TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Data Sciences

Antero Lukkonen 178058IABM

BENCHMARKING ENVIRONMENT FOR SERIATION METHODS

Master Thesis

Technical Supervisor

Innar Liiv

PhD

Tallinn 2021

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author:	Antero Lukkonen	....
		(signature)
Date:	10.05.2021	

Annotatsioon

Maatriksite ümberkorrastamine on arvutuslikult kallis andmeteaduse metodoloogia, mille eesmärgiks on maatriksi kujul esitatud andmete sisemise struktuuri avaldamine, läbi ümberjärjestamise viisil, et järjestus oleks optimaaln vastavalt mõõdikuks olevale funktsioonile.

Arvutusliku ressursimahukuse leevendamiseks on loodud eri algoritme, nende variatsioone ning realisatsioone. Tihti on kasutusel heuristikud, mis võimaldavad parendada kiirust, samal ajal ohverdades täpsust. Tasakaalu leidmine täpsuse ja kiiruse vahel ja jõudluse suurendamine on päris maailma rakendustes oluline. Seotud probleemiks on sobiliku algoritmi leidmine antud valdkonna ning sisendi suuruse jaoks.

Metoodiline lähenemine nendele probleemidele on võrdlusuuring ja selles sisalduvad alamprobleemid arvutipõhise eksperimenteerimisega. Nendeks alamprobleemideks on reprodutseeritavus, uuestiarvutatavus ja taaskasutus.

Töös pakutakse välja järjestuste võrdlusuuringuid võimaldav keskkond. Peale ümberkorrastamisele spetsiifilise äriloogika eraldamist isoleeritud taaskasutatavatesse komponentidesse, jõutakse üldistatud arvutipõhiste eksperimentide käivitamise probleemistikuni ning pakutakse välja põhimõtteline arhitektuur, mis lähtub funktsioonidest ja funktsioonide komponeerimisest.

Kuna eelnevad sarnased tööd keskenduvad kasutajaliidestele ning koostöö funktsioonidele, siis antud töö katab vähem puudutatud arvutuslikku hajusrakendust, mis lubaks massilist algoritmide käivitamist ja analüütikat.

Kuna teadusliku ja insenerivaldkonna areng baseerub paljuski eelnevatele tulemustele toetumisel ja iga tööriista edu sõltub selle kasutatavusest ning laiendatavusest, siis töös pakutakse välja, et arvutipõhised eksperimendid peaksid olema valmistatud kasutades tavapäraselt kätte saadavaid komponente, millel on olemas aktiivne kasutajaskond ka väljaspool teaduslikku maailma. Need tööriistad peavad võimaldama korratavust, taaskasutust, taasarvutust ning komponeerimist, et lahendada järgmisi ja keerulisemaid ülesandeid.

Lahendusena pakutakse välja konteinerid ja konteinerite orkestreerimine. Arhitektuuri realiseeriv prototüüp on loodud Microsoft Azure [1] pilve platvormil, kasutades Docker [2] konteinerid ja Kubernetes [3] konteinerite orkestreerimise platvormi, et demonstreerida tarkvara maailmas levinud komponentide kasutatavust.

Lõpuks käivitatakse loodud prototüübil hulk eksperimente, et demonstreerida arvutuslikku ja analüütilist võimekust. Samas on viimistletud tarkvaratoote loomine ressursside puudumise tõttu väljaspool fookust. Eelnevaid võrdlusuuringuid kasutatakse sisendite allikana ning prototüübi tulemuste valideerimisel. Eksperimentide tulemusi võrreldakse eelnevate tööde järeldustega.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 50 leheküljel, 5 peatükki, 27 joonist, 8 tabelit.

Abstract

Seriation is a computationally expensive data science methodology, with a purpose of revealing the inner structure of data represented in a matrix form, through optimizing for a given function. To remedy the high tax on compute resources a number of algorithms and implementations has been created with various heuristics to improve the speed of execution, while sacrificing accuracy. Improving algorithms, finding the most suitable algorithm for a problem domain and input size is crucial. Methodical approach to this problem is called benchmarking and it contains the subproblems of computer based experimentation and reproducibility, recomputability and reusability. In this thesis an architecture for a benchmarking environment for seriation algorithms is proposed. By extracting the seriation domain specific functions into black box isolated components, a generalized problem of computer based experimentation is reached and a conceptual architecture, that derives from concepts of functions and function composition is proposed. As the previous work largely focuses on user interfaces and collaborative aspects of such environments, this thesis focuses on the computation backend, that allows scalable distributed mass execution of seriation algorithms in a reproducible way.

As the basis of scientific and engineering advancement is building on the work of others and the success of any toolkit is dependent on the ease of use and extensibility, the thesis advocates, that computer based experiments should be done using standard off the shelf tools, that enable composing them to be used in further research with easy reusability.

Containerization together with a container orchestrator is proposed as the solution and implementation for the architecture. A prototype environment is created, in Microsoft Azure [1] cloud platform, to show that implementation is possible with widely used components known from software engineering world. Docker [2] is used for containerization and Kubernets [3] as the orchestrator.

Finally a series of experiments are executed on the prototype to demonstrate the computational and analytical capabilities, while a fully functional software as a service is out of scope for this work, due to resource constraints. Previously published research papers on similar topics are used as input sources. Results from the experiments are compared to results from previous research, to validate correctness of the built tool.

The thesis is in english and contains 50 pages of text, 5 chapters, 27 figures, 8 tables.

Table of abbreviations and terms

CBEx	Computer Based Experiment
PCA	Principal component analysis
SQL	Structured Query Language

Table of Contents

List of Figures

List of Tables

1. Introduction

This thesis is exploring technical solutions for doing benchmarking in a data science technique called seriation, by exploring the general challenges related to computer based experimentation.

In paragraph 2 seriation is covered more in depth. As a short context, seriation is a descriptive analytical technique, the purpose of which is to arrange comparable units in a single dimension such that the position of each unit reflects its similarity to the other units [4]. Such units form a matrix data structure with n rows and m columns. There are $n! * m!$ possible arrangements. Manual reordering of such matrixes is clearly not effective, when they get bigger, so a number of computer implementations exist.

As the amount of data is growing daily, algorithms and implementations need to be constantly improved. New-ones need to be invented. Algorithms and implementation have also parameters, that can be adjusted to get variations. This amounts to running a vast number of experiments, requiring the analysis of a large amount of results, making research labor intensive. Many steps in the process of such research seem to be repetitive and are good candidates for automation.

Vitruvius, a Roman architect and military engieer, born around 80–70bc, wrote in his tenth book on architecture: "The difference between machines and engines is, that machines need more workmen and greater power to make them take effect. Engines on the other hand, accomplish their purpose at the intelligent touch of a single workman" [5]. Computers are good machines at doing repetitive tasks. Computers won't do research for us yet, but letting computers do the repetitive work, of wich there are many in empirical research, is well within our reach.

Another complexity related to empirical research that plagues computer based research as well as any other, is the challenge of reproducibility. The complexities of recreating the experiments and analyzing results is a considerable waste in engineering as well as research. It is common to build on the previous theoretical work, but it is more complex to build on the more material artifacts created in previous research. Computer based research is in this area in a special position. It should be far easier to re-build experiments with software, than it is with complex physical objects. Yet there are many difficulties related to doing so. These will be covered with more detail in section 2.3.

The thesis has the following goals it attempts to achieve:

- Suggest engineering practices, that would improve computer based experimentation
- Benefit the field of seriation, by contributing towards an extensible distributed computing backend for benchmarking
- Generalize an accessible computer algorithm benchmarking environment backend architecture
- Build a prototype of the backend architecture
- Run a set of benchmarks on the prototype and present results
- Compare the results with previous research to find validation

To achieve this this thesis is structured broadly in the following way:

1. Introduction to the seriation method and a survery of the previous work
2. Overview of computer based experimentation
3. Overview of engineering tools as solution candidates
4. Architecture description of the environment
5. Description of implementation
6. Results of the experiment
7. Discussion, conclusions and further work

2. Theoretical framework

To set the stage for achieving the goals, an introduction to the domain of seriation, computer based experimentation and an overview of plausible tools is necessary.

2.1 Introduction to seriation

Seriation is applicable to various fields of research like archeology, cartography, social sciences and manufacturing [4]. It enables to find patterns in otherwise seemingly unrelated groups of data, by reordering rows and columns of matrixes. The process is stopped where the arrangement fits a defined criteria. Historically this stopping point was chosen subjectively, depending on the person who did the work.

A thorough history and description of the technique is in-depth covered by I. Liiv [6]. As an illustration for the subject matter, let's take a classical example dataset from J. Bertin [7], that represents the evolving of towns to cities. The dataset is fictional and is meant for demonstration purposes. It answers questions about presence of a property for any settlement.

Example questions that the dataset in table 1 is able to answer:

1. Does a settlement A contain a highschool?
2. Is there a veterinary in B?

The cells of table 1 represent a binary condition, where 0 represents the absence of a property and 1 represents the existence of a propery. A small section of the full dataset is given to save space.

Such a table can be represented also in a matrix form 2.1.

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \tag{2.1}$$

Table 1. Townships with presence or absence of a property

	Highschool	Railway station	One room school	Veterinary
A	0	0	0	1
B	0	1	0	1

To make the dataset visually more expressive, the cells can be colorcoded so, that a 1 is rendered black and a 0 is rendered white. The resulting image in Figure 1 is commony known as a bertin plot, after J. Bertin who introduced this way of visualizing. Figure 1 represents the full unordered dataset.

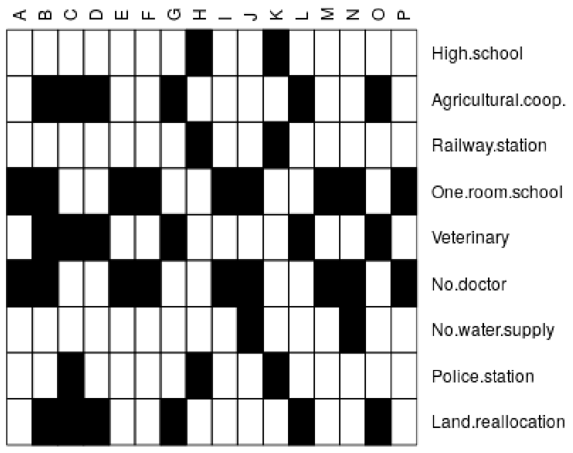

Figure 1. Bertinplot of the townships dataset

To find a pattern in this seemingly unrelated group of data, a seriation algorithm called PCA (Principal Component Analysis), implemented with the free software environment for statistical computing and graphics - R [8], in package Seriation [9], is applied, revealing the structure in Figure 2.

It appears, that there are groups of settlements with similar properties. H and K both have a high school, police station and a railway station. There is also a water supply and they have doctor, but no veterinary. Interpreter of these facts might conclude based on prior experience, that this is characteristic to bigger cities. Applying similar logic to N and J

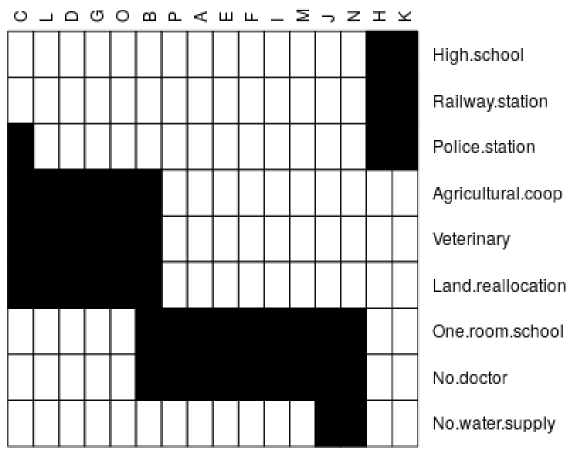

Figure 2. Bertinplot of the reordered townships dataset

suggests, that they could be small villages.

The reordering in Figure 2 is one out of many possible and it might be argued, that the reordering given by Bertin himself, using manual reordering, in Figure 3 gives a clearer hint, that this is an evolution of settlements from rural to urban.

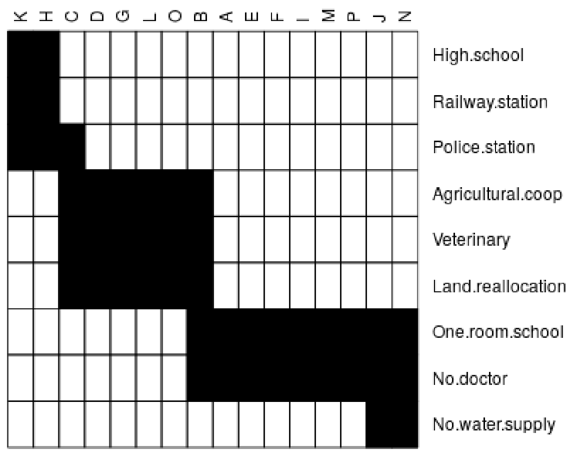

Figure 3. Bertinplot of the townships dataset, ordered by Bertin

J. Bertin himself assessed that a direct graphic processing of matrixes is possible until the size reaches 120 rows and 120 columns [10]. He proposed various machines and tools to help with that.

Since then a number of algorithms have been developed to automate the seriation process. And there exists a number of implementations of the algorithms in different programming languages and computer platforms. The most abundant source for these algorithms is the mentioned Seriation [9] package of the R [8] statistics software platform.

A formal definition of the underlying problem to solve in seriation is necessary, to implement seriation algorithms.

Definition 2.1.1. Seriation is a combinatorial optimization problem, where the goal is to find a permutation function **F** that optimizes the value of a given loss function **L** on equation 2.2, or merit function **M** on equation 2.3 on a matrix **A**.

$$F^* = argmin_F L(F(D)) \tag{2.2}$$

$$F^* = argmax_F M(F(D)) \tag{2.3}$$

The functions L and M are called evaluation functions and they are also one of the ways to assess the performance of the seriation algorithm.

The $n!$ permutations for n objects makes the search space too large for brute forcing all combinations for real world datasets in an attempt to find the most optimal solution. Classical method to approach such problems are heuristics. Results therefore can be approximations or exact.

Some algorithms fit some data better than others. Ideally, we would have general purpose algorithms that extract the underlying structure regardless of the problem domain. In practice this is not necessarily so, similarly to the subjectivity of assessing the beauty of a reordered matrix expressed by a bertin plot.

2.2 Related work on seriation

To propose an environment and technical architecture for a system, that automates part of the work involved in inventing new seriation algorithms and improve existing ones, it is necessary to learn, what is seriation as the domain subject and how this type of research is conducted classically. The related work will serve as the main tool to extract general principles, requirements and validation for the proposed architecture.

The chosen related work is loosely grouped into categories:

- Seriation as a concept
- Comparison of existing algorithms
- Proposing new algorithms
- Proposing collaborative environments

This categorization is based on the weight assigned to each of the categories in the corresponding article, based on the subjective opinion of the author and serves a descriptive purpose.

2.2.1 Seriation as a concept

Into this category fall the works that touch seriation as field of study or application. I. Liiv gives in his article [6] a comprehensive overview of seriation, evolving from a dating method in archeology to a general purpose data mining tool. The article outlines, how seriation is an interdistiplinary method that helps in decision making processes.

G. Toth and S. Amari-Amir rise awareness of the seriation method in the field of chemometrics. In their article [11] they state: "The overview and the assessment of the different merit/loss functions, algorithms, visualization methods, and numerical indicator values seem to be an enormous task where Liiv's unifying article is only the first step." A set of seriation methods from the R [8] package Seriation [9], PAST [12] and an orginially developed method are applied to various chemistry related datasets. For performance comparison, they resorted to a parallel visual check of the data matrix, the object distance matrix, and the variable correlation matrix. The conclusion they make is, that the result of seriation is something that can be justified by human visual perception and not by numbers as long as a comparative study has not cleared up the role of indicators. The source code of the developed algorithm was sadly not easly obtainable.

The R [8] package Seriation [9] focused article by M. Hahsler introduces an infrastructure for creating seriation methods [13]. It is also one of the richest available repository of method implementations together with evaluation methods. This work is more a tutorial focused on practical use of seriation.

2.2.2 Comparison of methods

In this category, the goal is to compare existing methods, to provide objective choices for researchers. No new methods or improvements are introduced.

Michael Hahsler provides a review of the currently most popular seriation criteria and methods for one-mode two-way data using a consistent formulation as an combinatorial optimization problem in operations research [14]. The R [8] package Seriation [9] is used for running experiments and concludes, that different seriation methods highlight different structural aspects of the data, and it can be useful to explore them in the decision making process. Runtime performance is assessed using a "wall clock". Quality of seriation results is assessed using anti-Robinson events and hamiltonian path length. Similarity of the resulting matrixes is assessed using the *Kendall rank order coefficient* [15].

2.2.3 Proposing new algorithms

In the field of genome sequencing a DNA strand is reconstructed from randomly sampled sub-fragments (reads) whose positions within the genome are unknown [16]. The article shows, how this can be mapped to a seriation problem. Algorithms that solve this specific problem efficiently are proposed with runtime performance assessment and deviation from an ideal solution using the Kendall rank order coefficient [15].

J.Liiv in his doctoral thesis explores the idea of using *Kolmogorov complexity*, wich is the length of the shortest effective description of an object, as a way to evaluate the quality of a matrix reordering [4]. As a sufficient approximation for calculating the Kolmogorov complexity, the result length of a compression algorithm gzip [17] is used. Results are evaluated with three criterias and agreement between these criterias is reported. In addition a variation of conformity analysis using SQL (Structured Query Language) is presented.

2.2.4 Proposing new environments

Behrish, Schreck, and Pfister propose as a novel approach a method for user guided matrix reordering [18]. It is a critical view on matrix reordering algorithms, that are based on heuristics and optimization criteria. They argue, that such algorithms behave as black boxes and domain knowledge is required to make educated decisions, what algorithms to choose. The tool they propose compiles a set of interactive, automatic and semi-interactive apporaches to matrix reordering into one user interface. In their processing pipeline, a matrix is projected into 2 dimensional space as vertices, the projection is interactive and allows for applying seriation methods onto parts of it. Manual reordering of parts is possible. Finally the vertexes are re-assembled to a matrix. The prototype gives access to about 70 seriation algorithms and 15 evaluation algorithms, that can be applied to input matrixes on demand. Code is not available and although the authors claim to be using a microservices architecture, it is not clear, what efforts are required to add more methods.

I.Liiv, R. Opik, J. Ubi and J. Stasko propose another web-based tool to support seriation [19]. The focus of this tool is on collaboration, data annotation and turning data into knowledge. The proposed tool sets its weight onto user interface and visualization. In principle, the tool is open for extension and allows for adding new seriation methods as precompiled binaries. Software architecture is monolithic. The same computer that executes the tool, also executes the algorithms. A standing out property of this article is the available source code, including 6 seriation algorithms implemented in the c language.

2.2.5 Summary

The articles produce one or many of the following artifacts:

- Input dataset descriptions
- Inputs datasets
- Evaluation methodology description
- A set of algorithm names
- Algorithm source code
- Visualizations of results in form of various plots
- Numeric results of evaluation

As a general observation, the articles do lack some artifacts of interest.

Even if the algorithms used are well described, the implementations are rarely included. Comprehensive tools, like the user guided matrix reordering solution [18] don’t share the code in an easily available source. An exception is the visual matrix explorer [19].

Also the input data is always not publicly available. Using the artifacts of existing reasearch for further work is cumbersome, due to having to re-create the experiments based on documentation. Similarly reproducing the results is complex if not impossible.

2.3 Related work on computer based benchmarks and experiments

For data science applications, W. Dai and D. Berleant [20] summarized 7 principles for a good benchmark:

- Relevance: Benchmarks should measure important features
- Representativeness: Benchmark performance metrics should be broadly accepted by industry and academia
- Equity: All systems should be fairly compared
- Repeatability: Benchmark results should be verifiable
- Cost-effectiveness: Benchmark tests should be economical
- Scalability: Benchmark tests should measure from single server to multiple servers
- Transparency: Benchmark metrics should be readily understandable

In their best practices article J. Fehr, J. Heiland et. al. state "Like experiments in natural sciences, a CBEx (Computer Based Experiment) should be designed in a way that is robust against uncertainties, i.e., such that it can be replicated to give the same results" [21].

More formally, this is defined as the three R's of Open Science [21]:

- Replicabiliy
- Reproducibility
- Reusability

Based on [21], the three R's can be defined as follows:

Definition 2.3.1. Replicability is the basic capability to repeat a CBEx and obtain the same numerical results. This is similar to the falsifiability of a theory.

Definition 2.3.2. Reproducibility is the practical challenge of reproducing the experiment

Commonly, reproducibility means making available the detailed documentation and building blocks. In software these are the formal algorithm descriptions and implementation used. Also hardware descriptions, if results are hardware dependent, which presents a class of own problems, like how to assemble this hardware.

Definition 2.3.3. Reusability allows to create added value through composability, by using the artifacts of the experiment.

To expand on that, the Recomputation Manifesto [22] states 6 tenets to follow, to more easily achieve the three R's in the context of a CBEx:

- Computational experiments should be recomputable all the time
- Recomputation of recomputable experiments should be very easy
- Tools and repositories can help recomputation become standard
- It should be easier to make experiments recomputable than not to
- The only way to ensure recomputability is to provide virtual machines
- Runtime performance is a secondary issue

While the first 5 points are about enabling deterministic execution of experiments, the last point, about runtime performance being a secondary issue is interesting. It is trivial that an executable experiment with deterministic results has to be available before any performance related evaluation can be done. Being secondary, is due to intrinsic difficulties in black box performance assessment of algorithms. P. Posser demonstrates in his performance study of exact algorithms for maximum clique [23], that running code on different machines will lead to different relative algorithmic performance, even when machines are calibrated and results rescaled. Only way to obtain comparable results, is to recreate the executables and run them on the original machine.

At minimum, executables should be made available. J.Fehr et. al. argue in [21], that source code availability is similar to materials and methods in any other research. The article [21] touches many aspects of what could be called best practices of software engineering. The artifacts, that a CBEx based research should produce are by [21]:

- Source code
- README - a brief description of the functionality and guide how to install
- LICENCE - states the rights for using and modifying the artifact
- RUNME - how to replicate the results, tests
- CITATION - guide how to cite the software project
- AUTHORS - authors and main contributors
- DEPENDENCIES - the third party dependencies of the software
- CODE - various other metadata about the project

GitHub [24], the source code repository provider that crossed the 100 million repository mark in 2018 [25] states in their guidelines [26] similar artifacts:

- README
- LICENCE

- ACKNOWLEDGEMENTS - Equivalent of AUTHORS
- Automated tests and builds - Equivalent of RUNME

These principles and guidelines are in practical software engineering called a definition of done. Similarly, as computer based experimentation is software engineering, these could be called a definition of done for research with CBEx. These principles are achievable with common automation and documentation. A naive approach would still violate the requirement of easy reproducibility, definition 2.3.2, producing yet another program that needs to satisfy these requirements. It would help the given researcher, but not necessarily benefit researchers to come. Given the diverse landscape of operating systems and languages is an inevitability and good thing, similarly as how diversity works in nature, the efforts should be around achieving a common standard interface, rather than one common implementation. In the end there will always remain the lowest common denominator - a tool that everyone needs to have, but this is similar to the requirement of having a computer.

This following sections address the standardization and common interface problem in a world of diverse platforms, by looking at it from systems thinking and human cognition perspective.

2.4 Standardization and automation

As noted previously, experimental computer science research, like any other, depends greatly on the experiment setup, inputs, processes, methods and instruments. Inevitably it suffers also from the problem of disturbances due to environment and taking measurements. The experimentation conditions have to be as similar as feasible, to obtain compareable results. The tools used for experiments need to be standardized and equipment rigorously described. In case of a CBEx, one way to divide equipment is:

- Hardware that runs software
- The runtime environment that makes it possible to run the experiment - operating system and supporting code
- The software that makes experiment code runnable in the environment - a compiler or interpreter

A high-level process of running a CBEx, is on figure 4 in BPMN [27] notation.

The key takeaway is, that from these steps, only analyzing and drawing conclusions is the

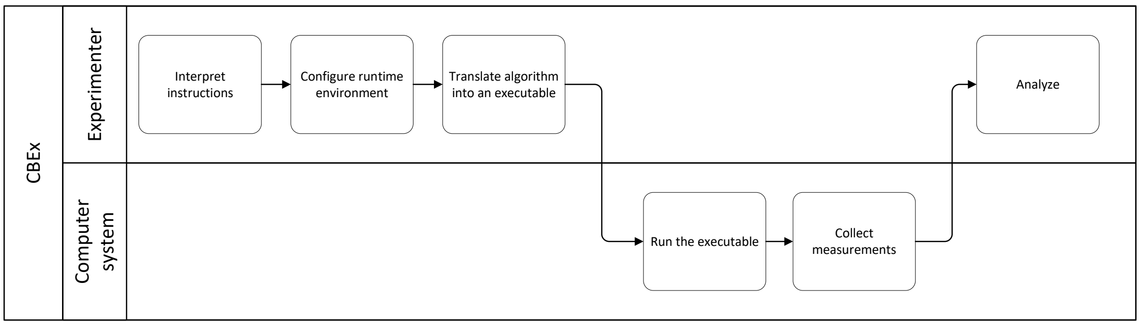

Figure 4. A classical CBEx process

phase where a human is strictly necessary. In terms of Vitruvius [5], a computer is used as a machine and a human behaves as the engine. In context of benchmarking, this means orchestrating inputs, algorithms, evaluation methods and results. On figure 5 a closer to the ideal model of a CBEx is given, where most of the tasks have been moved to the computer system side.

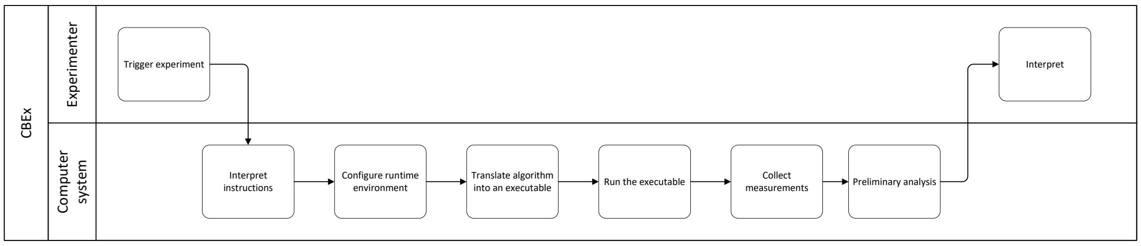

Figure 5. A more ideal sketch CBEx process

2.5 Units of computation

A fundamental problemsolving technique is decomposing complex problems into simpler ones. The inverse is about taking simple solutions and composing them to produce solutions to complex problems. This way of approaching problems is according to [28] a limitation of our short term memory and cognitive abilities and not the intrinsic requirement of the world. Software and producing software is a complex system as they satisfy the criterias set by P. Collier [29]. A complex system that works is invariably found to have evolved from a simple system that worked. The inverse proposition also appears to be true: A complex system designed from scratch never works and cannot be made to work. You have to start over, beginning with a working simple system [30]. In the context of this thesis, it means that there needs to be a set of proven to work and simple to use building blocks that can be orchestrated, to solve a complex problem. It is necessary to define the basic building blocks, the entities that are automated and implement a standard interface. These building blocks should be treated as black boxes. Definition 2.5.1 summarizes, what these these building blocks are.

Definition 2.5.1. Unit of computation is an atomic, black box building block, with a well defined interface, that maps its inputs to output and can produce side-effects.

This is analoguous to a mathematical function, or a function in functional programming, as a field closer to computer based experiments. For the purposes of this thesis and because the idea is to work with black boxes, a function is defined as follows:

Definition 2.5.2. Function $f : X \to Y$ is a mapping of elements from the input X into output Y elements.

Functions can be composed to create further units of computation, compositions as defined in definition 2.5.3.

Definition 2.5.3. Composition of function $f : Y \to Z$ and $g : X \to Y$ is the function $e : Y \to Z = f \circ g$

The third function e is the solution for a more complex problem. Functions can roughly be divided into pure (definition 2.5.4) and impure (definition 2.5.5)

Definition 2.5.4. Pure function is a function that given the same inputs, will always produce the same outputs without any side effects (definition 2.5.6).

Definition 2.5.5. Impure function is a function that is not pure.

Definition 2.5.6. Side effect is any observable change in the world in addition to the mapping of inputs to outputs.

The benefit of pure functions is, that it makes reasoning about them and about the compositions possible. Working with such functions is arguably simpler, as there is no need to account for unwanted changes, they approach the mathematical definition of a function and similar tools can be applied. In context of software, input and output to files, network can be treated as side-effects. In real-world programs side-effects can not be avoided, as without side-effects, a program is unable to produce an observable effect. The goal is rather to write as much as possible without side-effects and push them to the edges of the program. An example of a side-effect in context of reordering a matrix, where the expected output is another matrix, would be writing a file. Such a function is not pure. But on the abstraction layer of certain black boxes, where the standard way of producing output, is writing to a file, it will not be a side effect any more and the function will be pure for given purposes. For example, if a file written with a specific name to a specific place is defined as the output.

Pure functions are the lowest elements of truly reusable software. A composition of functions is a program and the next layer of composition is the composition of programs. Ideally, given input and output are well defined, it becomes possible to reason about whole programs similarly to functions.

2.6 Units of computation for CBEx

The last chapter introduced programs as composable units of computation. Given programs can be written for a variety of platforms and in a variety of languages a function expressed as source code or even as an executable program is not a satisfactory unit of computation. Programs do not exist in isolation, they run in a runtime environment. In addition to composing programs, there is also a need to compose runtime environments.

In Recomputation Manifesto [22], the author emphasizes the use of virtual machines as a means to capture the runtime environment. This is a specific technology with multiple implementations. Other principles in the manifesto are conceptual. At the time of writing, virtual machines were the popular technology and it is safe to assume, that also here a concept of capturing the runtime environment is meant. Since then another also long existing technology called containerization, for capturing the runtime environment has reached maturity.

There are differences in these two technologies as summarized by definitions 2.6.1 and 2.6.2

Definition 2.6.1. Virtual machines capture a full software environment together with virtualized hardware into an image file, that can be shared and started later on different physical machines, that support virtualization, resulting in the same behaviour and execution environment.

Definition 2.6.2. Containers capture the software environment into an image file, that can be shared and started later on different physical machines, that support a container runtime, resulting in the same behaviour and execution environment.

Containers, as it's widely understood today is very similar to virtualization, yet a technically different approach. A full comparison is not the subject of this paper, but it is worth to show key differences.

Properties of a virtual machine:

- Virtualizes hardware

- Runs on top of specialized software called hypervisor
- Requires a full operating system
- Is isolated in its entirety

Properties of a container:

- Includes only the files necessary for the functionality that is containerized
- Runs directly on the host operating system
- Applications are isolated

In summary, a virtual machine provides an abstract machine and a container provides an abstract operating system. Container technology, while gaining popularity in recent history, dates back to 1979, when Unix 7 released the chroot command, that allows to change the root directory of a process, encapsulating it inside this directory. FreeBSD added in 2000 the concept called "jails", that are a partition of the system, and allowed adding an ip address to such a partition [31]. In windows operating system a mechanism called AppContainer exists, that allows isolating processes. It is used for example to create safe runtime environments that prevent potentially malicious code from impacting other resources on the host machine [32].

The choice between the two technologies comes down to non-functional requirements, which stem from the properties of both. Virtual machines tend to require more host system resources such as memory and storage space, as they contain the full machine. Because containers run on top of a host operating system, they can be run on virtual machines, to get the best of both worlds.

2.7 Container technology overview

To introduce containers, a short list of commonly used containerization technologies is given. This list is not exhaustive as this space is moving fast and many smaller new implementations of the technology emerge and disappear often.

2.7.1 LXC

LXC is a userspace interface for the Linux kernel containment features. Through a powerful API and simple tools, it lets Linux users easily create and manage system or application containers. LXC containers are often considered as something in the middle between a chroot and a full fledged virtual machine. The goal of LXC is to create an environment as close as possible to a standard Linux installation but without the need for a separate kernel. [33].

LXC is a platform with the most market share of 34.75% [34].

2.7.2 Docker

Docker is a platform for developers and sysadmins to build, run, and share applications with containers [2]. It commands the second biggest market share of 25.2% [34].

The main components of Docker are:

1. Image - a set of instructions for how to create a container
2. Container - a runnable instance of an image
3. Daemon - manages Docker objects such as containers and images. Exposes an interface to take commands
4. Registry - stores images

The solution also provides tools to interact with the Daemon.

2.7.3 Rkt

Rkt is an application container engine developed for modern production cloud-native environments [35]. The market share of Rkt is 5.86% [34]. The main components of Rkt are:

1. Image - a set of instructions for how to create a container
2. Container - a runnable instance of an image
3. Command line client - manages Rkt system directly, instead of delegating to a central daemon

Main high level difference is that there is no central daemon process that controls the system, but the client code itself takes care of managing Rkt objects. There is also no central image registry. Rkt project has been deprecated by the governing body, so it is not wise to invest in this technology.

2.7.4 Singularity

Singularity is a container platform. It allows you to create and run containers that package up pieces of software in a way that is portable and reproducible [36]. This is a platform, that has been created from ground up, keeping high performanc computing in mind. Meaning it is well suited for running in clusters and includes certain architectural choices, that make sense in these environments.

Being a very domain specific platform, it does not figure on mainstream market share analysis.

The main components of Singularity are:

1. Image - a set of instructions for how to create a container
2. Container - a runnable instance of an image
3. Command line client - manages Singularity system directly, instead of delegating to a central daemon

Additional important characteristics:

1. Single file format - with the purpose of being easier to share and a caveat of being larger in size.
2. Linux focus - tooling and documentation does not touch popular platforms from Microsoft and Apple.

For interoperability, there exist tools that enable conversion to and from other container formats, like Docker.

2.7.5 Open Container Initiative

Every technology has a lifecycle and an end of life. Choosing in favor of a technology stack includes the risk, of this stack being deprecated, illustrated by the deprecation of the Rkt. Having a vibrant and vast community as well as organizational support for an implementation will help, but standardization will enable evolution and interoperability.

For containers this is Open Container Initiative. The Open Container Initiative is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes [37]. A vast majority of containerizarion technologies implement and contribute to the standards.

2.7.6 Summary

There are many container platforms, that target a special flavour of user, a certain workflow, or are just the newest fashion. The open container initiative standard enables moving between these platforms and increases chances, that the container image will also be runnable in a somewhat not so immediate future. An example of this is the convertibility of Docker and Singularity images. Consolidating to a single platform is an elusive goal, it is more important to be able to evolve and embrace change. There will always be the next thing and restricting choice would hinder evolution. This thesis uses in it's prototype the very common Docker platform. Nothing would prevent the same architecture to also work with the Singularity images.

2.8 Composing containers

To be useful as a unit of computation, containers need to support composition given by definition 2.5.3. Candidates for achieving this are known from general communication methods between computer processes:

- Shared volumes - definition 2.8.1
- Container network - definition 2.8.2
- Inter process communication - definition 2.8.3

Definition 2.8.1. Shared volume is a file system like structure exposed to containers by the container runtime and allows the containers to see the same files and directories.

Definition 2.8.2. Container network is a private network formed between containers, giving them access to each other using TCP/IP protocols.

Definition 2.8.3. Inter process communication are operating system exposed methods for processes to signal each other. Different operating systems provide different capabilities.

It is certainly possible to compose containers manually, using one of the given methods, by writing special code, that imperatively orchestrates calling multiple containers. As this is a common problem, domain specific tools have been created for such purposes. This chapter is about introducing some commonly used tools to achieve that.

2.8.1 Docker compose

Compose is a tool for defining and running multi-container Docker applications [38]. Traditionally this has been a light weight way of defining development and testing environments. In addition to standard Docker components, it adds two more:

1. docker-compose - command line client and interpreter for the special compose file
2. compose file - a domain specific configuration file for defining all required containers and their connections

2.8.2 Docker swarm

A cluster of Docker Engines that act as managers and workers [39], it adds the following components:

1. manager - manages cluster membership and work delegation
2. worker - runs the containers

These components work to keep the requested state of the cluster, that includes keeping a certain number of containers running, making sure the requested operations will be retried if they happen to fail.

2.8.3 Kubernetes

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications [3]. It is a vast system with many components and detailing all of them is not the goal of this paper, but a general overview is below.

The principal component of a Kubernetes cluster is a Pod, definition 2.8.4.

Definition 2.8.4. Pod Is the smallest unit of compute in Kubernetes. It contains one to many containers.

Containers in a Pod can form a composition through the use of init containers, definition 2.8.5.

Definition 2.8.5. Init containers are containers that run only once in defined sequence during a Pod lifecycle, before the main container starts.

Other systemic components are:

- kube-apiserver - Exposes the management api. All control goes through api server
- etcd - as a distributed system, Kubernetes needs a distributed consistent storage for it's state
- kube-scheduler - takes care of assigning Pods to Nodes.
- kube-controller-manager - runs controllers, that are responsible for responding to various events in the cluster, such as maintaining the requested state of the cluster.
- cloud-controller-manager - is the integration layer between a possible cloud provider like Microsoft Azure of Amazon Web Services [40].

Work in a Kubernetes cluster will be handled by a Node, definition 2.8.6.

Definition 2.8.6. Node is a computer, registered in the Kubernetes cluster, that provides a compute environment for other cluster participants.

A node runs in addition to the operating system services, the following components:

1. Node
   (a) kubelet - An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod
   (b) container runtime - software that runs containers, such as Docker.
   (c) kube-proxy - handles networking requirements

3. Benchmark environment for seriation

Having introduced seriation, challenges and requirements around CBEx, concepts and tools that potentially allow solving them, it is time to look at principial functional requirements of a benchmarking system for seriation.

Existing research around seriation methods asks the following questions:

Question 3.0.1. Given data represented in a matrix form A, what is the inner structure of the given data?

Question 3.0.2. Given an input matrix A, what is the optimal implementation for a given seriation algorithm?

Question 3.0.3. Given an input matrix A, is there an method that suits better a given situation?

Question 3.0.1 is the primary question of seriation, as described in detail in chapter 2.1. Finding answers to the questions 3.0.2 and 3.0.3 can be called benchmarking. Definition 3.0.1 formalizes these questions to an activity.

Definition 3.0.1. Benchmarking is the process of comparing the results of an algorithm implementation against an ethalon result.

The purpose of asking these questions is to turn facts into knowledge. Problem domains can be formed and when matched to algorithms that work best, will result in a set of recommendations that makes it faster to get from questions to answers. If an algorithm has parameters, then knowing the problem domain allows to have insight, what parameters to choose for an algorithm. Also an implementation, that finds the solution, but takes infinite time to find it, is of no practical use. In the case of seriation, when input datasets grow, the brute force processing times grow exponentially. This quickly becomes impractical. A classic way of solving this is to apply a heuristic as a short-cut. An example of such a general purpose algorithm is the Bond Energy Algorithm or BEA [41]. Heuristic approaches don't usually result in the ideal ordering, rarther an approximation, sacrificing accuracy for speed.

Algorithms can be analyzed broadly as a white-box or black-box object. A black-box approach to algorithm benchmarking means assessing the performance solely based on inputs, outputs and measurements, that an observer can make. A white-box approach analyzes the algorithm complexity, with access to source-code and models of the algorithm. The subject of this thesis is a black-box approach, as the goal is to empirically explore performance of seriation algorithms.

3.1 Domain concepts

The following core domain concepts are necessary to allow for reasoning about the questions in previous chapter.

Definition 3.1.1. Ranking is a monotonically decreasing weakly ordered sequence of objects in a set.

Definition 3.1.2. Evaluation is a mapping $e : A \to \mathbb{R}$, where A is a matrix of Definition 2.1.1 and $e \in \{L, M\}$ of Definition 2.1.1.

Definition 3.1.3. Ranking of seriation methods R_f is a ranking of tuples $t_r = (f, e(A))$, where the order between two elements is defined by the value of $e(A)$.

Result of Evaluation is an exact value. Additional characteristics of the seriation process itself are handled by measurement. Examples of such measurements are the wall clock runtime and used memory.

3.2 Benchmarking

Processes involved are described in form of business process modelling notation or BPMN [27]. Only processes that will be automated in prototype, are described. These processes will be the base for system components and map almost one to one to component diagrams in chapter 3.3. The diagrams are accompanied with small commentary. The abstraction level is kept fairly high.

Process 3.2.1. Adding an input matrix A. The system will in parallel calculate all reorderings for this input and then analyzes the produced reorderings according to Process 3.2.5. See Figure 6.

Process 3.2.2. Adding a seriation method. All input matrixes A are reordered using this new method and again outputs are analyzed in bulk. See Figure 7.

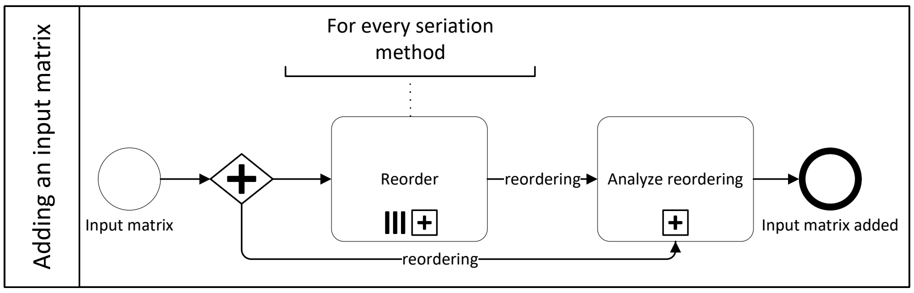

Figure 6. Process 3.2.1 Adding an input matrix

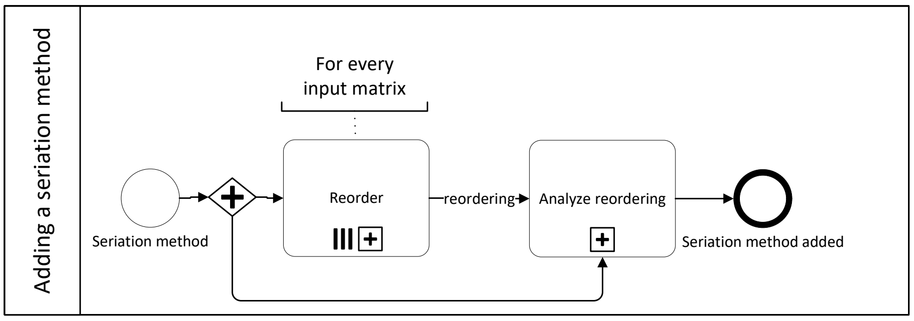

Figure 7. Process 3.2.2 Adding a seriation method

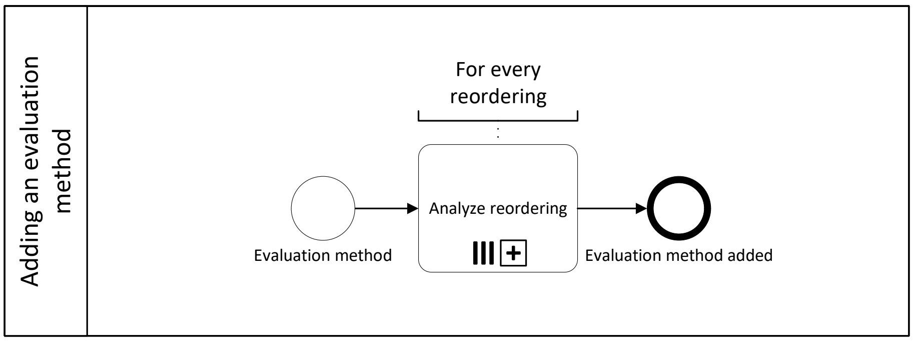

Figure 8. Process 3.2.3 Adding an evaluation method

Process 3.2.3. Adding an evaluation method. All already existing reorderings are evaluated. See Figure 8.

Reordering and evaluation both contain an identical subprocess Process 3.2.4 for executing the algorithm to map inputs to results. On figure 9 there are generic tasks, that are about working with input and output and a task called Process, where mapping inputs to output happens. This is where seriation and evaluation algorithms are executed.

Process 3.2.4. Map input to output. This process is about finding the raw data to answer the question 3.0.1, about revealing the inner structure of a matrix. See figure 9.

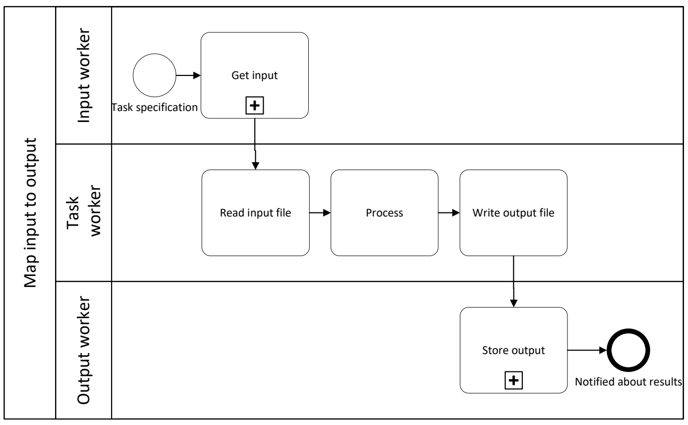

Figure 9. Process 3.2.4 Calculate seriation and evaluation results

By now, the necessary raw reorderings are generated. To answer questions about the optimal implementation 3.0.2 and domain suitability 3.0.3, further analysis is needed.

Process 3.2.5. Analyze reordering. This process consists of finding all evaluations for a reordering and analytics, to turn raw data into knowledge. See figure 10.

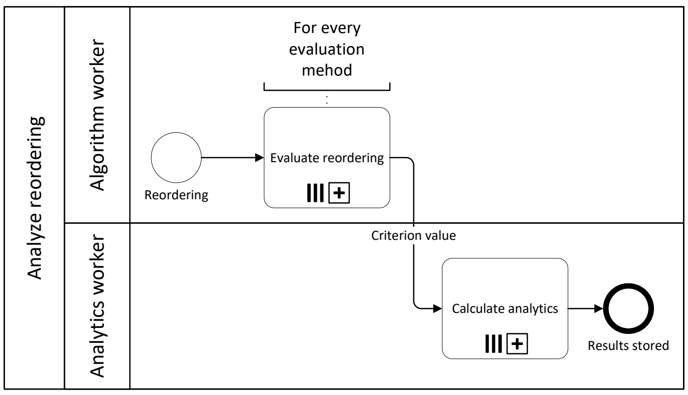

Figure 10. Process 3.2.5 Analysis of reordering

Figure 11 shows finding the final ranking. Similarly to an athletics competition, where every seriation is a competitor, an evaluation method is a event and an input is a competition.

This ranking table serves as a basis of further analytics - finding the top and bottom seriation methods.

Process 3.2.6. Analyze evaluation results. For every input matrix in the system, rankings (definition 3.1.1) are calculated.

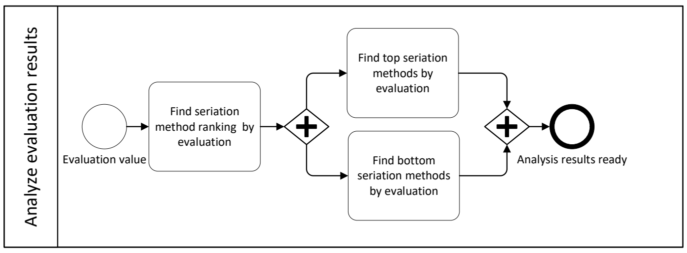
Analyze evaluation results
Evaluation value
Find seriation method ranking by evaluation
Find top seriation methods by evaluation
Find bottom seriation methods by evaluation
Analysis results ready

Figure 11. Process 3.2.6 Analyze evaluation results

3.3 Implementing components for the business processes

The main components of the system with interfaces they require and provide are described using the Unified Modelling Language component diagram notation. A component is defined by definition 3.3.1. Interactions of components are described in chapter 3.4.

On diagrams, the components contain their stereotypes in «» brackets. The components and interfaces are logical, they don't imply a particular implementation or a communication protocol.

- User - A human actor that takes on various roles.
- Infrastructure - generic components that enable functions like storage and messaging.
- Pod - a unit of comuptation as given by Definition 2.8.4.
- Container - See Definition 2.6.2.

Definition 3.3.1. Component is a high level part of the system that can be independently changed, as long as the interface is satisfied.

System user roles:

Component 3.3.1. Method author wants to benchmark a new version of a method.

Component 3.3.2. Data analyst is interested in extracting reports about benchmarking results, to answer benchmarking questions.

Infrastructure components:

Component 3.3.3. Queue is a messaging solution with first in first out semantics.

Component 3.3.4. Database is a queryable storage for structured data.

Component 3.3.5. Blob storage is a service for storing unstructured binary data, such as an image or a comma separated file.

Component 3.3.6. Kubernetes is a container orchestrator, described in chapter 2.8.3.

Functional components:

Component 3.3.7. API/UI takes input and makes output accessible to users and external systems. See figure 12.

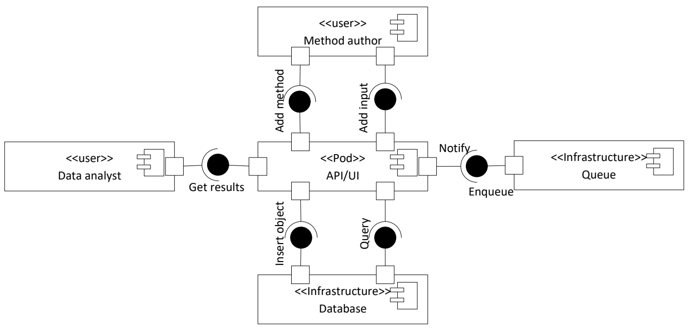

Figure 12. Component 3.3.7 API/UI

Component 3.3.8. Work dispatcher contains an algorithm for applying methods to inputs, via scheduling a job for each one. See figure 13.

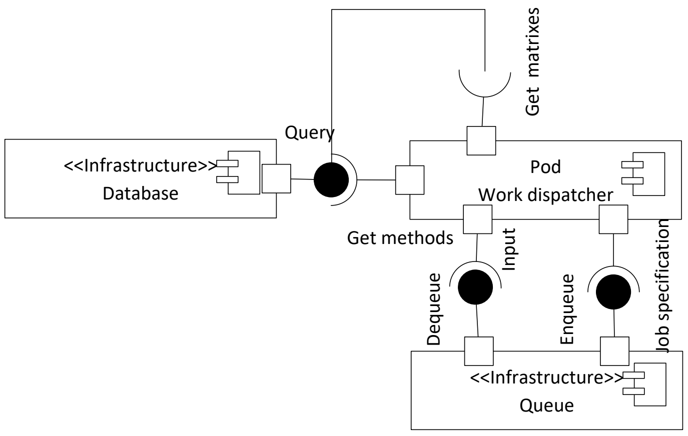

Figure 13. Component 3.3.8 Work dispatcher

A work dispatcher component 3.3.8 on figure 13 is needed to decide what seriations and evaluations need to happen. This can be all to all in case of the process 3.2.1, or a subset to subset, in case of the processes 3.2.3 and 3.2.2. Since the component needs to know about existing methods and inputs, it needs access to the database component 3.3.4. Work is handled by jobs, that are in turn scheduled in the system by the job dispatcher component 3.3.9.

Component 3.3.9. Job dispatcher ranslates the job description into an acceptable format for a job controller component 3.3.10. See figure 14.

Component 3.3.10. Job controller makes sure jobs in the system are executed in a resilient manner, with retries. It tries to ensure at least once semantics, that at least one instance of the job executes successfully. See figure 14.

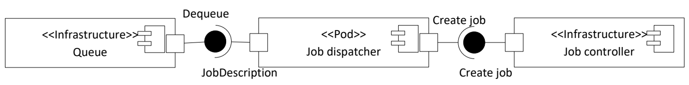

Figure 14. Component 3.3.9Job dispatcher

Component 3.3.11. Task worker takes care of executing jobs, the seriation and evaluation methods. It uses a blob storage component 3.3.5 for storing unstructured output and a database component 3.3.4 to store evaluation results and metadata about unstructured results. See figure 15.

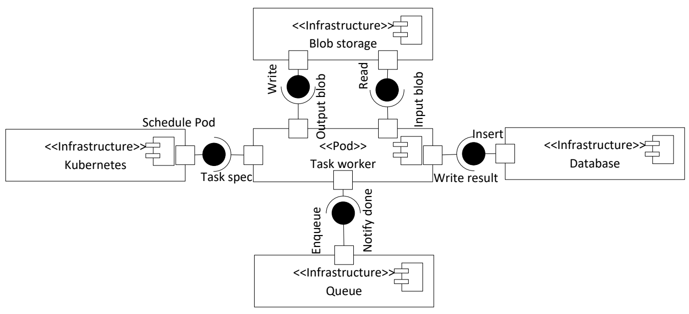

Figure 15. Component 3.3.11 Task worker

After storing the results, it needs to notify other system participants, that it has done its work using a queue component 3.3.3. A detailed description of internals of the task worker is on figure 16.

The input worker component 3.3.12 handles downloading the unstructured input, specified by the task specification. Work is done in the algorithm worker component 3.3.13.

Component 3.3.12. Input worker ensures that inputs for the main worker 3.3.13 are available.

Component 3.3.13. Algorithm worker maps input to output.

The output worker component 3.3.14 stores the unstructured results to blob storage component 3.3.5, numeric results and metadata into database component 3.3.4.

Component 3.3.14. Output worker takes the output produced by task worker component 15 and ensures it gets stored. It also notfies downstream actors about task completion.

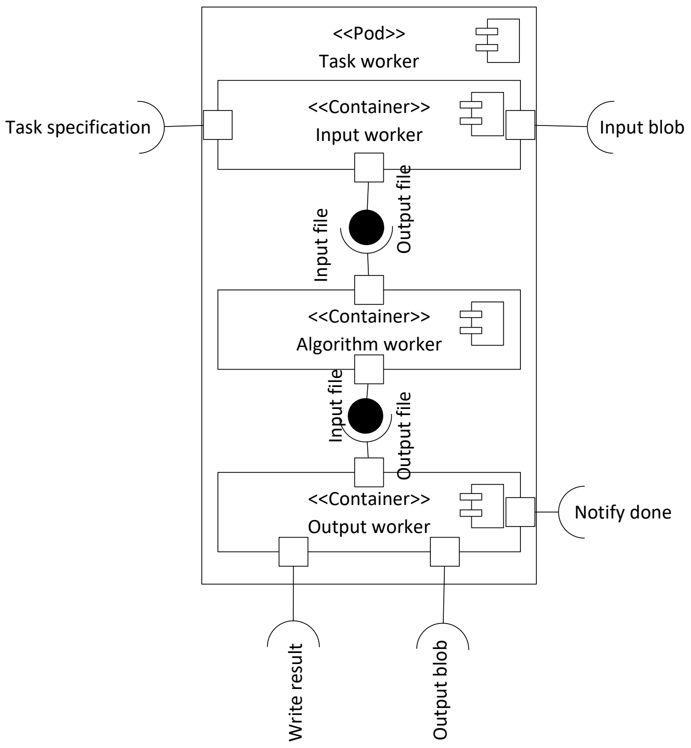

Figure 16. Component 3.3.11 Task worker internals

3.4 Interactions of components

Primary component interactions are described with UML sequence diagrams. Adding a seriation methord is shown on figure 17. This sequence covers also reordering inputs, evaluating results and analytics. As it would not add new information, these sub-sequences are not covered on separate diagrams.

A single task of seriation or evaluation is handled by identical components with an identical sequence on figure 18.

3.5 Implementation

A prototype implementation for the architecture in chapter 3.2 was created using the C# programming language targeting the .NET Core 3.0 [42] runtime.

In the spirit of prototyping, the technology choices and runime were chosen keeping the rapid iteration and simplicity for the author in mind. Programming language was chosen on the basis of authors prior experience, to reduce the amount of new things to learn.

All components above were packaged as Docker [2] container images and composed as relevant Pods, see definition 2.8.4, for running them in a managed Azure Kubernetes

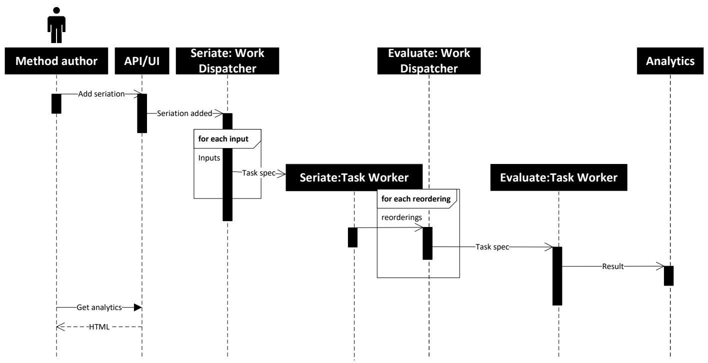

Figure 17. Sequence to add a seriation method to the system

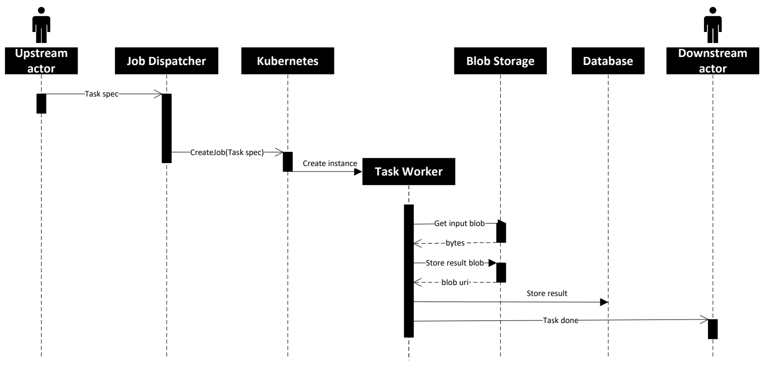

Figure 18. Sequence to process a reordering or evaluation

Service [43] cluster.

Choice for Microsoft Azure [1] is due to funding. The author has monthly credit in Microsoft Azure cloud platform, that can be used for educational and personal purposes. As the core components are cloud platform agnostic, then this his will not change the applicability of the architecture on other platforms.

Infrastructure component implementation choices follow naturally from Azure platform offerings and are given in table 2. With the exception of the queue implementation, where the criteria for choice was simplicity of local development.

Table 2. Infrastructure component implementations

Component	**Implementation**
Queue 3.3.3	RabbitMq messaging queue [44]
Database 3.3.4	Microsoft CosmosDb object oriented database [45]
Blob storage 3.3.5	Microsoft Azure Storage [46]
Job controller 3.3.10	Kubernetes Jobs [47]

In the prototype, a variety of algorithms are included, packaged as docker container images. Inputs are gathered from related work and the the R[8] statistics package. Table 3 summarizes the counts of different entities.

Table 3. Prototype statistics

Inputs	**Seriation methods**	**Evaluation methods**	**Reorderings**	**Total operations**
42	38	12	1596	22344

The code is available from GitHub https://github.com/antero-lukkonen/serikube.

Examples of seriation algorithms implemented with containers are demonstrated in the chapter .1.1 and chapter .1.2.

3.6 Algorithms

Thanks to the popular R[8] package Seriation [9], there is a number of seriation and evaluation algorithms available. A subset of them is included in the prototype. To prove the capability of using different programming languages, algorithms Minus and Plus Technique and Count Ones written in C are added [19]. The algorithms are packaged as Docker images, that satisfy the input and output interfaces described in the chapter 3.3. These images are uploaded to the Docker image registry with a fully qualified name "serikube.azurecr.io/serikube". Table 4 lists seriation methods in the porototype along with the Docker image name with the repository name prefix omitted to save space.

Table 4. Seriation methods in the prototype

Method	**Docker image**
BEA_TSP	r-seriate-matrix
BEA	r-seriate-matrix
PCA	r-seriate-matrix
TSP	r-seriate-dist
OLO	r-seriate-dist
HC	r-seriate-dist
GW	r-seriate-dist
ARSA	r-seriate-dist
R2E	r-seriate-dist
Original ordering	noop
Count ones	kurtmoser/loenda1
Plus	kurtmoser/plusstehnika
Minus - standard	kurtmoser/miinustehnika
Minus - 0-con	kurtmoser/miinustehnika
Minus - 1-con	kurtmoser/miinustehnika
ROC2 C++	tanelpipar
Modified ROC C++	tanelpipar
Zodiac C++	tanelpipar
ART C++	tanelpipar
Random	r-seriate-matrix
QAP_BAR	r-seriate-dist
SPIN_STS	r-seriate-dist
HC_single	r-seriate-dist
GW_complete	r-seriate-dist
SPIN_NH	r-seriate-dist
MDS_nonmetric	r-seriate-dist
Spectral	r-seriate-dist
MDS_metric	r-seriate-dist
MDS_angle	r-seriate-dist
PCA_angle	r-seriate-matrix
HC_complete	r-seriate-dist
HC_average	r-seriate-dist
OLO_complete	r-seriate-dist
Spectral_norm	r-seriate-dist
VAT	r-seriate-dist
QAP_LS	r-seriate-dist
QAP_2SUM	r-seriate-dist
QAP_Inertia	r-seriate-dist

The image names convey meaning:

- r-seriate-matrix - R based seriation method that expects a matrix of values as input
- r-criterion-matrix - R based evaluation method that expects a matrix of values as input

- r-seriate-dist - R based seriation method that expects a matrix of eucleidian distances as input
- r-criterion-dist - R based evaluation method that expects a matrix of eucleidian distances as input

Similarly to seriation methods, a wealth of evaluations comes from the R Seriation package. A variety of using the gzip [17] compression algorithm as an evaluation is included, in the spirit of using the Kolmogorov complexity as proposed by I. Liiv [4].

Table 5. Evaluation methods in the prototype

Method	**Docker image**
Path_length	r-criterion-dist
Least_squares	r-criterion-dist
Moore_stress	r-criterion-matrix
Neumann_stress	r-criterion-matrix
Gradient_raw	r-criterion-dist
Gradient_weighted	r-criterion-dist
Inertia	r-criterion-dist
ME	r-criterion-matrix
Gzip	gzip
PImage size	r-seriate-pimage-size
PImage	r-seriate-pimage
Bertinplot	r-seriate-bertinplot
AR_deviations	r-criterion-dist
AR_Events	r-criterion-dist

3.7 Experiment results

The prototype was used to run experiments on the inputs gathered from previous work. The aim of these experiments was to replicate previous research and find validation for the proposed architecture. For practical reasons, only a selection of results produced by the prototype are included in result tables. Results presented below are made available also through user interface of the prototype. Summary statistics of the performed experiments is in table 3.

On table 6 is a selection of three best ranked seriation methods by evaluation method for the input Jaccard index for incidence matrix for 59 graves and 70 artifacts [48].

Table 6. Top evaluations for input matrix Jaccard index for incidence matrix for 59 graves and 70 artifacts (Hodson, 1968).

Method	Top evaluation 1	Top evaluation 2	Top evaluation 3
ARSA	1 AR_Events	1 AR_deviations	1 Least_squares
ART C++	3 Gzip	20 Moore_stress	20 ME
BEA	13 ME	14 Neumann_stress	15 Path_length
BEA_TSP	1 ME	1 Neumann_stress	1 Moore_stress
Count ones	11 Gzip	23 Inertia	25 AR_deviations

Table 7 gives similar results for three worst ranked seriation methods for the same input.

Table 7. Bottom evaluations for input matrix Jaccard index for incidence matrix for 59 graves and 70 artifacts (Hodson, 1968).

Method	Bottom evaluation 1	Bottom evaluation 2	Bottom evaluation 3
ARSA	31 Moore_stress	29 Path_length	28 ME
ART C++	36 AR_Events	36 Gradient_weighted	36 AR_deviations
BEA	38 Gzip	33 Least_squares	33 Inertia
BEA_TSP	38 AR_deviations	38 Gradient_weighted	38 Gradient_raw
Count ones	38 Path_length	37 Neumann_stress	37 Moore_stress

To demonstrate drilling down into each individual ranking by a particular evaluation method, figure 19 shows the full ranking of seriation methods based on anti-Robinson events.

The prototype currently implements runtime measurements using a wall clock method. To get more reliable results, seriation is executed 10 times and the median with interquartile range is given on figure 20. Dots outside the boxes are outliers.

Figure 21 shows median anti-Robinson events with interquartile range for a selection of

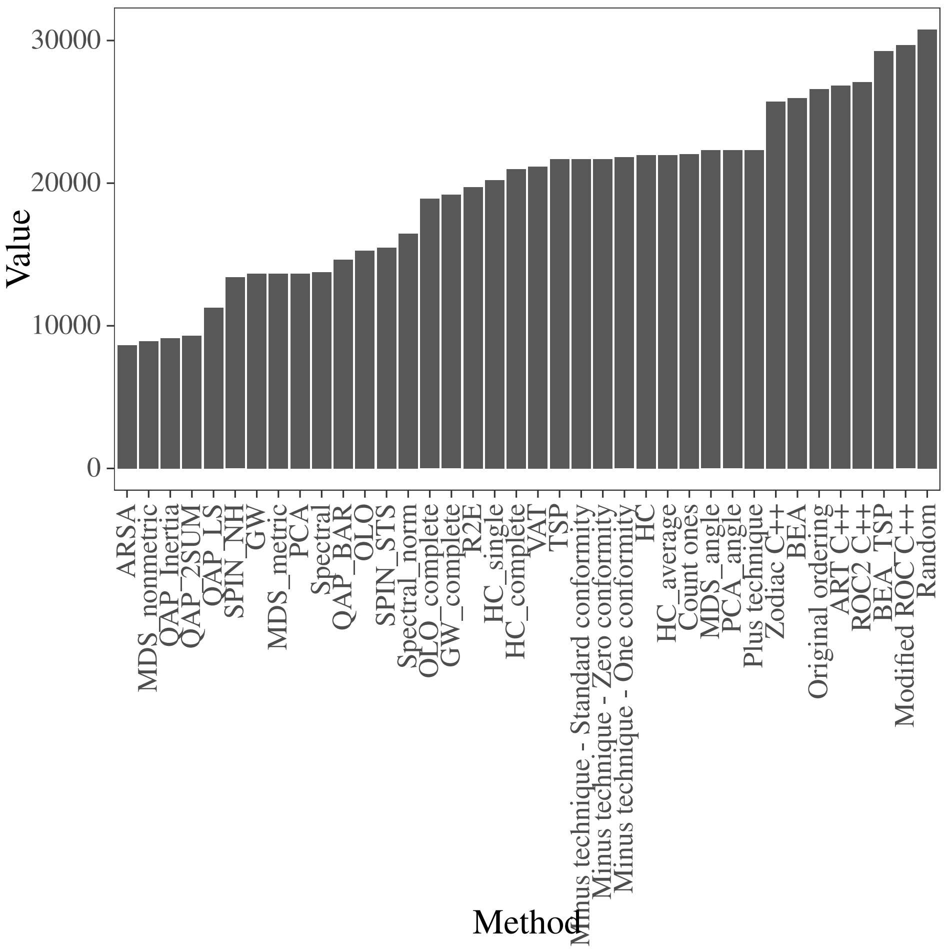

Figure 19. anti-Robinson events for seriation methods, for input Jaccard index for incidence matrix for 59 graves and 70 artifacts [48]

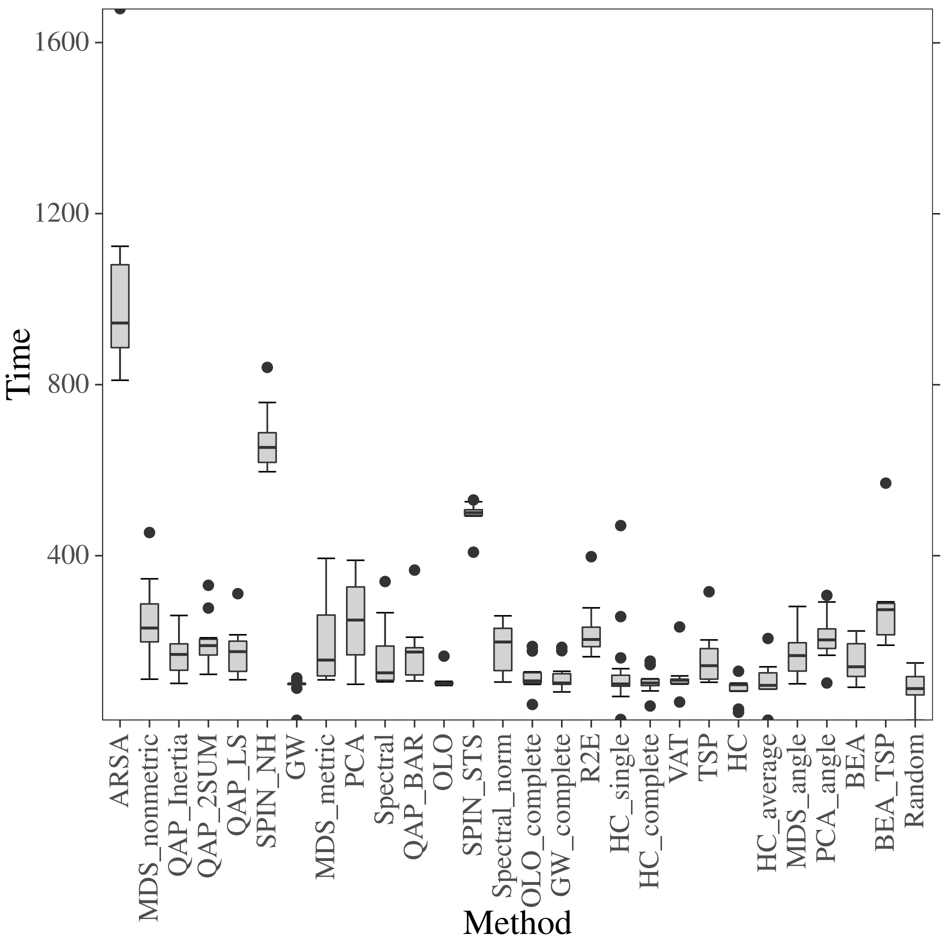

Figure 20. Runtime performance by seriation method, for input Jaccard index for incidence matrix for 59 graves and 70 artifacts [48]

inputs in the system given in table 8. Inputs are chosen from the comparison article by M. Hahsler [14].

Table 8. Inputs used for median anti robinson events plot

Input
Psych24 - Pearson correlation between results of 24 psychological tests given to 145 seventh and eighth grade students in a Chicago suburb [9]
Irish - Euclidean distances of scaled results of eight referenda for 41 Irish communities [9].
Munsingen - Jaccard index for incidence matrix for 59 graves and 70 artifacts [48].
Zoo - Euclidean distance for 17 features for 101 animals [49]
Iris - Euclidean distances (scaled) for Fisher's Iris dataset with 150 flowers and four features [9]
Wood - Euclidean distance for sample of the normalized gene expression data for six locations in the stem of Popla trees [9].

3.8 Discussion

The author went into this paper with a hypothesis, that there is an overlap between challenges in compute based experimentation and challenges in sustainable software engineering. Similarly as theoretical research is standing on the shoulders of previous theoretical work, so could executable artifacts serve as building blocks for further research.

In software engineering, libraries and other types of reusable components are used for reproducing behavior. Frameworks based on these components are used to let engineers focus on solving the business problems. Standards ensure that different components and systems can communicate and spcific implementations can change, so that systems don't break.

Containerization was found to be a viable option to be used in computer based experiments. The 6 tenets to follow in [22] are fulfilled with containers, allowing relatively easy execution and a more standard way of sharing and packaging experiments and serve as viable black box building blocks, the equivalent of a library, that is able to encapsulate a whole runtime. No study was conducted to validate this statement, rather the author considers them fulfilled by the fact, that it was possible to include multiple methods, built with different languages, running on different operating systems and recompute all the results without human involvement multiple times over. Implementation complexity of the seriation methods remains constant and is independent of an orchestrator.

The goal of proposing a benchmarking environment for seriation methods and while doing so, suggest an architecture for general purpose benchmarking was achieved by moving the domain specific concepts into containers as building blocks. This naturally left a

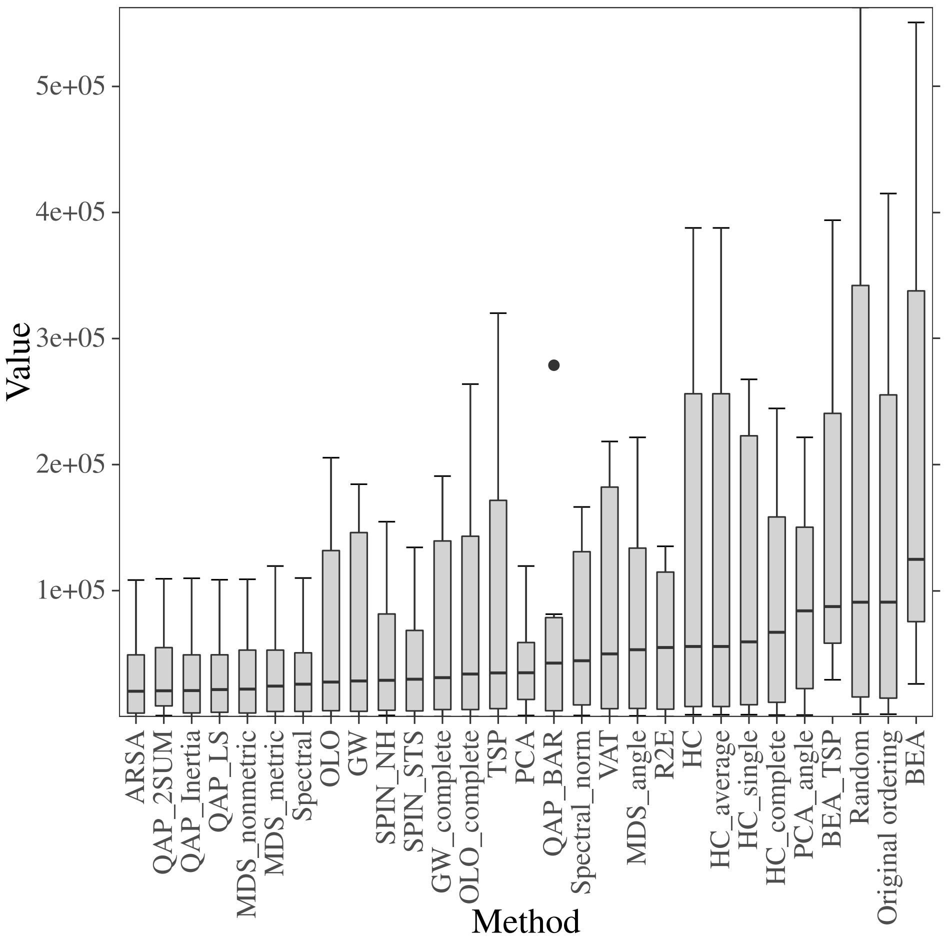

Figure 21. Median anti-Robinson events for seriation methods, for inputs in table 8

set of domain agnostic requirements, that revolve around mapping inputs to outputs and connecting these mappers in a defined way. Similarly as functions in mathematics and functional programming can be composed. Container orchestrators are ready made tools to achieve that if the units of computation are containers. The popular Kubernetes [3] was chosen after a study of existing orchestrators as it was most feature rich and programmable. It became also evident, that when treating containers as a unit of computation, a general data processing pipeline emerges, where all domain specific functionality is abstracted away into containers. In this way the architecture starts to resemble a generic data processing pipeline with sources, transformers, stores and aggregators.

The architecture and implementation choices enabled to build a prototype, that was used to do an extensive set of experiments and do preliminatry analysis of the results. Inputs, reordering and evaluation methods from previous work were containerized and inserted into the prototype. In implementation phase, the bet on commonly used and available implementations of Docker and Kubernetes paid off in a relative ease for creating the prototype. An alternative path to take would have been to use a ready made workflow engine or a data pipeline for these components. The choice of building from scratch was made due to the following factors:

- It was unknown if this would be the case a priori and comparing workflow engines would be a resource consuming task
- Curiosity and fun factor for the author, to learn what it would mean to build such an engine
- Opportunity to use familiar tools, languages and environments
- Practical reasons - funding for compute resources

As the container community is revolving around common standards provided by the Open Container Initiative [37] and the engineering world is betting on them it is probable that the approach remains valid for at least the near future and due to the principles of economy a transition path will be provided if the directions will change. It is unreasonable to expect the scientific community to be able to also cater for production quality engineering tools. This must be a collaboration both ways.

After completion of the prototype, a series of experiments based on previous studies was executed, with an attempt to replicate the results. Due to inavaiability of some of the inputs and the incompleteness of informal descriptions of the experiment setup, the results could not as expected, be exactly replicated, but side by side comparison reveals strong similarities. In summary, the results aligned with similar comparison studies done previously.

To demonstrate the benchmarking capability of the system, on table 6 it is visible that the seriation method ARSA (simulated annealing heuristic) performs best according to the evaluation method anti-Robinson events. This is expected and agrees with theory, as ARSA optimizes for anti-Robinson events. Multiple methods achieve a compareable result to the best reordering given by ARSA. Such small differences are not enough to make a choice. Other aspects of obtaining the reordeing might outweigh the small improvement in result accuracy, in a real worls setting. A multi dimensonal ranking is necessary to rank the results further. A way to use the runtime performance of the algorithm. In the prototype this was done using a wall clock method to measure the running time of the algorithm. This is shown on figure 20. MDS_nonmetric, QAP_Inertia and QAP_2SUM are good tradeoffs between quality of the result and runtime. This correlates with the finding of M. Hahsler [14].

The same seriation method ARSA is on the 31st position by Moore stress evaluation for the same input as it is visible on table 7.

Measurement of the runtime performance using a wall clock is imprecise using the prototype architecture. It is affected by every other process that that runs on the same computer at the time of the measurement. This is illustrated by the outliers visible on the same graph. The prototype is a massively shared environment as opposed to an isolated system, wich would be more suitable for measuring execution time. The compute cluster consists of virtual machines that run containers. Multiple containers are scheduled in parallel on these virtual machines.

On figure 19 is the full spectrum on seriation methods evaluated by anti-Robinson events for the same input. At end of the spectrum is random reordering. In total 5 seriation methods make the situation worse than the original ordering. This illustrates the need of knowing how the seriation methods perform and how they are suitable for a specific problem domain. The prototype has not tagged input datasets with a domain identifier, the input on these figures and tables is from the field of archeology.

It is also interesting to know if these results hold globally, for more inputs in the system, or is this just a domain specific outlier. Figure 21 shows the median anti-Robinson events for all seriation methods. ARSA is still the leader and also other methods hold their places quite well when compared to the single input figure. As it was not possible to obtain all matrixes used in the article by M. Hahsler, it is impossible to obtain exactly the same results. M. Hahsler reports the QAP to be the best reordering. The benchmarking environment places three variants of the QAP algorithm at the top, right after ARSA. On figure 21 the best seriation method is ARSA and QAP_2SUM is the close second.

4. Contributions

Contributions of this thesis can be summarized as follows:

- a proposal to approach computer based experimentation with tools familiar in software engineering
- a suggestion to solve reproducibility, replicability and reusability using containerized software for computer based experiments
- a general architecture for bulk experimentation with software alogorithms on various platforms and operating systems;
- a prototype of the architecture using Microsoft Azure [1] cloud provider, Docker [2] containers and Kubernetes [3] container orchestrator;
- a reproduction of previous research using the prototype, that resulted in a comparable outcome;
- source code with example containerized seriation algorithms and the prototype implementation;

5. Next steps

As recent years have shown, for example in deep learning space, empirical research continues to be relevant in compter science. We will surely gain more insight into algorithms, that we don't understand formally, only to find new areas, where we have a vast number of free variables to tune and experiments to run and where the inner structure is complex enough, for the research community only be able to approach it through experimentation. So investment in tools that allow more efficient work and gamification seems valuable.

The architecture and prototype provided in this thesis are a long way from a productized tool ready for adoption. Further research can not solve productization. A reasonable outcome can be development of conventions and standards. Also envangelization of the approach and further validation.

More important than a concrete implementation is to converge onto a set of standard and interoperability models, that would allow to have a certain stability from the researcher perspective and at the same time gives the freedom for implementation to change, as technology changes. This is illustrated by the Open Container Initiative, that gives a standard way to work with containers [37], with a relative certainty, that the code written today, will still run in the longer term, even if the underlying execution platforms change. Similar Open Computer Based Experimentation Initiative would bring value into the space.

This thesis did not reach a description of such standard, but it achieved initial conventions and validation that it is possible with off the shelf components.

The author sees as more important areas of research and investment as follows:

- Standardization of the experimentation worklfow engine. A benchmarking protocol.
- Standardization of the extension and integration model.
- Optimized resource usage of the system. Fair use protocols for a shared execution environment.
- Improving the runtime measurement problem. The noisy neighbor problems in containerized environments.

Bibliography

[1] Microsoft Corp. *Microsoft Azure*. " (https://azure.microsoft.com/en-us/)". 2021.

[2] *Orientation and setup*. (https://docs.docker.com/get-started/). Docker Inc., 2020.

[3] The Kubernetes Authors. *Production-Grade Container Orchestration*. " (https://kubernetes.io/)". 2021.

[4] I. Liiv. "Seriation and Matrix Reordering Methods: An Historical Overview". In: *Wiley InterScience* (2010), p. 70.

[5] Marcus Vitruvius Pollo. *Ten Books on Architecture*. Pinnacle press, 30-50BC.

[6] I. Liiv. "Seriation and Matrix Reordering Methods: An Historical Overview". In: *Wiley InterScience* (2010), p. 70.

[7] J. Bertin. *Graphics and graphic information processing*. Walter de Gruyter and Co, 1981, pp. 32–33.

[8] *The R Project for Statistical Computing*. (https://www.r-project.org/). The R Foundation, 2020.

[9] Michael Hahsler. *Infrastructure for Ordering Objects Using Seriation*. (https://www.rdocumentation.o 8). 2020.

[10] J. Bertin. *Graphics and graphic information processing*. Walter de Gruyter and Co, 1981, p. 37.

[11] Gergely Tóth and Sasan Amari-Amir. "Seriation, the method out of a chemist's mind". In: *Journal of Chemometrics* (Feb. 2018), e2995. DOI: `10.1002/cem.2995`.

[12] Paul D. Ryan Øyvind Hammer David A.T. Harper. "PAST - Paleonthological statistics software package for education and data analysis". In: *Palaeontologia Electronica* (June 2001).

[13] Christian Buchta Michael Hahsler Kurt Hornik. *Getting things in order: An introduction to the R Package seriation*. 2008.

[14] Michael Hahsler. "An experimental comparison of seriation methods for one-mode two-way data". In: *European Journal of Operational Research* 257 (2017), pp. 133–143.

[15] M. Kendall. “A New Measure of Rank Correlation”. In: *Biometrika* 30.1/2 (1938), pp. 81–93.

[16] Antoine Recanati et al. *Robust Seriation and Applications to Cancer Genomics*. 2018. arXiv: `1806.00664 [math.OC]`.

[17] *GNU Gzip*. Free Software Foundation, Inc. 2020.

[18] Michael Behrisch, Tobias Scgreck, and Hanspeter Pfister. “GUIRO: User-Guided Matrix Reordering”. In: ().

[19] Innar Liiv et al. “Visual matrix explorer for collaborative seriation”. In: *WIREs Comp Stat* 4 (2012), pp. 85–97.

[20] Wei Dai and Daniel Berleant. “Benchmarking Contemporary Deep Learning Hardware and Frameworks: A Survey of Qualitative Metrics”. eng. In: *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*. IEEE, 2019, pp. 148–155. ISBN: 9781728167374.

[21] Jörg Fehr et al. “Best practices for replicability, reproducibility and reusability of computer-based experiments exemplified by model reduction software”. In: *AIMS Mathematics* 1.Math-01-00261 (2016), p. 261.

[22] Ian P. Gent. *The Recomputation Manifesto*. 2013. arXiv: `1304.3674 [cs.GL]`.

[23] Patrick Prosser. “Exact Algorithms for Maximum Clique: A Computational Study”. In: *algorithms* 5 (2012), pp. 545–587.

[24] Inc GitHub. *Where the world builds software*. " (https://github.com/)". 2020.

[25] Inc GitHub. *Thank you for 100 million repositories*. " (https://github.blog/2018-11-08-100m-repos/)". 2020.

[26] Inc GitHub. *Starting an Open Source Project*. " (https://opensource.guide/starting-a-project/)". 2020.

[27] *Business Process Model and Notation*. Object Management Group. 2020.

[28] G. A. Miller. “The magical number seven, plus or minus two: Some limits on our capacity for processing information”. In: *Psychological Review* 63 (1956), pp. 81–97.

[29] Paul Cilliers. *Complexity and Postmodernism : Understanding Complex Systems*. eng. London: Routledge, 1998. ISBN: 0-415-15286-0.

[30] John Gall. *How systems work and especially how they fail*. eng. Quadrangle/The New York Times Book Company Inc., 1975.

[31] Robert N. M. Watson Poul-Henning Kamp. *Jails: Confining the omnipotent root*. " (https://docs.freebsd.org/44doc/papers/jail/jail-9.html)". 2021.

[32] Microsoft Corp. *AppContainer isolation.* " (https://docs.microsoft.com/en-us/windows/win32/secauthz
isolation)". 2021.

[33] linuxcontainers.org. *What's LXC?* " (hhttps://linuxcontainers.org/lxc/introduction/)". 2021.

[34] Datanyze. *Containerization market share.* " (https://www.datanyze.com/market-share/containerization–321)". 2021.

[35] *Rkt - Overview.* (https://coreos.com/rkt/). Red Hat Inc., 2020.

[36] *Introduction to Singularity.* (https://sylabs.io/guides/3.7/user-guide/introduction.html). Sylabs Inc, 2021.

[37] *Open Container Initiative.* (https://opencontainers.org/). The Linux Foundation, 2021.

[38] *Overview of Docker Compose.* (https://docs.docker.com/compose/). Docker Inc., 2020.

[39] Docker Inc. *Swarm mode overview.* " (https://docs.docker.com/engine/swarm/)". 2021.

[40] Amazon Web Services Inc. *Amazon Web Services.* " (https://aws.amazon.com/)". 2021.

[41] Shweitzer McCormick and White. "Problem Decomposition and Data Reorganization by a Clustering Technique". In: (1972).

[42] *.NET Free. Cross-platform. Open source.* (https://dotnet.microsoft.com/). .NET Core, 2020.

[43] Microsoft Corp. *Azure Kubernetes Service.* " (https://docs.microsoft.com/en-us/azure/aks/intro-kubernetes)". 2021.

[44] RabbitMQ. *RabbitMQ is the most widely deployed open source message broker.* " (https://www.rabbitmq.com/)". 2021.

[45] Microsoft Corp. *Azure Cosmos DB.* " (https://azure.microsoft.com/en-us/services/cosmos-db/)". 2021.

[46] Microsoft Corp. *Azure Blob Storage.* " (https://azure.microsoft.com/en-us/services/storage/blobs/)". 2021.

[47] The Kubernetes Authors. *Jobs.* " (https://kubernetes.io/docs/concepts/workloads/controllers/job/)". 2021.

[48] Frank Roy Hodson. *The La Tène cemetery at Münsingen-Rain.* Stämpfli, 1968.

[49] M. Lichman. *Euclidean distance for 17 features for 101 animals.* " (http://archive.ics.uci.edu/ml/index.p
2020.

Appendices

.1 Example seriation Docker files

This appendix contains examples of docker files that are Docker image specification. The goal is to illustrate what it means to create a reproducible experiment.

.1.1 Gzip evaluation method

```
FROM alpine:3.7
RUN mkdir -p /home/input
    && mkdir -p /home/work
    && mkdir -p /home/output
COPY testinput.csv /home/input/input.csv
ENTRYPOINT [
    "/bin/sh",
    "-c",
    "gzip -c /home/input/input.csv
    | wc -c > /home/output/output"]
```

Figure 22. Gzip used as evaluation method

.1.2 R based seriation

This is a more complex example, where parent Docker container is used on figure 23, to encapsulate the complexities of reading input and writing output, handling csv format and measuring execution time. An R script on figure 24 is used as template and includes a specific algorithm R script on figure 26. Figure 25 shows the Docker file that inherits from the standard R based parent and provides only the parts required by the parent. This is a demonstration of a way to standardize by convention. The result is a Docker file that takes ta seriation method name in the package Seriate as input, looks for a file called input.csv in the working directory, executes the algorithm and writes the reordering as output.csv back into the working directory. On figure 27 is a Powershell script to build and run this setup.

```
FROM rocker/r-ver:3.6.1
RUN R -e "\
 install.packages('seriation')"
RUN mkdir -p /home/input
    && mkdir -p /home/work
    && mkdir -p /home/output
COPY main.r /home/work/main.r
COPY testinput.csv /home/input/input.csv
ENTRYPOINT ["Rscript", "/home/work/main.r"]
```

Figure 23. Parent docker file for R based seriation methods

```
library(seriation)
args = commandArgs(trailingOnly=TRUE)
method = args[1]
matrixCsvFile = '/home/input/input.csv'
matrix = as.matrix(read.table(file=matrixCsvFile, sep=","))
source('/home/work/algorithm.r')
tryCatch({
    t1 = Sys.time()
    permutation = algorithm(matrix, method)
    write(
        as.numeric(difftime(Sys.time(), t1, units="secs")),
        file = "/home/output/timespan.reorder")
    if (length(permutation) == 1) {
        permutation = c(permutation, NA)
    }
    reordering = permute(matrix, permutation)
    write.table(
        reordering,
        file = "/home/output/output",
        col.names = TRUE,
        row.names = TRUE,
        sep=",")

}, error = function(err) {
    print(err)
    write(gettext(err), file = "/home/output/error")
})
```

Figure 24. Base R script that implements a template for all other R based seriations

```
FROM serikube/r-seriate
COPY algorithm.r /home/work/algorithm.r
```

Figure 25. Docker file that inherits from parent

```
algorithm <- function (x, method) {
    return (seriate(as.matrix(x), method))
}
```

Figure 26. algorithm.r - R script that implements seriation with matrix of values type of inputs

```
$imageName = "serikube/r-seriate"
docker build $PSScriptRoot --tag="$imageName"
docker build $PSScriptRoot/Matrix --tag="$imageName-matrix"
docker run --rm -v ${PSScriptRoot}:/home/output
    "$imageName-matrix" BEA
Get-Content $PSScriptRoot/output
```

Figure 27. Powershell script that illustrates building and running BEA algorithm with the above setup

.2 Lihtlitsents

Mina, Antero Lukkonen

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "BENCHMARKING ENVIRONMENT FOR SERIATION METHODS" , mille juhendaja on Innar Liiv
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

10.05.2021