Università
della
Svizzera
italiana

Faculty
of Informatics

Bachelor Thesis

August 27, 2020

VisualDBLP

A tool to visualize how researchers collaborate over time

Piermarco Barbè

Abstract

Websites such as dblp provide archives of computer science publications. However, finding information about how different researchers collaborate between themselves is not intuitive.
Creating an animated co-authorship graph is even harder, and an automatic way of creating such a graph does not exist. Visualizing this information can support the user of such a platform understand how a specific set of author cooperated over time.
For this purpose we developed VisualDBLP: A web-based platform that provides an evolutionary co-authorship graph depicting how a set of researchers collaborate over time. In this graph the nodes represent the authors and the edge between two authors shows, if present, the co-authorship relation between two authors. The radius of a node will reflect the number of publications of the author it represents.

Advisor
Prof. Dr. Michele Lanza
Assistant
Dr. Roberto Minelli

Advisor's approval (Prof. Dr. Michele Lanza): Date:

Contents

1 Introduction

The dblp[1] archive is an on-line reference for bibliographic information on major computer science publications. dblp was created in 1993 and initially it was served using the NCSA HTTP Server, which was a technology that was getting popular in those years. The very first documents published on dblp were few table of contents regarding Data Bases and Logic Programming, which gave the acronym that still survives today: DBLP.

1.1 dblp: Computer Science Bibliography

Nowadays, dblp contains more than four million publications and two million authors and the rate of growth of these numbers remains constant. Given the size of the database it is hard to have a an immediate overview of the co-author relationships of a certain subset of unknown researchers.

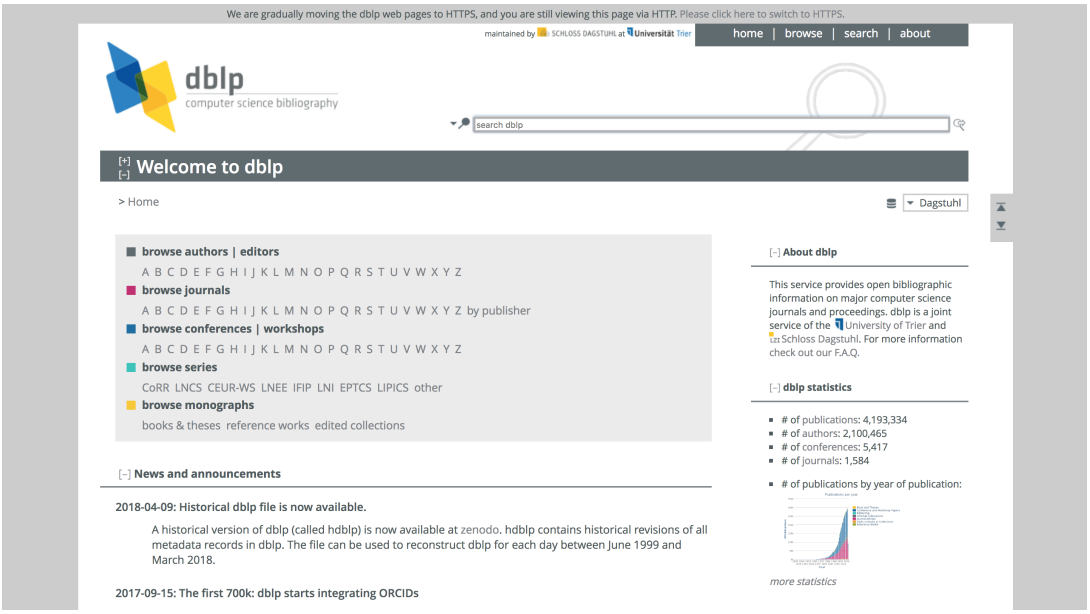

Figure 1. The homepage of dblp

1.1.1 The Dataset

dblp publicly releases a dump of its database in XML format, which contains all bibliographic records. To parse and query this dataset, dblp offers a client-side API written in Java, along with detailed information on the correct use and few examples.

```
DblpInterface db = new SimpleLazyCoauthorGraph();
Person p = db.getPerson("homepages/09/8523");
System.out.println(p.numberOfPublications());
```

The example above shows a DblpInterface is instantiated and assigned in the first line. After the execution of this line of code, the dataset is loaded in the main memory. it is now possible to query the dataset and handle the results as we want. This snipped of code simply prints the number of publications relative to the researcher having the id "homepages/09/8523" on dblp. From dblp HTTP API, we can query authors, papers, and venues. The list of API offered from dblp is the following:

- `http://dblp.org/search/publ/api` for publication queries
- `http://dblp.org/search/author/api` for author queries
- `http://dblp.org/search/venue/api` for venue queries

[1] dblp website: `http://dblp.org/`

dblp HTTP API uses different kinds of search methods and some logic parameters which can be used within the query, in order to execute more queries at once. The different search methods are:

- **Case Insensitive Prefix Search**: The query term will be expanded, ignoring the difference between lowercase and uppercase letters, with any term that matches the prefix equal to our query.
  For example:

  http://dblp.org/search/publ/api?&q=crypto

  will return results for “cryptology”, “cryptography” and more terms with the same prefix.

- **Exact Word Search**: Appending a “$” symbol to the query parameter the exact word search is activated, and therefore we will exclude the expansion used in the previous search method.
  This query, for example:

  http://dblp.org/search/publ/api?&q=crypto**$**

  will not return the same result list as before, but just any result which have, in the title, at least one occurrence of the word “crypto” .

- **Logic AND**: A space between words in the query parameter will make this space be interpreted as a logic AND by the dblp HTTP API, so:

  http://dblp.org/search/publ/api?&q=crypto%20currency

  will not return every “cryptocurrency” document on dblp, but both document about “crypto” and “currency”.

- **Logic OR**: Using a pipe symbol (|) it is possible to have documents relative to at least one term set as query parameters.

Few query functionalities has been disabled, like the “phrase search” operator and the “not” operator. It is still possible to get to the right result combining “and” operators.

More parameters can be set while using dblp HTTP API.

- **format**: Sets the format in which the http response should be sent. This parameter can be set to “XML” or “JSON”.
- **h**: Sets the maximum number of results contained in the http response.
- **f**: Sets the offset of results contained in the http response. Mainly used for pagination.
- **c**: Sets the number of completion terms that the http response should contain.

2 State of the Art

2.1 Data Visualization

Data visualization is an active field of research whose goal is to communicate information clearly and efficiently to users. The human brain is capable of processing information in parallel while using a graphical representation of data, but it is not capable of doing the same for textual representation [H+11]. A reader of a journal article may use the visuals to get a snapshot of the article's contents without necessarily go over the whole article. Similarly, understanding a few thousands lines of JSON requires a different quantity of effort than looking at a graph that represents that JSON, for example.
The dorsal stream is one of the fastest thinking parts of the brain: One picture shown for few moments is enough to trigger a lot of reaction in the brain, while text, audio and video require a lot deeper and slower engagement to potentially deliver the same message [HH12].
Selecting the right visualization technique for data is crucial as failing at this may mislead the user or make this not completely understanding what your visualization aims to deliver [KK11]. Roadsigns, maps and weather forecasts are just few examples of the most common successful data visualization way of deliver information.

Cleveland and McGill say it is easiest for us to compare objects in terms of a common scale, like an axis [CM84]. Data contains entities represented by visual objects and the position of these visual objects is also fundamental. Kevin Jordan and Diane Schiano [JS86] found that when lines of varying sizes were represented close together, those seemed slightly different to users, but if disposed farther apart, the users felt like noticing a bigger difference.

2.2 Visualizing dblp

VisualDBLP is not the first software that can represent graphically dblp but not every application for visualizing dblp is taking in consideration time spans, like VisualDBLP do. Some applications are no more online and therefore inutilizable. This website [Cuh] shows different dblp visualization techniques, but none of them work anymore. Figure 3 shows the examples you can find in this page. Few applications [BPW15] are platform dependent and nowadays this is really big defect: applications should be platform independent as much as possible. Being VisualDBLP a web application, this inconvenience is avoided.
Other applications, even if platform independent, are no more available. Few of them, though, are proper visualization tools, including different graph styles and many visualization techniques. Ebony [LLB10] is a tool for studying and visualizing relationships among authors, groups of authors and entire research areas, taking advantage of the information stored in the dblp database. Figure 2 shows an example of what Ebony can produce.

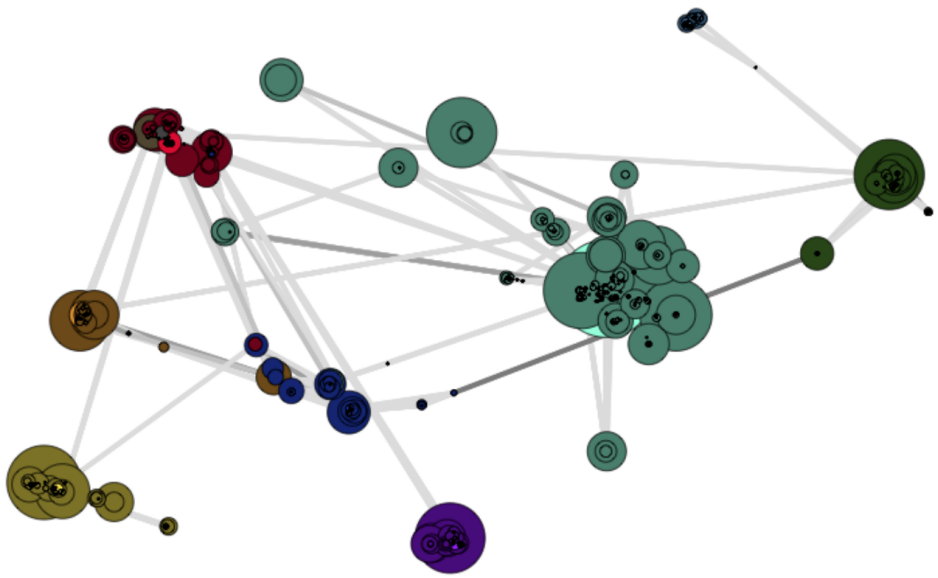

Figure 2. Example of visualization produced by Ebony

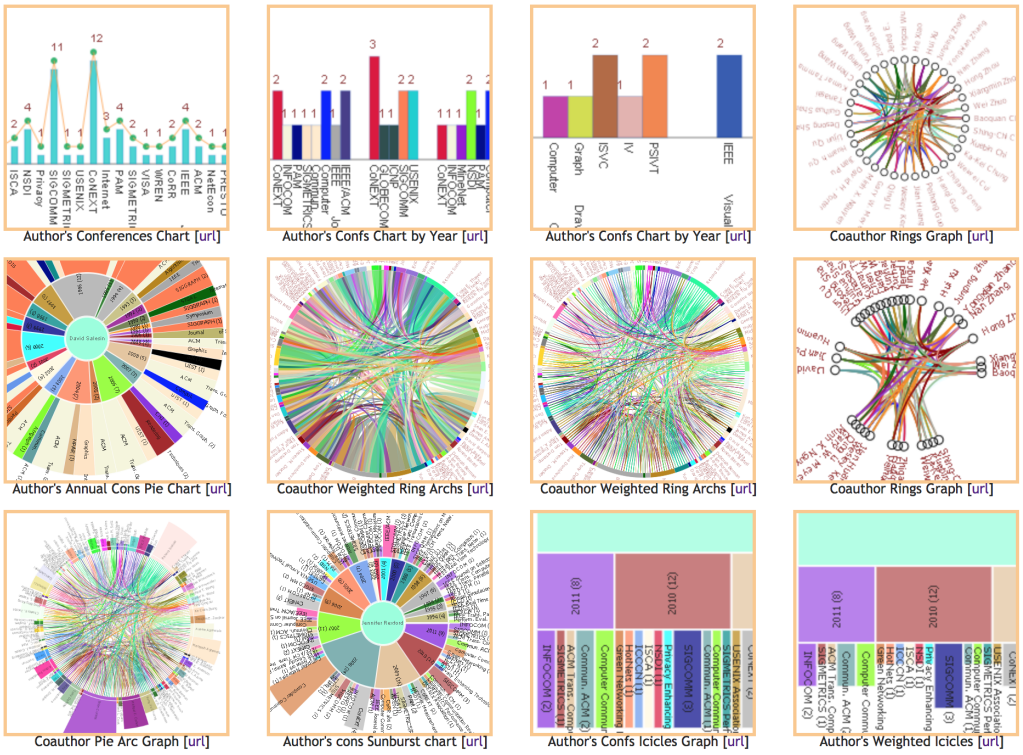

Figure 3. Different dblp visualization techniques

3 Architecture

3.1 particles.js: Explanation and Ad-Hoc Functionalities

particles.js[2] is a JavaScript library for creating interactive and animated graphs in the web browser exploiting an HTML5 canvas. Inside this canvas element, particles.js can draw three different objects: particles (or nodes), edges and background.
Different properties can be given to nodes: size, colour and opacity, speed, direction, and attraction with other particles are just a few. The edge linking two nodes can have different settings too, like the minimum length of this in order to be shown, the colour and the opacity, and the width. The user can finally apply style to the background. Combining the settings for these three elements, different effects can be applied to a particles.js instance and few are ready to play with in the particles.js website.
The core of the library uses the "requestAnimationFrame" method, a method tells the browser that the library wishes to perform an animation and requests that the browser calls a specified function to update an animation before the next repaint. This function gets called sixty times per second in order to draw sixty times the canvas and the particles in it.
Particles.js deals with objects having a number of properties. The most important are:

- radius, expressed in pixels: It reflects the quantity of publications written by a specific researcher.
- position, described by "x" and "y" coordinates, which defines the centre of the particle relative to the canvas in which the particle is drawn.
- colour, described by a RGB code.

Figure 4 shows the particles.js website: At the top left corner we can see a monitor for performances of particles.js, counting frames per second and the number of particles drawn; to the right side we can see the settings panel for the particles.js instance.

[2]particles.js website: `https://vincentgarreau.com/particles.js/`

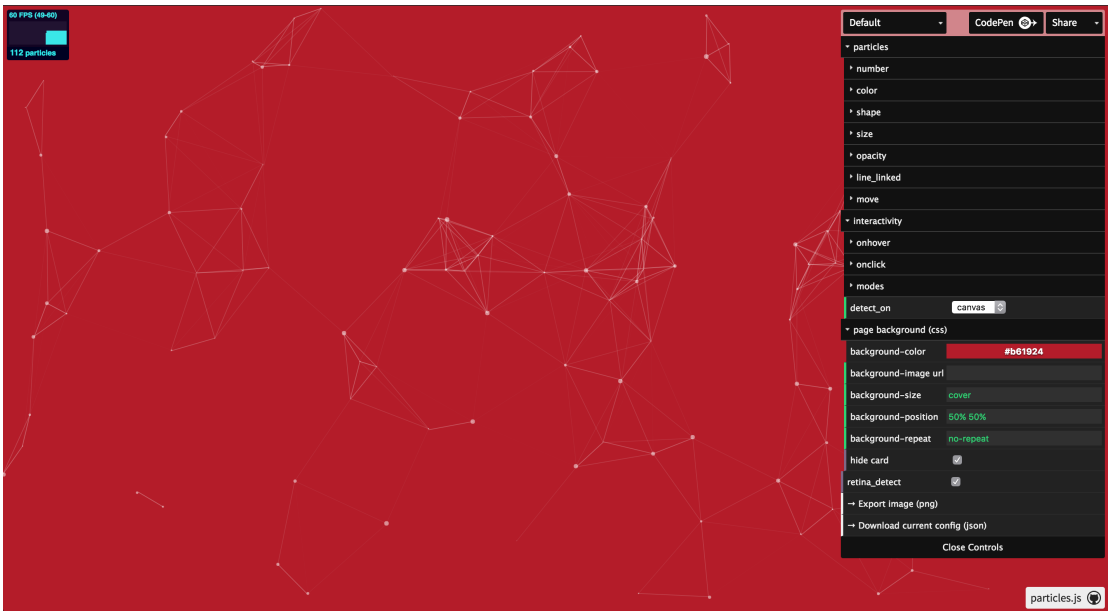

Figure 4. particles.js website

3.2 VisualDBLP

VisualDBLP depicts how a set of researchers collaborated over time, using the dblp HTTP API for gathering data. Any researcher present on dblp can be selected and analysed, the graph representing researchers fully customisable and the output can be easily integrated on any HTML page.
Figure 5 shows the VisualDBLP GUI: On the left side, from the top to the bottom, we can see a search box, a list where results get rendered and a list where selected authors are added. On the right side we can find a particles.js instance running in the same way as it would run on its website, with the relative control panel aside.
VisualDBLP uses a customized version of particles.js described in section 3.4.2 and 3.4.3. Since the number of particles must reflect the number of authors in the selected authors list, from the particles.js settings is not anymore possible to select the number of particles: this value will increase or decrease as soon as an author is added or removed from the selected authors list.

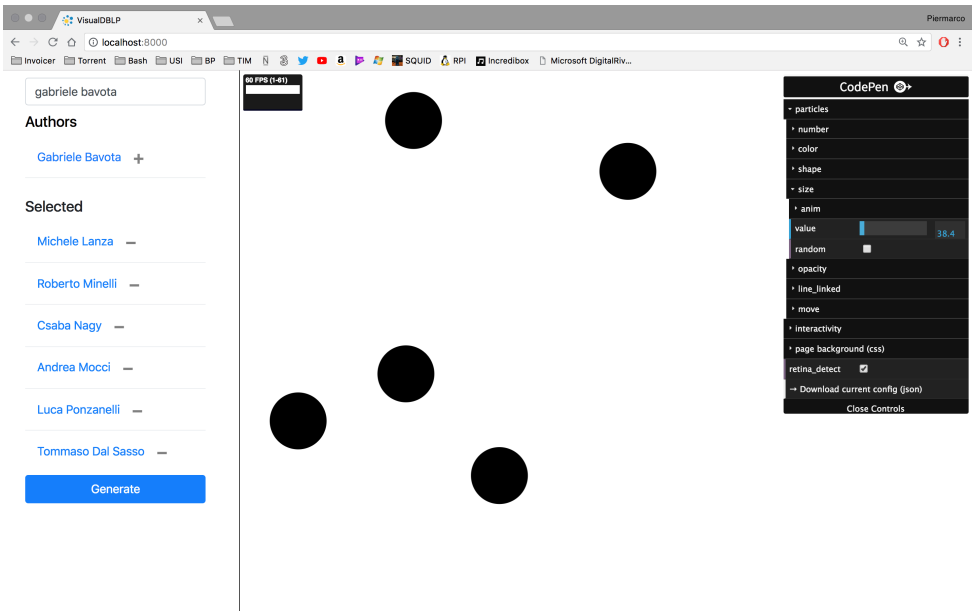

Figure 5. The VisualDBLP web interface

3.3 Choosing a Graphic Library

Different libraries are able to represent dynamic entities in an HTML document. Few libraries are purely used for data visualization, in fact some of them offer different kind of visualization techniques. In this section we will talk about the different libraries that have been experimented with VisualDBLP, even if only particles.js became part of this application.
particles.js was the library that most matched our needs, having the particles movement and the edges visualization by default. The effect of the movement of the particles can remind a lava lamp using the right settings, making the particles move smoothly and making them bounce when reaching the border of the canvas.

3.3.1 First Proposal: Sigma.js

Sigma is a JavaScript library dedicated to graph drawing. It makes easy to visualize entities like networks on Web pages.
From Sigma.js we took inspiration on how to deliver a JSON to a graph, and this structure essentially is the same that VisualDBLP uses. However, the way Sigma.js represents entities is not what we were looking for: Nodes are static, and we were looking for something dynamic, so we decided to search for something else. Figure 6 shows the result of the visualization of the Software Institute members using Sigma.js.

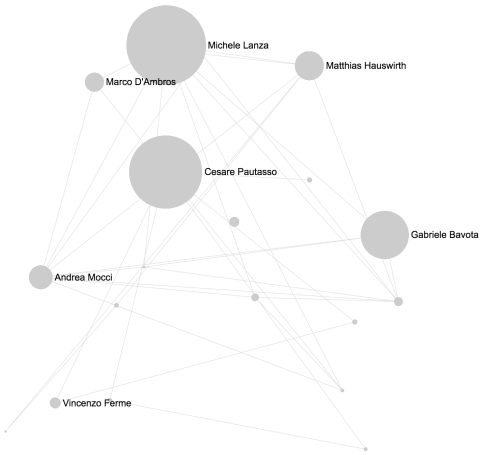

Figure 6. The Software Institute members represented with Sigma.js

3.3.2 Second Proposal: D3.js

D3 is one of the most powerful and widely adopted JavaScript libraries. The New York Times makes use of D3.js for delivering visual contents [3]. More D3 examples can be found in the D3 GitHub repository.[4] Using a Force-Directed Graph[5] is possible for representing the Software Institute members and the relationship correlation between them, but since the result should have more an aesthetic purpose rather than actually delivering information, we decided to look for something different again.
The data structure used by D3.js is similar to the one used from Sigma.js, which made the VisualDBLP JSON structure definitive. Figure 7 shows the result of the visualization of the Software Institute members using D3.js.

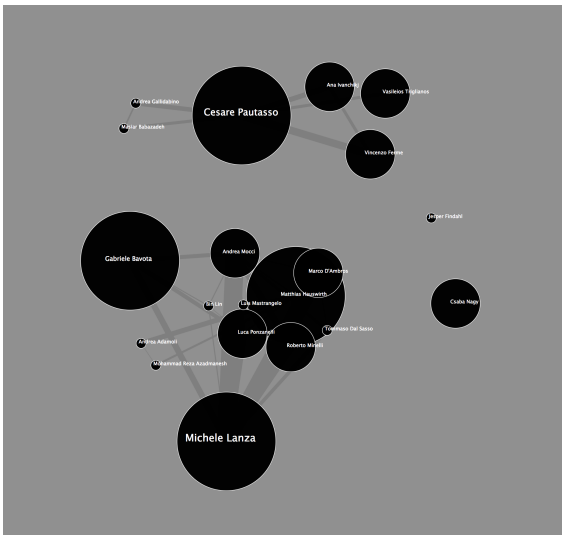

Figure 7. The Software Institute members represented with D3.js

3.3.3 The Core

VisualDBLP is a web application, and its core is written in JavaScript. Figure 8 illustrates the core architecture of VisualDBLP.
The user selects a list of researchers, using the dblp HTTP API relative to author queries in which we discussed about in section 1.1.1. A jQuery "onchange" event is set to the search box, so whenever the query string changes, the value in the search box is used as exact query parameter for retrieving author names. The HTTP response from the dblp API contains a set of authors, that are handled by VisualDBLP and rendered in scrollable result list visible on the left of the web GUI: Researchers which are selected from the user will be from now on defined as the "interest set".
As soon as the user is satisfied with selecting the authors, an HTTP request is sent to the HTTP API relative to publications, so that VisualDBLP can retrieve the list of publications written by each author in the interest set: data from each HTTP response is stored by VisualDBLP. In Figure 4 this is named as the "Set of papers".

Once we have the "Set of papers", we also have this information:

- The first paper published in the aggregate set of publications.
- The last paper published in the aggregate set of publications.
- How authors collaborated in writing papers.

VisualDBLP will iterate from furthest year through the nearest year, generating what in Figure 4 is labeled as "Yearly snapshots". Each snapshot contains:

- The number of publications written by each author in a given year.
- How many publications have been co-authored by any pair of author in the interest set.

[3] An example of the usage of D3.js from the Ney York Times: https://archive.nytimes.com/www.nytimes.com/interactive/2012/02/13/us/politics/2013-budget-proposal-graphic.html?hp
[4] https://github.com/d3/d3/wiki/Gallery
[5] https://bl.ocks.org/mbostock/4062045

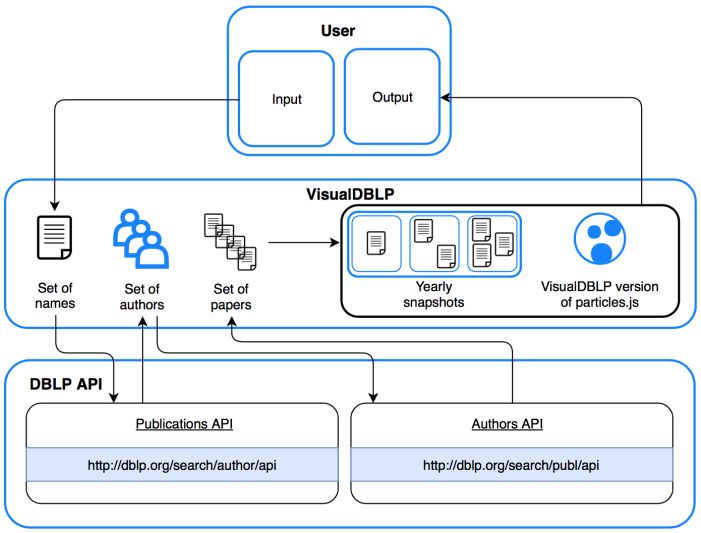

Figure 8. The VisualDBLP architecture

Once this process is completed, VisualDBLP produces a JSON with a structure visible in Figure 7: each field is an year and the value of each field represents a snapshot taken in that year. For each year, we have 2 different fields:

- **nodes**: Contains the interest set and the statistics about each author, like:
  - **id**: Needed to identify different researchers.
  - **size**: Represents the number of publications of that author.
  - **label**: The name of that researcher.
- **links**: It is a list of undirected links with this structure:
  - **source**: The index of an author in the "nodes" field.
  - **target**: The id of another author.
  - **size**: The number of publications in common between the "source" author and the "target" author.

Figure 9 shows two different snapshots taken on the same interest set: We can see that the set of researchers stays the same, but the values are different: this is due to the the increase of the number of publications of each author.

3.3.4 Growing the Radius with a Linear Animation

VisualDBLP maps the number of publication of a researcher to the size of the radius of the respective particle. Another field for each particle is initialised: it is the value that represents how many pixels the radius of the particle has to grow to have a smooth animation while growing its size. Such field is defined as radiusStep.
For example, imagine a particle has to grow from a radius of 10 to a radius of 50, and 60 frames are needed in order to execute the animation linearly. Then, for each frame, the radius of a particle has to be incremented by the same value, so, in this example:

$$\frac{50-10}{60} = 0.66$$

The radiusStep of this particle must be set to 0.66. This calculation is executed whenever there is a change of radius of a particle relative to a particular researcher.

3.3.5 The setInterval Function

To update the radius of the particles, a setInterval function is initialised and the interval of the execution of this function can be set by the user using the VisualDBLP web UI. Even the year of the snapshot that has to be visualized gets incremented by this function, in order to visualize the whole JSON created by VisualDBLP, as this function gets called. When the ending year is reached and the statistics of this last year has been visualized, the animation can

```
"2016": {
  "nodes": [
    {
      "size": 186,
      "id": "Paolo_Tonella",
      "label": "Paolo Tonella"
    },
    {
      "size": 156,
      "id": "Michele_Lanza",
      "label": "Michele Lanza"
    },
    {
      "size": 136,
      "id": "Cesare_Pautasso",
      "label": "Cesare Pautasso"
    },
    {
      "size": 100,
      "id": "Carlo_A._Furia",
      "label": "Carlo A. Furia"
    },
    {
      "size": 76,
      "id": "Gabriele_Bavota",
      "label": "Gabriele Bavota"
    },
```


(a) Part of a snapshot in 2016


```
"2018": {
  "nodes": [
    {
      "size": 195,
      "id": "Paolo_Tonella",
      "label": "Paolo Tonella"
    },
    {
      "size": 165,
      "id": "Michele_Lanza",
      "label": "Michele Lanza"
    },
    {
      "size": 153,
      "id": "Cesare_Pautasso",
      "label": "Cesare Pautasso"
    },
    {
      "size": 105,
      "id": "Carlo_A._Furia",
      "label": "Carlo A. Furia"
    },
    {
      "size": 101,
      "id": "Gabriele_Bavota",
      "label": "Gabriele Bavota"
    },
```


(b) Part of a snapshot in 2018

Figure 9. Two snapshots of the same interest set taken in different years

stop for an amount of time specified once again by the user, using the VisualDBLP web UI. The animation can then resume from the beginning year, creating a never ending loop which is the effect that VisualDBLP aims to deliver. If the user is not willing to displaying all the snapshots taken from VisualDBLP, there is a setting for deciding from which year the loop will start and at which year this will stop, again using the VisualDBLP web UI.

4 Evaluation

4.1 Performance

4.1.1 Creating the JSON: Theta Complexity Analysis

Let's define as $year_m$ the first year in which a paper has been published by one or more researcher in the set of our interest, and as $year_M$ as the last one. VisualDBLP must iterate over

$$year_M - year_m$$

years, so the complexity is

$$\Theta(year_M - year_m)$$

Thus, for each year, an additional informations between each author has to be extracted, so we define as

$$authors$$

the size of the set of researchers we are interested in. Comparing each author with any other author has a complexity of

$$O(authors^2)$$

because each author has to be compared with at most $authors - 1$ authors.
Then, talking about the list of publications done by each of the authors, we can iterate over the smaller of the 2 lists: The number of papers written by one and the other author can be at most the length of the smaller list.
So, begin a_1, a_2 the two authors we are considering in the comparison, being $P(a_1)$ the publications of a_1, and

$$min(P(a))$$

the size of the smaller of the two lists, an extra

$$\Theta(min(P(a)))$$

is added to our computational complexity. Therefore, the final complexity of creating this JSON is:

$$O((year_M - year_m) \times (authors^2) \times (min(P(a))))$$

4.2 Limitations

- **dblp API Request Limit**
  Since VisualDBLP uses dblp API, a fixed amount of request over a certain amount of time is set by dblp. Even if during the developing of VisualDBLP this limit has never been exceeded, a user may abuse of dblp API and make VisualDBLP unusable.
- **particles.js CPU Boundaries**
  particles.js is CPU bounded. The CPU takes care of drawing particles, update their position, speed, radius size, and more. If too many authors are selected (and consequentially too many particles are drawn), the user may experience slow downs in particles movement.

4.3 Future Work

VisualDBLP is an application that relies on the dblp HTTP API. Having a backend that simulates the dblp APIs would heavily benefit this project for many reasons:

- **Users, login and logout**
  Having the possibility of creating user, and therefore login and logout functionalities, would allow people to save executed researcher and graph customisations, even if this is possible yet using Codepen[6], a web service HTML, CSS and JavaScript coding.
- **Caching**
  A caching mechanism may be implemented to improve server efficiency: For example, if a user queries VisualDBLP with a given author and a given time lapse, this result may be cached, so that the next time some other user executes the same query, the cached result is returned.

[6]Codepen website: `https://codepen.io/`

- **Using smaller XML publications instead of the dblp dataset**
  The dblp dataset is an XML file larger than 2 GB. It would be possible to get the dblp bibliography of an author when a query is executed on this and have this bibliography locally saved.
  The whole bibliography contains author information and author publications, in a single XML file.
  In this way, the server uses a constant amount of memory, saves a lot of disk space (since a lot of authors and publications present in the dblp dataset would not be present on this server) but potentially can deliver the same service.

- **A WebGL implementation of particles.js**
  particles.js is lightweight, but not computationally. Just for one single particle many lines of code are executed, and the CPU may suffer. WebGL enables web content to use an API based on OpenGL ES 2.0 to perform 2D and 3D rendering in an HTML canvas in browsers that support it without the use of plug-ins, making render these particles by the GPU, which is by far the best component in the computer architecture for such jobs. This would remove the **particles.js CPU boundaries** described in section 5.2, pushing even further the limit of the number of researchers we may give as input to VisualDBLP.

5 Case Study: website of the Software

As a proof of concept we decided to apply VisualDBLP to the Software Institute members and then integrate the result of VisualDBLP on the Software Institute website. The Software Institute is part of the Faculty of informatics of the Università della Svizzera italiana.
Currently, 22 people are part of the Software Institute, and their website offers a list of publications written by members of the Software Institute, but it does not bring to the a website visitor how people in the Software Institute cooperate.
The current Software Institute website[7] has as home page a fill width image of CodeCity [WL08].
Our goal was to display the Software Institute history from the point of view of its members and their co-author relationship.
Figure 10 and 11 show a prototype of how the Software Institute homepage will change: The static image has been removed and the Software Institute instance of particles.js can be seen working as this loops through years.

Figure 10. The Software Institute homepage with Codecity as background.

Figure 11. The Software Institute homepage with particles.js representing its members.

[7] Software Institute website: `https://www.si.usi.ch/`

6 Conclusions

dblp provides computer science publications and it is one of the main platform for this use. Computer science researchers papers can be found on this platform, easing other researchers while searching for documents which relate to a specific argument.
Visualizing relations between authors on dblp is not intuitive and dblp does not offer a method for retrieving such information considering a temporal context.
To this aim we developed an application that is able to visualize how different authors collaborate in writing papers over time. VisualDBLP is not the first application able to visualize dblp co-author relationships, but on top of our knowledge it is the first to depict them considering a temporal context, allowing the user to fully customize the graph which will be used for visualizing such information. As a proof of concept we used VisualDBLP to visualize the co-author relationship inside the Software Institute.

7 Acknowledgments

I would like to thank Professor Michele Lanza for giving me the opportunity to work once again with him and the Software Institute.
More thanks to Dr. Roberto Minelli, which helped me during all the development phases of VisualDBLP .
Special thanks to my family, friends, and colleagues which always stood next to me.

References

[BPW15] Michael Burch, Daniel Pompe, and Daniel Weiskopf. An analysis and visualization tool for dblp data. In *Information Visualisation (iV), 2015 19th International Conference on*, pages 163–170. IEEE, 2015.

[CM84] William S Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American statistical association*, 79(387):531–554, 1984.

[Cuh] Lixiujun Cuhk. Dblp online visualization. `http://pages.cs.wisc.edu/~lixiujun/samples/dblp/main`. Accessed: 2018-06-18.

[H+11] Peter Hoek et al. Parallel arc diagrams: Visualizing temporal interactions. *Journal of Social Structure*, 12(7), 2011.

[HH12] Martin N Hebart and Guido Hesselmann. What visual information is processed in the human dorsal stream? *Journal of Neuroscience*, 32(24):8107–8109, 2012.

[JS86] Kevin Jordan and Diane J Schiano. Serial processing and the parallel-lines illusion: Length contrast through relative spatial separation of contours. *Perception & Psychophysics*, 40(6):384–390, 1986.

[KK11] Muzammil Khan and Sarwar Shah Khan. Data and information visualization methods, and interactive mechanisms: A survey. *International Journal of Computer Applications*, 34(1):1–14, 2011.

[LLB10] Remo Lemma, Michele Lanza, and Alberto Bacchelli. Ebony: Visualizing the dblp database. Technical report, Università della Svizzera Italiana, Lugano, Switzerland, June 2010.

[WL08] Richard Wettel and Michele Lanza. Codecity: 3d visualization of large-scale software. In Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn, editors, *ICSE Companion*, pages 921–922. ACM, 2008. 978-1-60558-079-1.