DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Metrics for Job Similarity Based on Hardware Performance Data

Felix Fischer

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Metrics for Job Similarity Based on Hardware Performance Data

Metriken für Job Ähnlichkeit basierend auf Hardware-Performance Daten

Author:	Felix Fischer
Supervisor:	Prof. Dr. rer. nat. Martin Schulz (TUM)
Advisors:	M.Sc. Daniel Schürhoff (RWTH)
	Prof. Dr. rer. nat. Matthias Müller (RWTH)
Submission Date:	15.10.2020

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

London, 15.10.2020   Felix Fischer

Acknowledgments

I would like to express my gratitude to Prof. Dr. Martin Schulz and Prof. Dr. Matthias Müller for giving me the opportunity to write this thesis in cooperation with the RWTH Aachen University. Additionally, I want to thank these Professors and Alessio Netti for their constructive criticism and advice guiding me finding my path through this extensive subject area.

Special thanks go to my fellow student Severin Schüller who provided all of the benchmark runs used and tirelessly helped me review and execute code for querying data. His endless motivation and kind support is deeply appreciated.

Furthermore I want to thank Daniel Schürhoff for his optimism, organizational effort, valuable feedback and supervision during the conceptualisation as well as execution of this thesis.

Finally, I want to thank my family and friends, who supported and believed in me throughout my studies. Your friendship, encouragement and love brightens every day of my life.

Abstract

Modern High-Performance Computing systems are ever growing in scale. To better manage and get more insights into these systems, HPC institutions have turned to hardware performance data. The collection and utilization of this time series data can help in understanding what these systems and their users are doing and how to make them more efficient. Being able to measure similarity between jobs is a significant first step in the direction of increasing an HPC cluster's efficiency, as they are necessary to understand behavior of users or to detect anomalous behavior.

In this work we discuss different approaches of generating and evaluating job representations for job similarity scores on the example of existing data and infrastructure from the RWTH Aachen University. To accomplish this, we utilize and evaluate two dimensionality reduction techniques and two clustering algorithms. We propose a new measure to score clustering results with regards to job similarity, using benchmark runs. This score is then used to evaluate the different approaches.

Contents

1. Introduction

Modern High-Performance Computing (HPC) systems are ever growing in scale. With Folding@home haven broken the exascale barrier [fol20], it will not be long for dedicated supercomputers to follow suit. The current world leader, Fugaku, is leading this race with almost half an exa-Flop of double precision compute consuming around 28 MW of power [Top20]. But with scale comes complexity: Thousands of users, hundreds of thousands of jobs, potentially running on different hardware types. To efficiently allocate and use available resources is a huge research field.

To combat this ever growing complexity, HPC institutions have turned to hardware performance data. This time-series data characterizes the usage of hardware resources, like CPU, memory, and I/O. Examples include Flop/s, measured memory bandwidth, or the number of bytes read from disk. The collection and utilization of this data can help in understanding what these systems and their users are doing and how to make them more efficient. While this was realized as early as 2004 [Muc+05], most HPC centers only use this data for health monitoring purposes or manual analysis of single-job performance, and sometimes anomaly detection [Bor+19].

Being able to measure the similarity between jobs is the prerequisite for many further approaches to making HPC clusters more efficient. They are necessary for identifying shifts in general cluster usage, to understand behavior of user groups, and to detect outliers and anomalous behavior.

This thesis aims to further research in the area of performance monitoring by laying groundwork for and investigating job similarity based on hardware performance data. It will first cover the existing systems used to capture this data and its shape. Next it will scientifically approach the task of describing and reducing the dimensionality of this expansive data set, laying a solid foundation for future research. Delving in further, two dimensionality reduction and two clustering techniques are investigated. These are evaluated in respect for their use in defining job similarity with a custom scoring mechanism using benchmark runs. Lastly, this work describes in detail the applications and limitations of this approach as well as possible future work.

These goals will be achieved through usage of the existing data and infrastructure provided by the RWTH Aachen University. It currently maintains infrastructure to continually capture hardware performance metrics on their HPC cluster Cluster Aix-la-Chapelle (CLAIX) [RWT20b].

2. Background

The following sections will describe the required background knowledge for the rest of this thesis. The architecture of existing systems and their interplay is shown in sections 2.1 to 2.4. Next, this chapter will describe the contributions to the existing setup in the form of Python code used for data analysis in section 2.5.

To place these systems and further research into context, related work is described in section 2.6. Lastly, in section 2.7, we introduce a set of pre-existing benchmark runs, which will be used to evaluate the distance measures and cluster analysis in chapters 4 and 5.

To guide the sections about architecture and place this works additions into visual context, figure 2.1 illustrates the structure of all systems, as well as their dependencies.

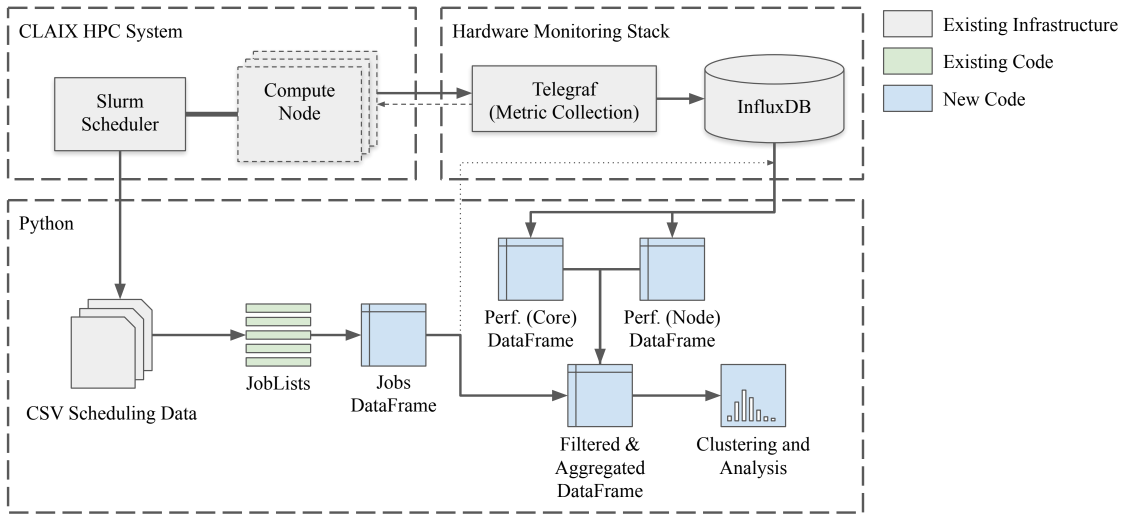


Figure 2.1.: Architecture of the Performance Data Collection Stack at the RWTH Aachen University: Data from CLAIX' scheduler Slurm and the hardware monitoring stack is parsed, filtered, and combined for data analysis and clustering purposes.

2.1. Hardware

The current computing hardware available in CLAIX is composed of two parts: CLAIX2016 and CLAIX2018 [RWT20b]. They are named after their year of installation and can be further subdivided into different node types [RWT20a]. The core data analysis in this work is restricted to CLAIX2016 nodes, even though it is likely more relevant to analyze the newest node type on CLAIX2018. This is done to avoid faulty data on CLAIX2018, further described in section 3.1. The CLAIX2016 cluster first appeared on the Top500 list in November 2016, where it placed 310th [Top17].

Within CLAIX2016, the most prevalent node type, *lnm* (600 nodes), is chosen to maximize available data with a single hardware configuration. These nodes contain two Intel Broadwell EP (E5-2650v4) processors with 12 cores each, have 128 *GB* of DDR4-2400 memory and are connected using an Intel Omni-Path network with a bandwidth of 100 Gb/s. As shown in equation A.1, a single node can reach a peak performance of 844.8 GFlop/s. All further benchmarks and data analysis, except for the benchmark comparison in section 2.6, deal with data stemming from this type of node.

2.2. Performance monitoring with Likwid

To collect all non I/O hardware performance data at the RWTH, Likwid [THW10] is used. Likwid is a portable performance monitoring tool for the GNU Linux operating system, that provides an API for reading CPU hardware performance counters. To collect hardware performance metrics in CLAIX, a C program previously developed by Jonas Hahnfeld is used.

While Likwid is usually invoked manually from the command line or using a script, this program makes use of Likwid's API directly and is remotely invoked, once a minute, by the collection server (Telegraf, see section 2.3).

Every invocation, this application measures hardware performance in the following five steps (in order of appearance), spending one second to measure each step:

1. Power per socket
2. Cache request and miss rate per core
3. Double precision Flop/s, clock rate, activity (boolean) per core
4. Single precision per core Flop/s
5. Memory bandwidth (read/write) per NUMA node

Here, a core is considered active when both the cycles per instruction and clock are not *nan*. All core specific metrics are only reported if their core is considered active.

This is taken into consideration in section 3.2, where this missing data is re-added before analysis.

While this data collection is always running per default, it is possible for users to explicitly turn off Likwid data collection (for example when wanting to profile their application themselves). In this case, the measurements are skipped to not interfere with the user. With no measurements available in later parts of the data processing, these jobs are filtered automatically.

2.3. Data collection and storage using InfluxDB and Telegraf

To collect and store the data available, a collection agent and database are needed. In the case of the RWTH, these roles are fulfilled by Telegraf [inf20a] and InfluxDB [inf20b] respectively. InfluxDB is a time series database designed to handle large amounts of sensor data and metrics. Telegraf is a server application that allows collection of metrics through a plugin architecture.

Telegraf manages the collection of data through so called collectors. These are polled exactly once every minute and the resulting data is inserted into InfluxDB. One of the collectors used is the previously mentioned C application using Likwid to read out hardware performance counters. Other scripts and publicly available plugins are used to capture more system and I/O data (e.g. bytes written to disk, packets sent over the network). A sizable collection of the available data is listed in section A.3.

Data for use in this thesis is limited in scope, as retention policies exist to delete core and node specific data after 180 and 540 days respectively. These time windows and the aforementioned recording frequency of once every minute were chosen to limit the database storage requirements to roughly 1.5 TB.

Back when this data collection system was first installed, no noticeable overhead on benchmark performance was measurable. While the measurement interval and duration may have an effect on noisiness of the data, investigating these effects are out of scope for this work.

2.4. Job scheduling data from Slurm

In order to define job similarity measures, the raw performance data must first be joined up with job scheduling data. This data is produced and provided by the Slurm Workload Manager [YJG03], the scheduling system used within CLAIX to efficiently allocate resources to users who wish to run jobs.

Slurm's scheduling data, which can be used to identify jobs with their runtimes and hosts, exists in the form of CSVs ordered by month and day. As jobs sometimes run over multiple days, causing data to be distributed among more than one CSV, these files need be to merged before further analysis. This thesis makes use of pre-existing scripts, developed by Severin Schüller, to access this scheduling data in the form of *JobLists* as seen in figure 2.1.

2.5. Working with hardware performance metrics

In order to analyze and explore hardware performance data at the University IT Center, the programming language Python is used. To not unnecessarily strain the database infrastructure with constant re-querying of the same data, a data pipeline that saved intermediate data representations was built.

Before this thesis, the data was queried and directly stored in an already aggregated fashion. It was held in memory using NumPy [VCV11] arrays and Python objects, which were then directly serialized to binary (".bin") files using of the pickle library.

During work on this thesis, the pre-existing data querying code was replaced by code utilizing pandas [tea20] dataframes. This library enables the writing of idiomatic data transformation and filtering code and interfaces well with machine learning libraries like scikit-learn [Ped+11] and Keras [Cho+15]. Using this library also made it possible to serialize data in the form of compressed parquet [Voh16] files (".parquet.gzip"). The immediate aggregation step was also removed to allow the custom aggregations described in section 4.1, based on completely raw data.

The *DataFrame* objects in figure 2.1 show all steps of this new data pipeline.

The difference in performance characteristics between these two code bases can be seen in 2.1. As each month of data only needs to be constructed once, the initial construction time is insignificant, although this can certainly still be optimized. The reduction in disk-size allowed for the storage and processing of raw node and core data, which would have otherwise consumed significantly more storage resources.

	Initial construction	Disk	Read	Write	Memory
existing	0.3h	31.2GB	378s	242s	104GB
new	2.82h	5.5GB	146s	1093s	36GB

Table 2.1.: Performance Comparison of Old and New Code, Node-Level data for 2020-07

2.6. Related work

In order to validate the chosen measurement interval and data collection methodology described in sections 2.2 and 2.3, we draw on work done to measure node-level perfor-

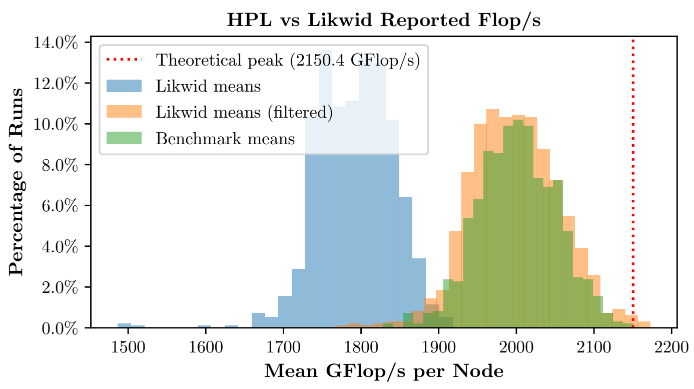


Figure 2.2.: Comparison of HPL and Likwid reported Flop/s on CLAIX2018

mance variability on the CLAIX2016 and CLAIX2018 clusters [Han19]. To continually measure performance variability across its nodes, the RWTH Aachen IT Center employs a periodically scheduled number of jobs executing High-Performance Linpack (HPL) [A P18] five times per job across roughly 1000 nodes, collecting and analyzing the resulting Flop/s measurement reported by HPL.

In figure 2.2, the distribution of these measurements for the month of July 2020 and CLAIX2018 is compared to the mean Flop/s measured and reported by Likwid for the same jobs. The theoretical peak performance of 2150.4 GFlop/s, seen in this plot, is calculated similarly to section A.1 using specifications from Intel [Inta], the RWTH Aachen IT Center [RWT20a], and WikiChip [Wik20]. For these calculations, the base AVX-512 clock of 1.4 GHz is used. Using AVX-512, an instruction set for single instruction multiple data (SIMD) vector operations, allows for higher throughput than normal operations, even at a lower guaranteed frequency. The measured Flop/s can be slightly faster than the theoretical value because of Intel's Turbo Boost Technology.

The mean of the distribution of average job Flop/s reported by Likwid initially differs greatly from the mean of the distribution of actual HPL measurements. After reviewing the reported Flop/s of a randomly picked node in these runs in figure 2.3 it can be seen that there are periods in between HPL runs with drops in reported Flop/s. Benchmark loading and verification times. As these are not factored into the mean Flop/s that HPL itself reports, they should be excluded in the Likwid measurements to accurately compare the two. After excluding all measured values below 10 GFlop/s, the now filtered distribution of Flop/s means resembles the distribution of HPL Flop/s measurements. Thus, the metric collection system described in the previous section can accurately capture the overall nature of a given job with fairly high confidence.

To embed the performance monitoring system used at the RWTH Aachen in the

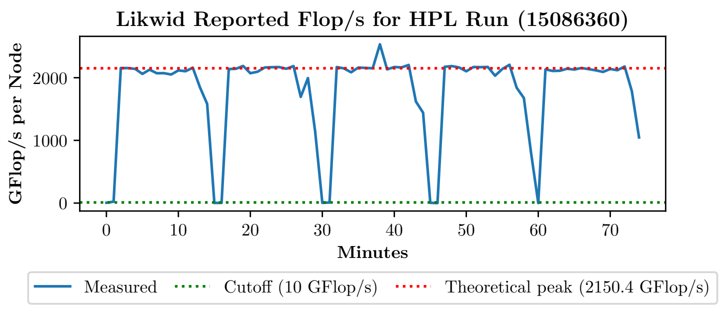


Figure 2.3.: Measured Flop/s over time for HPL Job 15086360 on CLAIX2018

broader scope of research, the metric collection stacks employed at the Erlangen Regional Computing Center (RRZE) and the Leibniz-Rechenzentrum (LRZ) in Munich are highlighted. The RRZE, maintained by the Friedrich-Alexander-Universität, similarly to CLAIX, also uses Likwid and InfluxDB [Röh+17]. They have recently developed the front-end user interface *ClusterCockpit* [Eit+19] with the main focus of detecting pathological jobs (e.g. user error in scheduling, severe load imbalances, idle jobs). The Leibniz-Rechenzentrum (LRZ) in Munich employs a completely different data collection and storage system, the *Datacenter Data Base* (DCDB) [Net+19]. *DCDB Wintermute* [Net+20], a system built on top of DCDB, was developed to enable online operational data analytics. Similar to the work completed in this thesis, the developers of *DCDB Wintermute* have applied Bayesian Gaussian mixture clustering on select performance metrics (power, temperature, and CPU idle times)[Oze+20]. This was done to identify non-optimal workload balancing across nodes and anomalous node power consumption for one node to demonstrate the capabilities of their system. Work in Munich also includes significant research into knowledge based methods for detecting performance bottlenecks and inefficiencies like load imbalance [Gui15].

While specific machine learning methods have been proposed in the context of the motivation of this thesis, to the best of our knowledge no attempt has been made at defining and evaluating general job similarity scores on a plethora of performance metrics. To give specific examples, others have predicted power consumption [YTU18] using clustering, detected known I/O patterns with supervised machine learning [Tun+17] and detected unseen general performance anomalies [Bor+19] using unsupervised learning with autoencoders. During the work on this thesis, an attempt to cluster HPC jobs, similar to the methodology of this thesis, has been published [HDF20]. This other work differs from this thesis with regards to the the methodology used for evaluation: Halawa et al. evaluate their clustering results using cohesion and separation measures. In contrast, this thesis first defines desirable characteristics of a

Keyword	SPEC MPIM2007	SPEC MPIL2007
GemsFDTD	113.GemsFDTD	145.lGemsFDTD
TF	129.tera_tf	129.tera_tf
leslie	107.leslie3d	143.dleslie
lu	137.lu	137.lu
su3imp	104.milc	142.dmilc
tachyon	122.tachyon	122.tachyon
zeusmp	132.zeusmp2	132.zeusmp2
wrf2	127.wrf2	147.l2wrf2
idle	n.a.	n.a

Table 2.2.: Used SPEC MPI® 2007 Benchmarks and Naming

good similarity score and then using these to construct a scoring method for clustering results across chapters 4 and 5.

2.7. Benchmarks

To evaluate the consistency of job similarity scores and different unsupervised clustering method in later sections, we make use of pre-existing labeled benchmark runs. Specifically this work makes use of the existing runs listed in table 2.2. These benchmarks are all part of the SPEC MPI® 2007 benchmark suite, which consists of two separate classes: MPIM2007 and MPIL2007 - containing a total of 18 benchmarks [SPE07].

The runs used in this work do not include all benchmarks in the benchmark suite. Each row in the aforementioned table matches a benchmark from the medium or large category with its respective counterpart. All MPIM2007 and MPIL2007 jobs were run on a single and four nodes respectively. The number of nodes a benchmark was run on, as well as the keyword column in table 2.2 will from here on be used to identify a specific benchmark in the form of the labels: *1_<Keyword>* and *4_<Keyword>*. All individually identifiable benchmarks were run 5 times.

Additionally to these benchmarks, ten minute idle runs on one and four nodes will be used. These will be referred to as *1_Idle* and *2_Idle*.

3. Data preparation

After previously describing the setting of this work, the following section will start working with the collected data. Before further analyzing existing hardware performance data, filtering, cleaning, and feature engineering is required. First, the raw data needs to be filtered down to a consistent yet maximal set. Next, any missing data needs to be dealt with. Lastly, new columns are derived from the existing ones to better capture the properties of a single job.

3.1. Filtering

As the recorded double precision $Flop/s$ data on CLAIX2018 sometimes contains faulty values, CLAIX2018 is excluded from any further analysis. These erroneous measurements first started appearing on the 16.05.2019, when Intel used the Flop hardware performance counter as a spare register to patch a bug in microcode relating to transactional memory (TM). As a result, this counter does not contain valid data when AVX-512 is used in conjunction with TM. Efforts have since been made to continually disable TM on CLAIX18. This has not been completely successful, as it turns back on when an application uses it and there exists no BIOS setting to force this feature to stay disabled.

In order to guarantee homogeneous measurement data while ensuring maximal data, we further restrict ourselves to a single hardware type in CLAIX2016. The main node type: lnm (600 nodes), described in section 2.1, is chosen.

Because of the retention policies mentioned in section 2.3, per core data was only available starting the middle of March 2020. With the most recent query being completed in the middle of September 2020, the data used for this thesis spans the time frame from the 15.03.2020 until the 15.09.2020. While it would have been preferable to include cache request and miss rates, these are not included in further analysis, as these measurements have only been active since 23.06.2020. Including them implies removing half of the available data. This amount of data was valued higher than the additional information that the cache metrics provide.

3.2. Missing data

As mentioned in section 2.2, per core data is only recorded for active cores. In order to correctly calculate aggregations, the values for inactive cores are padded with zeroes.

3.3. Derived columns

All I/O and InfiniBand metrics listed in the appended table A.1 can be used to extract further information about performance behavior. The transmit, receive, read, or write speed and amount of packets, reads, or writes can be combined to calculate average packet/read/write size. For example, average disk write size is computed taking the ratio of the *disk_write_bytes* with the *disk_writes* column.

To correctly capture the statistical properties of these derived columns, it is necessary to compute them before further aggregation in section 4.1. This is the case as the mean of a ratios $\mu_{(a/b)}$ does not necessarily equal the ratio of means μ_a/μ_b. The same holds true for the standard deviation of ratios $\sigma_{(a/b)}$, which is not equal to the ratio of standard deviations σ_a/σ_b. This is further illustrated with an example in section A.2.

4. Dimensionality reduction

After all of the data preparation steps, the data set consists of the following dimensions:

- jobs (59714 total)
- nodes (ranging from 1 to a max of 256)
- cores (24 cores per node)
- time (varying from a few minutes to a max of 10 days, one data point per minute)
- features (27 per node + 3 per core)

The time-series data for a given job is highly dimensional with varying shape and length. Defining a distance measure using this data directly is nearly impossible, as two jobs can have different runtimes and amounts of nodes.

In section 4.1 this data set is reduced down to a single row of data per job. An effort is made to preserve statistical properties of the different dimensions. Given the vast variety of distributions across aggregated features, in section 4.2 two scaling approaches are introduced. These aim to scale features, such that no one feature dominates the others. Afterwards, in sections 4.3 and 4.4 the number of features is further reduced through the use of PCA and autoencoders. This is done to both address the curse of dimensionality and to keep clustering runtimes low. Lastly, in section 4.5 the previously defined distance measures are interpreted and analyzed.

4.1. Aggregation

To transform the data for a given job to a consistent shape and length, statistical aggregations over the different dimensions are completed. The mean and standard deviation (std) are applied to the *time*, *cores*, and *nodes* dimensions in the given order.

Table 4.1 shows a general overview of the aggregations applied. Here *Core-Level-Metrics* and *Node-Level-Metrics* refers to the different metrics listed in A.1. The derived *Node-Level-Metrics* from section 3.3 are also included here. The different aggregations in the table are chosen to reflect metric properties that differentiate jobs from one another. If they are not extracted here, this information would be lost and not accessible to further dimensionality reduction and clustering. The standard deviation of the *time*, *core*, and *node* dimensions are taken to represent what can be interpreted as imbalance. This information would be lost if data was only aggregated using the mean.

While other methods of aggregating time-series data are proposed in section 6.2, this work focuses only on extracting general statistical properties through aggregation. A more detailed analysis of possible statistical aggregations (e.g. the mean core imbalance of time imbalances) or the detection and use of different code sections (e.g. data loading or check-pointing) for aggregation is out of scope for this work.

Time	Cores	Node	Meaning
Core-Level Metrics			
mean	mean	mean	global mean
mean	mean	std	node imbalance
mean	std	mean	mean core imbalance
std	mean	mean	mean time imbalance
Node-Level Metrics			
mean	-	mean	global mean
mean	-	std	node imbalance
std	-	mean	mean time imbalance

Table 4.1.: Aggregations and their Representations

Through the process of aggregation some duplicate (when scaled) columns are produced. This is the case because some data (e.g. DP-Flop/s) is present in both the per-core and per-node data. These duplicates are simply removed.

At the end of aggregation, every job has the same shape of only 108 feature columns.

4.2. Standardization and Normalization

Next it is necessary to scale the aggregated data. The features produced through the aggregation steps described in the previous section all have widely differing distributions. The aim of standardization and normalization is to transform features such that no single feature dominates others in further approaches and evaluations.

Standardization transforms the data of a feature such that its distribution has a mean of zero and standard deviation of one.

Normalization scales data linearly so that the minimum and maximum data points map to zero and one, respectively.

Both can be useful, depending on the data at hands and methods used. To empirically test which method is better suited for the problems in this thesis, both are used.

4.3. Principal component analysis

To achieve reasonable runtimes in later clustering approaches and to avoid the curse of dimensionality, the 108 features per job are further reduced. This is done using PCA, described in this section and autoencoders, described in section 4.4.

Principal component analysis (PCA) is a widely used dimensionality reduction technique. It estimates the original data through a linear transformation into the basis of so-called principal components. These components are calculated, sorted, to maximize the captured variance in data. This approach is often used in cluster analysis with the assumption that components containing the most variance, capture the most informative features of a given data set. For this reason, and its simplicity and availability, this method is applied in this thesis.

4.4. Autoencoders

Autoencoders [Kra91; Cho15] are a type of neural network, which aim to learn efficient data encodings through unsupervised learning.

The typical architecture of an autoencoders, displayed in figure 4.1, and used throughout this thesis is very simple. They are fully connected, feed-forward neural networks consisting of an encoder and decoder section. The encoding part is made up of an input layer with the dimensionality of the input data (108 features) and several (optional) hidden layers to learn and encode correlations and dependencies of different features. Additionally, it contains the bottleneck layer, which is also shared as an input to the decoder. This has the desired reduced dimensions. The decoder further contains the reverse set up of the encoding stack, finally outputting data in the original shape.

To perform dimensionality reduction, a given autoencoder is trained using the mean squared error (MSE) as a loss function. After successfully training the network to reproduce a given input on the output side, the bottleneck layer can be used as a compressed representation of the original data.

ReLU is used as an activation function throughout the network, as it has been shown to be very powerful and is the current state of the art. When data is scaled using normalization, the last layer of the autoencoder has to use a different activation function to account for this. Because during normalization data is scaled to values between zero and one, the *Sigmoid* function is used, as it produces the same range of values.

In order to achieve a sparser encoding of data, it is also possible to add a regularizer constraint to the bottleneck layer. This can be desirable to avoid overfitting and learning a model that is more complex than required. The regularization used here tries to

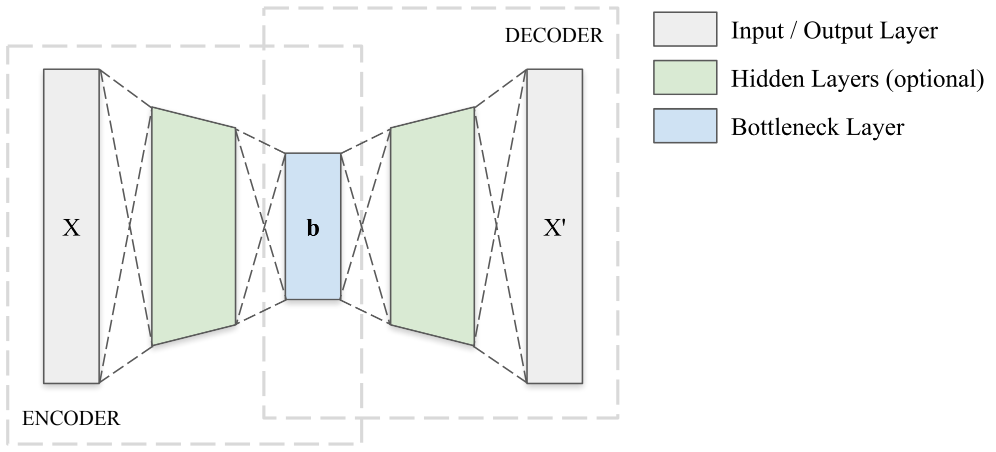


Figure 4.1.: Network Architecture of a Typical Autoencoder

achieve a more sparse encoding by constructing the loss function to additionally minimize the L1 norm of the bottleneck layer.

In the next sections different network depths of hidden layers will be called *simple*, *shallow*, and *deep*. These names refer to the configuration and size of the extra hidden layers and the configurations can be seen in table 4.2.

Name	Shape
Simple	[]
Shallow	[64]
Deep	[256, 128, 64, 32, 16]

Table 4.2.: Extra Hidden Layer Sizes

4.5. Evaluation using MSE

The number of components to reduce the original data down to is determined by using the PCA rule of thumb of selecting as many components as needed for retaining 90% of the variance. This threshold is reached at 16 components. This number was then taken and applied across all other methods.

To evaluate how well different techniques capture the original data, the loss of information is compared. To correctly estimate future performance, 20% of the shuffled data is removed before training autoencoders and fitting PCA. These 20% are then used exclusively for evaluation in this section. Alternatively the same could be accomplished using cross validation. This was omitted because of time limitations.

The error for a given encoding is calculated using the MSE. This is achieved by first decoding back to the original 108 dimensions. To not favor one feature over another based on distribution, the original and reconstructed data are both transformed using standardization (fitted on the original data). Standardization is chosen over normalization to avoid the under representation of features with very large outliers. Lastly, the MSE is calculated using these scaled representations.

Dim. Red.	Scaler	Reg.	Depth	MSE
autoencoder	norm	none	deep	0.173488
			shallow	0.21619
			simple	0.314946
		applied	deep	0.259047
			shallow	0.249772
			simple	0.393544
	std	none	deep	0.339015
			shallow	0.299305
			simple	0.300781
		applied	deep	0.388993
			shallow	0.374349
			simple	0.328651
pca	norm	n.a.		0.608582
	std	n.a.		0.570563

Table 4.3.: Mean Squared Error of Dimensionality Reduction Techniques

Table 4.3 shows the mean squared error for different dimensionality reduction configurations. Note, that all autoencoding configurations were trained five times and only the model with the least MSE is selected and used. The mean squared error rate was overall lowest for the autoencoder trained on normalized data with no regularization of the bottleneck layer and a deep network architecture. A potentially non-expected result is that autoencoders minimizing MSE on normalized data perform better than autoencoders minimizing MSE on standardized data. This is surprising as the worse performing network is minimizing the exact metric they are being compared with. This cannot be attributed to overfitting, as the models are evaluated on data different from the data they are trained on. Overall it can be said that simple networks achieve a lower MSE on standardized data, while deep networks synergize more with normalized data. Autoencoders with regularization turned on perform generally worse than their counterparts with it turned off. PCA performed worse than all autoencoders.

4.6. Evaluation using pairwise benchmark distances

While a model with low MSE represents a high quality encoding of the original data, it does not necessarily mean that said model is able to capture properties desirable in a distance measure of jobs. To further evaluate these dimensionality reduction techniques, specifically with regards to their value as distance measures, we fall back to the benchmarks introduced in section 2.7. To illustrate job similarity properties of the techniques, heatmaps of the euclidean distances between benchmarks are used. These allow for quick visual identification of key features desirable in a job similarity score.

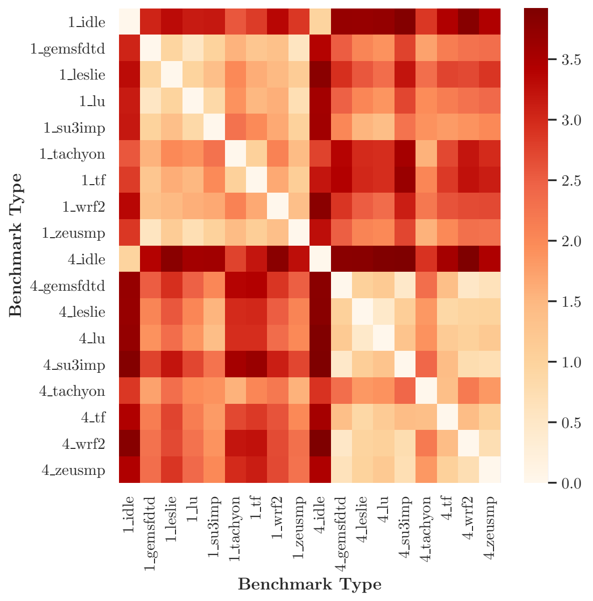
Figures 4.2 and 4.3 show the pairwise euclidean distances of benchmark runs and mean benchmark runs using the best and worst performing methods from section 4.5. These two models consist of an autoencoder (using normalized data, no regularization, and a deep network architecture) and PCA (using standardized data). They are selected to contrast the models' MSE measures with their applicability to job similarity. Heatmaps for other dimensionality reduction techniques are appended in section A.4. For every benchmark type, the mean of benchmark runs is calculated using the arithmetic mean. Note that for both the *1_idle* and *4_idle* jobs only one run is available.

One very intuitive characteristic of a good job similarity score is that an idle job should be very different from high performance benchmarks. This is definitely the case for the two dimensionality reduction techniques displayed. The *1_idle* and *4_idle* jobs have, on average, the highest distances to all other jobs. On all heat maps, they are visually distinguishable as dark red lines.

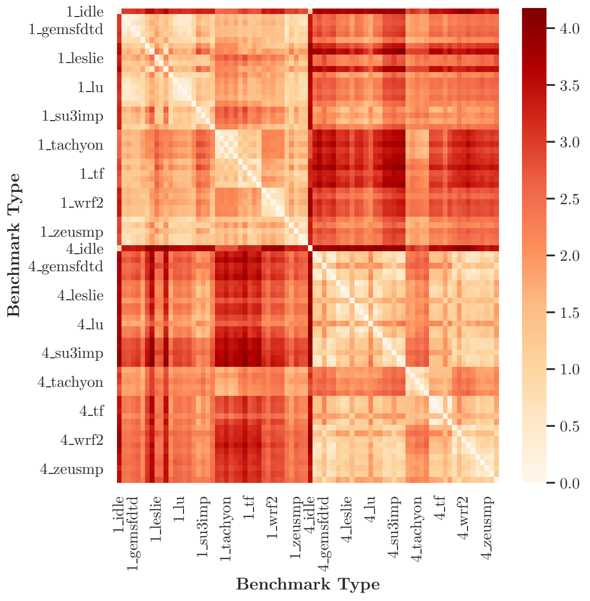
Another property that should definitely be present, is the similarity of benchmark runs of the same benchmark type. As these all had the exact same hardware, code, and environments the distance between any two runs of the same type should ideally be zero. Optimally, the distance heat map of individual runs will look exactly the same (except for the two idle runs) as the heat map for the run means. However, this is not the case and a distinct level of noise between individual runs can be observed. In general the autoencoder captured more of this noise than PCA did.

This noise is somewhat expected and could be caused by a multitude of reasons: Firstly the operating system as well as the hardware can introduce variability in performance [Han19]. Additionally, this noise could be due to the sampling nature of the data collection systems described in sections 2.2 and 2.3. With a sampling interval of only one second and a job changing job phases many times a second, some inaccuracy and noise is to be expected.

Additionally, it might be desirable for a distance measure to show similarity between single and multi node runs of the same benchmark type. If these distance measure were able to capture this, it would show in the diagonals of the lower left and upper right quarter. These should be lighter than other pairwise distances between single



(a) Pairwise Distances of Mean Benchmark Runs



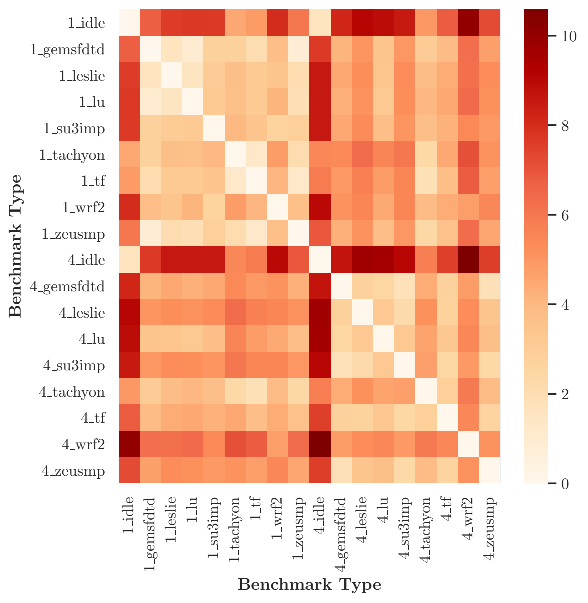
(b) Pairwise Distances of Individual Benchmark Runs

Figure 4.2.: Pairwise Benchmark Distances for Best Performing Dimensionality Reduction Technique (Autoencoder: normalized, deep, not regularized)

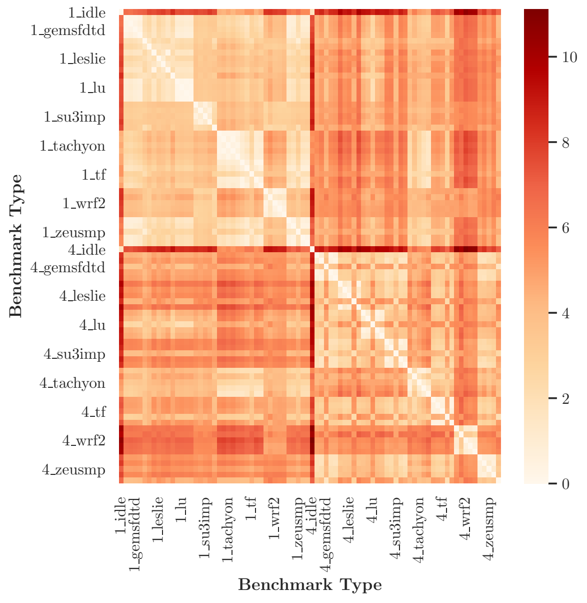
and multi node jobs. This cannot be observed for both of the dimensionality reduction techniques, likely because of the large effect that cross node MPI communication and synchronization has on the measured metrics.

While the cross node communication might obscure similarities to single node workloads, it definitely helps to distinguish single and multi node benchmarks in general. Both methods show an increase in distance measured between these two categories of jobs. This effect is visualized by a significantly darker lower left and upper right quarter in all heat maps.

Overall this visual analysis shows that measures like MSE, or the lossiness of dimensionality reduction techniques, does not map well to desirable features of job similarity measures. Thus, it is necessary to develop a new job similarity score to compare different methods. One possible measure to gauge the quality of job similarity scores through clustering is constructed and introduced in chapter 5.



(a) Distances of Mean Benchmark Runs



(b) Distances of Individual Benchmark Runs

Figure 4.3.: Pairwise Benchmark Distances for Worst Performing Dimensionality Reduction Technique (PCA: normalized)

5. Cluster Analysis

With the result of the dimensionality reduction in chapter 4, jobs can now be clustered.

This chapter's main aim is to objectively evaluate the multitude of approaches that can be taken to transform and cluster the available data. First, two clustering algorithms and the motivations for choosing them are described in sections 5.1 and 5.2. These are then applied to all distance measures described in chapter 4. Lastly, an approach for scoring clustering results using benchmark runs from 2.7 is introduced. This is consequently applied to all combinations of dimensionality reduction and clustering algorithms to identify the best overall approach.

5.1. K-Means

K-Means [Mac67; Llo82] was selected as a clustering algorithm, as it is very well established and easy to understand. This makes it a prime candidate for setting a baseline for comparisons with other benchmarks in the future. K-Means is an iterative algorithm. Each iteration consists of two steps:

1. All data points are assigned to the closest cluster center.
2. The cluster centers are recalculated using the new assignments.

Initially, cluster centers are randomly initialized.

5.2. Bayesian gaussian mixture model

A gaussian mixture model (GMM)[MB88] on the other hand, is a model used to statistically represent sub-groupings of data through a mixture of Gaussian distributions. These models are usually fitted to existing data using iterative expectation maximization (EM). This algorithm fits parameters of distributions so that the mix of all most likely to produces the data at hand.

One variant of the greater family of GMMs is the Bayesian gaussian mixture model (BGMM) [Ped+11] used in this work. This model estimates an upper bound on the number of components in a mixture. While the documentation states that the "ideal" number of components is usually not well defined during data exploration, the use of this algorithm guarantees that the number of components used is not forcibly bound to the number of components specified, but can decrease to lower numbers.

These models were chosen, because of their advantage in statistically representing data. They can represent clusters of wildly different, overlapping shapes, which is not possible with K-Means. Additionally, their applicability to HPC data has successfully been demonstrated by Ozer et al. [Oze+20], who used these models to detect performance anomalies of hardware nodes.

5.3. HD-Scoring

Having introduced different clustering algorithms, the results they are able to produce, in conjunction with the dimensionality reduction techniques described in chapter 4, need to be objectively evaluated.

While metrics like the silhouette score [Rou87] are able to assign a score to a cluster result, these scores have no connection or relevance to job similarity, which is the preferred metric in this work. For this reason, more desirable and directly quantifiable clustering qualities are defined using the known benchmark runs introduced in section 2.7.

One desirable quality of a clustering result with respect to capturing job similarity is the homogeneous grouping of known similar jobs. Secondly, known different jobs should ideally be placed in diverse groupings. In this section we first define measures to capture these properties mathematically using the Simpson index [Sim49] and then propose an overall measure that combines these properties. In section 5.4 this new measure will then be used to empirically evaluate clustering results.

5.3.1. Homogeneity

To evaluate homogeneity of a given clustering each type of SPEC benchmark, introduced in section 2.7, will serve as a test grouping. All runs of a single benchmark should be assigned the same label.

The Simpson index λ, defined in equation 5.1, when applied to the clustering result of a specific benchmark, represents the probability that any two randomly selected runs (with replacement) receive the same label. This measure has the advantage of being a intuitively understandable quantity when applied to classification problems. Here, λ_b is the Simpson index of a given benchmark b, n represents the number of unique labels, and p_{ib} is defined as the probability of the label i being assigned to a benchmark b. Optimally, all runs for a given benchmark receive the same label and λ_b is maximal and equal to 1. In the worst case, all runs receive a different label and λ_b is minimal and equal to $(1/r)^2$ with r being the number of runs (assuming $r \leq n$).

$$\lambda_b = \sum_{i=1}^{n} (p_{ib})^2 \tag{5.1}$$

$$\tag{5.2}$$

To aggregate these scores across all benchmarks, we introduce a new measure α in equation 5.3. It is defined as the mean of all Simpson indices λ_b, where m represents the number of benchmark types. Given the same amount of runs r for each benchmark, the minimum and maximum of this aggregate is the same as the minimum and maximum of the Simpson index itself. It represents the probability of choosing, with replacement, two runs with the same labels, given a random benchmark type.

$$\alpha := \frac{1}{m} \sum_{b=1}^{m} \lambda_b = \frac{1}{m} \sum_{b=1}^{m} \sum_{i=1}^{n} (p_{ib})^2 \tag{5.3}$$

5.3.2. Diversity

To measure the diversity of a clustering we use the probability of any two runs, with replacement and regardless of benchmark type, receiving a different label. This value β can be calculated as seen in equation 5.4. It is one minus the Simpson score for all labels λ_{all}, not segregating by benchmark type.

$$\beta := 1 - \lambda_{all} = 1 - \sum_{i=1}^{n} (p_i)^2 \tag{5.4}$$

5.3.3. Combined HD-Score

To combine these two measures, we can consider the probability of the combination of the events described in sections 5.3.1 and 5.3.2. As both of these events are independent from each other, the combined probability is simply the product of the individual probabilities. This combined probability can be calculated as described in equation 5.5. As the events described by this probability are desirable, a score closer to one is ideal and a score close to zero is poor.

This score could further be weighted by calculating the probability that one of the above mentioned events happens more than once. However, this works restricts itself to the non-weighed probabilities to not further complicate the evaluation process.

$$\gamma = \alpha\beta \tag{5.5}$$

This combined probability score is further used to evaluate the combinations of different dimensionality reduction techniques and clustering algorithms.

5.4. Evaluation

Having defined a scoring function that measures desired qualities related to job similarity in the previous section 5.3, these scores are now applied to clustering results.

As both clustering algorithms described in sections 5.1 and 5.2 are dependent on random initial conditions, they are run five times for each set of hyper-parameters tested and the best scoring run is chosen.

The number of clusters and components range from two to nine. Because of runtime constraints, larger numbers are not tested here.

First, we will asses which combinations of dimensionality reduction methods and clustering algorithms, scored in table 5.1, perform the best. It is immediately obvious, that, on average, BGMM performs much better than K-Means. This, of course, comes at some cost: BGMMs implement a much more complex algorithm and take much longer to fit. Looking at the results of BGMMs, an inverse correlation to the MSE performance of the dimensionality reduction techniques can be observed. Techniques with a worse MSE perform better than those with a better MSE. This demonstrates that measures like MSE do not map well to desirable features of job similarity measures, as previously suggested in section 4.6. Regularization in autoencoders seemed to slightly boost scores of BGMMs. This is likely because regularization prevented some overfitting causing the model to generalize better. Overall, PCA with standardization clearly out-performs all other methods. This is surprising as it is the simplest method tested. It shows that success in machine learning is not necessarily tied to the complexity of the model and simple approaches should always be tried before moving on to more complex solutions.

Next, the effect of the number of clusters on scores in table 5.1 is investigated. On average, score increases with an increasing amount of clusters. Homogeneity and diversity as separate scores are appended in section A.5. When looking at these, it becomes clear that the correlation of score and the number of clusters stems from the diversity measure: While homogeneity only slightly decreases with the number of clusters tested here, diversity increases. This effect seems to plateau for numbers of clusters close to nine, suggesting that a higher number of clusters does not help dividing the data into more sections.

Lastly, we take a closer look specifically at homogeneity for different amounts of clusters. Homogeneity does not suffer significantly with an increasing amount of clusters. Instead, it stays fairly constant at its initial divergence from the optimum. This initial divergence is likely caused by the run-to-run variability of benchmarks, which has a non insignificant effect on the aforementioned distance measures. This noise was previously noted in section 4.6. To further improve the results of methods applied in this thesis, investigation and reduction of this noise is very likely necessary.

To summarize the contents of this chapter and evaluate them on a grander scale, we first defined desirable properties that clustering results of jobs in HPC should have. These were then used to introduce a scoring method for clustering results that can be used to evaluate combinations of dimensionality reduction techniques and clustering algorithms. This methodology was then successfully applied to identify the most promising approach: PCA with standardization as scaler, 8 clusters, and a BGMM was able to achieve an HD-Score of 0.7. As can be seen in the tables in section A.5, the result of this approach guarantees that, on average, two benchmark runs of the same type receive the same label approximately 95 percent of the time. This seems reasonable, given the previously identified noise in job representations. With future work reducing this noise, this score can likely be improved. Additionally, any two benchmark runs have a 74 percent probability of being differently labeled. These scores show that this dimensionality reduction technique, PCA, was able to capture underlying properties of jobs that are essential to segregating jobs from one another in a meaningful way. The clustering technique of BGMMs was identified to be well suited to this task.

Dim R.	Scaler	Reg.	Depth	Number of Clusters							
				2	3	4	5	6	7	8	9
K-Means											
autoenc	norm	none	deep	0.37	0.55	0.48	0.49	0.55	0.55	0.54	0.54
			shallow	0.00	0.49	0.49	0.51	0.49	0.49	0.49	0.53
			simple	0.00	0.00	0.04	0.04	0.44	0.44	0.58	0.59
		applied	deep	0.04	0.51	0.54	0.55	0.34	0.34	0.24	0.51
			shallow	0.04	0.28	0.35	0.39	0.55	0.55	0.58	0.56
			simple	0.00	0.04	0.51	0.53	0.48	0.47	0.58	0.58
	std	none	deep	0.10	0.10	0.21	0.40	0.42	0.44	0.48	0.51
			shallow	0.08	0.10	0.30	0.54	0.53	0.50	0.53	0.53
			simple	0.00	0.10	0.12	0.47	0.52	0.53	0.53	0.53
		applied	deep	0.00	0.10	0.31	0.42	0.42	0.42	0.42	0.48
			shallow	0.06	0.12	0.14	0.28	0.40	0.39	0.39	0.38
			simple	0.12	0.27	0.29	0.35	0.35	0.42	0.41	0.32
pca	norm	n.a.		0.04	0.24	0.42	0.38	0.54	0.54	0.54	0.53
	std	n.a.		0.45	0.44	0.49	0.48	0.52	0.51	0.61	0.52
Bayesian Gaussian Mixture Model											
autoenc	norm	none	deep	0.00	0.06	0.00	0.08	0.49	0.48	0.48	0.57
			shallow	0.34	0.36	0.02	0.49	0.55	0.58	0.50	0.50
			simple	0.28	0.47	0.51	0.51	0.53	0.52	0.52	0.53
		applied	deep	0.28	0.02	0.43	0.48	0.54	0.54	0.53	0.59
			shallow	0.30	0.32	0.53	0.42	0.48	0.54	0.56	0.42
			simple	0.33	0.35	0.58	0.56	0.58	0.60	0.63	0.63
	std	none	deep	0.30	0.48	0.46	0.51	0.49	0.52	0.55	0.45
			shallow	0.40	0.45	0.45	0.59	0.59	0.54	0.64	0.66
			simple	0.41	0.50	0.47	0.53	0.54	0.58	0.58	0.62
		applied	deep	0.10	0.46	0.41	0.47	0.50	0.48	0.50	0.56
			shallow	0.33	0.43	0.41	0.65	0.65	0.63	0.65	0.69
			simple	0.44	0.49	0.67	0.55	0.54	0.62	0.65	0.60
pca	norm	n.a.		0.49	0.52	0.55	0.53	0.58	0.58	0.60	0.61
	std	n.a.		0.49	0.64	0.66	0.67	0.65	0.70	0.70	0.68

Table 5.1.: HD-Scores of Dimensionality Reduction Techniques and Clustering Algorithms

6. Conclusion

This thesis aimed to lay groundwork for and to investigate job similarity based on hardware performance data.

To achieve this, chapter 2 covered important background information, both on the existing systems and data collected at the RWTH Aachen University used in this thesis. To validate the data available, it is compared to data from previous work measuring hardware performance through HPL benchmarks. Further, benchmarks were introduced, which later helped with the definition of desirable properties of job representations with which to define job similarity scores.

Chapter 3 dealt with necessary preparation of raw data. In two cleaning steps, this data was then filtered and missing values of inactive core-data was filled in. Afterwards, further information about performance behavior was derived from existing features and added to the data set. This was necessary, as these features would have gotten lost in further aggregations. At the end of this pre-processing step, data was clean, but still highly dimensional and irregular in shape and length: Data for a two jobs could span across different amounts nodes for different periods of time. Some data existed for every core on a given node, some data was present only on the node-level.

A first approach to reduce this dimensionality to a more manageable level was taken in chapter 4. Here, the irregularities in shape and length were removed through aggregation of the different dimensions of the data with statistical measures. Once the data shape was consistent, data was first scaled, as to not bias for or against certain features in further analysis. Then two dimensionality reduction techniques, principal component analysis and autoencoders were introduced and applied to reduce dimensionality further. These approaches were then evaluated, both using the mean squared error and visually using heatmaps of pairwise distances between the aforementioned benchmarks. While the former captures goodness of fit for data encodings, when compared to the latter, it was found to not translate well to properties one might seek for job similarity measures.

Chapter 5 added cluster analysis to the set of tools that can be used for comparing different job representations with regards to how well they represent job similarity. After describing two algorithms that can be used to cluster the data at hand, a novel approach at scoring different techniques using benchmark runs was introduced. This combined both of the desirable properties of homogeneity within benchmark runs of the same type and diversity between runs overall to derive a combined metric, the so called *HD-Score*. Through the use of this score, it was subsequently possible to analyze different combinations of dimensionality reduction techniques and clustering algorithms. Lastly, principal component analysis in combination with standardization as a scaling method and Bayesian Gaussian mixture models as the clustering algorithm was identified to best capture job similarity of the underlying data.

6.1. Engineering limitations

This section deals with the limitations imposed by the existing systems and data.

Any data analysis can only be as good as the data that is available. The data used in this thesis is limited in scope due to several factors described in chapter 3: The sampling rate and measurement intervals were given and chosen with performance overhead and disk space in mind. The existing settings could likely be tuned to output data at a higher frequency with larger measurement intervals, as the existing settings were found to not introduce a measurable overhead at the time of installment. Further, data concerning cache hit and request rates was excluded in this work. This was done, as data from the first half of 2020 did not yet have the collection of these metrics enabled. With time, enough data with cache metrics will amass to include these in future analysis. Additionally, data from CLAIX2018 was discounted because of faulty Flop/s measurements. With solid mitigations, it might be possible to use this data in the future. Alternatively, it might be possible to detect jobs for which data is faulty using statistical methods. These could then be excluded these from the data set. This would open up analysis on newer hardware, making results of future work directly applicable to cutting edge systems. Lastly, more hardware performance counters and other metrics could be added to or derived from existing features. Specifically more low-level data like instructions per cycle per core or memory bandwidth usage per NUMA node could help better capture properties of HPC jobs.

Another limitation, the reason for this work's problem statement itself is the non-existence of labels that constitute a "good" job similarity score. While similar jobs could theoretically be labeled manually, this process is very costly. This structural challenge was tackled through the use of existing and labeled benchmark runs. With these, it was possible to describe desirable properties of job similarity and ultimately define a scoring system for different methodologies in section 5.5.

6.2. Future work

This thesis showed novel and promising approaches to job similarity scores in HPC. Still many further research avenues can be explored. In this section we aim to give an overview of possible extensions and uses of the results of this work. Additionally, we propose new lines of promising research in the realm of data science applied to unlabeled hardware performance data.

The results gained in this thesis could be applied to save personnel and computing costs. The distance metrics and clustering approaches can be used to detect abnormal behavior for a given user. By making these measures and labels available to users, they could self identify abnormal behavior in reoccurring jobs. This information, in addition to pointing them to dashboards and other resources for profiling their jobs in more detail, can decrease time to fault detection and unnecessary debugging runs. All of this could increase HPC cluster efficiency.

Further, changes to the approach used in this thesis are proposed. In future work, other dimensionality reduction techniques (e.g. T-SNE, self-organizing maps) and clustering approaches (e.g. hierarchical or density based) could be applied. These can potentially produce better job representations as they model data differently.

To further improve distance measures defined in this thesis, features that cause run-to-run variability could be singled out and corrected for. In section 4.6, noise in benchmark runs of the same type was identified. With variability being caused by a multitude of factors such as turbo-boost [AMK16] and interference across nodes caused by shared IO systems (e.g. file system)[Lof+10], this is is not an easy task. Simply removing features with goal of minimizing distance between runs of the same benchmark type can lead to the loss of separation between different benchmark types. To avoid this problem, wrapper methods [KS95] for feature selection could be used in conjunction with the scores defined in this thesis.

The results of this work can also serve as a starting point for further research. For example, given job representations of the same job on two different hardware types, a mapping from hardware type to another could be learned. This is desirable, as it would allow the simple transfer of results from one architecture to another. It would also be a significant step in the direction of defining a hardware agnostic job similarity score.

Using these transformations, energy efficiency of clusters could be improved. Models that are able to predict power usage from hardware performance counters already exist [Che+14]. Together with models transforming job representations from one node type to another, models predicting total energy consumption on different hardware configurations can be built. These predictions can then be provided to users, which in turn can choose to run their job on the most energy efficient hardware available.

In order to easily interpret job representations, it is desirable to look at dimensions that capture programmatic differences in jobs. Without knowing how a program's properties map onto hardware performance counters, it can be difficult and time consuming to manually define features that differentiate jobs in a desired manner. Using labeled data of e.g. many benchmark runs or jobs that are purposefully altered to induce these differences, it could be possible to extract dimensions or components of special interest without expert knowledge. These dimensions could then be used to define job representations that are easily interpreted using reference jobs.

This methodology could also be applied to gauge job optimization potential. Properties of badly optimized jobs, like node imbalance, and bad cache efficiency often have effects on other metrics like MPI communication and memory bandwidth. However, it can be fairly trivial to make optimized code less efficient, through e.g. the use of non-optimal kernels or blocking sizes. Using many purposefully altered benchmarks, the dimensions describing various inefficiencies could be extracted. These could then be applied to other jobs to gauge their optimization potential.

List of Figures

List of Tables

Bibliography

[A P18] A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary. *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*. `https://www.netlib.org/benchmark/hpl/`. [Online; accessed 27-September-2020]. Dec. 2018.

[AMK16] B. Acun, P. Miller, and L. V. Kale. "Variation Among Processors Under Turbo Boost in HPC Systems." In: *Proceedings of the 2016 International Conference on Supercomputing*. ICS '16. Istanbul, Turkey: Association for Computing Machinery, 2016. ISBN: 9781450343619. DOI: `10.1145/2925426.2926289`.

[Bor+19] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini. "Anomaly Detection Using Autoencoders in High Performance Computing Systems." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (July 2019), pp. 9428–9433. DOI: `10.1609/aaai.v33i01.33019428`.

[Che+14] G. T. Chetsa, L. Lefèvre, J.-M. Pierson, P. Stolf, and G. Da Costa. "Exploiting performance counters to predict and improve energy performance of HPC systems." In: *Future Generation Computer Systems* 36 (2014), pp. 287–298.

[Cho+15] F. Chollet et al. *Keras*. `https://keras.io`. [Online; accessed 28-September-2020]. 2015.

[Cho15] F. Chollet. *Building Autoencoders in Keras*. `https://blog.keras.io/building-autoencoders-in-keras.html`. [Online; accessed 28-September-2020]. May 2015.

[Eit+19] J. Eitzinger, T. Gruber, A. Afzal, T. Zeiser, and G. Wellein. "ClusterCockpit — A web application for job-specific performance monitoring." In: *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. 2019, pp. 1–7.

[fol20] @foldingathome. *Tweet 1242918035788365830*. `https://twitter.com/foldingathome/status/1242918035788365830`. [Tweet; accessed 29-September-2020]. Mar. 2020.

[Gui15] C. B. Guillén Carías. “Knowledge-based Performance Monitoring for Large Scale HPC Architectures.” Dissertation. München: Technische Universität München, 2015.

[Han19] L. Hanneken. “Untersuchung der Node-Level Performancevariation auf dem CLAIX2016 und CLAIX2018 Cluster der RWTH Aachen University.” en. In: *Bachelorarbeit* RWTH Aachen University (2019), pp. 2019–. DOI: `10.18154/RWTH-2019-09153`.

[HDF20] M. S. Halawa, R. P. Díaz Redondo, and A. Fernández Vilas. “Unsupervised KPIs-Based Clustering of Jobs in HPC Data Centers.” In: *Sensors* 20.15 (July 2020), p. 4111. ISSN: 1424-8220. DOI: `10.3390/s20154111`.

[inf20a] influxdata. *InfluxDB*. `https://docs.influxdata.com/telegraf/`. [Online; accessed 26-September-2020]. 2020.

[inf20b] influxdata. *InfluxDB*. `https://docs.influxdata.com/influxdb/`. [Online; accessed 26-September-2020]. 2020.

[Inta] Intel. *Intel® Xeon® Platinum 8160 Processor*. `https://ark.intel.com/content/www/us/en/ark/products/120501/intel-xeon-platinum-8160-processor-33m-cache-2-10-ghz.html`. [Online; accessed 28-September-2020].

[Intb] Intel. *Intel® Xeon® Processor E5-2650 v4*. `https://ark.intel.com/content/www/us/en/ark/products/91767/intel-xeon-processor-e5-2650-v4-30m-cache-2-20-ghz.html`. [Online; accessed 30-September-2020].

[Kra91] M. A. Kramer. “Nonlinear principal component analysis using autoassociative neural networks.” In: *AIChE Journal* 37.2 (1991), pp. 233–243. DOI: `10.1002/aic.690370209`. eprint: `https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/aic.690370209`.

[KS95] R. Kohavi and D. Sommerfield. “Feature Subset Selection Using the Wrapper Method: Overfitting and Dynamic Search Space Topology.” In: *KDD*. 1995, pp. 192–197.

[Llo82] S. Lloyd. “Least squares quantization in PCM.” In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.

[Lof+10] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf. “Managing Variability in the IO Performance of Petascale Storage Systems.” In: *SC ’10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. 2010, pp. 1–12.

[Mac67] J. MacQueen. “Some methods for classification and analysis of multivariate observations.” In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, Calif.: University of California Press, 1967, pp. 281–297.

[MB88] G. Mclachlan and K. Basford. *Mixture Models: Inference and Applications to Clustering*. Vol. 38. Jan. 1988. DOI: 10.2307/2348072.

[Muc+05] P. J. Mucci, D. Ahlin, J. Danielsson, P. Ekman, and L. Malinowski. “PerfMiner: Cluster-Wide Collection, Storage and Presentation of Application Level Hardware Performance Data.” In: *Euro-Par 2005 Parallel Processing*. Ed. by J. C. Cunha and P. D. Medeiros. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 124–133. ISBN: 978-3-540-31925-2.

[Net+19] A. Netti, M. Müller, A. Auweter, C. Guillen, M. Ott, D. Tafani, and M. Schulz. “From facility to application sensor data.” In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Nov. 2019). DOI: 10.1145/3295500.3356191.

[Net+20] A. Netti, M. Müller, C. Guillen, M. Ott, D. Tafani, G. Ozer, and M. Schulz. “DCDB Wintermute: Enabling Online and Holistic Operational Data Analytics on HPC Systems.” In: *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing* (June 2020). DOI: 10.1145/3369583.3392674.

[Oze+20] G. Ozer, A. Netti, D. Tafani, and M. Schulz. “Characterizing HPC Performance Variation with Monitoring and Unsupervised Learning.” In: June 2020.

[Ped+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python.” In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[Röh+17] T. Röhl, J. Eitzinger, G. Hager, and G. Wellein. “LIKWID Monitoring Stack: A Flexible Framework Enabling Job Specific Performance monitoring for the masses.” In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)* (2017), pp. 781–784.

[Rou87] P. Rousseeuw. “Rousseeuw, P.J.: Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. Comput. Appl. Math. 20, 53-65.” In: *Journal of Computational and Applied Mathematics* 20 (Nov. 1987), pp. 53–65. DOI: 10.1016/0377-0427(87)90125-7.

[RWT20a] RWTH Aachen IT Center. *Batch Systems, Hardware (RWTH Compute Cluster Linux)*. `https://help.itc.rwth-aachen.de/service/rhr4fjjutttf/article/e018f684c5624ae6b9bf7f0994d399f2/`. [Online; accessed 26-September-2020]. 2020.

[RWT20b] RWTH Aachen IT Center. *CLAIX / RWTH Compute Cluster*. `https://www.itc.rwth-aachen.de/go/id/eucm`. [Online; accessed 26-September-2020]. 2020.

[Sim49] E. H. Simpson. "Measurement of Diversity." In: *Nature* 163 (June 1949), p. 688. DOI: `10.1038/163688a0`.

[SPE07] SPEC. *SPEC MPI2007 Documentation*. `https://www.spec.org/mpi2007/docs/index.html`. [Online; accessed 27-September-2020]. Oct. 2007.

[tea20] T. pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: `10.5281/zenodo.3509134`.

[THW10] J. Treibig, G. Hager, and G. Wellein. "LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments." In: *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*. San Diego CA, 2010.

[Top17] Top500. *CLAIX (PHASE I, 2016) - NEC HPC1812-RG-2, XEON E5-2650V4 12C 2.2GHZ, INTEL OMNI-PATH*. `https://www.top500.org/system/178934/`. [Online; accessed 27-September-2020]. 2017.

[Top20] Top500. *SUPERCOMPUTER FUGAKU - SUPERCOMPUTER FUGAKU, A64FX 48C 2.2GHZ, TOFU INTERCONNECT D*. `https://www.top500.org/system/179807/`. [Online; accessed 27-September-2020]. 2020.

[Tun+17] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun. "Diagnosing Performance Variations in HPC Applications Using Machine Learning." In: *High Performance Computing*. Ed. by J. M. Kunkel, R. Yokota, P. Balaji, and D. Keyes. Cham: Springer International Publishing, 2017, pp. 355–373. ISBN: 978-3-319-58667-0.

[VCV11] S. Van Der Walt, S. C. Colbert, and G. Varoquaux. "The NumPy array: a structure for efficient numerical computation." In: *Computing in Science & Engineering* 13.2 (2011), p. 22.

[Voh16] D. Vohra. "Apache parquet." In: *Practical Hadoop Ecosystem*. Springer, 2016, pp. 325–335.

[Wik20] WikiChip contributors. *Xeon Platinum 8160 - Intel*. `https://en.wikichip.org/wiki/intel/xeon_platinum/8160`. [Online; accessed 29-September-2020]. Feb. 2020.

[YJG03] A. B. Yoo, M. A. Jette, and M. Grondona. “SLURM: Simple Linux Utility for Resource Management.” In: *Job Scheduling Strategies for Parallel Processing*. Ed. by D. Feitelson, L. Rudolph, and U. Schwiegelshohn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60. ISBN: 978-3-540-39727-4.

[YTU18] K. Yamamoto, Y. Tsujita, and A. Uno. “Classifying Jobs and Predicting Applications in HPC Systems.” In: *High Performance Computing*. Ed. by R. Yokota, M. Weiland, D. Keyes, and C. Trinitis. Cham: Springer International Publishing, 2018, pp. 81–99. ISBN: 978-3-319-92040-5.

A. Appendix

A.1. CLAIX2016 *lnm* node peak performance calculation

The process and quantities needed to calculate a systems peak performance using specifications from Intel [Intb] and the RWTH Aachen IT Center [RWT20a] is shown in A.1. This equation calculates the peak performance of the *lnm* node type contained in CLAIX2016

$$\begin{aligned} peak_{flop/s} &= 2 \text{ [number of sockets]} \\ &\quad * 12 \text{ [number of cores]} \\ &\quad * 2 \text{ [AVX2 fused multiply-add (FMA) Units per chip]} \\ &\quad * 4 \text{ [AVX2]} \\ &\quad * 2\,\text{Flop [FMA]} \\ &\quad * 2.2\,\text{GHz [base clock]} \\ &= 844.8\,\text{GFlop/s} \end{aligned} \tag{A.1}$$

A.2. Statistical quantities of ratios

To statically and accurately represent columns that are derived from the ratio of others, it is necessary to calculate the values these column contain before reducing them to their statistical properties such as means and standard deviations. In this work, this knowledge is applied to derive columns such as packet size in 3.3 before aggregation in section 4.1.

The need for this can be shown for both the mean and standard deviation. An illustration for two columns a and b, as well as their ratio a/b is shown for rows of data in figure A.1.

sample	a	b	a/b
1	6	3	2
2	1	2	0.5

$$\begin{aligned} &\text{Mean:} \\ \mu_{(a/b)} &= (2+0.5)/2 = 1.25 \\ \neq \mu_a/\mu_b &= ((6+1)/2)/((3+2)/2) = 3.5/2.5 = 1.4 \\ &\text{Standard deviation:} \\ \sigma_{(a/b)} &= \text{std}(2, 0.5) = 3/(2\sqrt{2}) \\ \neq \sigma_a/\sigma_b &= \text{std}(6,1)/\text{std}(1,2) = (5/\sqrt{2})/(1/\sqrt{2}) = 5 \end{aligned}$$

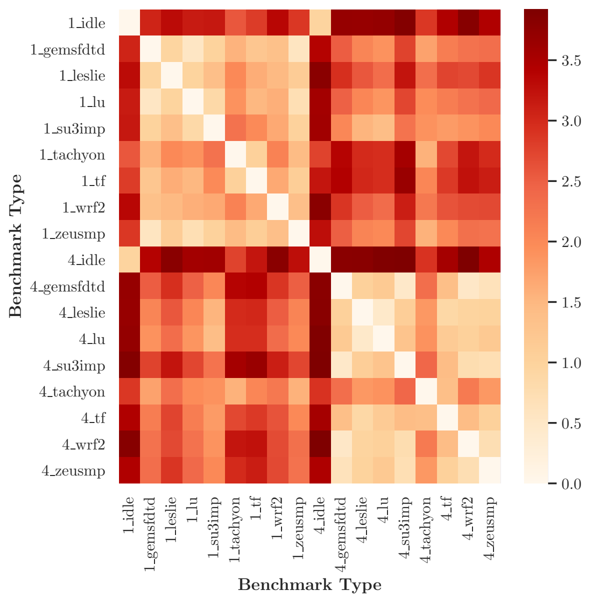
Figure A.1.: Example showing the necessity of deriving columns before applying the mean or standard deviation

A.3. Queried metrics

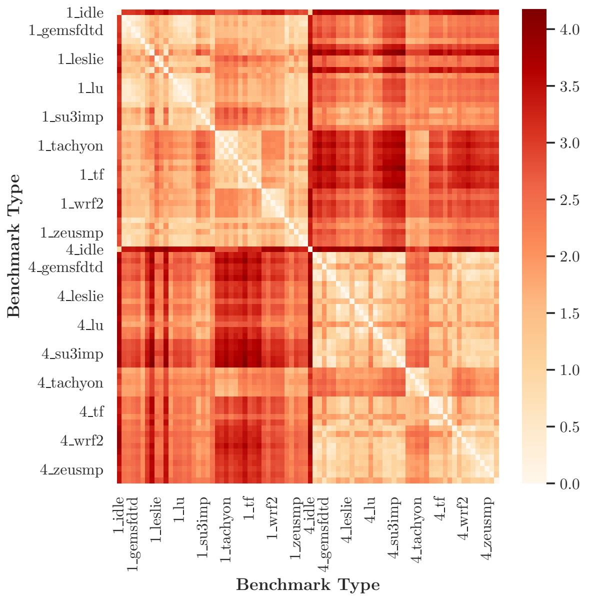
Category	Metric name	Explanation	Unit
Node-Level Metrics			
CPU	clock	CPU clock speed	Hz
	cores_active	Number of active cores	#
	cpu_usage	CPU usage	%
	cpu_usage_iowait	I/O wait	%
	flops_dp_aggregated	Summed double precision Flop/s	MFlop/s
	flops_sp_aggregated	Summed single precision Flop/s	MFlop/s
Memory	mem_active	Memory consumption	B
	mem_bw_read	Memory read speed	MB/s
	mem_bw_write	Memory write speed	MB/s
Socket	power_draw	Summed socket power draw	W
System	temperature	Average temperature readings from various sensors	C
Infiniband Interconnect	inf_rcv_data	Data received	B/s
	inf_rcv_packets	Packets received	Hz
	inf_xmit_data	Data transmitted	B/s
	inf_xmit_packets	Packets transmitted	Hz
Network I/O	net_recv_bytes	Data received	B/s
	net_recv_packets	Packets received	Hz
	net_sent_bytes	Data send	B/s
	net_sent_packets	Packets sent	Hz
Disk I/O	disk_read_bytes	Data read	B/s
	disk_reads	Number of reads	Hz
	disk_write_bytes	Data written	B/s
	disk_writes	Number of writes	Hz
Lustre I/O	lustre_read_bytes	Data read	B/s
	lustre_read_req	Read requests	Hz
	lustre_write_bytes	Data written	B/s
	lustre_write_req	Write requests	Hz
Core-Level Metrics			
CPU	clock	CPU clock speed	Hz
	flops_dp	Double precision Flop/s	MFlop/s
	flops_sp	Single precision Flop/s	MFlop/s

Table A.1.: Metrics queried from InfluxDB

A.4. Heatmaps of pairwise benchmark distances using different dimensionality reduction techniques

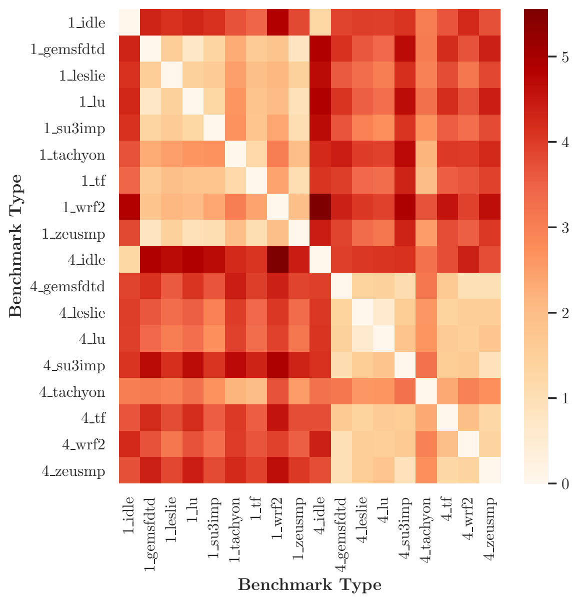


(a) Distances of Mean Benchmark Runs

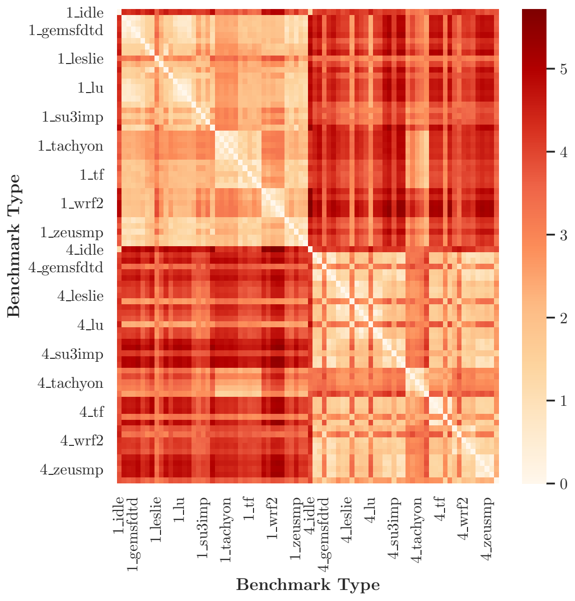


(b) Distances of Individual Benchmark Runs

Figure A.2.: Pairwise Benchmark Distances (Autoencoder: normalized, deep, not regularized)

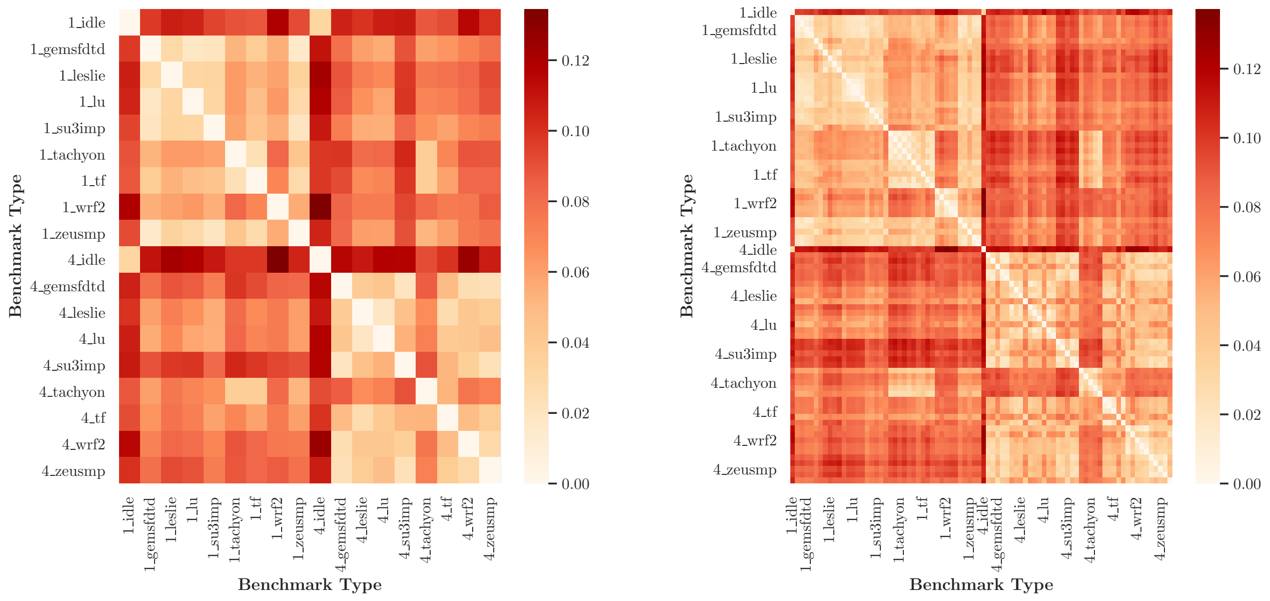


(a) Distances of Mean Benchmark Runs



(b) Distances of Individual Benchmark Runs

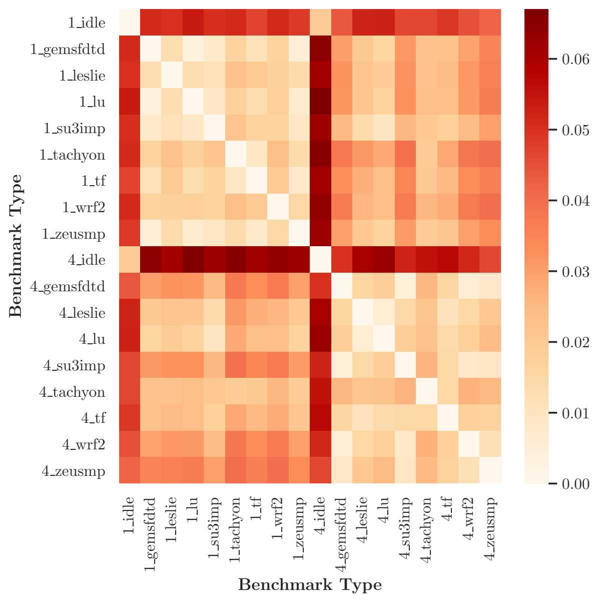
Figure A.3.: Pairwise Benchmark Distances (Autoencoder: normalized, shallow, not regularized)



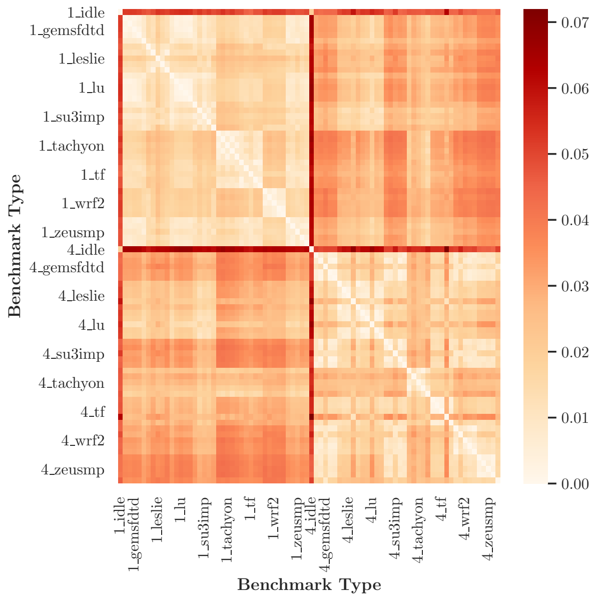
(a) Distances of Mean Benchmark Runs

(b) Distances of Individual Benchmark Runs

Figure A.4.: Pairwise Benchmark Distances (Autoencoder: normalized, shallow, regularized)

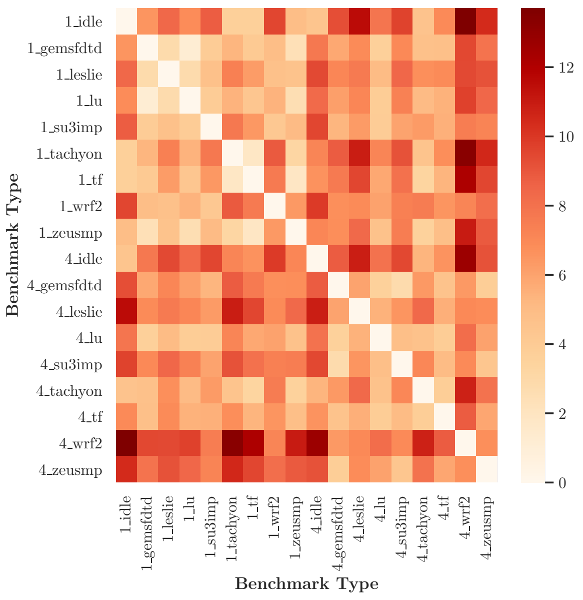


(a) Distances of Mean Benchmark Runs

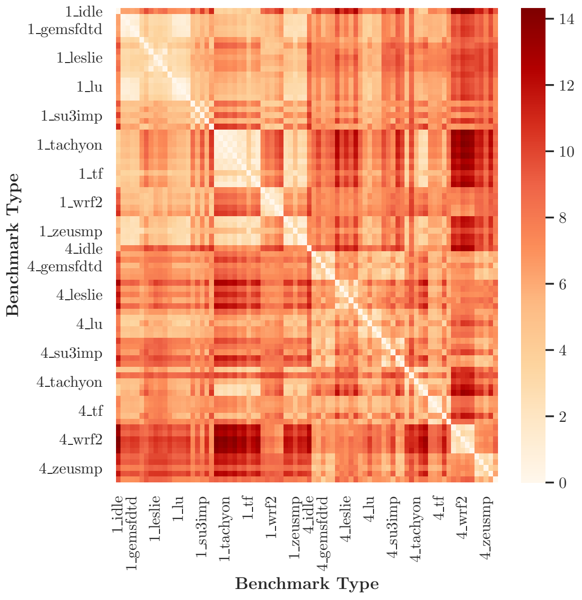


(b) Distances of Individual Benchmark Runs

Figure A.5.: Pairwise Benchmark Distances (Autoencoder: normalized, deep, regularized)

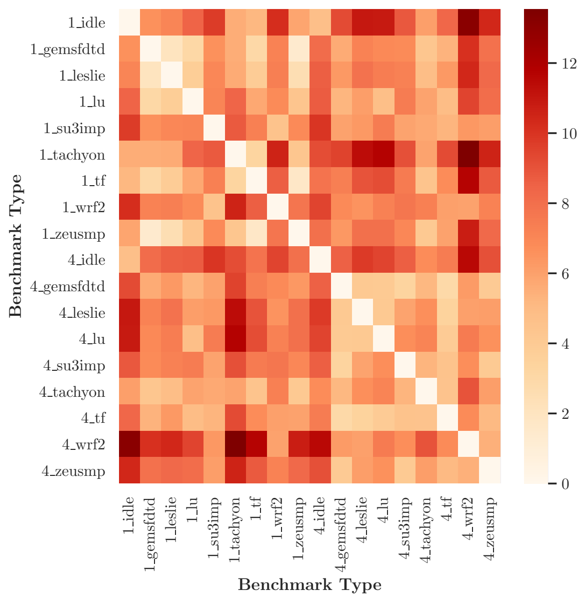


(a) Distances of Mean Benchmark Runs

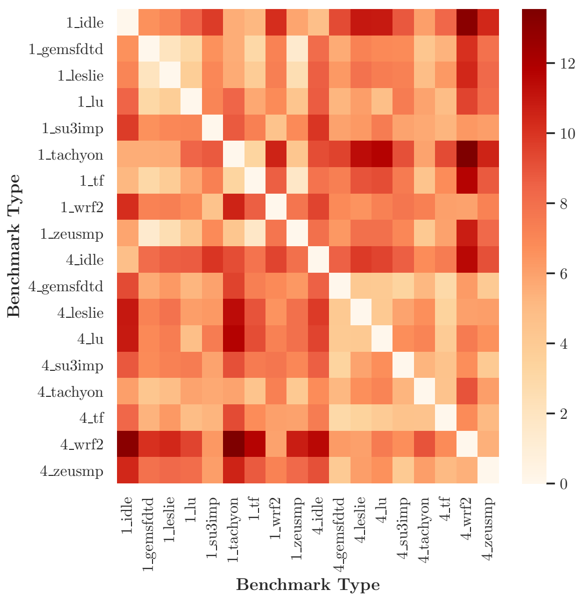


(b) Distances of Individual Benchmark Runs

Figure A.6.: Pairwise Benchmark Distances (Autoencoder: standardized, shallow, not regularized)

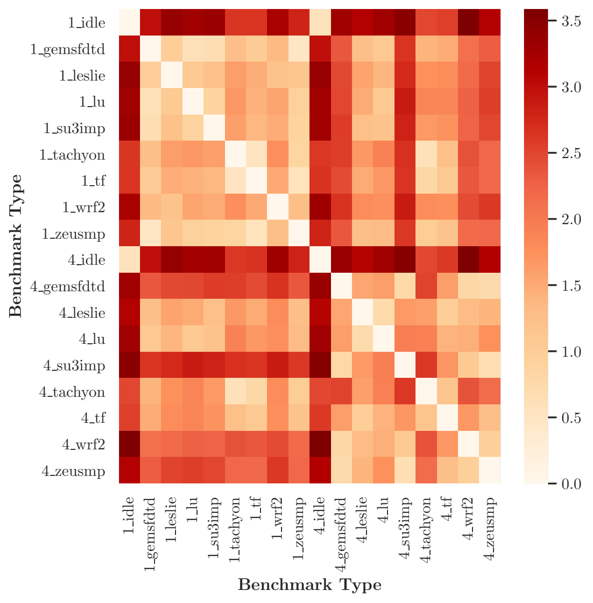


(a) Distances of Mean Benchmark Runs

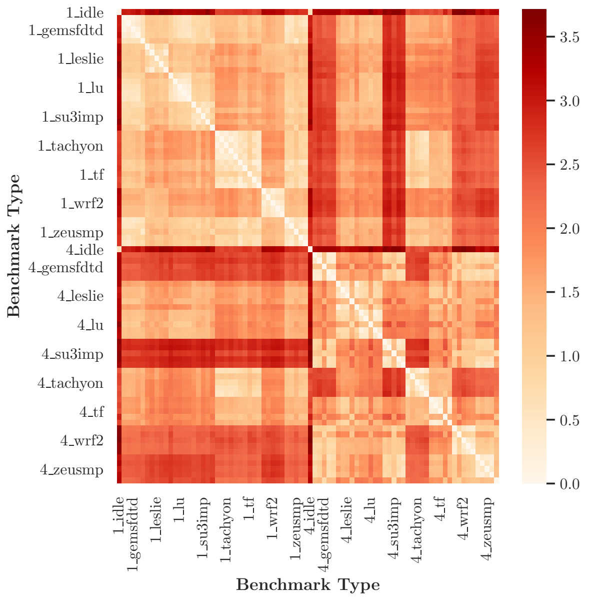


(b) Distances of Individual Benchmark Runs

Figure A.7.: Pairwise Benchmark Distances (Autoencoder: standardized, simple, not regularized)

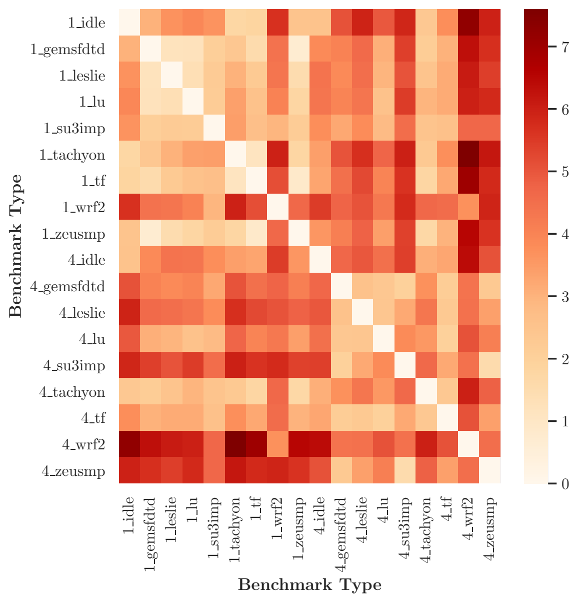


(a) Distances of Mean Benchmark Runs

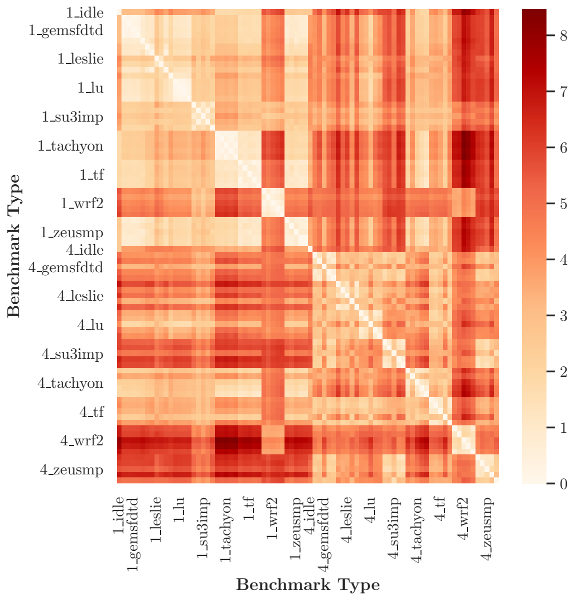


(b) Distances of Individual Benchmark Runs

Figure A.8.: Pairwise Benchmark Distances (Autoencoder: normalized, simple, not regularized)

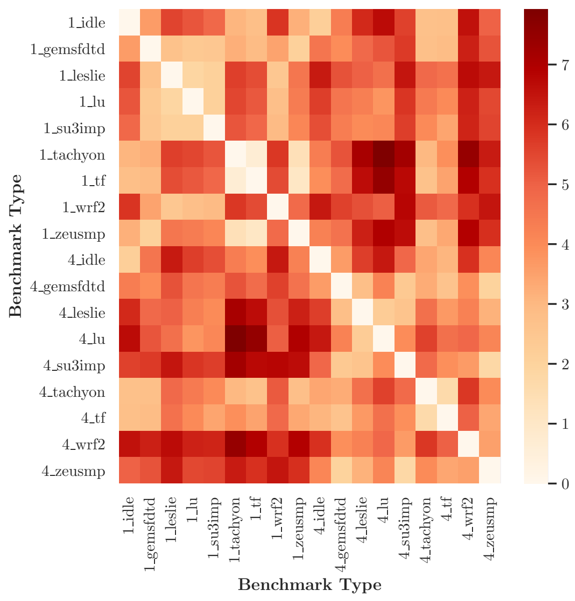


(a) Distances of Mean Benchmark Runs

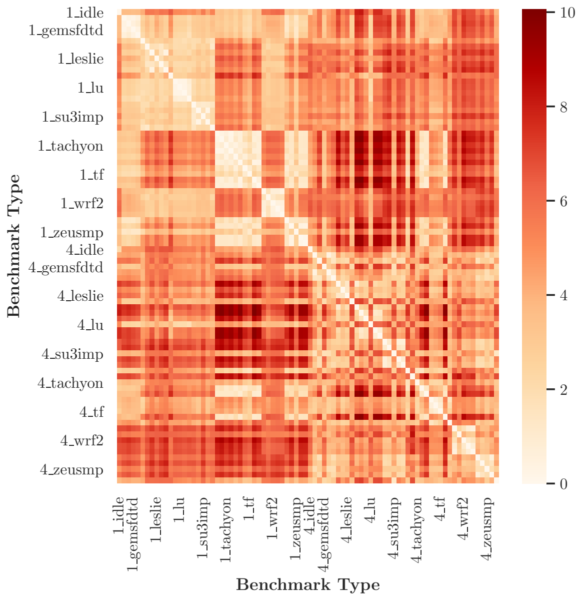


(b) Distances of Individual Benchmark Runs

Figure A.9.: Pairwise Benchmark Distances (Autoencoder: standardized, simple, regularized)

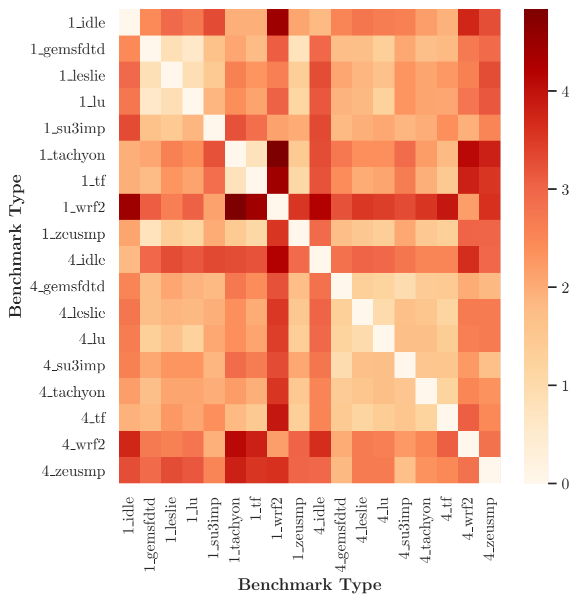


(a) Distances of Mean Benchmark Runs

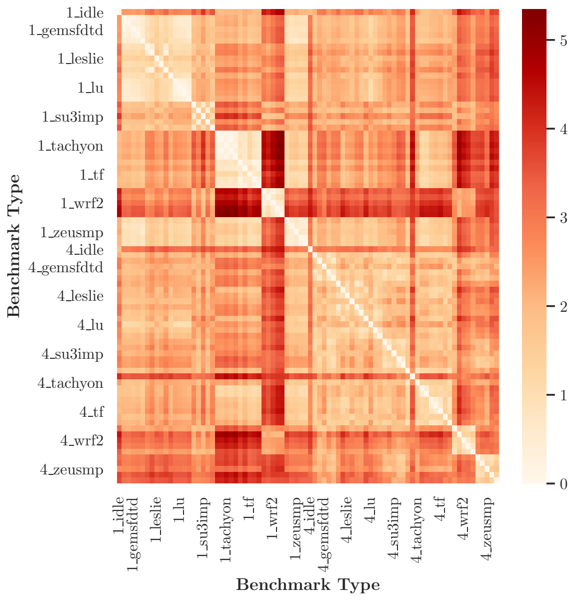


(b) Distances of Individual Benchmark Runs

Figure A.10.: Pairwise Benchmark Distances (Autoencoder: standardized, deep, not regularized)

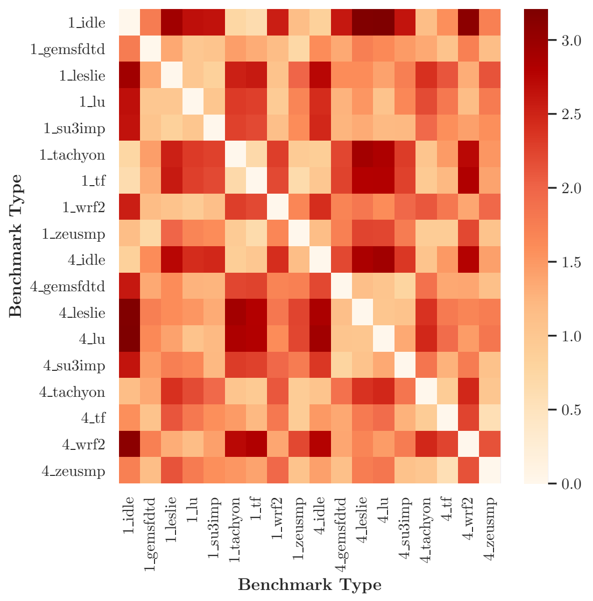


(a) Distances of Mean Benchmark Runs

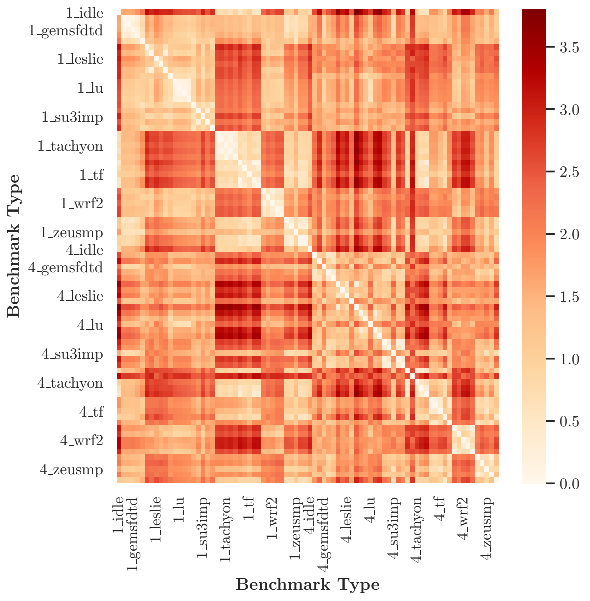


(b) Distances of Individual Benchmark Runs

Figure A.11.: Pairwise Benchmark Distances (Autoencoder: standardized, shallow, regularized)

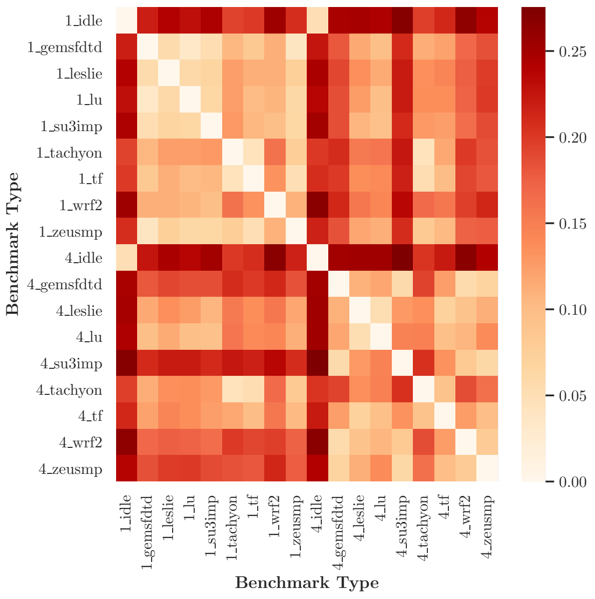


(a) Distances of Mean Benchmark Runs

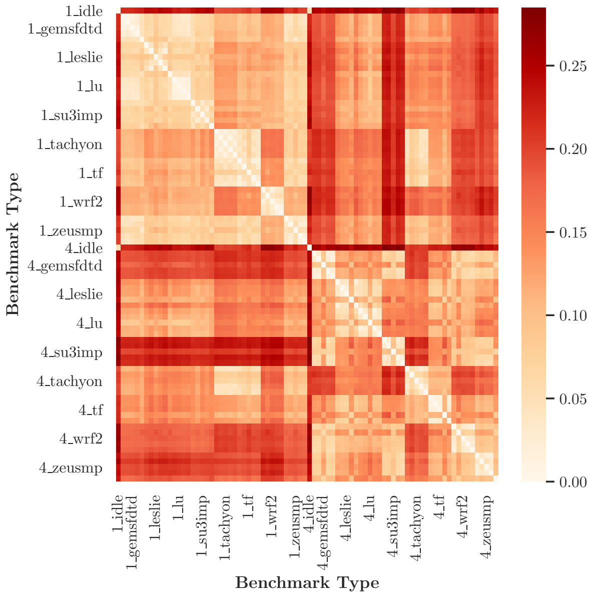


(b) Distances of Individual Benchmark Runs

Figure A.12.: Pairwise Benchmark Distances (Autoencoder: standardized, deep, regularized)

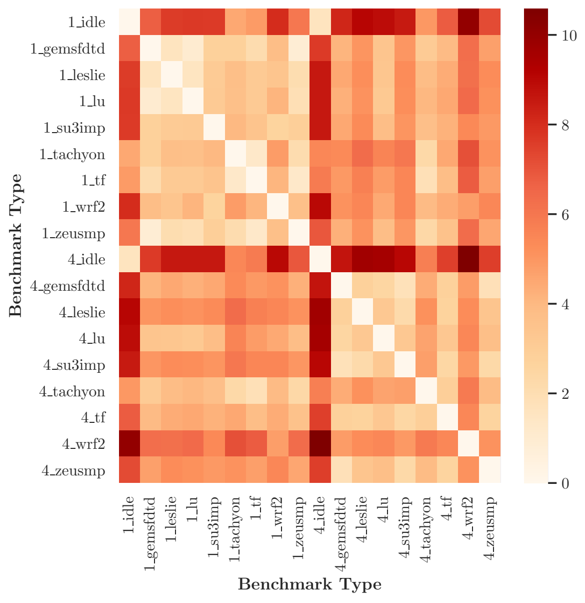


(a) Distances of Mean Benchmark Runs

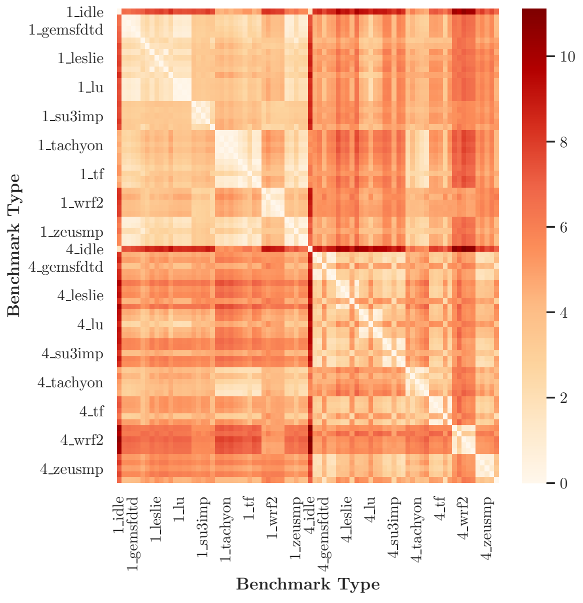


(b) Distances of Individual Benchmark Runs

Figure A.13.: Pairwise Benchmark Distances (Autoencoder: normalized, simple, regularized)

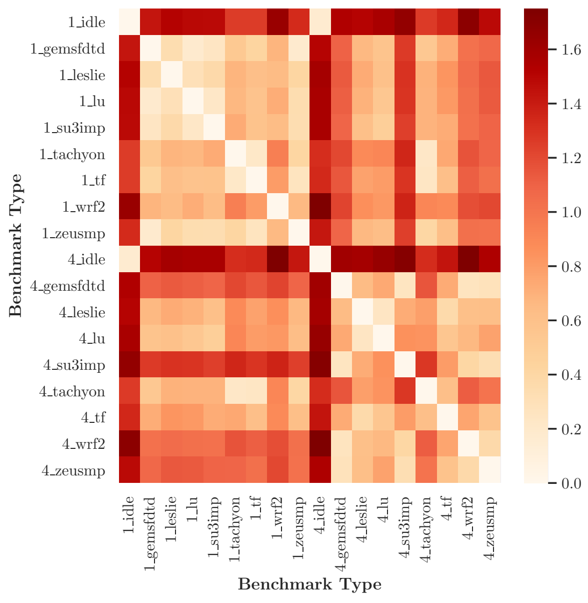


(a) Distances of Mean Benchmark Runs

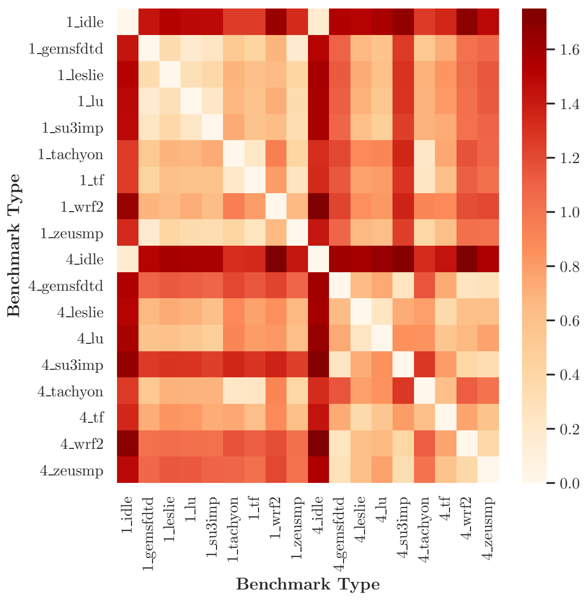


(b) Distances of Individual Benchmark Runs

Figure A.14.: Pairwise Benchmark Distances (PCA: standardized)



(a) Distances of Mean Benchmark Runs



(b) Distances of Individual Benchmark Runs

Figure A.15.: Pairwise Benchmark Distances (PCA: normalized)

A.5. Homogeneity and diversity scores

Dim R.	Scaler	Reg.	Depth	Number of Clusters							
				2	3	4	5	6	7	8	9
K-Means											
autoenc	norm	none	deep	0.94	0.90	0.96	0.94	0.88	0.87	0.86	0.85
			shallow	1.00	0.98	0.98	0.98	0.94	0.94	0.94	0.94
			simple	1.00	1.00	1.00	1.00	0.94	0.92	0.94	0.89
		applied	deep	1.00	0.98	0.97	0.94	0.90	0.90	0.91	0.88
			shallow	1.00	0.90	0.82	0.72	0.78	0.84	0.94	0.93
			simple	1.00	1.00	0.98	0.88	0.94	0.95	0.89	0.88
	std	none	deep	1.00	1.00	0.94	0.79	0.77	0.81	0.75	0.80
			shallow	0.98	1.00	0.90	0.95	0.88	0.80	0.81	0.81
			simple	0.98	0.94	0.93	0.92	0.89	0.92	0.90	0.87
		applied	deep	1.00	1.00	0.87	0.86	0.86	0.86	0.86	0.88
			shallow	0.98	0.98	0.97	0.92	0.88	0.90	0.90	0.86
			simple	1.00	1.00	0.98	0.94	0.95	0.95	0.97	0.95
pca	norm	n.a.		1.00	0.95	0.94	0.97	0.88	0.88	0.88	0.82
	std	n.a.		0.95	0.96	0.91	0.94	0.94	0.94	0.94	0.92
Bayesian Gaussian Mixture Model											
autoenc	norm	none	deep	1.00	0.95	1.00	0.94	0.97	0.95	0.98	0.96
			shallow	0.89	0.93	0.98	0.96	0.86	0.93	0.96	0.96
			simple	0.96	0.84	0.86	0.86	0.86	0.93	0.80	0.89
		applied	deep	0.93	1.00	0.86	0.90	0.87	0.87	0.91	0.85
			shallow	0.85	0.80	0.88	0.86	0.85	0.84	0.84	0.89
			simple	0.97	0.88	0.93	0.91	0.90	0.94	0.90	0.87
	std	none	deep	0.90	0.94	0.94	0.96	0.86	0.84	0.78	0.85
			shallow	0.89	0.86	0.94	0.88	0.84	0.88	0.87	0.87
			simple	0.90	0.94	0.89	0.82	0.88	0.85	0.93	0.83
		applied	deep	1.00	0.85	0.83	0.81	0.70	0.70	0.81	0.76
			shallow	0.94	0.94	0.94	0.88	0.86	0.86	0.86	0.87
			simple	0.92	0.92	0.94	0.89	0.92	0.89	0.91	0.90
pca	norm	n.a.		0.98	0.86	0.86	0.88	0.86	0.93	0.88	0.86
	std	n.a.		1.00	0.98	0.98	0.93	0.90	0.95	0.95	0.93

Table A.2.: Homogeneity Scores of Dimensionality Reduction Techniques and Clustering Algorithms

Dim R.	Scaler	Reg.	Depth	Number of Clusters							
				2	3	4	5	6	7	8	9
K-Means											
autoenc	norm	none	deep	0.40	0.61	0.50	0.52	0.62	0.63	0.63	0.63
			shallow	0.00	0.50	0.49	0.52	0.52	0.52	0.52	0.57
			simple	0.00	0.00	0.04	0.04	0.47	0.48	0.62	0.66
		applied	deep	0.04	0.52	0.56	0.59	0.37	0.37	0.26	0.58
			shallow	0.04	0.31	0.42	0.54	0.71	0.66	0.62	0.61
			simple	0.00	0.04	0.52	0.61	0.51	0.49	0.65	0.65
	std	none	deep	0.10	0.10	0.23	0.50	0.55	0.55	0.64	0.64
			shallow	0.08	0.10	0.34	0.57	0.60	0.62	0.66	0.66
			simple	0.12	0.29	0.32	0.38	0.39	0.46	0.46	0.37
		applied	deep	0.00	0.10	0.36	0.49	0.49	0.49	0.49	0.55
			shallow	0.06	0.12	0.14	0.30	0.45	0.44	0.44	0.44
			simple	0.00	0.10	0.12	0.50	0.55	0.56	0.55	0.56
pca	norm	n.a.		0.04	0.25	0.44	0.40	0.61	0.61	0.61	0.64
	std	n.a.		0.48	0.46	0.54	0.51	0.55	0.54	0.65	0.56
Bayesian Gaussian Mixture Model											
autoenc	norm	none	deep	0.00	0.06	0.00	0.08	0.51	0.50	0.49	0.59
			shallow	0.39	0.38	0.02	0.51	0.64	0.62	0.53	0.53
			simple	0.29	0.56	0.60	0.60	0.62	0.56	0.65	0.60
		applied	deep	0.30	0.02	0.50	0.53	0.62	0.62	0.58	0.69
			shallow	0.35	0.41	0.60	0.49	0.57	0.64	0.66	0.47
			simple	0.34	0.40	0.62	0.61	0.64	0.64	0.69	0.72
	std	none	deep	0.34	0.51	0.48	0.53	0.57	0.63	0.70	0.53
			shallow	0.45	0.53	0.48	0.67	0.70	0.61	0.73	0.76
			simple	0.48	0.52	0.75	0.67	0.61	0.73	0.70	0.73
		applied	deep	0.10	0.54	0.49	0.58	0.72	0.69	0.62	0.73
			shallow	0.35	0.46	0.44	0.73	0.76	0.73	0.75	0.79
			simple	0.45	0.54	0.51	0.60	0.59	0.66	0.63	0.69
pca	norm	n.a.		0.49	0.61	0.64	0.60	0.67	0.62	0.69	0.71
	std	n.a.		0.49	0.65	0.67	0.72	0.73	0.73	0.74	0.73

Table A.3.: Diversity Scores of Dimensionality Reduction Techniques and Clustering Algorithms