Bernd MALLE, BSc.
Mat.Nr. 0130547

GRAPHINIUS
A Web based graph exploration and analysis platform

Master's Thesis

to achieve the university degree of

Master of Science (MSc)

Master's degree programme:

Software Development and Business Management

Supervisor:
Assoc. Prof. Dr. Andreas HOLZINGER
Institute for Information Systems and Computer Media
Graz University of Technology

Graz, April 29, 2016

This page intentionally left blank

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, DATE _____

YOUR NAME

This page intentionally left blank

Acknowledgements

This page intentionally left blank

Abstract

Keywords
KEYWORDS

ÖSTAT classification
ÖSTAT CLASSIFICATION

ACM classification
ACM CLASSIFICATION

This page intentionally left blank

Kurzfassung

Schlüsselwörter
KEYWORDS GERMAN

ÖSTAT Klassifikation
ÖSTAT CLASSIFICATION

ACM Klassifikation
ACM CLASSIFICATION

This page intentionally left blank

Table of Contents

Motivation

Choosing a suitable topic for a Master's Thesis is not a readily decided matter. On one hand a student desires to show some insight into contemporary and maybe even complex problems, but on the other hand the task has to be achievable within a certain timeframe. The project should be theoretical enough to be of interest even for professors while at the same time (at least in Software Development) be practical enough to remaining motivating to the student. In the case of this Thesis I tried to combine scientific and engineering aspects into a project which could be expanded in future endeavors while keeping it sufficiently self contained to represent a work of its own.

Scientific motivation

Having been a member of the HCI-KDD.org group for over 2 years now, I have developed a genuine interested in graph theory, machine learning, HCI as well as their applications in modern information systems - not least in the context of biomedical applications. Although finalizing my Master's study at a relatively advanced age, I am still open to pursuing a PhD in those areas. Therefore, it seemed logical to tackle some problems related to those fields; however significant progress in such matters are not easily achieved even by professional scientists, leave alone a single MSc student on a deadline...

For this reason I intended to contribute to the future work of Professor Holzingers Team by concerning myself with scientific matters without the pretension of being capable of improving on complex research efforts. As a result, I built on work already conducted in my Master's project by underpinning it with a more general and expandable Software architecture. Assisting (data) scientists by providing a better underlying infrastructure to accelerate their experimental iterations and making demanding processing steps available to researchers outside the field of computer science or software development, has been dubbed 'data engineering' over the recent

years [article citations].

Popular projects like Apache Hadoop or Spark, Google's recently released TensorFlow and many other, more specialized clowd-based research platforms, fall under this category - albeit they boast very different properties and advantages and are therefore suitable for different, although overlapping, use cases. The Graphinius library (and future platform) presented in this thesis is unique in the sense that it allows computations directly on the client, but without requiring any installation, by utilizing a piece of software contained in any browser - the JavaScript virtual machine (JSVM).

Technological / engineering motivation

As a student of Software Development and Business Management the two aspects of modern architectures and their repercussions on the next generation of business models are fascinating and of great importance to me. Especially the emergence of powerful JavaScript virtual machines in modern browsers as well as access to the GPU from inside a browser sandbox open up new opportunities for startup companies in many fields that were hitherto restricted to conservative Software Development paradigms. Physics simulations, machine learning tasks, the visualization and manipulation of complex data structures as well as console games can now be implemented on a general, ubiquitous platform without much loss of performance or usability. In the field of traditional Web applications, servers can act more and more as simple database-abstracting backends with near-zero computational and scalability requirements - especially if combined with cheap, global content delivery networks. On the other hand, handing much of the business logic over to the client side, an ever increasing complexity in cooperation between backend services, as well as emerging standards like websockets enabling realtime communication paradigms as publish/subscribe over traditional networking architecture also put challenges to a guild of developers used to write mainly server-side code such as Java, PHP or Ruby on Rails.

As a consequence, the following thesis is an attempt at merging my scientific curiosity with my drive towards new, exciting Software Development methodologies into a working, expandable prototype of a future Web based research platform. I

am glad if I was partially able to live up to that goal.

Introduction

The following sections shall give a brief introduction into how and why project Graphinius came into existence, what challenges we want to tackle building it, as well as the general objectives and potential new applications it might enable or make feasible in the future.

What is Graphinius?

Graphs are a fundamental tool of mathematics and can be applied to a very diverse field of modern scientific areas: network routing (information, traffic, logistics), social network / community analysis, image processing, NLP operating on graphs of document spaces and topic clusters as well as fraud detection via belief propagation networks (all of the mentioned and a few others will be described in Chapter 4).

A relatively novel addition to that spectrum is the emerging field of computational biology, in which we can find PPI, metabolics, or connectome graphs. Since BioMed Researchers are usually no tech experts, an intuitive, UI-based research platform could facilitate rapid experimental iterations. Graphinius aims to be such a Web-based, graph theoretical research platform offering real-time in-browser computations as well as tightly integrated visualization, interaction, and manipulation of graph structures.

In this thesis I will mainly introduce GraphiniusJS and its underlying design principles; however, I am in the lucky position of already having supported the development of a graph visualization library in the context of a colleague's Master's Project. This WebGL / Three.js based library called GraphiniusVis uses low-level datastructures and concepts, enabling it to visualize up to 20k nodes / 60k edges inside a browser fluently even on middle-class laptops. I will also provide an outlook on the whole, emerging Graphinius platform and its exciting capabilities for researchers and students.

The history of Graphinius

When I met Andreas Holzinger in November 2013 he presented me with a 'crazy' idea, as he put it: To analyze dermatological images not by traditional image processing methods, but by first transforming them into a graph structure and subsequently apply graph mining algorithms with the goal of obtaining results at least comparable to the quality achievable as of date. My first experiments in what I quickly termed "graph extraction" from images' were conducted in Matlab and progressed rather modestly; being a developer used to employ C-style programing languages and Object oriented paradigms, Matlab seemed a peculiar (and very costly) way to perform specific, handwritten algorithms which did not lend themselves especially well to matrix representations and calculations. Moreover I was concerned with the fact that even if I succeeded in doing a great job in extracting usable graphs (what that meant was not clear to me at that time) I would not easily be able to share my work with colleagues outside the research community. Providing C++ / Java Code on Github for developers having working installations of those environments is certainly standard by today, but installations of Matlab (or Octave with specific add-on packages installed) are not widespread. Furthermore, in case my results would be interesting to the public as well, a live demonstration of the software at work might be desirable.

After years of having utilizing Web based technologies and continuously researching new frameworks and improvements in that area, it seemed to me that if a whole document suite could be implemented in JavaScript (and even more demanding projects like an entire virtual machine inside the browser [CITE WEBSITE]), the time might be ripe to apply modern JavaScript Engines to the task of image as well as graph processing. Against my expectations Professor Holzinger did not have any objections and I was given green light to develop a simple prototype together with a colleague. This early work was focused purely on extracting graphs out of images without additional image preprocessing steps or subsequent graph analysis. As we would implement mathematical algorithms, we decided on TypeScript as a typed superset of JavaScript for our development process; no frameworks or numerical libraries were used. Within a little more than a month, we had an example website up and running which could load images into a canvas, segment them by either a Wa-

tershed or a Kruskal-based region merging algorithm and extract a graph from the label image; the results are available at http://berndmalle.com/imgextract and have been published in Holzinger, Malle, and Giuliani, 2014. As far as execution time was concerned, our crude and non-optimized JavaScript implementation fell just a little short of a comparable algorithm implemented in Matlab, with great upward potential for using in-browser optimizations or even GPU-accelerated processing.

Of course there were downsides too - JavaScript engines are naturally single-threaded even in 2015, so long-running intense computing tasks tend to block other functions. This is even true for other Browser-intern modules like the rendering engine or input event handling, so that the whole user experience is severely diminished when computing demanding operations. However, a possibility to avert this behavior is the usage of WebWorkers which have been introduced as a working draft as early as 2009, but until recently haven't been widely supported. Also, possible downsides to their usage in situations where large datastructures need to be copied from the main thread to a worker (and back) have to be considered. More interestingly though, it became clear early on that implementing our suite of algorithms as a Web based platform would bring with it several major advantages over traditional software approaches amongst which are easy reproducability, effortless scalability and online meta machine learning opportunities; all of those advantages will be discussed in detail in subsequent chapters.

Building upon those insights Professor Holzinger and I came up with a project called iKnodis.net - Interactive Knowledge Discovery in Networks) - which was designed to cover the whole processing pipeline from image preprocessing over graph extraction to graph analysis and result visualization. In the process of defining those modules it became clear that single components of the processing pipeline would have to be interchangable so that other students be able to contribute new algorithms from future student and research projects to the different stages of operations. This generated the challenge of coming up with a flexible pipelining architecture that would itself be aware of the pre/postconditions in each stage of computation and upon satisfaction of all the constraints be able to connect the different parts together as well as executing them utilizing Webworkers in the background. On this basis I realized another potential advantage of a Web based research platform: by affording users the opportunity to contribute their own algorithms for certain parts

of the pipeline via file upload, it would be possible to gather a large collection of algorithms in a much shorter period of time as compared to researchers working in isolated teams.

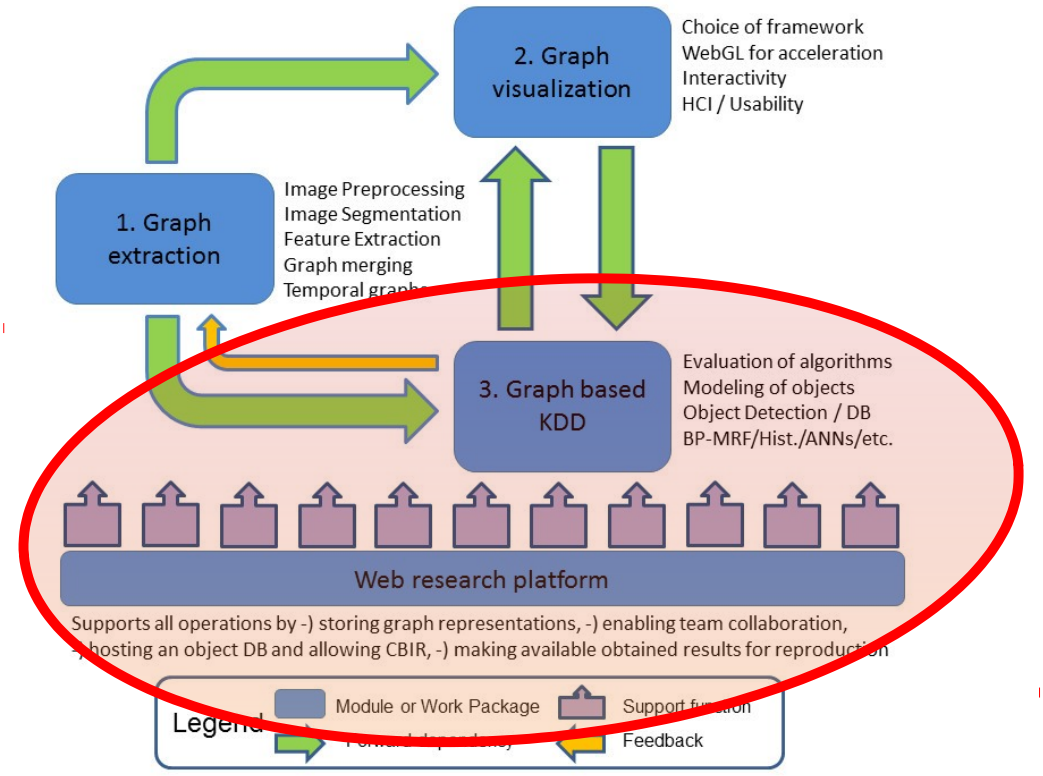


Figure 2.1: Former project iKNOdis architecture overview

In the process of creating an FWF proposal out of iKnodis, the project was split into two separate projects in order to make them amenable to independent smaller funding efforts as well as different groups of developers working on their respective areas of interest yet maintaining their focus towards achieving the common goal. This resulted into iKNOdis now spanning graph extraction out of images as well as 3-dimensional visualization of the resulting graph structures via a Browser based method.

The remaining modules of building a Web Based research platform as well as establishing graph mining algorithms able to tackle object recognition / image classification tasks usually covered by traditional image processing techniques, have been

outsourced to project OGMA, the Open Graph Mining Architecture. Although it will be in the context of OGMA that this Master Thesis will play out, the resulting smart computational pipeline will also encompass the algorithms of iKNOdis. Indeed, a graph extraction algorithm already implemented during my Master's Project Holzinger, Malle, and Giuliani, 2014 will be part of the test cases for the practical implementation found in later chapters of this work.

How this thesis is structured

The rest of this chapter is composed of a short description of how composite machine learning tasks are handled today followed by the question if a transition towards Web based technologies is feasible given the narrow implementation choices in that field. We will then explore in detail what benefits researchers as well as the whole research community could derive from conducting experiments on such a platform; therefore we have to outline potential properties and focal points of such a technology. Upon a short discussion of how the gathering of metadata about diverse ML/KDDM experiments as well as their results could open up new opportunities, we will envision a future recommender system for composing machine learning pipelines based on a heuristic approach. This chapter will close with a brief overview of the general properties of the smart computational pipeline (henceforth called "SCP") under development.

Chapter 3, "Related work", will consider past and present approaches to recognizing objects in images as well as conduct image classification utilizing graph based approaches and will therefore show the validity of OGMA's objectives. Furthermore, contemporary efforts of building machine learning pipelines as well as the data and insights they are based on will be investigated. An overview of existing software solution will conclude chapter 2.

In Chapter 4, "SCP General Design", we will lay the technical and scientific foundation for the design of a smart computational pipeline for OGMA. Building upon a high level overview of machine learning pipelines and their different stages in theory, we will be able to derive general requirements that any piece of software handling those processes will have to fulfill. Further, the question will arise as to how to represent algorithmic "packages" for the sake of this undertaking and how

dependencies in the I/O of each pipelining stage should be resolved automatically. Following an analysis of how to use existing package manager algorithms to fulfill those requirements, the question of executing a valid processing pipeline within a browser environment considering modern users' expectations of fluidity and asynchronicity will be dealt with. At the end of this chapter, we will propose a data model for storing the experiment metadata for further analysis and find a description of the algorithms used for testing the pipeline later on.

The next chapter 5, "SCP Software Design", will elaborate on the actual software implementation as well as the software development process followed in this project. First a set of requirements for modern Web based software will be compiled; based on this checklist an exploration of possible suitable technologies for both server and client will be conducted and their respective strenghts and weaknesses revealed. After a detailed comparison I will argue for the particular combination of choices I made and will subsequently describe the development process used to implement the pipeline. An illustration of how Webworkers were used to execute the pipeline as well as how results were visualized will finalize this chapter.

The 6th chapter on "Results and Discussion" will wrap up the efforts undertaken in this project. I will first go into details about the project implementation and the difficulties and / or caveats that arose during my work. A detailed experimental setup will be described and the expected behavior of the SCP defined, while the ensuing section should verify those expectations and therefore validate the approach taken. A discussion of open and interesting challenges as well as an outlook on future possibilities to enhance and expand on this work will top of this chapter.

Eventually the 7th and last chapter, "Conclusion", will give a short recap on the motivation, goals, and main issues of this project and thus bring the thesis to its completion.

How Machine Learning / KDD are approached today

When taking the famous Machine Learning MOOC tought by Prof. Andrew Ng from Stanford University on Coursera in 2013, one story he was conveying during a

section on optimizing Machine Learning Pipelines had especially caught my attention. As a specialist in high demand Prof. Ng is frequently consulting for Silicon Valley companies in matters of Machine Learning and Artificial Intelligence. On one of these occasions, the client company had been trying to optimize their ML pipeline for the better parts of 2 years without any significant improvements in their results. After looking at the different stages of their pipeline and conducting a so-called ceiling analysis, Prof. Ng concluded that two developers had spent 18 months on optimizing their background separation algorithm while the most significant potential for improvement really lay in a latter stage of the process. Based on this analysis the company was able to remedy the shortcomings in a relatively short amount of time.

This incident shows how much effort is potentially squandered by trying to implement sophisticated processing pipelines within isolated teams in a non-standardized fashion: Proprietary approaches - both in technology as in methodology - hinder the exchange of information with other members of the research community, thus opening up vulnerabilities to making mistakes which could have easily been avoided by considering the experience of other professionals. The following properties of data analysis / ML projects seem to give rise to such vulnerabilities:

- **Isolation.** Working on common machine learning problems in isolated teams without communication makes comparison of approaches as well as results unnecessarily hard. Dealing with errors at any stage of the pipeline takes more effort than necessary due to a lack of reference values, while achieving superb results has little to no effect on the potential of other experts.

- **Proprietary Software.** Countless professionals prefer developing data analysis pipelines in highly proprietary software environments like Matlab or Mathematica. This prevents an influx of solid, community-tested algorithms while preventing others from gaining knowledge acquired in such organizations (if they are unwilling to pay for the software).

- **Irreproducibility.** As unpublished code cannot be perfectly reverse engineered, experiments conducted in isolation can't be easily corroborated. This might be advantageous with respect to product development and patent pro-

cedures, but is usually detrimental to the efforts of researchers trying to get published and spreading their insights.

- **Lack of scalability.** Last but not least, heterogeneous and highly customized data processing pipelines might not lend themselves well to parallelization, which might prevent the use of such algorithms on quickly expanding datasets.

Papers to cite in this context:
Lorica, 2015 Meng, 2015 Lorica, 2013a *The Stanford Natural Language Processing Group* Sparks, 2014 Jones, 2013 Lorica, 2013d Lorica, 2013c Lorica, 2013b

A Web based approach - why?

How a Web based approach could benefit users, researchers and society

- **Ease of access.**
- **Effortless scalability.**
- **Built-in potential analysis.**
- **Server Side Meta Machine Learning.** As researchers start using our platform, their pipeline configurations, input descriptions, task specifiers (classification, text analysis etc.) as well as results will be stored on the server. Enough data provided, such a database might be amenable to the development of heuristics on the expected success of a particular algorithm invoked at a specific stage of the pipeline.
- **Heuristics-based recommendations.** Based on the last section, an optimal pipeline configuration might be computed provided only the input data (image, text, point cloud etc.) as well as the desired result. The Smart Computational Pipeline would then self assemble (given the user has no objections) and immediately commence the experiment.

General applications

education

algorithm prototyping

research platform

GRAPHINIUS as a platform

General Properties

Automatic real-time visualization (switchable)

Online documentation

Example graph datastructures

User profiles

Save and fork experiments

Distributable via URL (like e.g. Codepen)

Can write own algorithm and use on given graphs

Graph Properties

Mixed mode graph

A mixed mode graph - at least in Gephi vocabulary - is a graph that may contain directed as well as undirected edges at the same time. While many algorithms are defined on just undirected (e.g. Minimum spanning tree) or directed (e.g. percolation) edges exclusively, for many real world applications it is required to consider a combination of both - imagine traffic simulations with one-way streets or social networks in which people can be friends (undirected) and / or follow each other (directed). As Graphinius should be able to cover such applications, its core needs to be designed as a mixed-mode graph; the problem with this is that many algorithms have no standard implementation for a mix of both edge types, and so here and

there it was necessary to come up with a logical and pragmatic solution, even if it could not be verified by any textbook.

Node and edge types (filters)

A mixed-mode graph alone however, is not enough for more complex scenarios, even in every day's situations. Assuming a social network again, we would first think of humans as participating entities. Depending on the particular use case of the network however, other nodes might be resources such as books (author network), movies (Netflix), or any type of commodity (online shopping). Moreover, edges in such a network cannot only differ in mode of direction, but might represent a specific type as well (following someone vs. movie recommendation). From this emerges the need for graph filtering - or graph views - which expose only a specified subgraph to an executing algorithm, suitable to the particular situation. To stay with our example, if we need to find all users within three hops of friendship connections, we do not want to traverse all the edges representing recommendations (or messages, as there might be orders of magnitudes more of those present). In this case, we would execute a Breath-first-search against a view of the graph, which would present the BFS's logic with only the connections it is required to 'see'.

Object oriented

One design decision in writing any new (graph) library - as far as the author can judge from his personal research - lies in speed and memory vs usability. This concerns not so much the handling of nodes and edges themselves (many libraries have very good wrapper functions for dealing with basic primitives), but requirements like additional payload - e.g. the k-gram vector of a node representing a text document - or node & edge types themselves. In order to speed up execution of graph algorithms, advanced libraries use specialized data types like sparse matrices or fixed-length arrays; this on the other hand forces a programmer to hold additional data structures at hand for whenever more complex computations are needed. A good example for this would be computing the 'distance' of two nodes when defined not as the length of the shortest path between them, but as the cosine distance between their feature vectors.

Taking into account the special language properties of Javascript, with its great emphasis on first-order functions and closures which makes for a natural callback-driven algorithmic approach (see fig. [insert figure]), the author believes that an object oriented approach is the suitable one for Graphinius, which is backed by 3 different properties of the language:

- Despite not being 'traditionally' object oriented like Java or C++ for its absence of constructs like classes etc., Javascript is firmly OO - in fact, everything except primitives is always an object, including and especially functions.
- Accessing objects instead of flat memory should not incur too much runtime overhead anymore, since modern JSVMs have abandoned the flat model in favor of an object memory model themselves.
- As Graphinius is intended to be part of a learning, teaching and research platform, and not designed to handle large graphs of millions of nodes and upwards, the OO approach seems a natural fit since it allows implementing algorithms in a very intuitive way [comparison].

Online Editor

Build & mutate

fork & extend

publish via simple URL

Real world suitability

Collaborative platform (research teams)

Reproducibility / Corroboration of results

Prototyping

Algorithm Marketplace

personal algorithms

community algorithm database

Applications

Areas of application

Social networks

Social networks are today's natural candidates for graph based algorithms, as they have been rising to power and fame over the previous decade and a half. Of course most social graphs in use today are far too big for any client or server side application to handle, and are therefore only interesting to programmers and architects of database clusters, high performance grid-computing developers and data-center engineers. Because of this, I am going to confine myself to the topic of local sphere recommenders, where I believe small graph computing to be able to have some real world influence. In order to get to this point, we will first need to take a look at the shape and size of typical recommendation processes (themselves forming subgraphs of larger networks), which in the following section will be termed 'cascades'.

Network recommendation analysis

Leskovec, Singh, and Kleinberg, (2006) have examined recommendation networks crystallizing from purchases based upon previously received product recommendations. In order to do this, they employed an online shopping system observing the product categories of DVDs, Music, Books and Videos (VHS). Users of that system were modelled as nodes in a graph, with the graph initially being completely unconnected. In this system people who bought a product (and only actual purchasers) were able to recommend the bought product to as many people as they wanted via email; this resulted in a *temporary* recommendation edge added between the two (user) nodes. This edge was then handled according to the following two criteria:

1. Recommendations received after a product was already bought by the receiving person were immediately deleted.

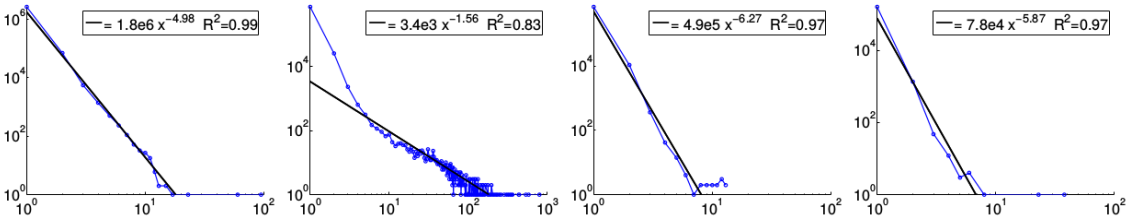
2. Recommendations received which did not result in the product being bought (during the observational period) were also deleted.

This procedure resulted not in a graph comprising all of the users and products bought throughout the system, but only a collection of - fragmented - subgraphs representing the recommendation cascades. The main question of the study then was to the size distribution of those cascades w.r.t. their count, and if properties of the original social network (e.g. density, degree distribution) had any influence on that distribution.

A second point of interest concerned the isomorphism classes of cascades, meaning their shape and size similarity. Therefore, a similarity measure had to be established, as the graph isomorphism problem is NP-hard and therefore impractical to use on a real-world study. This is why exact isomorphism matching was only used on cascades up to a graph size of 9 nodes; above that a graph *signature* was computed including singular values (via SVD) of the graph adjacency matrix up to a size of 500 nodes. Above that, the signature only consisted of the number of nodes and edges as well as a histogram of in- and out-edges per node (degree distribution). The relation between cascade amount and size can be seen in Figure 4.1 on page 33.

The results of this study after a two-year period can be summarized as follows:

- The largest cascade (which also form connected components) accounted for less than 2.5% of all nodes.
- Cascades did not only come in the form of trees (snowball effect) but form arbitrary graphs with splits, collisions as well as cycles.
- Splits are more common than collisions, however (as one would expect).
- The frequency of a cascade type (as computed by graph isomorphism) is not a strict monotonic function of cascade size, which points to the recommendation propagation process to be influenced by more subtle factors of the underlying social graph than just the network structure alone.
- Most cascades observed exhibited fewer than 9 nodes (with the exception of DVD recommendation cascades) and were of very small degree (just a little over 1 according to the visual representation found in the paper



(a) Book (b) DVD (c) Music (d) Video

Figure 4.1: Size distribution of recommendation cascades for four product categories

This diagram was taken from (Leskovec, Singh, and Kleinberg, 2006), page 7.

The above analysis holds several insights which in combination lead to a remarkable conclusion:

1. The cascades presented in the paper represent only 'successful' recommendations, i.e. the ones which receivers perceived as valuable enough to actually buy the product.

2. The goal of any recommender algorithm (regardless on which item space it operates) is to produce exactly such valuable recommendations.

3. Because most cascade sub-graphs were of very small size and degree, 'successful' recommendations can be assumed to originate from places in the direct neighborhood of a node.

The local sphere

The concept of a local sphere and computations applied to it comes from the author's (possibly incorrect, but natural) insight that the relevance of recommendations behaves as a function of node vicinity:

- Lets call the whole social graph and all interactions in it the 'global sphere'.
- Recommendations to users are then computed over the global sphere, which takes an amount of resources exponential to the size of the underlying graph.
- Let's further assume that 95% of all relevant (accepted) recommendations in a social network like facebook are those that are derived from the immediate local neighborhood of a node (less than 2 degrees, see section above..)

- This assumption is corroborated by the fact that two degrees are also what Facebook allows programmers (as of 2013) to query via their graph API from any authenticated user, apparently in an effort to prevent automatic traversal / exploration of their most valuable business asset.
- Let's call this immediate local neighborhood the 'local sphere'

Now let's also consider how modern publish/subscribe based frameworks (like Sails, Meteor, Hapi or Derby, only to mention some JavaScript libraries) handle data communication between server and client:

- The server offers some subscriptions on it's data, usually limiting access to items based on identity, authorization or user role provided by the client.
- The client defines some subscriptions on server-side data collections (tables), representing the client's wish for information regardless of it's status or authority.
- Publication as well as subscription can be seen as a mathematical subset of all the data in the database.
- An algorithm inside the respective framework resolves those (potentially conflicting) interests by computing the intersection set of the data provided / requested.
- The intersection data set is then pushed to the client (in our case the browser) as soon as it becomes available or is updated, which makes this model ideal for real-time interaction and communication between clients.
- The sum total of all the data pushed to the client is equivalent to the 'local sphere' we described earlier - HOWEVER - their inherent graph structure is lost during the transmission, so that the client can only see them as isolated fragments without context.

The combination of those two ideas now enables us to envision the following scenario:

- Instead of interpreting all data items in the local sphere as isolated entities, we retain their graph structure enabling the client to gain hitherto unachieved knowledge and insights into its already available data.
- We therefore need a graph library in the browser, not only to represent the local sphere graph, but also to analyze it in order to take intelligent actions that were previously reserved for the server-side (data center) infrastructure.
- No complex graph partitioning algorithm on the server is necessary, as we can use the natural set contraints inherent in any web application:
  - e.g. in a social network, the client will have access to all its immediate friends, social activities and interest groups
  - in a project management tool, the client naturally has access to the data of all team members, to-do lists, milestones, resources etc.

  If our 95%-relevance assumption mentioned earlier holds, we can achieve great scaling efficiency by introducing the local sphere concept:
  - the client can immediately perform computations like recommendations on the subgraph of the local sphere.
  - only recommendations accepted will have to be stored on the server (that is, cause additional network traffic).
  - the client is easily able to recompute the relatively small local graphs in real-time, offering responsiveness far beyond today's best (server-side) infrastructures.
  - as modern web frameworks transport all of the required data into the client store anyways, we do not add extra complexity to our servers and databases.
- On the other hand, questions of data security / privacy will have to be dealt with, as we are talking about preemptively filling the client memory with possibly otherwise unnecessary or superfluous data.

Needless to say, GraphiniusJS would be an ideal candidate to explore this concept further and could, if used appropriately on carefully modeled local spheres,

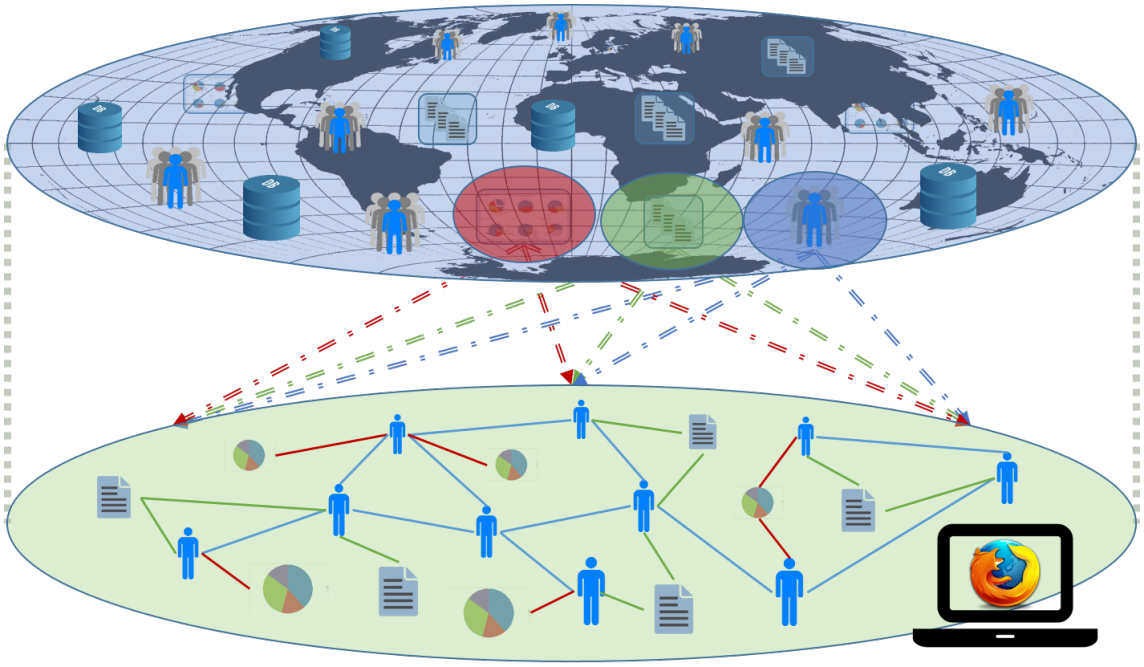


Figure 4.2: Local sphere projected from global sphere

Only a very small portion of the global graph is actually visible from any connected client. The sum of all viewable items however, if properly conveyed to the client (i.e. with their connection information preserved), could form a subgraph of the whole network called the 'local sphere', which would allow the browser to utilize the underlying graph structure to extract hidden knowledge and perform graph computations on its own.

enable start-ups to compete with much larger companies employing complex and very expensive machine learning infrastructures.

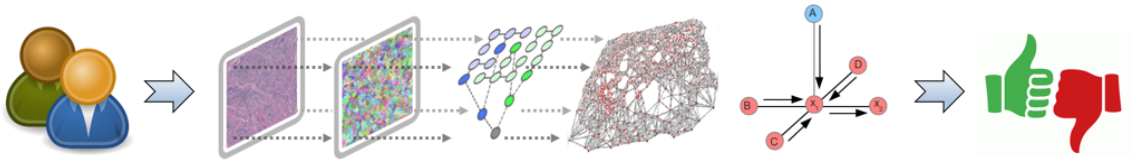


Figure 4.3: Graph based image classification example

1) a laser scan image of a nevus is oversegmented and 2) a graph extracted by interpreting region centroids as nodes and region adjacency as edges. 3) A belief propagation algorithm is applied to the resulting graph yielding 2) a converged state representing the nevus classification as benign or malignant.

Name	Age	Zip	Gender	Disease
Alex	25	41076	Male	Allergies
…	…	…	…	…

Figure 4.4: The three types of data considered in (k-)anonymization

Graph based image processing

Graph based NLP

Biomedical applications (protein networks etc.)

Anonymization

Node	Name	Age	Zip	Gender	Disease
X1	Alex	25	41076	Male	Allergies
X2	Bob	25	41075	Male	Allergies
X3	Charlie	27	41076	Male	Allergies
X4	Dave	32	41099	Male	Diabetes
X5	Eva	27	41074	Female	Flu
X6	Dana	36	41099	Female	Gastritis
X7	George	30	41099	Male	Brain Tumor
X8	Lucas	28	41099	Male	Lung Cancer
X9	Laura	33	41075	Female	Alzheimer

Node	Age	Zip	Gender	Disease
X1	25-27	4107*	Male	Allergies
X2	25-27	4107*	Male	Allergies
X3	25-27	4107*	Male	Allergies
X4	30-36	41099	*	Diabetes
X5	27-33	410**	*	Flu
X6	30-36	41099	*	Gastritis
X7	30-36	41099	*	Brain Tumor
X8	27-33	410**	*	Lung Cancer
X9	27-33	410**	*	Alzheimer

Figure 4.5: Tabular anonymization: input table and anonymization result

Fraud detection

Polo chau's BP in MRF for spam classification work...

Application specific requirements

Data Structures

Data Cleaning

Preprocessing

Graph generation (ER model...)

Feature Selection

Data Mining

Postprocessing

Visualization

Interaction / user feedback (iML)

Related Work - Survey of graph libraries

C++

Boost Graph Library (BGL)

Lemon (C++)

igraph (c, python, R)

GraphTool

Java

GraphStream

JUNG

JGraphT

Grph

Python

Networkx

Ruby

graphy

RGL

Other (Matlab, R, whatever)

The R graph package (discontinued, just mention)

Architecture of GRAPHINIUS

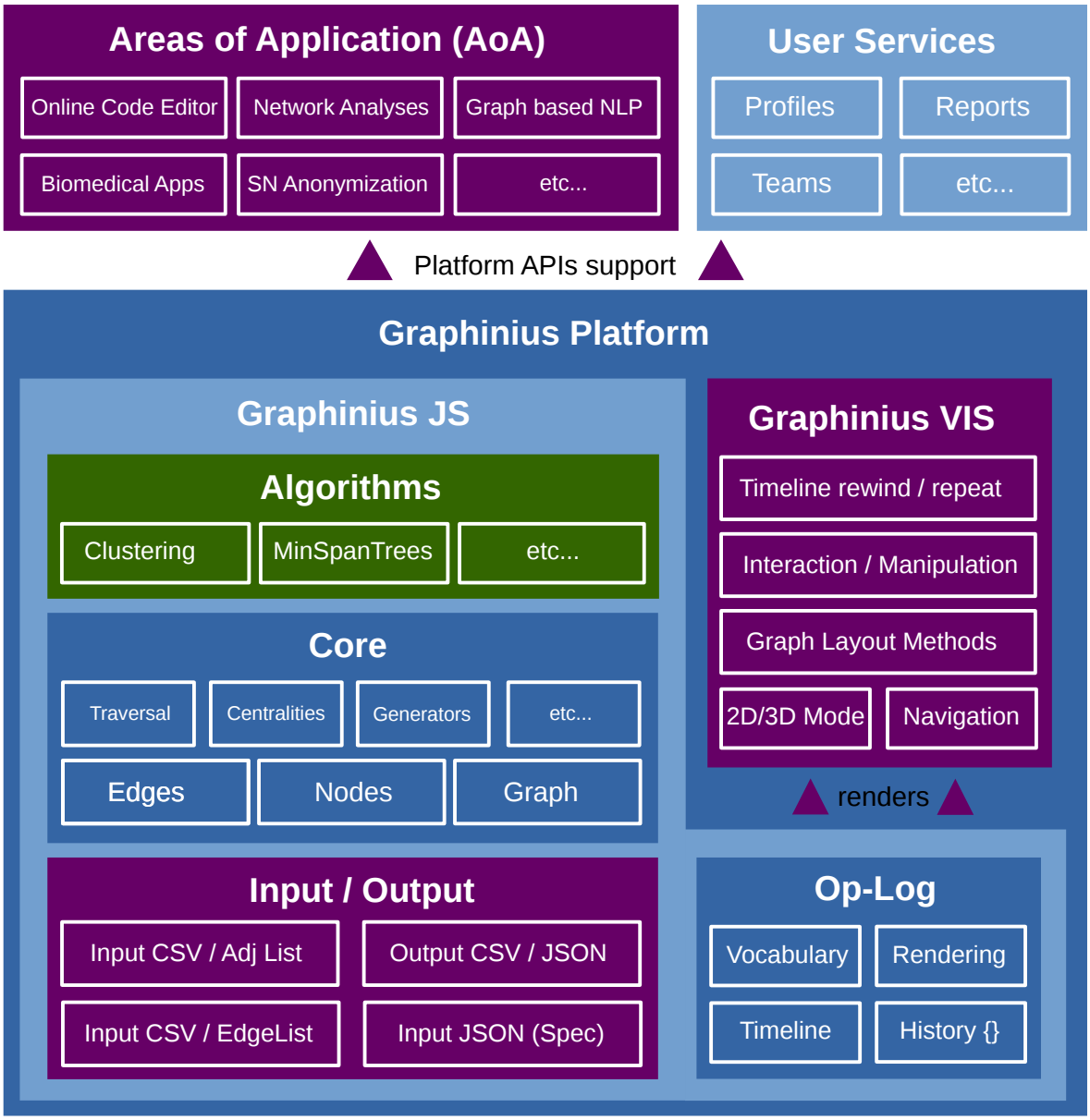


Figure 6.1: Graphinius platform architecture overview

Graphinius Base

As the name suggests, the Base offers all the functionality necessary to develop graph-based applications on top of it, including basic graph-computations and algorithms as well as visualization. It is therefore logically composed of those two modules, Graphinius JS and Graphinius VIS as well as a mechanism of communication between those two.

Graphinius JS

Graph input readers

CSV

```
1    A, B, u, C, u, A, d, B, d, D, d
2    B, A, u
3    C, A, u, A, d
4    D, A, d
```

Figure 6.2: Adjacency list including edge direction

CSV Edge Lists use the simple format of [StartNode, EndNode [,directed]].

JSON

Graph Core

Edges

Nodes

Graph

Traversal

BFS / DFS implementations... [figure: callback-based DFS]

Degrees

Centralities

Generators

Algorithms

Clustering

MinSpanTrees

Shortest Paths

The Op-Log

Timeline

History Object

Vocabulary

Rendering Mechanism

Graphinius VIS

2D/3D Mode

Navigation

Graph Layouts

Interaction / Manipulation

Timeline rewind / repeat

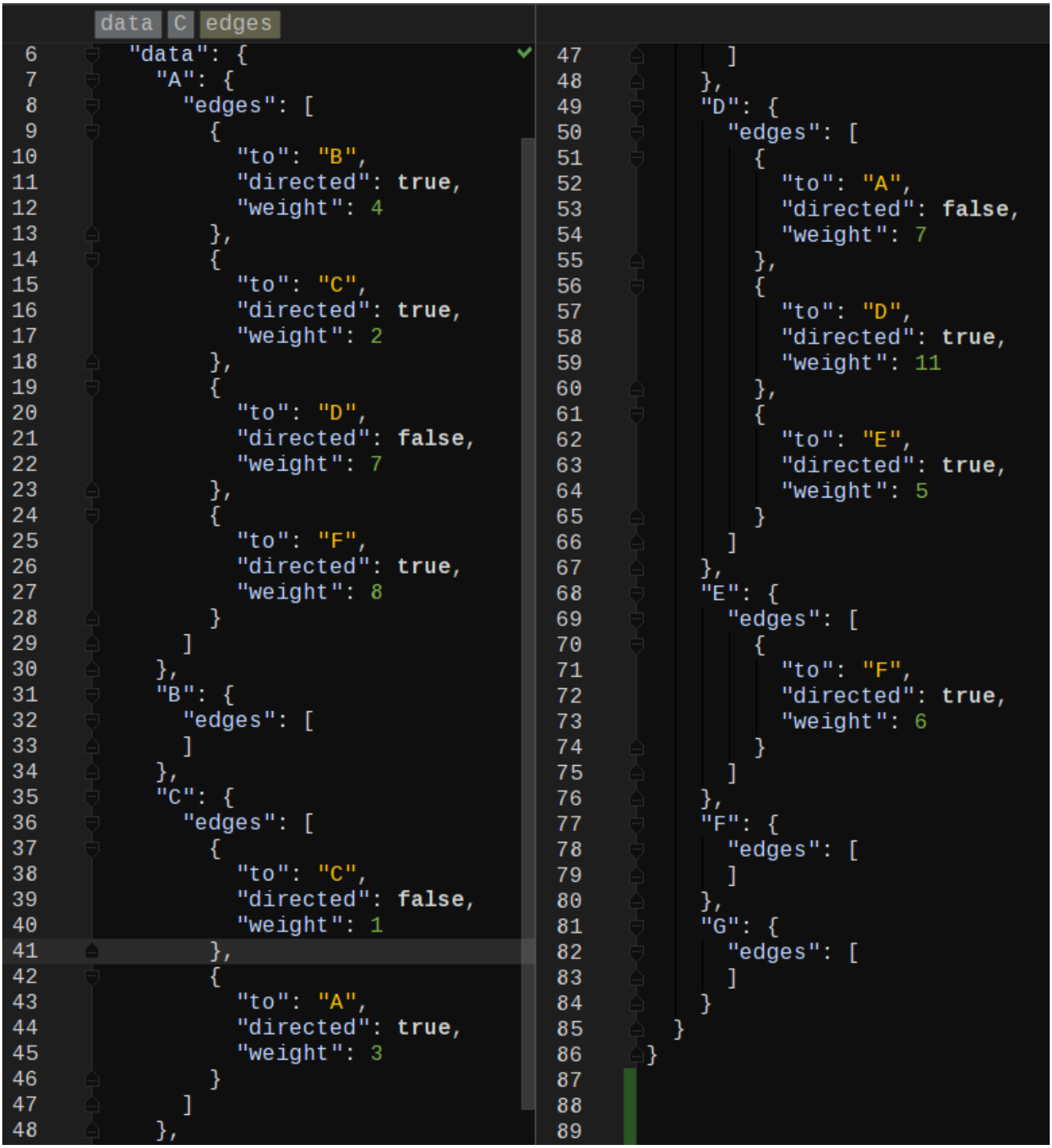


Figure 6.3: Sample graph in the Graphinius JSON format
Apart from the 'to' node, direction and weight, any node can exhibit an arbitrarily large feature vector containing any type of information (like patient data, word vectors, etc.). Another special sub-object which the input reader is looking for is the 'coords' object, which specifies the coordinates used in the constant layout renderer of the GraphiniusVIS library.

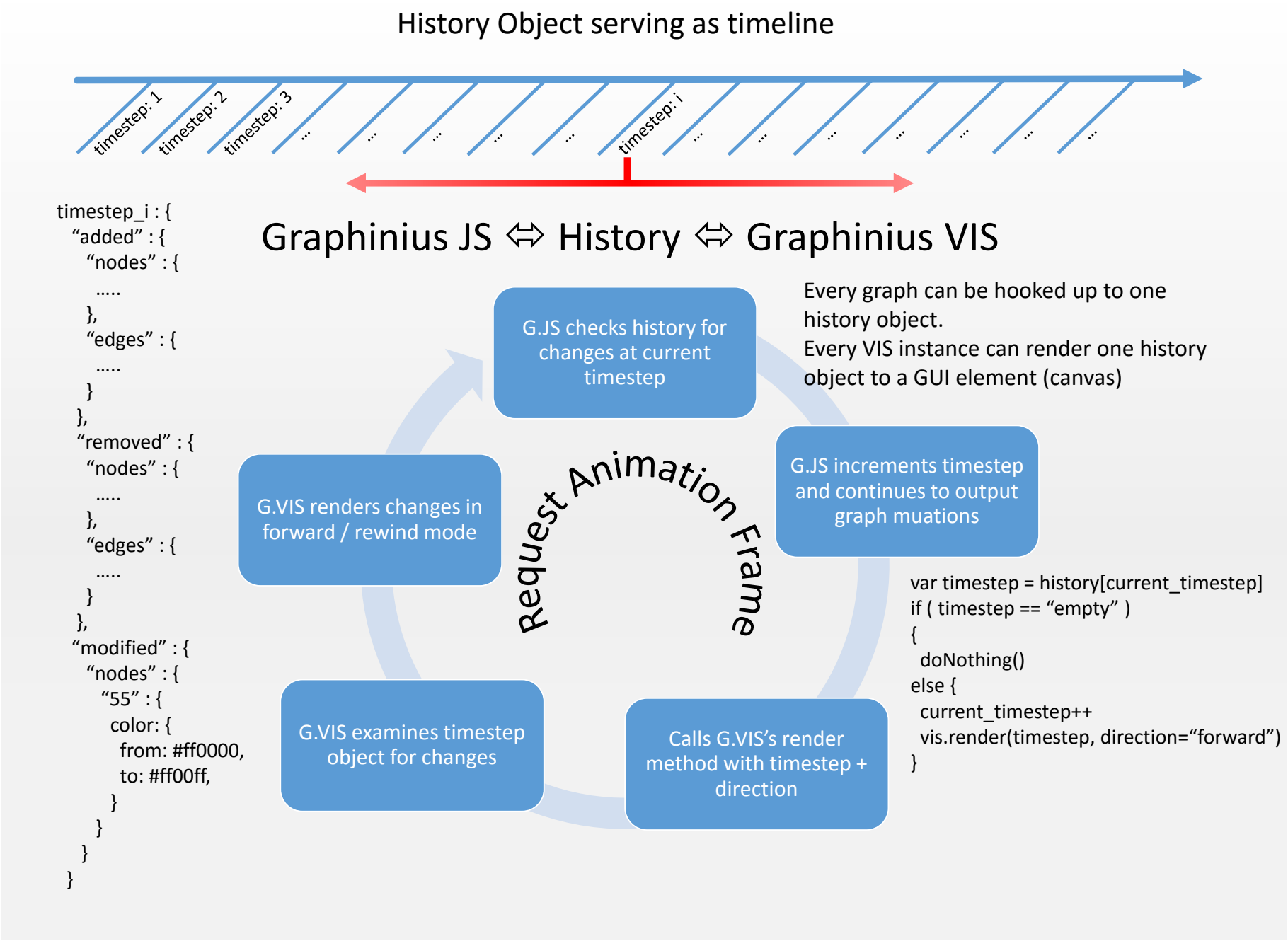


Figure 6.4: Graphinius JS <-> VIS communication via Op-Log

Areas of Applications (AoA)

Online Editor

Biomedical Applications

SN Anonymization

Platform Services

Personal Profile

Teams

Output / Reports

Software Requirements & Survey

In order to assess which technologies to use for a new project, one first has to take into account the kind of software product to build, the sector of the economy it will be used in as well as the specifics and constraints of the environment it's going to operate and interconnect in. Let us first take a closer look at those points:

- **The kind of product** to construct will often determine some core technologies: Building a messenger app requires real-time behavior some statistical product suite would never make use of. Likewise, an autonomous control system of a space probe will also depend on time-critical components, but in a different way than a messenger app, relying strongly on a constant rate of throughput, whereas a message flow will not be critically disturbed by a lag or latency disruption every now and then.

- **The industry sector** largely determines requirements in the form of compliance or industry certification. For instance, whereas the security concerns in a normal end-user centered application might be dealt with with relatively moderate levels of effort, applications employed in the financial or even medical sectors will in all probability have to satisfy additional security demands such as audit trail systems or compliance to specific data formats and standards.

- **The technical environment** a system is operated in will influence its shape and behavior as well. A relatively disconnected and isolated system like a statistical module (which e.g. outputs some results on a nightly basis) will be modeled differently than a web-based, cloud-oriented service incorporating many interfaces and API calls to dependent background or partner services.

In the following sections, we are not considering the entire SW development workflow from an economic / managerial point of view, but just the technological aspects of it. Let us first realize the differences between software development today and the way it was usually conducted as short as only 2 decades ago.

The traditional software development process as followed throughout the first decades of the existence of our field has been relatively simple: upon setting a goal for functionality or any other measurable entity (code module, UI section), a continuous iterative process of writing some lines of code, re-compiling, testing (automated or manual) and bug fixing was all, or most, that was necessary to arrive at some usable product. Modern applications, however, especially web-based ones (and that includes all kinds of mobile apps that have seen their rise over last decade) operate on many different moving parts:

- Some server-side backend which coordinates incoming requests and provides consistency across business logic and database layers. This is probably the part with the greatest similarity to traditional, client-only or centralized software (development). I would also include old-fashioned web 'applications' (and certainly websites) in this environment, as a browser-based GUI alone without much processing or business logic going on, does not really fall into the category of a distributed application.

- A client part in the form of a modern in-browser based app (like GMail, Google Docs or Office365) or any mobile app executing on a contemporary mobile device.

- Some background-services, mostly in separated modules distributed over one or many servers worldwide, including interfaces to dependent services, isolated REST services (like a search portal for medical professionals inside a larger healthcare application) or microservices: sub-components of the business logic implemented directly on a database level, as implemented in the Foxx application micro-framework inside the multi-model database ArangoDB (Dohmen and Hackstein, 2014).

- Where needed, a visualization module will have to be provided which resides on the client side but is logically separated from the 'normal' business and communication logic of that module. In browsers, this can either be written in normal DOM code, SVG, Canvas, or WebGL (we don't want to mention earlier technologies that are fortunately falling from grace rapidly...).

In addition to this generic complexity, we have to deal with a different workflow cycle even on the level of individual developers: Whereas 15 years ago somebody could set up some HTML files, include some JavaScript files, iteratively add new snippets of code and check the results by reloading the browser, even this small part of the development cycle has changed dramatically over the past 10 years - new Meta-languages like Typescript or Coffeescript on the language side, HTML-meta-markups like HAML, CSS preprocessors like Sass/Scss/Less as well as the integration of modern testing libraries makes a simple browser reload a technique of the past.

Those new methods provide great opportunities (but also challenges) even for the single programmer, which require a whole execution and deployment infrastructure, as depicted in Figure 7.1 and described in the following sections.

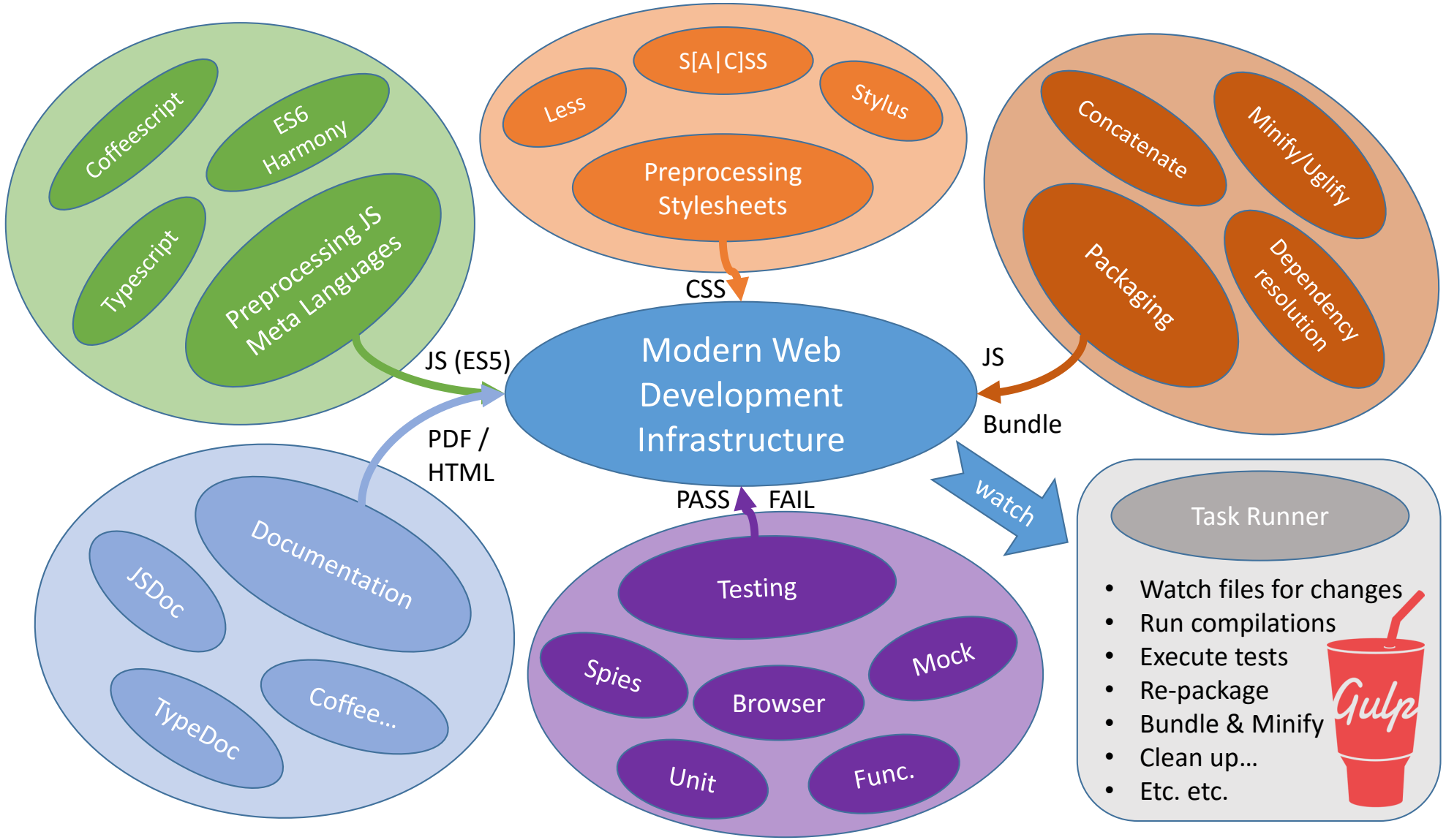


Figure 7.1: Modern Web Development Component Diagram

Graphinius JS

The components presented in this section correspond with the graphical elements in Figure 7.1.

Preprocessing (compiling) JS Meta Languages

Javascript / ES6

Although the JavaScript language was put together by Brendan Eich of the Netscape company within 3 weeks in 1994 (when it was initially called LiveScript, and in 1995 renamed to JavaScript in a marketing attempt to jump on the Java-hype bandwagon), it turned out to be a nifty little language for general in-browser development and computations. JavaScript is a prototype based language, which means that it uses pointers to parent objects instead of class instantiations, features first-order functions (functions which can generate functions) and function-as-object passing, which is ideal for callback-based implementations of the visitor pattern, even more so as JS functions are automatically closures (functions or lambdas that have, regardless of their execution context, full access to their original definition scope).

After almost two decades however, the JS development community felt that requirements on modern web-based product had increased so drastically, that traditional JavaScript could only partially serve them anymore. Problems are - amongst others: 1) the lack of an explicit type system (which is crucial for larger software projects), 2) the lack of an implicit module system allowing for requiring other files or packages (except in the form of browser script tags), 3) the lack of a chaining system for async callbacks (which resulted in the notorious 'pyramid of doom'), 4) the somewhat peculiar functional scoping, which is mostly an entrance barrier to developers of other languages, as well as 5) a general lack of elegant language constructs (like deconstruction of objects into variables etc.).

While some of those problems have been addressed even in the context of ECMAScript 5 (like the introduction of promises to replace nested callback functions), many shortcomings could only be addressed by external libraries, which polluted the workflow with additional dependencies not needed in more sophisticated lan-

guages and often increasing the JS download size by hundreds of kilobytes, which can become a problem on mobile devices.

ECMAScript 6 (codename harmony) is an overhaul of the JavaScript language featuring classes, a new keyword for the familiar block scoping ('let'), as well as an integrated module system allowing to require external files even inside the browser. The greatest obstacle to using ES6 today is the lack of complete support across all browser vendors - this is where ES6-to-ES5 compilers like *Traceur* or the more popular *Babel* come in. As far as syntax goes, ES 6 cleans up some keyword usages in order to make code more readable (samples taken from http://es6-features.org/):

```
1    odds  = evens.map(function (v) { return v + 1; });
2    pairs = evens.map(function (v) { return { even: v, odd:
         v + 1 }; });
3    nums  = evens.map(function (v, i) { return v + i; });
```

Listing 7.1: ECMAScript 5 (usually referred to as 'JavaScript') version of functional programming using the natively built-in mapping function.

```
1    odds  = evens.map(v => v + 1)
2    pairs = evens.map(v => ({ even: v, odd: v + 1 }))
3    nums  = evens.map((v, i) => v + i)
```

Listing 7.2: ECMAScript 6 equivalent to the above code.

Coffeescript

Coffeescript was an attempt to make JavaScript code more readable as well as writable. It was apparently inspired by the clean syntax used in modern scripting languages like Ruby or Python and adopted the use of whitespace as control characters like Python (but not Ruby). Most of the language was based on using 'syntactic sugar' to abbreviate otherwise verbose JS code. For instance, the *this* variable was replaced by the @ sign, the return statement at the end of a function became superfluous, and the lambda operator -> was introduced as a shortcut

for the *function* keyword in normal JS. Examples taken from http://ricardo.cc/2011/06/02/10-CoffeeScript-One-Liners-to-Impress-Your-Friends.html

```
1     [1..10].map (i) -> i*2
2     i * 2 for i in [1..10]
```

Listing 7.3: Two versions of the same mapping functionality in CoffeeScript

Typescript

Testing

Jasmine

Mocha / Chai(!)

Mocking Library (Sinon)

Selenium(??)

Build system for browser

Browserify

request.js

webpack

Manual

Build system

Grunt

Gulp

Documentation

JSDoc

TypeDoc

Graphinius VIS

Description of Nicole's choices and decisions...

```
// Project configuration.
grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    watch: {
        typescript: {
            files: ['**/*.ts'],
            tasks: ['compileTS']
        },
        tests: {
            files: ['**/*.js'],
            tasks: ['mocha']
        }
    },
    uglify: {
        build: {
            src: 'dist/JSAlgorithms.js',
            dest: 'dist/JSAlgorithms.min.js'
        }
    },
    shell: {
        mocha: {
            options: {
                stdout: true,
                stderr: true
            },
            command: 'mocha'
        },
        compileTS: {
            options: {
                stdout: true,
                stderr: true
            },
            command: 'tsc **/*.ts'
        },
        copyMin: {
            options: {
                stdout: true,
                stderr: true
            },
            command: 'cp dist/JSAlgorithms.js /var/www/html/g
        }
    }
});
grunt.registerTask('mocha', 'shell:mocha');
grunt.registerTask('compileTS', 'shell:compileTS');
grunt.registerTask('build', ['concat', 'uglify', 'shell:copyM
```

```
// Compile Typescript to Javascript
gulp.task('build', ['clean'], function () {
  return gulp.src(paths.typescripts, {base: "."})
             .pipe(ts(tsProject))
             .pipe(gulp.dest('.'));
});

// Generate Documentation (type doc)
gulp.task("tdoc", ['clean'], function() {
  return gulp
    .src(paths.typesources)
    .pipe(tdoc({
      module: "commonjs",
      target: "es5",
      out: "docs/",
      name: "Graphinius"//,
      // theme: "minimal"
    }));
});

// Packaging - Node / Commonjs
gulp.task('dist', ['tdoc'], function () {
  var tsResult = gulp.src(paths.distsources)
                     .pipe(ts(tsProject));
  return merge([
    tsResult.dts
                .pipe(gulp.dest('.')),
    tsResult.js.pipe(gulp.dest('./dist/'))
  ]);
});

// Packaging - Webpack
gulp.task('pack', ['dist'], function() {
  return gulp.src('./index.js')
    .pipe(webpack( require('./webpack.config.js') ))
    .pipe(gulp.dest('build/'));
});

// Uglification...
gulp.task('bundle', ['pack'], function() {
  return gulp.src('build/graphinius.js')
    .pipe(uglify())
    .pipe(rename('graphinius.min.js'))
    .pipe(gulp.dest('build'));
});
```

Figure 7.2: Comparison between Grunt & Gulp build systems

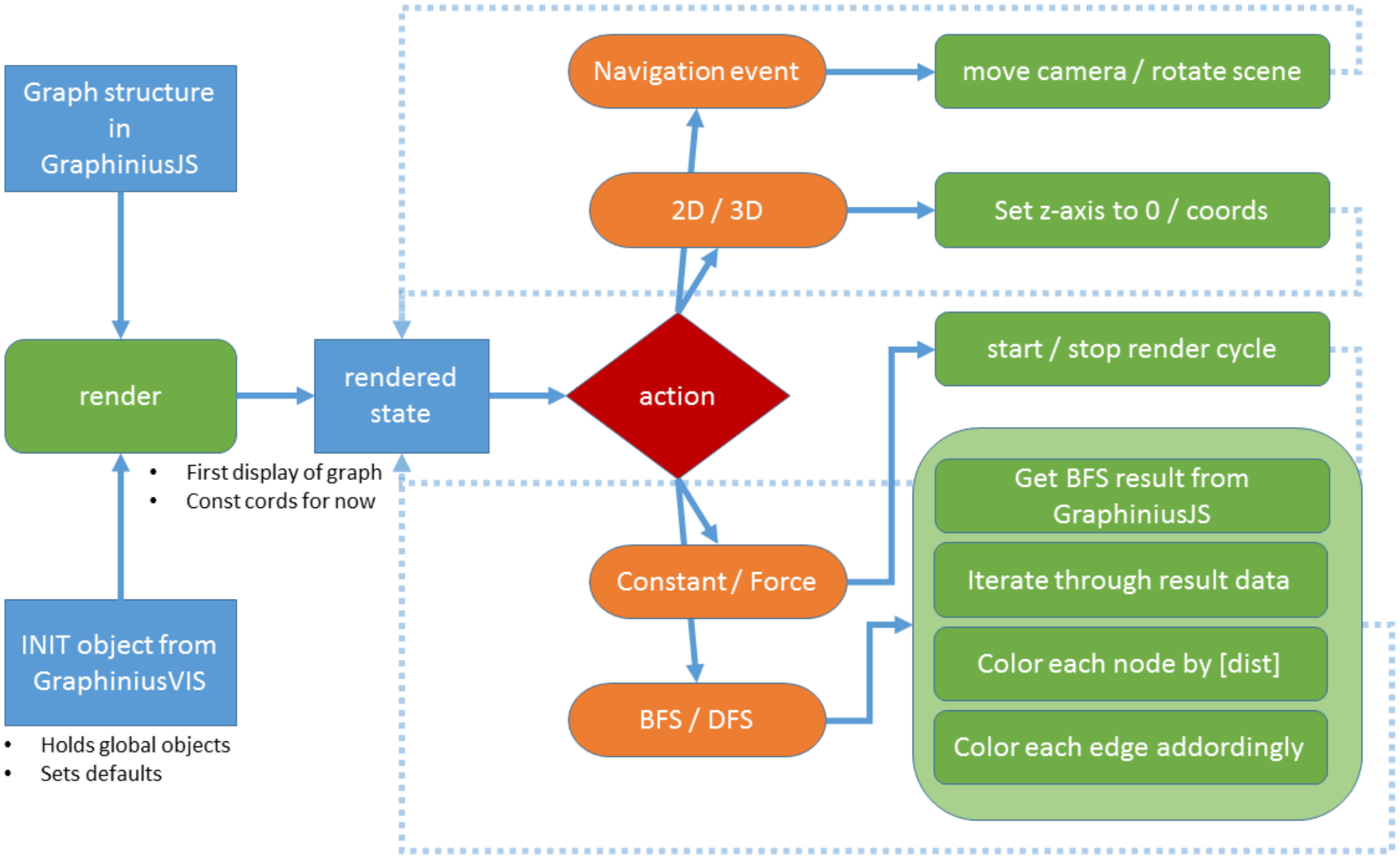


Figure 7.3: Graphinius VIS control flow

Implementation

Development approach

The levels on which testing can occur as defined in Myers and Sandler, 2004 are Unit testing (Module testing), Integration testing as well as Functional (overarching) testing...

Agile

scrum

What I did...

Web Development approach

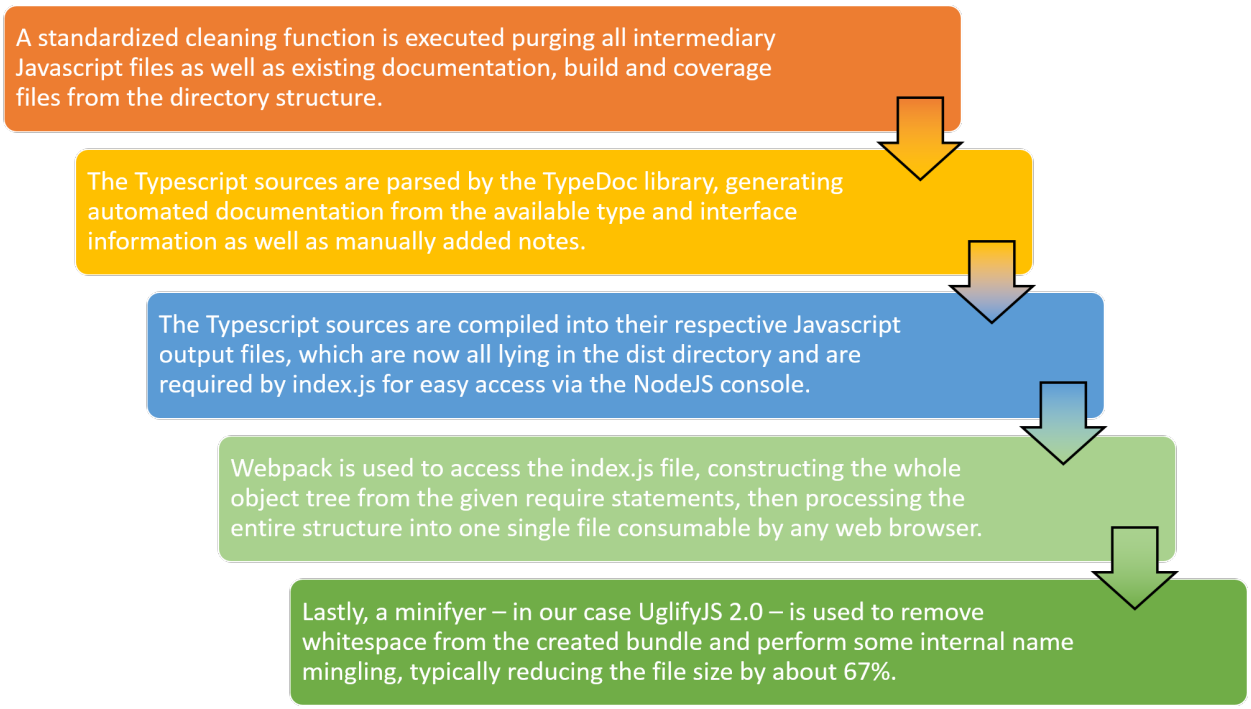


Figure 8.1: The GraphiniusJS Bundling process

Testing approach

BDD...

Unit tests

Functional tests

Mocks used for browser code testing

Spies (Sinon)

Implemented Algorithms

Degree Distribution

Core Search - Graph Traversal

Breadth first search

Depth first search

Best (priority) first search

Topological Sorting / SCC

Shortest paths

Clustering

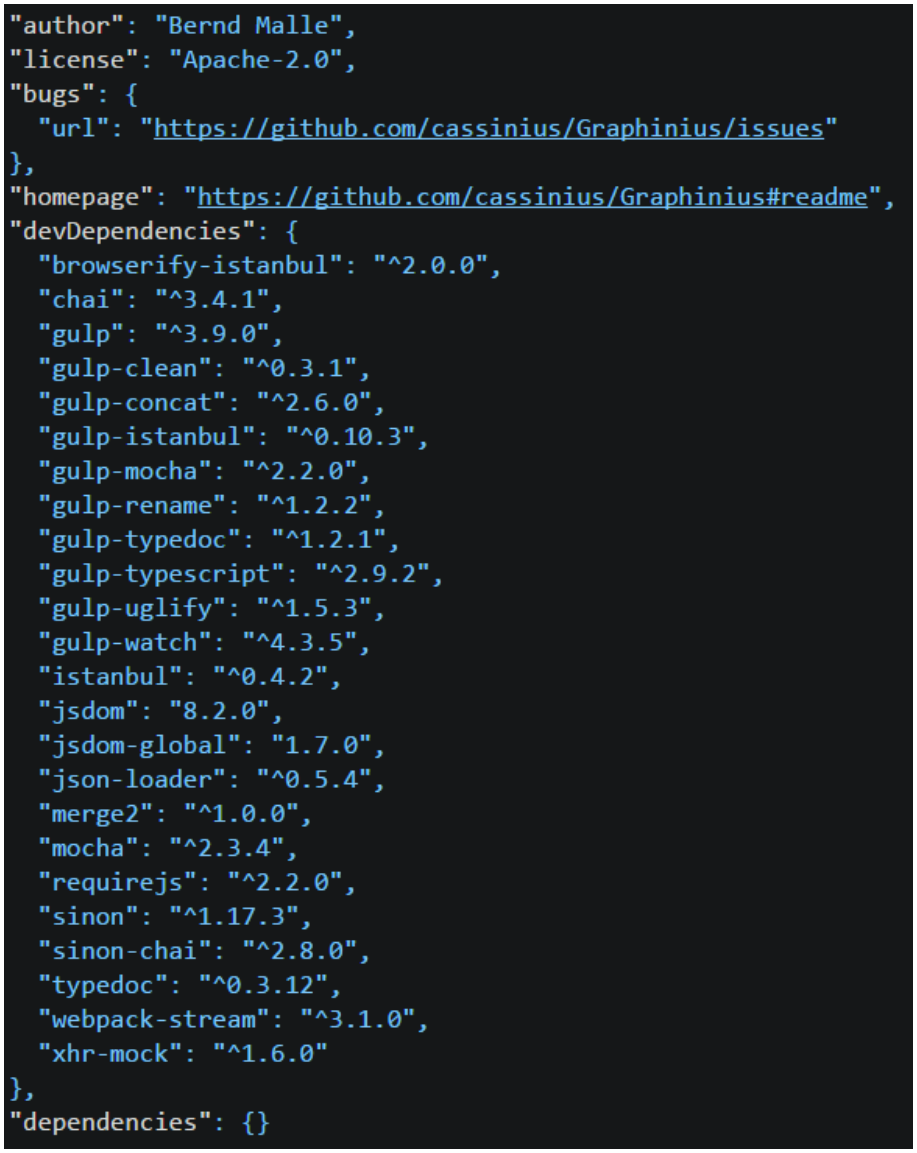


Figure 8.2: GraphiniusJS development and runtime dependencies
As is clearly visible, I focused on managing all complexity during development time, resulting in zero dependencies for the runtime JS bundle.

Implemented Use Cases (Demo)

Manual editing (predefined structures)

1. **Build graph yourself**
2. **Load predefined graph**
3. **Watch visualization update**
4. **run through whole algorithm at once (fluent vis??)**

Graph extraction from images (graphs in preprocessing)

1. **Input image**
2. **Image preprocessing**
3. **graph based segmentation**
4. **visualization**

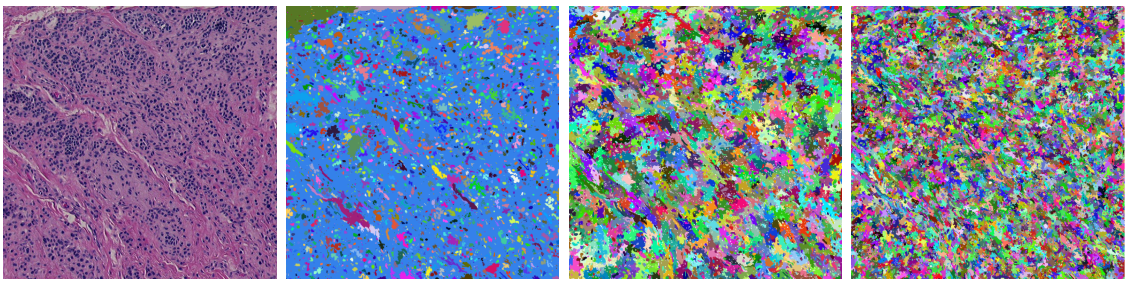


Figure 9.1: Kruskal MST based region merging

Result of applying a Kruskal based region merging algorithm to an image of numerous small scale regular structures. (1) Input image, (2) Result with parameters $k = 1150, s = 0, m = \infty$, (3) Result with parameters $k = 150, s = 5, m = 500$, (4) Result with parameters $k = 50, s = 2, m = 150$.

Anonymity: SaNGreeA (with iML)

1. **Process input data into suitable structure**
2. **Enhance structure with graph information (random)**
3. **Anonymize via SaNGreeA**
4. **prepare individual cost function via iML**
5. **Anonymize via SaNGreeA modified**
6. **Compare results**

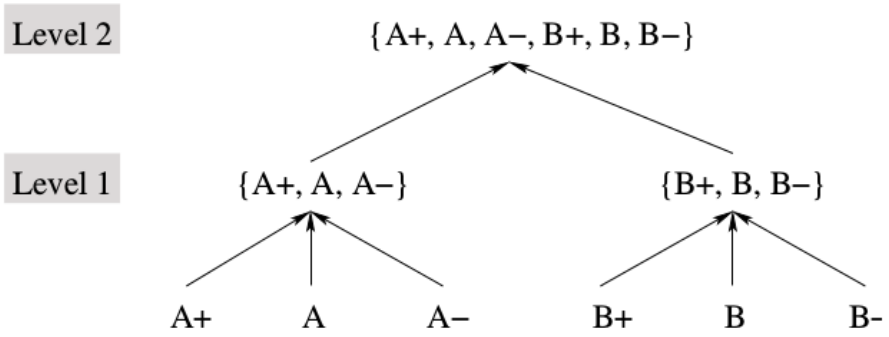


Figure 1: A possible generalization hierarchy for the attribute "Quality".

Figure 9.2: Example of a typical generalization hierarchy

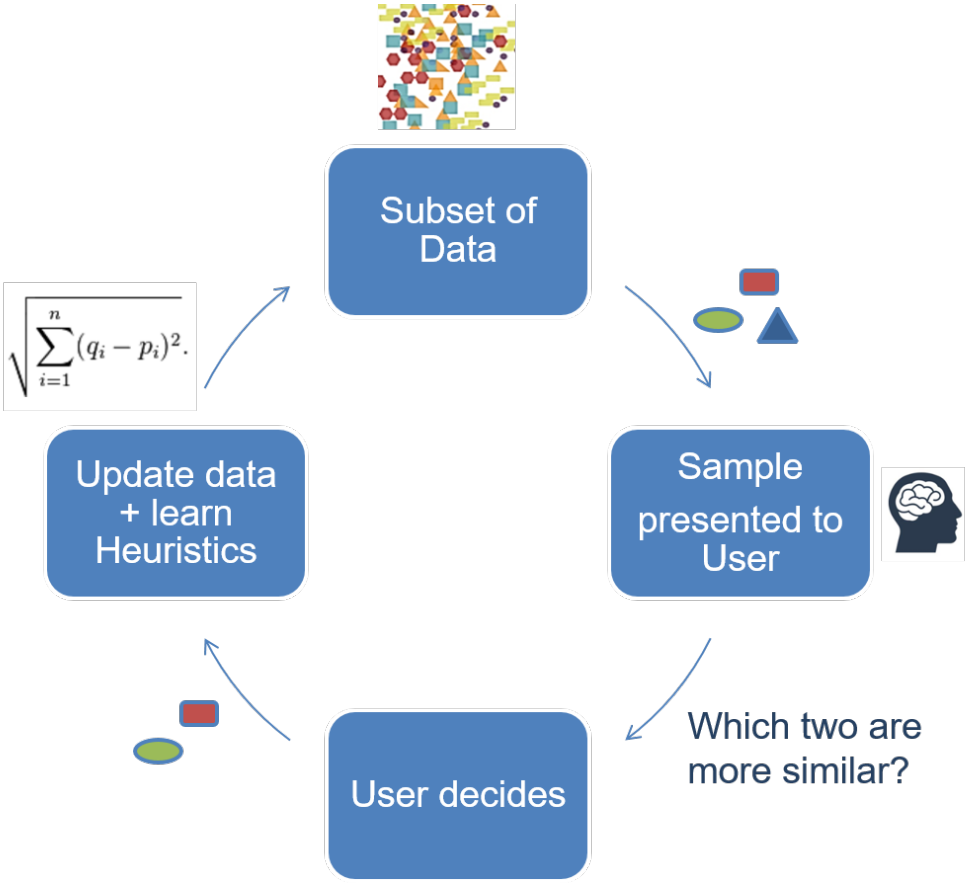


Figure 9.3: Anonymization augmented by IML (human in the loop)

Results and Discussion

Time to implementation

Complexity / size of code

Test coverage

Size of input graphs

Execution speed in various scenarios

if poss., compare with python implementations ;)

Open Problems & Future Work

Future Work

Considering GRAPHINIUS an arbitrarily extendable computing platform (which uses graphs as underlying, universal data structures), there are many possibilities to build upon this work, ranging from small improvements to the introduction of fundamentally new infrastructure, transcending the use of contemporary graph libraries.

The use of a centralized, Web based graphical workflow system will prove especially useful in exploiting and propagates the experience of individual users, as it bundles not only data, but also the settings and results of all experiments conducted on that platform.

Graph generators (graph types)

Graph classifiers

JSVM based grid computing

General processing / ML pipelines

Heterogeneous data linkage

Many research fields comprise several sub-problems which are amenable to different machine learning approaches and feature their own, distinctive input data sets. Coming from various, different data sets featuring their distinct attribute domains, they probably are - via their time-, space- or other dimensions, interlinkable with one another. Usually, studies are only concerned about using a single one of those data sources and applying different methods to it. However, a more holistic approach would be to fuse those data sets along one or more dimensions (or any other mean-

ingful ruleset) in order to achieve a richer representation of the underlying problem. A resulting data-set might take the form of a graph structure, in which individual entities from the originating sets are linked by meaningful connection rules, which in themselves will have to be learned.

Meta machine learning

Meta-learning applies learning algorithms on data collected about machine learning experiments. As one of the first papers regarding this topic, (Rice, 1975) defined the "Algorithm Selection Problem", which was first recognized as a meta learning problem by the machine learning community. He describes five spaces in which the Algorithm Selection Problem plays out:

- **The problem space**: This is the set of all possible input problems (datasets + desired result class).
- **The feature space**: Features of a specific problem or family of problems. In dermatological imaging those would be defined by the imaging method (laser-scan vs. stanza), the scale of the objects to be detected (single cells vs nevi) etc.
- **The algorithm space**: The set of all algorithms suitable for the specific problem (features) to be worked on.
- **The performance measures** (the metric space): The set of possible measurements that could describe the quality of a solution (runtime performance, accuracy, ..).
- **The criteria space**: The weighing of different performance measures considered for a particular solution.

In most scenarios concerning Graphinius we will be concerned with the selection of workflow components and their parameters based on the nature of the specific area of application, the tackled problems, their features, available algorithms, preprocessing methods, parameters as well as performance measures.

Hyper heuristics

Hyper heuristics are a way of selecting or configuring algorithms by searching a space of lower level heuristics instead of searching the solution (parameter) space itself. Hyper heuristics are different from meta learning in that they work independently of the problem domain and therefore promise to be generally applicable; the challenges lie in producing algorithms that do not need to be optimal, but rather good-enough, soon-enough, cheap-enough (Burke et al., 2003).

Although many research fields in themselves are not broad enough to be a suitable proving ground for hyper heuristic research, the Graphinius platform will provide us with meta-data about experiments in many diverse areas. We fully agree with Burke et al., 2013 who concludes that there is still little interaction between research communities, a problem whose solution could lead to the extension of algorithms to both new problem domains and new methodologies through cross-fertilization of ideas.

Meta ML / heuristics database

The greatest advantage of using Meta ML in combination with a centralized, Web based workflow system lies in the fact that users may profit from their colleagues' meta data by building up a collective Meta ML database: input data plus algorithm parameters plus success metrics.

Algorithmic recommender

Conclusion

GraphiniusJS API

List of Figures

Listings

List of Tables

Bibliography

Burke, Edmund et al. (2003). *Hyper-Heuristics: An Emerging Direction in Modern Search Technology*. Ed. by Fred Glover and GaryA Kochenberger. New York: Springer, pp. 457–474. DOI: 10.1007/0-306-48056-5_16. URL: http://dx.doi.org/10.1007/0-306-48056-5_16.

Burke, Edmund K et al. (2013). "Hyper-heuristics: a survey of the state of the art". In: *Journal of the Operational Research Society* 64.12, pp. 1695–1724. ISSN: 0160-5682. DOI: 10.1057/jors.2013.71. URL: http://www.palgrave-journals.com/doifinder/10.1057/jors.2013.71.

Dohmen L., Edlich I.S. and M. Hackstein (2014). "A Declarative Web Framework for the Server-side Extension of the Multi Model Database ArangoDB". In:

Group, Stanford NLP. *The Stanford Natural Language Processing Group*. Stanford University. URL: http://nlp.stanford.edu/.

Holzinger, Andreas, Bernd Malle, and Nicola Giuliani (2014). "On graph extraction from image data". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8609 LNAI, pp. 552–563. ISSN: 16113349. DOI: 10.1007/978-3-319-09891-3_50.

Jones, Zachary M. (2013). *GNU Make for Reproducible Data Analysis*. Pennsylvania State University. URL: http://zmjones.com/make/.

Leskovec, Jure, Ajit Singh, and Jon Kleinberg (2006). "Patterns of Influence in a Recommendation Network". In: *Proceedings of the 10th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*. PAKDD'06. Singapore: Springer-Verlag, pp. 380–389. ISBN: 3-540-33206-5, 978-3-540-33206-0. DOI: 10.1007/11731139_44. URL: http://dx.doi.org/10.1007/11731139_44.

Lorica, Ben (2013a). *Data Analysis: Just one component of the Data Science workflow*. O'Reilly Media, Inc. URL: http://radar.oreilly.com/2013/09/data-analysis-just-one-component-of-the-data-science-workflow.html.

Lorica, Ben (2013b). *Data Analysis: Just one component of the Data Science workflow*. O'Reilly Media, Inc. URL: http://radar.oreilly.com/2013/09/data-analysis-just-one-component-of-the-data-science-workflow.html.

Lorica, Ben (2013c). *Data Science Tools: Fast, easy to use, and scalable*. O'Reilly Media, Inc. URL: http://radar.oreilly.com/2013/03/fast-easy-to-use-scalable-data-science-tools.html.

Lorica, Ben (2013d). *Data scientists tackle the analytic lifecycle*. O'Reilly Media, Inc. URL: http://radar.oreilly.com/2013/07/data-scientists-and-the-analytic-lifecycle.html.

Lorica, Ben (2015). *Building and deploying large-scale machine learning pipelines*. O'Reilly Media, Inc. URL: http://radar.oreilly.com/2015/01/building-and-deploying-large-scale-machine-learning-pipelines.html.

Meng, Xiangrui (2015). *ML Pipelines: A New High-Level API for MLlib*. Databricks Inc. URL: https://databricks.com/blog/2015/01/07/ml-pipelines-a-new-high-level-api-for-mllib.html.

Myers, Glenford J. and Corey Sandler (2004). *The Art of Software Testing*. John Wiley & Sons. ISBN: 0471469122.

Rice, John R (1975). "The algorithm selection problem". In: *Advances in Computers* 15, pp. 65–117.

Sparks, Evan (2014). *ML Pipelines*. UC Berkeley. URL: https://amplab.cs.berkeley.edu/ml-pipelines/.