* <http://lucene.apache.org/java/docs/>

**Apache Tomcat**

* <http://tomcat.apache.org/>

**CiteSeerX Beta**

* <http://citeseerx.ist.psu.edu/>

**ZVTM (Zoom-able Visual Transformation Machine)**

* <http://zvtm.sourceforge.net/>

**Java Development Kit**

* <http://java.sun.com/javase/downloads/index.jsp>

**OCLC OAI Harvester2 (Java)**

* [http://www.oclc.org/research/activities/past/orprojects//harvester2/harvester2.htm](http://www.oclc.org/research/activities/past/orprojects/harvester2/harvester2.htm)

## Instructions for Access

Visit the following URL:

<http://rosina.cs.washington.edu:8080/>

NOTE: you will need to use a Java-enabled web browser with the latest JRE installed.