Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Adaptive task parameters for Workflow Engines

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)

eingereicht von:	Tom Freudenreich		
geboren am:	10.02.1997		
geboren in:	Berlin		
Gutachter:	Prof. Dr. rer. nat. Lars Grunske		
	Dr. Thomas Vogel		
eingereicht am:	30. Okt. 2023	verteidigt am:	

Abstract. Research in modern science often relies on the use of scientific workflow software. These kinds of systems support researchers in the conduction and automation of computationally heavy experiments, distributed storage of experimental data and the analysis of it. Also, pre- and post-processing tasks and visualizations can be part of workflows. To instruct the workflow software, researchers have to create scripts that describe the steps and tasks within a workflow. Such workflow steps often consist of the execution of a third-party tool, that fulfils a task which is part of the research process. Choosing parameter values for such tasks, is often a manual and time-consuming endeavour. This thesis wants to examine a novel approach in which the selection of these parameter values is automated adaptively to the execution context. Scientist often have knowledge about feasible ranges and sets of parameter values for their domain. We want to define a data structure that allows workflow developers to document this knowledge. Furthermore, the goal of this thesis is to extend the workflow engine Nextflow in such a way that it takes in this data structure, adaptively chooses values for parameters and restarts the corresponding task until no anomaly is detected during execution. We were able to achieve this goal with a working implementation. This was also capable of finding a correct parameter(e-value) in a real world bioinformatics BLAST workflow, faster than a human. It is concluded, that the implementation of these adaptive parameters, can reduce the development time of scientific workflows if the user specifies a feasible range of values for the parameter.

Contents

1 Introduction

Many domains of modern science are bound to handling of vast amounts of data and conducting of complex computer simulations, calculations and other forms of data processing. As such, studies on important present research topics with impact on everyday-life are often executed on large computer clusters and software systems. For example, cancer-research in medicine[1] or simulations for climate change research[2]. It has been identified, that "In the future, the rapidity with which any given discipline advances is likely to depend on how well the community acquires the necessary expertise in database, workflow management, visualization, and cloud computing technologies"[3]. This recent shift to computer facilitated research has led to the new paradigm of "e-Science".
A core feature of this new paradigm is the use of so-called "scientific workflow management systems". This is software that supports researchers in the automation of their various daily tasks, such as the execution of computer based experiments, storage and distribution of experiment data, pre- and post-processing, analysis and visualization[4]. In this context, a *workflow* describes an arrangement of these research tasks and the dependencies in-between them.
This work aims to devise a methodology that will accelerate the creation and use of scientific workflows - thus ultimately benefiting the speed of research teams. Furthermore, it shall ease the development of workflows and as such boost the accessibility of workflow software.

1.1 Motivation

A scientific workflow is usually constituted out of smaller work units that are often called *tasks*. These typically resemble the execution of a third-party tool that a domain scientist uses in his research work. The used tools can be very diverse. So are their parameters, as they additionally differ in type, structure, naming-conventions and range of possible values. Even tools, that solve the same problems, can have different interpretations for parameters with the same name. It can thus be concluded, that it can be a challenge for workflow developers to overview all the different parameters and configurations of third-party tools for their research branch. In addition to the manifold appearances of parameters, it is also difficult to find correct values for them[5].

Researchers usually have an idea of feasible ranges and sets of values for parameters of their science domain. They have gained that knowledge through for example past experiments, from articles of related research or their own logical deduction. But during workflow development, trying out all these values one at a time until a favourable one is found, is a very time-consuming effort, as the execution of scientific workflows usually is bound to long computing times.

1.2 Goal and Scope of the Thesis

In this thesis, the objective is to automate the selection of parameter values for tasks within a workflow. Users of workflow software should be supported by an algorithm, that will test values for a task parameter, until the task execution completes without an error. Every time the execution fails, the parameter value should be adapted automatically. As aforementioned, researchers usually have an understanding of feasible ranges and bounds of valid values for task parameters. Hence, the focus of the thesis will be, to harness this knowledge and (in worst case exhaustively) try out all elements inside these bounds, until the third-party tool of a task has been executed successfully. If the user has further knowledge about the structure of the values in-between the bounds, he can choose a search strategy and/or input settings that tune the adaptation performance. The focus of the thesis is set on anomaly detection, not parameter optimization.

The approach shall be implemented as an extension of the workflow system *Nextflow* and only control single parameters of tasks and processes (and not hyperparameters of the script). For this purpose, an appropriate data structure for the user knowledge about the parameter space must be defined. Further, an algorithm for the adaptation process has to be designed, that decides, how a task execution failure or success is handled.

We want to investigate two research questions:

1. *Feasability*: Is it possible to create such an implementation?
2. *Real world usefulness*: Would such an implementation save time, during the development of a workflow?

   Undoubtedly, it is not possible to make a general statement about overall reduced research process time, since every research project holds individual challenges. Not in every workflow creation, a support for selecting parameters is needed. Thus, we set the scope of the research question, to only focus on the process of parameter selection during workflow development. The user should be supported in that and the time needed, to find parameter values that allow successful task execution should be reduced.

2 Background

In the following chapter, the basic concepts and terms that are the fundament of the thesis topic and pursued implementation are defined and explained.

2.1 Scientific Workflows and Workflow Software

The concept of *workflows* originated from the business world and later was adopted and further enhanced by the scientific community. The *Workflow Management Coalition* (WfMC) defined Workflows in 1996 as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules"[4].
Workflows, that are subject to this thesis, describe complex compilations of computational experiments and pre- and post-processing tasks. In the last decades, numerous software tools have emerged that support researchers in conducting of these operations. *Deelman and colleagues* comment on these developments, and define: "All science campaigns of sufficient complexity consist of numerous interconnected computational tasks. A **workflow** in this context is the composition of several such computing **tasks**. A workflow management system (**WMS**) aids in the automation of those operations, namely managing the execution of constituent tasks and the information exchanged between them."[6].
Core part of a WMS is the workflow **engine**, which manages the orchestrated execution of the tasks, according to a workflow definition. This is often expressed in a structured, programming or domain specific language(see chapter 2.2). Although abstractly viewed, many WMSs essentially serve the same purpose, each science branch has its own preferred user interfaces and domain inherent specific features. There are also many execution environments exist, for example: high performance clusters(HPC) or more modern approaches with dynamic computation resource allocation on cloud servers. Thus, a vast landscape of workflow software systems has been created. Ambitions to define standardized interfaces to achieve interoperability, have been undertaken[7][8].

2.2 Domain Specific Languages and Nextflow

The workflow engine accepts a parsable description of the tasks it should execute and the dependencies in-between them. In this thesis, we will call this description a *workflow script*. A workflow system usually can accept different types of scripts. Often, these scripts are structured files, that contain the calculation steps and invocations of third-party tools, including their parameters or requirements to the execution environment. The *Pegasus* system, for example, uses XML[9]. Some systems also offer an interface, that is programmable with a programming language, like the *Fireworks* system by Python[10].
A domain specific language(DSL) is defined as "a computer programming language of limited expressiveness focused on a particular domain" [11] To allow a fast research process, many workflow systems created domain specific scripting languages, that are

easy to use for researchers, that belong to this area. The programming language Groovy can easily be modified, to create DSL's[12].
The workflow engine Nextflow uses such a Groovy extension, to offer an easily understandable scripting language for workflow developers and researchers. The program's documentation states: "The Nextflow syntax has been specialized to ease the writing of computational pipelines in a declarative manner"[13]. More information on the Nextflow system itself can be found in chapter 3.1.5.

2.3 Parameter Adaptation

First, some terms that are important in the context of the thesis, have to be distinguished and their use explained:

- Execution *success:* In the context of this thesis, a success shall be the completion of task execution with a returned result value. An *anomaly* or a *failure* is the premature termination, usually also accompanied by an error object.
- *Correct* parameter value: Correct values for parameters, are those allowing a successful execution. *Incorrect* ones are those that trigger anomalies.
- *Parameter space*: Usually this space is the entirety of all possible values a parameter can be configured with. In the thesis, this term is also used to refer to the set of parameter values, that a user wants to test out in the adaptation process. Ideally, this should be significantly reduced in size in contrast to the entire parameter space and exclude *incorrect* parameter values.
- *Process*: This keyword is used by Nextflow to declare a task in a workflow script. Further, throughout the thesis, the term is used synonymously with the word "task" and also sometimes refers to the Linux process of an executed task.

To our knowledge, there is no direct and complete feature of self-setting task parameters that seek a successful task execution in the well-known[14][15] workflow engines, like we pursue it in the thesis implementation. Though there are many similar approaches and features, that are discussed in the next chapter.

2.3.1 Search Strategy

Every time, a task with an adaptive parameter fails, the parameter value gets adjusted. The *search strategy* defines the policy that is used to decide which parameter value is used next, during the adaptation process. Since the thesis is only concerned with anomaly detection and not with performance optimization, cost functions for task execution will not be researched. Thus, a large part of traditional research on parameter selection approaches, based on gradients or combinatorial optimization, is excluded.

3 Related work

In this chapter, related research and preliminary work is discussed. Current popular workflow engines and their feature sets are examined to show that the thesis approach is novel. Furthermore, existing workflow engine features that achieve similar functionality like adaptive parameters get explored.

3.1 Feature Sets of Current Workflow Engines

Surveys of the feature sets of popular workflow engines have been done[9][16]. Different engines try to implement different capabilities, appropriate to the domain they are operated in(and the use cases and requirements inherent to the domain). Hence, a large landscape of workflow systems have been created and are still further developed. In the next section, some of the most capable and most used engines are presented. As the research for this thesis has been pursued, no system with an implementation of the exact approach anticipated in the introduction was found. However, some of the engines are capable of executing workflow scripts that could emulate the behaviour of adaptive parameters.

3.1.1 Galaxy

Galaxy is a popular workflow engine, used especially in the biomedical and bioinformatics domain. It offers easy accessibility via browser and allows running workflows in the cloud [17].
As opposed to “task” (as it is called in this thesis), a step in a workflow in Galaxy is referred to as a *“tool”*. Parameters and input data for tools, are set manually on the web GUI or are propagated output from previous workflow steps[18]. Loops and conditionals are not possible in a workflow[19]. Some tools have the capability to get *dataset collections* as an input to perform so-called *“batch operations”*[17]. In that way, a task can be executed repeatedly with different parameters. This feature is similar to the goal of this thesis, but lacks support for parameter bounds, as all values that shall be tried out and the ordering has to be specified explicit. Further, not all tools have configured there parameters to support dataset collections.

3.1.2 Pegasus

The workflow system *Pegasus* “is used intensively by various research communities, e.g. astronomy, biology, computational engineering”[14]. It offers special features, like optimized scheduling algorithms and provenance data for debugging.
Workflow descriptions are stored in XML or YAML format and can be created with an API available in Java, Python or Perl. Pegasus uses this abstract definition of the workflow, to create a *WEP*(workflow execution plan) with concrete tasks and necessary

data transfers for the executing system. Task dependencies are also part of the WEP and determine, among other things, the flow of data between task outputs and inputs. Parameters are set either with fixed values in the workflow description or are outputs of previous tasks[20]. According to our researched work on Pegasus and its application documentation([21]), no special data structure for bounded or adaptive parameters is available.
Nevertheless, fault discovery and recovery features are built in. After fault detection, a failed task can get retried a fixed number of times automatically. Also, a workflow can be resubmitted, so that past calculated results of successful tasks don't have to be computed again. Further, it is possible to specify an additional alternative data source[15].
Another feature called "hooks" allows to add routines that get executed if specific triggers are induced[22]. Also, with *Variable Expansion*, placeholders in the task script and parameters can be substituted at runtime[23]. The combination of those functionalities allows, to implement a form of adaptive parameters, but all logic for setting the parameter values has to be programmed explicit in the hook.

3.1.3 Kepler

Although a little bit older, but therefore also more stable, Kepler is a scientific workflow system based on the Ptolemy II engine and is used in many different domains[24]. Kepler is Java based and open source. A GUI is used to create workflows that can contain interleaved, conditional sub-workflows and loops(through so-called *repeat tasks*). Data flows between the *output ports* of tasks, to the *input ports* of tasks. These *data channels* can have different types, like: String, Integer, Files. Additionally, a task can have parameters, that are set to a fixed value. What makes Kepler unique as a workflow system, is that one can choose the *model of computation* for every workflow. This sets the execution semantics of the workflow. Example models are: Process Network, Synchronous Data-Flow or Discrete Events [25]. In the current version of Kepler(2.5), no parameter bound data type is supported for task parameters. However, it is possible to implement own custom logic(like adaptive parameters) by extending the task classes in Java. Also, exception handling is user defined, which allows actions like resubmission and starting alternative tasks on anomalies, but no built-in parameter adaptation.
Figure 1 contains an example screenshot, that visualizes Kepler features in a geology rock classification workflow.

3.1.4 Fireworks

Fireworks is a workflow software that is often used "for running high-throughput calculation workflows at supercomputing centers" e.g. for computational chemistry and materials science calculations[10].
In this system, workflow definitions are formulated in python scripts. A step in a workflow is called a "Firework" and is described by a python command or JSON file, that specifies a set of tasks that shall be executed and also values for their parameters.

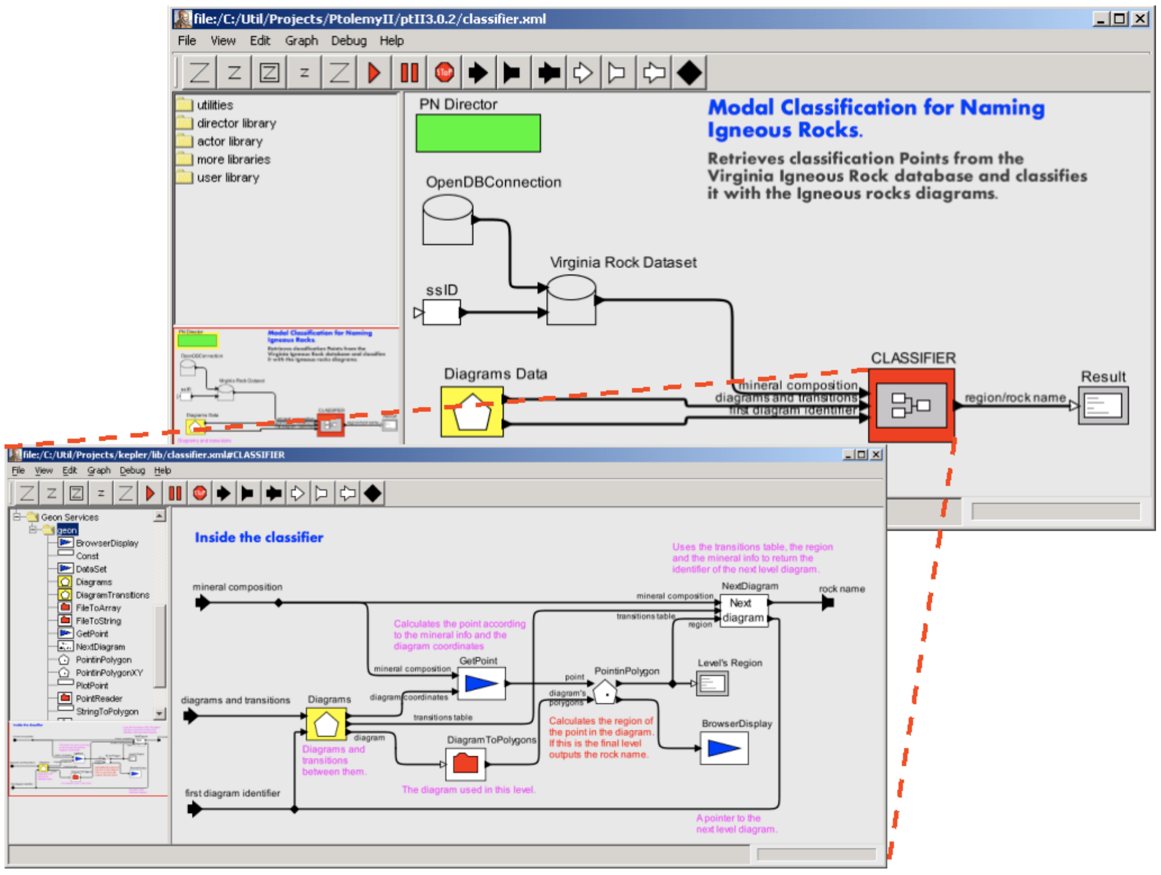
file:/C:/Util/Projects/PtolemyII/ptII3.0.2/classifier.xml
File View Edit Graph Debug Help
utilities
director library
actor library
more libraries
user library
PN Director
Modal Classification for Naming Igneous Rocks.
Retrieves classification Points from the Virginia Igneous Rock database and classifies it with the Igneous rocks diagrams.
OpenDBConnection
Virginia Rock Dataset
ssID
Diagrams Data
CLASSIFIER
mineral composition
diagrams and transitions
first diagram identifier
region/rock name
Result
file:/C:/Util/Projects/kepler/lib/classifier.xml#CLASSIFIER
Geon Services
BrowserDisplay
Const
DataSet
Diagrams
DiagramTransitions
FileToArray
FileToString
GetPoint
NextDiagram
PointinPolygon
PointinPolygonXY
PlotPoint
PointReader
StringToPolygon
Inside the classifier
Uses the transitions table, the region and the mineral info to return the identifier of the next level diagram.
mineral composition
Calculates the point according to the mineral info and the diagram coordinates
GetPoint
NextDiagram
Next diagram
rock name
transitions table
region
point
PointinPolygon
Level's Region
diagrams and transitions
Diagrams
diagram coordinates
diagram's polygons
Calculates the region of the point in the diagram. If this is the final level outputs the rock name.
BrowserDisplay
Diagrams and transitions between them.
DiagramToPolygons
The diagram used in this level.
first diagram identifier
A pointer to the next level diagram.

Fig. 1: Example Kepler geology workflow

Workflow developers are able to implement adaptive parameters using the python script. But again, the logic has to be implemented fully, and there exists no data structure in Fireworks, that is already built-in, that simplifies the implementation of parameter adaptation.

3.1.5 Nextflow

Nextflow focuses on scalability and reproducibility. It is designed specifically for bioinformaticians familiar with programming[26]. A variation of groovy is used as a domain specific language, to formulate workflows[13]. A task is also referred to as a "process" in this language. Nextflow implements a *dataflow* model in which data gets passed between in- and outputs of processes, that depend on each other. So-called *Channels* are employed for this connection. Channels can further be supplied with fixed collections of values.
In that way, tasks get executed every time when parameters and input data are available in connecting channels. With a channel supplied by fixed input data, it is not possible to test a range of values for correct parameters. As soon as a process is executed with an incorrect parameter, the workflow stops executing. It is not possible to predict,

which value is handled from the Channel first, since this depends on the order, in which the OS schedules the processes. So, during the development of a Nextflow script, it is not feasible to use a Channel to test out multiple values for a task. The Nextflow application would have to be started with the same workflow repeatedly, which would cost a lot of time.
To provide the dataflow model, Nextflow also makes the restriction, that a process is only allowed to be declared and used once in a workflow. So, although groovy constructs like conditionals and loops exist, it is not possible with them to traverse a range of values and start a process repeatedly with parameters set to these values. Thus, no simple way to implement adaptive parameters is given.

3.2 Similar Approaches for Thesis Use Case

3.2.1 Workflow features supporting parameter selecting

Popular and modern workflow engines offer helpful features to find correct and optimal parameters, or to even fill in values for task parameters automatically in different scenarios.

Workflow Steering and Parameter Fine Tuning

Current workflow engines offer a feature called *"Workflow steering"*, that is used to fine tune parameters in workflows. For that, usually, a technique called *Checkpointing* is implemented: A workflow execution is stopped in certain conditions(e.g. on a breakpoint or if a maximum number of task-retries is reached) and the already computed part of the workflow is saved.
That way, the part does not have to be executed again and multiple parameters can be tried out for the current task, in shorter time[9] [27][5].
Nextflow implements checkpoints(“-resume”) and task retries on errors [28] [29].

Workflow Provenance

Data Provenance is a record of the history of the creation of a data object[9].
Gil et al. present an approach, using provenance metadata for the workflow system ”Wing”, including automatic parameter setup. *Nextflow* implements various provenance features[30]. It has to be researched, if this approach is compatible with the implementation restrictions, given in the introduction.

3.2.2 Dynamic Workflows

Some workflow engines(e.g. *Fireworks*) support *”dynamic workflows”*, that are able to modify themselves during execution. The conditional nature of this feature is also used, to create *“self-healing” systems*, for example to react on execution failures or to start refinement calculations [10]. So, the workflow engine can independently(without manual human intervention) decide to start additional processes.

This offers the user the possibility to create its own adaptive routines and parameter selections. As stated in the introduction, a goal of this thesis is to conceive a dedicated data structure for parameter bounds. In the thesis it is pursued, to offer the user a simple interface to implement adaptive parameters in their workflow scripts. Thus, in contrast to *Fireworks* approach, it should be possible to define those without having to create own adaptive program logic.

3.2.3 Workflow Optimization

Another field of research is the workflow optimization: Tasks get reordered in a workflow, unneeded task get deleted, or performance creating tasks get inserted, to increase overall execution efficiency and decrease resource use and execution time[31]. The optimizations that some workflow schedulers do, can be separated into two groups:

- **Static:** Those are applied before the execution.
- **Dynamic:** Those are applied during execution.

There is not much research on task parameter optimization or automatic value selection during these optimization phases. Some work is done on hyperparameters[32].

3.2.4 Using Application Parameter Databases

For a lot of third-party tools and software, the parameter structure and also valid, correct or popular values for parameters are already documented.

CLI Autocompletion

A related but more pragmatic field, is addressing the autocompletion of CLI commands. For a given part of a command string, possible endings(with a high chance of being useful for the user) shall be suggested.
In many terminals and shells, configuration files and scripts exist, that store valid parameter values for CLI applications and are used to offer completion[33]. Some shells (like [34]) offer auto suggestions, based on command history and completions.
The website *fig.io* uses more advanced approaches, like a large open database of auto completions for CLI applications[35].
Also AI models are used, to create whole commands from textual descriptions[36].

The *nf-core* Pipeline Library

The *nf-core* library is a collection of ”collaborative, peer-reviewed, best-practice analysis pipelines”[37].
Part of the pipeline documentation is also the process configs and tunable third-party tool parameters(most often CLI options)[38]. In so-called ”nextflow schema”-files, also community approved default values are stored[39]. (an example scheme: [40])
This data may be used as a source for good parameter values and can otherwise also

be a training set, for learning an association between parameter name and common values.

3.2.5 Machine Learning Approaches

The use of machine learning techniques to improve workflow engines has been studied[5]. There are approaches that try to reduce the number of job executions for parameter space explorations[41]. Those harness data from previous workflow executions and execution cost functions, to prune the parameter space. Neither is generally available in the realm of this thesis's implementation.

4 Implementation

The Nextflow engine that executes the workflows, was programmed in Groovy and is publicly available under the Apache 2.0 Licence[42]. To add adaptive parameters to the codebase, different implementation approaches are possible. First, it has to be decided, which information a data structure describing an adaptive parameter has to hold. This is described in chapter 4.1. The following chapter identifies, how Nextflow can be extended, to adjust the workflow execution to allow repeated task executions with adapted parameter values. Different strategies, for choosing the next value to be tested, will be studied. In the last chapter, it will be elaborated, how the adaptation process will handle the different task execution results. The complete extended version of the Nextflow code base can be found on GitLab: https://gitlab.informatik.hu-berlin.de/freudeto/bachelor-thesis.git

4.1 Data Structure

As stated in the introduction, this thesis will focus on enabling a Nextflow user, to specify his knowledge of a possible range of values for a task parameter. This specified space can then be traversed, to test out these values until a favourable parameter value is found for the task. A data structure representing this parameter space, can have different appearances, depending on the data type of the parameter.

Nextflow can parse and execute workflows that are formulated in the Nextflow scripting language. This domain specific language(DSL) is an extension of Groovy and as such employs the typical data types, i.e. integers, float, booleans and strings. In the realm of this thesis, the parameter spaces, that get explored adaptively, get divided into:

- *Numerical spaces:* These shall be specified by the user by supplying the bounds of an interval. A lower and upper bound given with integers, represent a finite set of possible values that can be traversed by the adaptive algorithm. Using floating point numbers for the bounds, an additional discretization is needed first.

- *Categorical spaces:* These shall be given directly by an array of possible values. The user can specify a list of values, even of different data types.
  Example: A user wants to supply a boolean "false" value as a parameter to a program, but does not know the representation of the boolean state for the program argument. He could supply the categorical space [false, "false", 0, null]. As such, the adaptation algorithm should automatically choose the correct representation. This example is also used as an experimental setup in chapter 5.2.

Since the Nextflow DSL is based on a dataflow model, parameters of tasks in workflows get supplied via input channels. These channels can have different types. The scope of the thesis, only focusses on the *val* input type.

4.1.1 Adaptive Parameter Specification

To enable the user to capture his knowledge about a parameter space, three callable methods have been added to the Nextflow scripting language. The `BaseScript` class, which inherits from the `groovy.lang.Script`, is the foundation for every Nextflow script. As such, a user creating a Nextflow script can access the methods of `BaseScript`. For the identified use cases of *numerical* and *categorical* parameter spaces, the following three methods have been added to `BaseScript` and allow specifying an adaptive parameter:

- `adaptparam_int(String forProcess, int from, int to, String withSearchStrategy, Integer startAt=null)`
- `adaptparam_float(String forProcess, double from, double to, int maxSteps, String withSearchStrategy, Double startAt=null)`
- `adaptparam_categorical(String forProcess, ArrayList categories)`

In Table 1, the signatures of the instantiating methods are explained in more detail. Also, an example of usage is shown in listing 1.

Name	Purpose
forProcess	Contains the name of the process, in which the adaptive parameter should be used.
from	Gives the lower bound of numerical parameter spaces.
to	Gives the upper bound of numerical parameter spaces.
maxSteps	(Is only used in adaptparam_float) Is used to discretize continous intervals, that are given by floating point numbers. The resulting set, contains *maxSteps* elements, drawn equidistantly from the interval.
withSearchStrategy	Name of the searchStrategy for the adaptation process. Since only two searchStrategies exist, an script compilation error is thrown if an unknown name is supplied here. The functionality of search strategies will be discussed in more detail later.
startAt	(Is only used, when the search strategy is “sorted”) This marks the parameter value between the lower and upper bound, for which the user assumes the highest chance, that it allows a successful process execution. Since it can only be used with the “sorted” search strategy, its default value is `null`.
categories	(Is only used in adaptparam_categorical) The array of values, that shall be adaptively tried out.

Table 1: Settings, that can be specified during creation of an adaptive parameter

```
process myProcess {
  input:
    val parameter1
    val parameter2
  output:
    stdout

  """
    ...
  """
}

workflow {
  adaptparam1 = adaptparam_int(
    forProcess="myProcess",
    from=0,
    to=5,
    withSearchStrategy="random")

  result = myProcess("fixed string parameter", adaptparam1)
}
```

Listing 1: Exemplary creation of adaptive parameter

4.1.2 Parameter Values and Meta Data

To store an adaptive parameter in memory, an `AdaptParam` class has been added. In Table 2 the class members representing the current state and the identification of the adaptation process are explained. For each of the three methods discussed in the previous chapter, a constructor for the `AdaptParam` class exists. These constructors initialize the state members, by calculating the array of values(and their order), that get tested during the adaptation process. Additionally, the settings of the adaptation process, that were submitted by the user, get saved in further class members, that are displayed in Table 3. These are used for logging and eventually for analysis.

The public method `getNextValue()` can be used, to access the parameter value, that should be used for the immediate process execution. During the call of the method, `currentValueIndex` and `currentValue` get incremented and updated accordingly. Since the type of the elements from the `values` array can vary, the return value is of the universal type `Object`.

4.1.3 Access During Execution

During the start of Nextflow, a `Session` object is created, that can be accessed by all classes during runtime. The `Session` class follows the singleton pattern and holds

Name	Type	Purpose
name	String	Is used for identification, when using multiple `AdaptParam`.
process	String	Contains the name of the workflow process, where the adaptive parameter is used.
values	Array of Object	After initialization, this array contains all values, that shall be tried out adaptively for a parameter of the process.
currentValueIndex	Integer	This index holds the state of the adaptation process and indicates the present value that is being tested.
currentValue	Object	The element of the `values` array, at the `currentValueIndex`.

Table 2: `AdaptParam` members, describing the state of the adaptive parameter.

Name	Type	Purpose
minimum	Object	This contains the lower bound of a numerical parameter space. Since integers as well as doubles can be used for the bounds, the abstract `Object` type from Groovy is used.
maximum	Object	Like that, also a maximum for the upper bound exists
searchStrategy	Enum	Name of the search strategy, that determines, in which order the values of the given parameter space are traversed.
startAt	Object	If the "sorted" search strategy is used, the startAt value marks a position between the lower and upper bound, where the user suspects the value, that is most probable, to allow a successful process execution.

Table 3: `AdaptParam` members, describing the meta information of an adaptive parameter.

information of the current workflow execution state.
To give the tasks of a workflow an opportunity, to access information of adaptive parameters at a central point, a further class `AdaptParamList` was added to Nextflow. This eases the creation of `AdaptParam` instances and stores them in a HashMap, where each entry is mapping from the process name, to the adaptive parameter object. The `Session` class was extended by a public `AdaptParamList` member. Using this, all processes of a workflow can retrieve their respective adaptive parameters during runtime,

including the current value, that shall be used for the next execution.

4.2 Search Strategy

For each of the three methods, that can be used to create an adaptive parameter, a constructor of the `AdaptParam` class exists. The general functioning of these are similar: Metadata is saved in the class members, and the `values` array is filled with the parameter values in a specific order. The simplest constructor is used by the *categorical* adaptive parameter. It just sets the `values` array, exactly like the user supplied it in the creation method `adaptparam_categorical`.
In the two constructors for the numerical adaptive parameters, first the `values` array gets filled with the numbers between the interval bounds. For `adaptparam_int` these numbers are all the integers between the bounds. `adaptparam_float` uses the `maxSteps` setting, to define a "step size" by the formula:

$$= \frac{|\text{upper bound} - \text{lower bound}|}{\text{maxSteps}}$$

Now this step size is used, to discretizes the continues interval given by the floating point bounds, into equidistant values. For the sake of simplicity, this is implemented just by *for* loops, adding the step size to the lower bound for `maxSteps` times.
After the `values` array is now initialized with the appropriate numbers, these get placed in a specific order according to the chosen `searchStrategy`:

4.3 Adaptability

The implementation of workflow execution done by the Nextflow engine is extensive and complex. Many approaches for implementing adaptive parameters are possible. In the realm of this thesis, a less intrusive implementation has been pursued.

4.3.1 Variable Injection

The main idea used by the thesis implementation is, that the methods, used to create the adaptive parameters in `BaseScript`, return the name of the `AdaptParam`, which is prefixed by a $ symbol and thus resembles a variable in Bash syntax. This allows users who want to use adaptive parameters, to simply supply the value returned by the method as process arguments. Nextflow, as it naturally operates, then takes these and embeds them into the process script. During process execution, the values for the Bash variables are submitted as environment variables. For that, the `BashWrapperBuilder` has been modified, to set the current values for adaptive parameters as environment variables, while preparing the launch of the next workflow task. As described in 4.1.3, the `AdaptParamList` can be retrieved from the globally accessible `Session` object. For every process, it is now possible to get the `AdaptParam` via the `getAdaptParamForProcess` method.

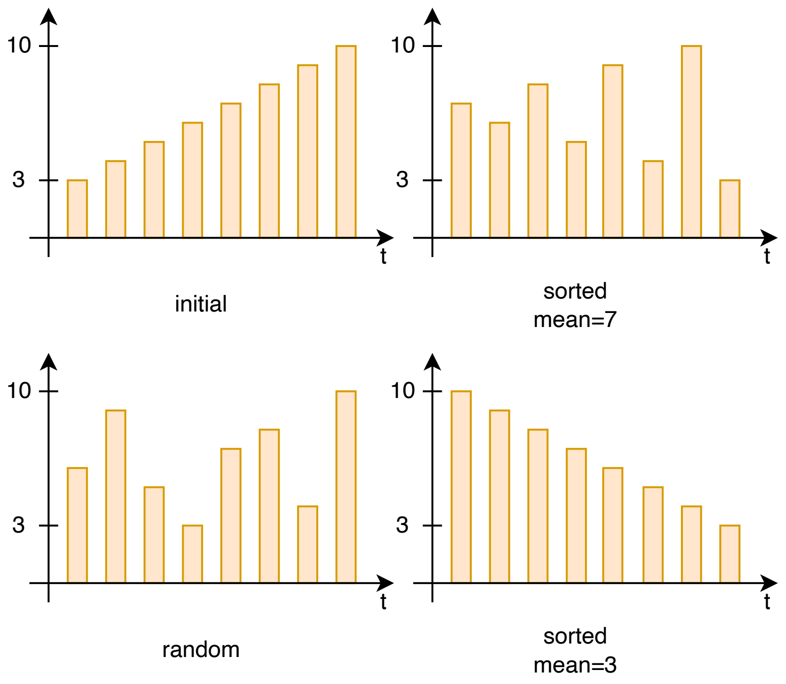

Fig. 2: A given numerical parameter space with bounds from 3 to 10 is visualized, with the corresponding parameter value array in the upper left corner. The other diagrams show the same array, after the algorithms of the search strategies have been applied.

Search strategy	Implementation
random	This search strategy reorders the elements of the `values` array using the Fisher–Yates shuffle algorithm[43]. The user can employ this strategy, if he knows that the values between the supplied bounds are feasible, but not what regions inside the bounds offer the greatest probability of successful process execution.
sorted	For this strategy, additionally to the parameter value interval, the user can supply a value inside the bounds, that he thinks, is the most probable to be a favourable value, for successful process execution. For floating point numbers, the nearest discretized value is used. The `startAt` value, marks this favourable value. For example, if the user suspects correct values in the beginning of the interval, he should choose a `startAt` value near to the lower bound. Starting from this element of the array, all other elements are sorted according to their numerical distance. As such, a value with a small distance comes further to the front in the array and an element with a large distance further to the back. If the user wishes, to just sort the values ascending respective descending, he can use the lower respective upper bound as the `startAt` value.

4.3.2 Workflow Task Retry

Processes in Nextflow have properties that are set by default values, but also can be changed using directives in the workflow script [44]. The `errorStrategy` property describes, how the Nextflow engine should react on errors during process execution. The default mode "`TERMINATE`", kills all pending and running tasks. As such, if during the adaptation process a value for a parameter has been set, that causes an error, Nextflow would terminate and end the workflow execution. Another `errorStrategy` is `RETRY`, which restarts a process after an error has been caught. This is the most suitable of the error strategies for adaptive parameters, but has an issue: Another configurable setting is the `maxRetries` with default value 1 and the `maxErrors` config. These decide the number of retries for a process with `RETRY` as error strategy, respectively the number of summed errors of all process executions(a process can launch different execution instances) for `maxErrors`[45]. Thus, a new errorStrategy `ADAPT` has been added, that acts like the `RETRY` mode, but ignores the limitations of `maxRetries` and `maxErrors`. This new error strategy is selected automatically for a process, if it uses adaptive parameters.

4.3.3 Successful adaptation

After a successful workflow task execution, i.e. when no error caused a premature end of runtime, the value of the adaptive parameter shall not be altered further. The

workflow should then continue normally and the successful value be logged. For that, the `checkTaskStatus` method of the `TaskPollingMonitor` class was extended. This procedure is called repeatedly by Nextflow, to probe the statuses of all running processes and find execution completions(including abortions).
Additionally, to the usual task completion log message, now the value of the adaptive parameter is printed to the log, if one was present in the process and the exit status was positive(i.e., if the exit code is equal to zero, like defined in the POSIX specification[46]). For an example log, please see listing 4 in the appendix section A.

5 Validation

The aim of this chapter is to describe experiments, that are able to validate, whether the implementation described in the preceding chapter can fulfil the requirements of the thesis-goal. First it is explained, how the implementation can be executed. Afterwards, experimental setups are outlined, that shall show the feasibility and usefulness of adaptive parameters.

5.1 Reproducibility

The experiments conducted within the thesis, where done on university machines(`Gruenau 2`), with an environment outlined in Table 4. The commit hash of the offical Nextflow Git repository, also stated in the table, is the initial code base that has been altered. The applied code changes described in the implementation chapter can be compiled with the command `make compile` inside the repository. After successful compilation, the program then can be started with `./launch.sh run <script_name>.nf` .

System	Intel(R) Xeon(R) Gold 6254 CPU @ 3.10GHz
OS	openSUSE Leap 15.3
	(Linux version 5.3.18-150300.59.106-preemp)
JDK	openjdk version ”11.0.17”
Master Commit Hash	6307f9b5
Nextflow version	version 23.06.0-edge
Date	2023-10-24

Table 4: Execution environment of Nextflow for validation.

5.2 Feasibility

The first research question of the thesis addresses the feasibility of adaptive task parameters for workflows. In order to proof the feasibility of an implementation of these, we will execute the developed Nextflow extension with a simple exemplary workflow. In the appendix section A is a listing with an appropriate Nextflow script that we developed for that. This toy example should also illustrate the use and functioning of adaptive parameters. In the upper part of the script, for every supported data type, a simple process is defined, that accepts one parameter and executes only without an error, if certain values are passed to it. The workflow can be set to execute one of these processes via command line argument. This results in three possible experimental setups, that should cover the key use cases of adaptive parameters.

5.2.1 Integer Adaptive Parameter

In this setup, the process only successfully executes, if the integer value 5 is passed to its parameter. We created an integer adaptive parameter for an interval between 0 and

10. The search strategy “random” will test all integers between the bounds in random order. But since all are tested until the task executes successfully, the *correct* one(5) should also be used. Thus, it is anticipated, that the process will be executed at least one, but probably multiple times, and that the adaptation algorithm will eventually find the correct value and end the workflow. Since bringing 10 values in a specific order, resembles a permutation, there are 10! possible sequences to test the values. And as the value order used by the adaptation algorithm is created by randomness, it is not feasible to test until all sequences were used. We chose, to execute the workflow 30 times, to establish a high probability, that the implementation works as expected and at the same time, don’t have an excessively long runtime. This experiment should validate, if the implementation can handle the positive scenario, where a user picks bounds, that include the correct parameter value.

5.2.2 Float Adaptive Parameter

In the second setup, only the floating point number 20 passed as a value to the parameter of the process will cause a successful execution. All other values let the process exit with an error code.
To validate, if the implementation can also handle a negative scenario, we created an adaptive parameter for floating point numbers with bounds, that not include the wanted correct parameter value. We chose a range from 0 to 5 with the “maxSteps” setting set to 10. This means, that the algorithm should test the process with the ten values from inside the bounds, that are equidistantly distributed.
Further, we set the search strategy to `sorted` and the `startAt` value to 1. With that, the ten values, that the adaptation algorithm can use, should be in an order, where the `startAt` value 1 is used first, and all other values in descending order, according to the numerical distance from it. The workflow should not finish successfully, but result in a termination due to the fact, that no appropriate parameter could be found.

5.2.3 Categorical Adaptive Parameter

In the final experiment to validate feasibility, the process will only finish its execution successfully, if the value passed to it is a string, with the content “false”. To cover all implemented parameter space types, we use the categorical adaptive parameter to test, if we can find a correct value for the process parameter. The three values 0(as an integer), *null*(as a groovy object literal) and the string “false” are used as the categories to test. The adaptive algorithm should test the process with the values in the same order, as they were passed to the adaptive parameter. Thereby, we anticipate a successfully executed workflow, with the correct parameter value found.

5.3 Usefulness

The second thesis research question addresses, if an implementation of adaptive task parameters, can be useful for researchers developing workflows, in the sense, that it saves time during the creation and use of scientific workflows. To validate, if the implementation reduces the time of selecting parameters for tasks, an exemplary popular workflow, which could be used in real world scenarios, is executed. The workflow covers the typical bioinformatics task of local sequence alignment with the BLAST program. It is such a well-known use of workflows, that it is presented as one of the six main examples on the landing page of the Nextflow website[47]. The workflow utilizes the `blastp` program of the NCBI BLAST software suite[1]. Its purpose is to find similar regions in protein sequences. The workflow uses it, to match the user provided protein sequences against a chosen protein database. Both, exemplary sequences to query and a database, are provided in the Github repository of the example, which is linked in the cited Nextflow site. We will use these for our experiment as well.

The workflow proceeds as follows: For every provided protein sequence from the input parameter, `blastp` queries the database and returns the top search hits as a table in a file. This file is then processed, to only include specific rows and columns. For every `blastp` invocation, the result file with the top hits is then also passed to the `extract` process, that then prints the matching sequence.

In the context of this validation experiment, we want to assume the role of a bioinformatics researcher that wants to extend this example for his own usage. The workflow shall be enhanced, so that the `blastp` tool also uses the e-value parameter. This is an alignment setting, that describes the "number of expected hits of similar quality (score) that could be found just by chance" where a smaller e-value, resembles better matches. Hereby, $1e - 50$ is considered of very good quality and 0.01 as still good[48]. In the assumed scenario, we limit the number of top hits to at least 5 and otherwise exit the `blast` process with an error. We want to find the smallest possible e-value, that still produces enough hits, so that the task still successfully completes.

5.3.1 Manual Parameter Selection

In the first experimental setup, the workflow gets executed repeatedly, with the e-value as a task parameter being set manually each time. For the scenario, of course it would of course be possible, to find a correct parameter value by luck. Thus, we will choose the e-values in the same order, as the implemented adaptive algorithm will do it with the settings described in the next section, to measure the differences in run times. To capture the complete workflow execution time of Nextflow, we use the Linux `time` command, when starting the script.

5.3.2 Adaptive Parameter Selection

In the second experiment, we execute an adjusted workflow, that uses an adaptive parameter. The respective script can be found in the appendix section B. We created an adaptive parameter for the `blast` process, that should approach the correct e-value

[1]https://blast.ncbi.nlm.nih.gov/blast/Blast.cgi

starting at $1e - 50$ in a maximum of 100 steps. We use the “sorted” search strategy to bring the e-values to check in ascending order. The upper bound is $1e - 35$.

6 Evaluation

In this chapter, the results of the experiments are evaluated.

6.1 Toy Example

First, we will describe the outcome, of the experiments testing the functioning of the implementation.

6.1.1 Integer Parameter

In the integer experiment, the workflow was executed multiple times because the result is influenced by randomness. All 30 runs of the workflow resulted in a return value of 0. This means, that in every workflow execution the correct parameter value(in this case, the integer 5) could be found, since Nextflow otherwise returns an error code 1 if an unhandled exception is thrown during runtime. If no parameter value for a successful task execution can be found, such an exception is thrown.
If we look at the `STDOUT` output log(appended in Appendix section A), we can see, that the implemented Nextflow extension, noticed the adaptive parameter creation during the workflow script parsing(line 4). Due to the way, how Nextflow is printing progress updates to `STDOUT`, log messages can sometimes be in confusing order. But it is still visible, that after the creation of the adaptive parameter, the task is started and its execution fails three times. The value of the adaptive parameter is automatically adjusted four times(including the initial setup and the final one, with the correct value), in the order: $1, 8, 4, 5$. Finally, a log message(line 14) presents the found correct value, that allowed complete and successful execution, which is 5. The process termination message states the exit code 255(e.g. line 15), which is the unsigned 8-bit integer value, of the `exit -1` command, used in the bash script of the tasks process.

6.1.2 Float Parameter

The experiment to validate the use of adaptive floating point number parameters, results in an uncompleted workflow execution. Again, at the beginning during the workflow script parsing, the adaptive parameter is found and setup. Matching the settings of the adaptive parameter, a maximum of 10 execution steps(and the inherent discretization into 10 possible parameter values to try out) are logged. But all process executions result in an error. The tried out values for the parameter and the erroneous process runs are reported. Finally, a warning is written to the log, that states, that the adaptive parameter was not able to find a correct value inside the user provided bounds. Afterwards Nextflow shuts down gracefully.

6.1.3 Categorical Parameter

Analogous to the preceding two experiments, the workflow with the categorical parameter finds the adaptive parameter declaration and logs a successful setup of it. Again, the task gets executed repeatedly, since the parameter value gets adjusted adaptively as long as the task execution result in errors. The values provided as categories are tried out in the same order, as they were given in the adaptive parameter creation. The last value, leads to a successful task execution, which ends the workflow.

6.2 Real World Workflow

For the real world experiment, the blast workflow is first executed with fixed e-values.

6.2.1 Manual Parameter Selection

The e-value parameter was hard coded in the workflow script, and Nextflow started. According to the result of the workflow execution, the parameter was adjusted and the workflow rerun. In table 5 the results and runtime of each configuration are listed. It also displays the order, in which the parameter values were used. Summarized, the first six chosen parameter values lead to an unsuccessful task and workflow execution and the last to a positive completion. In section B of the appendix, the output of one

e-Value	**runtime**	**task result**
1e-50	0m5,528s	error
1.0101010101011092E-37	0m5,271s	error
2.020202020202118E-37	0m4,798s	error
3.030303030303127E-37	0m6,235s	error
4.040404040404137E-37	0m5,309s	error
5.050505050505146E-37	0m5,234s	error
6.060606060606155E-37	0m5,791s	success
$\sum$	0m38,166s	

Table 5: List of manual started Nextflow runs including runtime, using an adjusted e-Value parameter in the script for each run.

of the unsuccessful executions can be seen. It is also exemplary for the erroneous runs, since their output did not differ much.
After the process of the task is started(run hash “bc/32add5”), an execution failure is recorded and also the cause: a negative exit code, logged. This corresponds to the insufficient number of alignment hits, that the `blastp` program returned with the chosen e-value, as it is less than 5 hits.

In the appendix also a log of the successful run can be found. No anomaly happens during execution and both the `blast` and `extract` process complete without error. Before the workflow ends, the matching sequences are printed.

6.2.2 Adaptive Parameter Selection

After a viable e-value was found using the manual method, the workflow was run with the adaptive parameter.
The experiment went as follows: The adaptive parameter creation and its settings is logged after the workflow setup. The `blast` process is executed 7 times(6 retries), and the chosen e-values from the adaptation mechanism can also be seen in the output. In the moment, the `blast` completes without error, the number of hits during alignment with the required e-value is reached. The workflow can continue and the `extract` process receives the top hits from the `blast` task. This process also finishes successful and the matching sequences are printed. The complete run of Nextflow, from launch to exit was $10,773s$.

7 Discussion

7.1 Interpretation of the Results

As the executed toy example workflow run without problems and as expected in the experiments, we think that it is sufficient, that the developed extension of Nextflow validates the pursued feasibility. The used experimental setups of the toy example did not contain all possible settings of the developed implementation. As such, the experiments did not fulfil a complete testing of all possible execution paths. But we think, that the chosen settings represent all common use cases of adaptive parameters, that were addressed by the thesis goal. In addition to that, the pre-existing functionality of Nextflow was not affected, since for example, the `extract` process of the experiment from chapter 5.3 executed normally and no other impairment during runtime was seen. The implemented search strategies worked as expected, and it is in the responsibility of the user, to choose a strategy, that is appropriate to his use case.
We were further able to insert an adaptive parameter into a real world BLAST workflow, like a user would apply it in a realistic scenario. We executed this experiment on university servers via `SSH`, which is also a fitting environment, since this kind of workflow is often used by bioinformatics researchers that could use scientific workflow software on similar hardware. Using the adaptive parameter, a correct value for the executed tool of the task could successfully be found. Since someone without our implemented feature would have to start the complete workflow repeatedly, we showed, that it is faster to use adaptive parameters and thus the pursued thesis goal was reached. In our case, the adaptive parameter setting was more than 3 times faster. We think that on much larger workflows, e.g. *rnaseq* from the nf-core repository, the impact and usefulness would be even higher! But of course, these differences in time are also bound to luck and intuition of the workflow developer. As such, the advantage in using adaptive parameters, depends on the selection of bounds, in contrast to the chosen concrete values if done manually. Summarized, our implementation cannot make the selection of parameters in workflows easier, since the user still has to conceive the adaptation bounds. But it can make the parameter value selection significantly faster, since with adaptive parameters the overhead of restarting the workflow is saved.

7.2 Future work

Adaptive Parameter Creation

We see it as a success, that the implementation makes it possible to make a parameter adaptive with just one added instruction to a Nextflow script. We concentrated on choosing intuitive names for the adaptation settings, like search strategy and bounds. Although we were satisfied with the result of the implementation, the thesis did not focus on a pleasant user experience in particular. We chose a simple input method, that did not require a modification of the DSL parsing. More sophisticated definitions of adaptive parameters and their parameter spaces should be possible. Our implementation only supported intervals as a method to record the user knowledge about a parameter space, but maybe other kinds of sets could also be useful. We think, potential future implementations could enhance on this. If the implementation of the thesis should be made publica, it maybe would be beneficial to realize it as a Nextflow Plugin, to offer an easy installation. In addition to that, it has to be discussed, which default values would support users of adaptive parameters the most.

Adaptation Algorithm

As mentioned in the thesis scope, only single parameter values of tasks were a goal. We think, that supporting multiple parameter values could be an useful extension in the future, that would require more research and planing. The complexity of search strategies would grow, as the number of combinations of parameter values would rise quickly. But often tool parameters can influence each other and as such, a support of the user with adaptive parameter could be useful.

Also, the `ADAPT errorStrategy` that we developed, allows our implementation to work, but also endless retries. In combination with an inappropriate process script, this can result in an infinite loop and resources hogging.

Meta analysis

Something this thesis does not research is, if there are performance differences between the different search strategies. Also, a lot of possible future research could be focused on the analysing, if adaptive parameters can compete with the other parameter selection methods referenced in chapter 3.

8 Summary

In this thesis, we were able to show, that it is feasible to implement adaptive task parameters for scientific workflows. We extended the code base of the Nextflow workflow engine and showed on various examples, that the new version was capable of executing workflows as usual, but also that it selected values for task parameters automatically. The created algorithm hereby harnessed a user supplied value range and found correct values within it, that allowed successful task execution. If the correct value is not within these bounds, the algorithm notified the user accordingly. The implementation is compatible to be used with value parameters passed in channels. Only a single parameter of a task can be adaptive, but multiple processes can have an adaptive parameter in a workflow script.
We evaluated, that on a common bioinformatics workflow, the implementation could adapt a task parameter to a correct value faster than a human could, in our case up to 3 times. Thus, we conclude, that adaptive parameters can be an improvement to the conventional workflow development process. In the day-to-day research process, the typical manual parameter selection is a tedious time-consuming task. Researchers that have an idea of a feasible value range, can evaluate this faster now using the implementation.

References

[1] Shiyong Lu and Jia Zhang. "Collaborative Scientific Workflows". In: *2009 IEEE International Conference on Web Services*. 2009, pp. 527–534. DOI: 10.1109/ICWS.2009.150.

[2] Marcin Płóciennik, Sandro Fiore, Giacinto Donvito, Michał Owsiak, Marco Fargetta, et al. "Two-level Dynamic Workflow Orchestration in the INDIGO DataCloud for Large-scale, Climate Change Data Analytics Experiments". In: *Procedia Computer Science* 80 (2016). International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA, pp. 722–733. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2016.05.359. URL: https://www.sciencedirect.com/science/article/pii/S1877050916308341.

[3] Gordon Bell, Tony Hey, and Alex Szalay. "Beyond the Data Deluge". In: *Science* 323.5919 (2009), pp. 1297–1298. DOI: 10.1126/science.1170411. eprint: https://www.science.org/doi/pdf/10.1126/science.1170411. URL: https://www.science.org/doi/abs/10.1126/science.1170411.

[4] Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields. *Workflows for e-science : scientific workflows for grids / Ian J. Taylor ... (ed.)* eng. London: Springer-Verlag London Limited, 2007. ISBN: 1846285194.

[5] Azita Nouri, Philip E. Davis, Pradeep Subedi, and Manish Parashar. "Exploring the Role of Machine Learning in Scientific Workflows: Opportunities and Challenges". In: *CoRR* abs/2110.13999 (2021). arXiv: 2110.13999. URL: https://arxiv.org/abs/2110.13999.

[6] Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D Carothers, Kerstin Kleese van Dam, et al. "The Future of Scientific Workflows". In: *Int. J. High Perform. Comput. Appl.* 32.1 (Jan. 2018), pp. 159–175. ISSN: 1094-3420.

[7] Rafael Ferreira da Silva, Henri Casanova, Kyle Chard, Ilkay Altintas, Rosa M Badia, et al. "A Community Roadmap for Scientific Workflows Research and Development". In: *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*. 2021, pp. 81–90. DOI: 10.1109/WORKS54523.2021.00016.

[8] Ulf Leser, Marcus Hilbrich, Claudia Draxl, Peter Eisert, Lars Grunske, et al. "The Collaborative Research Center FONDA". In: *Datenbank-Spektrum* 21.3 (Nov. 1, 2021), pp. 255–260. ISSN: 1610-1995. DOI: 10.1007/s13222-021-00397-5. URL: https://doi.org/10.1007/s13222-021-00397-5.

[9] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. "Workflows and e-Science: An overview of workflow system features and capabilities". In: *Future Generation Computer Systems* 25.5 (2009), pp. 528–540. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2008.06.012. URL: https://www.sciencedirect.com/science/article/pii/S0167739X08000861.

[10] Anubhav Jain, Shyue Ping Ong, Wei Chen, Bharat Medasani, Xiaohui Qu, et al. "FireWorks: a dynamic workflow system designed for high-throughput applications". In: *Concurrency and Computation: Practice and Experience* 27.17 (2015), pp. 5037–5059. DOI: https://doi.org/10.1002/cpe.3505. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.3505. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3505.

[11] Martin Fowler. *Domain Specific Languages.* 1st. Addison-Wesley Professional, 2010. ISBN: 0321712943.

[12] Apache Groovy project. *Domain-Specific Languages.* Accessed: 2023-10-22. URL: http://docs.groovy-lang.org/docs/latest/html/documentation/core-domain-specific-languages.html.

[13] S.L Seqera Labs. *Nextflow Basic concepts.* URL: https://www.nextflow.io/docs/latest/basic.html?highlight=groovy.

[14] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. "A Survey of Data-Intensive Scientific Workflow Management". In: *Journal of Grid Computing* 13.4 (Dec. 1, 2015), pp. 457–493. ISSN: 1572-9184. DOI: 10.1007/s10723-015-9329-8. URL: https://doi.org/10.1007/s10723-015-9329-8.

[15] Deepak Poola, Mohsen Amini Salehi, Kotagiri Ramamohanarao, and Rajkumar Buyya. "Chapter 15 - A Taxonomy and Survey of Fault-Tolerant Workflow Management Systems in Cloud and Distributed Computing Environments". In: *Software Architecture for Big Data and the Cloud.* Ed. by Ivan Mistrik, Rami Bahsoon, Nour Ali, Maritta Heisel, and Bruce Maxim. Boston: Morgan Kaufmann, 2017, pp. 285–320. ISBN: 978-0-12-805467-3. DOI: https://doi.org/10.1016/B978-0-12-805467-3.00015-6. URL: https://www.sciencedirect.com/science/article/pii/B9780128054673000156.

[16] Rina Singh, Jeffrey A. Graves, Valentine Anantharaj, and Sreenivas R. Sukumar. "Evaluating Scientific Workflow Engines for Data and Compute Intensive Discoveries". In: *2019 IEEE International Conference on Big Data (Big Data).* 2019, pp. 4553–4560. DOI: 10.1109/BigData47090.2019.9006223.

[17] The Galaxy Community. "The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2022 update". In: *Nucleic Acids Research* 50.W1 (Apr. 2022), W345–W351. ISSN: 0305-1048. DOI: 10.1093/nar/gkac247. eprint: https://academic.oup.com/nar/article-pdf/50/W1/W345/45189566/gkac247.pdf. URL: https://doi.org/10.1093/nar/gkac247.

[18] Helena Rasche Marius van den Beek. *Using Workflow Parameters (Galaxy Training Materials).* URL: https://training.galaxyproject.org/training-material/topics/galaxy-interface/tutorials/workflow-parameters/tutorial.html.

[19] Daniel Garijo, Pinar Alper, Khalid Belhajjame, Oscar Corcho, Yolanda Gil, et al. "Common motifs in scientific workflows: An empirical analysis". In: *Future Generation Computer Systems* 36 (2014). Special Section: Intelligent Big Data Processing Special Section: Behavior Data Security Issues in Network Information Propagation Special Section: Energy-efficiency in Large Distributed Computing Architectures Special Section: eScience Infrastructure and Applications, pp. 338–351. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2013.09.018. URL: https://www.sciencedirect.com/science/article/pii/S0167739X13001970.

[20] Ewa Deelman, Karan Vahi, Mats Rynge, Rajiv Mayani, Rafael Ferreira da Silva, et al. "The Evolution of the Pegasus Workflow Management Software". In: *Computing in Science I& Engineering* 21.4 (2019), pp. 22–36. DOI: 10.1109/MCSE.2019.2919690.

[21] Pegasus Team. *Introduction*. URL: https://pegasus.isi.edu/documentation/user-guide/introduction.html?highlight=arguments.

[22] Pegasus Team. *Hooks*. URL: https://pegasus.isi.edu/documentation/user-guide/monitoring-debugging-stats.html#hooks.

[23] Pegasus Team. *Variable Expansion*. URL: https://pegasus.isi.edu/documentation/reference-guide/variable-expansion.html.

[24] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, et al. "Kepler: an extensible system for design and execution of scientific workflows". In: *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004*. 2004, pp. 423–424. DOI: 10.1109/SSDM.2004.1311241.

[25] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, et al. "Scientific workflow management and the Kepler system". In: *Concurrency and Computation: Practice and Experience* 18.10 (2006), pp. 1039–1065. DOI: https://doi.org/10.1002/cpe.994. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.994. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.994.

[26] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, et al. "Nextflow enables reproducible computational workflows". In: *Nature Biotechnology* 35.4 (Apr. 1, 2017), pp. 316–319. ISSN: 1546-1696. DOI: 10.1038/nbt.3820. URL: https://doi.org/10.1038/nbt.3820.

[27] Marta Mattoso, Jonas Dias, Kary A.C.S. Ocaña, Eduardo Ogasawara, Flavio Costa, et al. "Dynamic steering of HPC scientific workflows: A survey". In: *Future Generation Computer Systems* 46 (2015), pp. 100–113. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2014.11.017. URL: https://www.sciencedirect.com/science/article/pii/S0167739X14002519.

[28] Evan Floden. *Demystifying Nextflow resume*. Accessed: 2023-06-13. June 2019. URL: https://www.nextflow.io/blog/2019/demystifying-nextflow-resume.html.

[29] *Nextflow processes.* Accessed: 2023-06-13. URL: https://www.nextflow.io/docs/latest/process.html.

[30] *Easy provenance reporting.* Accessed: 2023-06-13. URL: https://www.nextflow.io/blog/2019/easy-provenance-report.html.

[31] Georgia Kougka, Anastasios Gounaris, and Alkis Simitsis. "The many faces of data-centric workflow optimization: a survey". In: *International Journal of Data Science and Analytics* 6.2 (Sept. 1, 2018), pp. 81–107. ISSN: 2364-4168. DOI: 10.1007/s41060-018-0107-0. URL: https://doi.org/10.1007/s41060-018-0107-0.

[32] Douglas de Oliveira, Fábio Porto, Cristina Boeres, and Daniel de Oliveira. "Towards optimizing the execution of spark scientific workflows using machine learning-based parameter tuning". In: *Concurrency and Computation: Practice and Experience* 33.5 (2021), e5972. DOI: https://doi.org/10.1002/cpe.5972. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5972. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5972.

[33] *Bash documentation: Programmable completion.* Accessed: 2023-06-13. URL: https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html#Programmable-Completion.

[34] *ZSH offers auto completion of cli commands.* Accessed: 2023-06-13. URL: https://github.com/zsh-users/zsh-autosuggestions.

[35] *Fig.io offers a opensource database for CLI autocompletions.* Accessed: 2023-06-13. URL: https://github.com/withfig/autocomplete.

[36] *Fig.io offers ai support for automatic command createion.* Accessed: 2023-06-13. URL: https://fig.io/user-manual/ai.

[37] Philip A. Ewels, Alexander Peltzer, Sven Fillinger, Harshil Patel, Johannes Alneberg, et al. "The nf-core framework for community-curated bioinformatics pipelines". In: *Nature Biotechnology* 38.3 (Mar. 1, 2020), pp. 276–278. ISSN: 1546-1696. DOI: 10.1038/s41587-020-0439-x. URL: https://doi.org/10.1038/s41587-020-0439-x.

[38] *Pipeline configuration for nf-core.* Accessed: 2023-06-13. URL: https://nf-co.re/usage/configuration.

[39] *Introducing Tower Datasets.* Accessed: 2023-06-13. URL: https://seqera.io/blog/introducing-tower-datasets/#versionable-input-data-sources-for-nextflow-pipelines.

[40] *Nf-core schema file.* Accessed: 2023-06-13. URL: https://github.com/nf-core/demultiplex/blob/master/nextflow%5C_schema.json.

[41] Bruno Silva, Marco A.S. Netto, and Renato L.F. Cunha. "JobPruner: A machine learning assistant for exploring parameter spaces in HPC applications". In: *Future Generation Computer Systems* 83 (2018), pp. 144–157. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2018.02.002. URL: https://www.sciencedirect.com/science/article/pii/S0167739X17323294.

[42] *Nextflow Github Repository*. Aug. 2023. URL: https://github.com/nextflow-io/nextflow.

[43] Fransiskus Panca Juniawan, Harrizki Arie Pradana, Laurentinus, and Dwi Yuny Sylfania. "Performance Comparison of Linear Congruent Method and Fisher-Yates Shuffle for Data Randomization". In: *Journal of Physics: Conference Series* 1196.1 (Mar. 1, 2019), p. 012035. ISSN: 1742-6596; 1742-6588. DOI: 10.1088/1742-6596/1196/1/012035. URL: https://dx.doi.org/10.1088/1742-6596/1196/1/012035.

[44] *Nextflow processess directives*. Accessed: 2023-10-29. URL: https://www.nextflow.io/docs/latest/process.html#directives.

[45] *Nextflow processess maxErrors directive*. Accessed: 2023-10-29. URL: https://www.nextflow.io/docs/latest/process.html#maxerrors.

[46] IEEE and The Open Group. *The Open Group Base Specifications Issue 7, 2018 edition*. Accessed: 2023-10-24. URL: https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/stdlib.h.html.

[47] *BLAST pipeline*. Accessed: 2023-10-24. URL: https://www.nextflow.io/example3.html.

[48] *E-value I& Bit-score*. Accessed: 2023-10-24. URL: https://www.metagenomics.wiki/tools/blast/evalue.

Appendix

All listings and log outputs can also be found on GitLab: https://gitlab.informatik.hu-berlin.de/freudeto/bachelor-thesis.git

Appendix Section A

Toy Example Listing

```
process intProcess {
        input:
        val x
        output:
        stdout
        ”””
        if [ ${x} −ne 5 ]; then
        exit −1
        fi
        ”””
}

process floatProcess {
        input:
        val x
        output:
        stdout
        ”””
        bcResult=‘echo ”$x != 20.0” | bc‘
        if [ \$bcResult −eq 1 ]; then
        exit −1
        fi
        ”””
}

process stringProcess {
        input:
        val x
        output:
        stdout

        ”””
        CORRECT_VALUE=”false”

        if [[ ”${x}” != \$CORRECT_VALUE ]]; then
        exit −1
        fi
        ”””
}

workflow {
        if (params.experiment == 1) {
                adaptparam = adaptparam_int(forProcess=”intProcess”, from=0, to=10,
                     withSearchStrategy=”random”, mean=0)
                res = intProcess(adaptparam)
                res | view {it}
        }
        else if (params.experiment == 2) {
                adaptParam = adaptparam_float(forProcess=”floatProcess”, from=0, to=5, maxSteps=10,
                      withSearchStrategy=”sorted”, mean=1)
                res = floatProcess(adaptParam)
                res | view {it}
        }
        else if (params.experiment == 3) {
                adaptParam = adaptparam_categorical(forProcess=”stringProcess”, categories=[0, null
                     , ”false”])
                res = stringProcess(adaptParam)
                res | view {it}
        }
        else {
                print ”Please choose experiment”
        }
}
```

Listing 2: Nextflow script of the toy example

Start Script for Toy Example Experiment 1 (Integer Parameter)

```
#!/bin/bash
for I in {1..30}
do
    ./launch.sh run workflows/toyexample.nf --experiment 1
    echo "Experiment run # $I with result $?"
done
```

Listing 3: Start script for toy example experiment 1 (integer parameter)

Toy Example Output Experiment 1 (Integer Parameter)

```
$ time ./launch.sh workflows/toyexample.nf --experiment 1
N E X T F L O W  ~  version 23.06.0-edge
Launching `workflows/toyexample.nf` [golden_sanger] DSL2 - revision: b9fb0ee22e
Created adaptive parameter(int) for process intProcess testing from 0 to 10 starting at 0
executor >  local (4)
[ea/8909de] process > intProcess [ 75%] 3 of 4, failed: 3, retries: 3
Try adaptive parameter adaptParam0 with value 1
executor >  local (4)
[ea/8909de] process > intProcess [100%] 4 of 4, failed: 3, retries: 3
Try adaptive parameter adaptParam0 with value 8
Try adaptive parameter adaptParam0 with value 4
Try adaptive parameter adaptParam0 with value 5

Adaptive parameter found correct value: adaptParam0 with value 5
[6e/44e299] NOTE: Process `intProcess` terminated with an error exit status (255) -- Execution is
    retried (1)
[42/f4913f] NOTE: Process `intProcess` terminated with an error exit status (255) -- Execution is
    retried (2)
[95/7a2ead] NOTE: Process `intProcess` terminated with an error exit status (255) -- Execution is
    retried (3)


real    0m4,633s
user    0m8,099s
sys     0m1,661s
```

Listing 4: STDOUT of toy example experiment 1 (integer parameter)

Toy Example Output Experiment 2 (Float Parameter)

```
$ time ./launch.sh workflows/toyexample.nf --experiment 2
N E X T F L O W  ~  version 23.06.0-edge
Launching `workflows/toyexample.nf` [maniac_meucci] DSL2 - revision: 345429651e
Created adaptive parameter(float) for process floatProcess testing from 0.0 to 5.0 in 10 steps
    starting at 1.0
[a4/ab2286] process > floatProcess [100%] 11 of 11, failed: 11, retries: 10
Try adaptive parameter adaptParam0 with value 1.1111111111111112
Try adaptive parameter adaptParam0 with value 0.5555555555555556
Try adaptive parameter adaptParam0 with value 1.6666666666666667
Try adaptive parameter adaptParam0 with value 0.0
Try adaptive parameter adaptParam0 with value 2.2222222222222223
Try adaptive parameter adaptParam0 with value 2.7777777777777777
Try adaptive parameter adaptParam0 with value 3.3333333333333335
Try adaptive parameter adaptParam0 with value 3.8888888888888893
Try adaptive parameter adaptParam0 with value 4.444444444444445
Try adaptive parameter adaptParam0 with value 5.0
[a0/d047c2] NOTE: Process `floatProcess` terminated with an error exit status (255) -- Execution is
    retried (1)
[6a/4ee7a6] NOTE: Process `floatProcess` terminated with an error exit status (255) -- Execution is
    retried (2)
[34/ccc9d9] NOTE: Process `floatProcess` terminated with an error exit status (255) -- Execution is
    retried (3)
[07/9bc64c] NOTE: Process `floatProcess` terminated with an error exit status (255) -- Execution is
    retried (4)
[0c/cf4ab2] NOTE: Process `floatProcess` terminated with an error exit status (255) -- Execution is
    retried (5)
[89/17d737] NOTE: Process `floatProcess` terminated with an error exit status (255) -- Execution is
    retried (6)
[07/60c45c] NOTE: Process `floatProcess` terminated with an error exit status (255) -- Execution is
    retried (7)
[e0/1dc7ed] NOTE: Process `floatProcess` terminated with an error exit status (255) -- Execution is
    retried (8)
[e8/30b48b] NOTE: Process `floatProcess` terminated with an error exit status (255) -- Execution is
    retried (9)
[6b/1ba36b] NOTE: Process `floatProcess` terminated with an error exit status (255) -- Execution is
    retried (10)
WARN: No value for adaptive parameter found, that can execute process successful. Throwing error...
```

```
ERROR ~ Error executing process > 'floatProcess'

Caused by:
Index 10 out of bounds for length 10


-- Check '.nextflow.log' file for details


real    0m4,879s
user    0m7,573s
sys     0m1,597s
```

Listing 5: STDOUT of toy example experiment 2 (float parameter)

Toy Example Output Experiment 3(Categorical Parameter)

```
 $ time ./launch.sh workflows/toyexample.nf --experiment 3
N E X T F L O W  ~  version 23.06.0-edge
Launching `workflows/toyexample.nf` [dreamy_sinoussi] DSL2 - revision: 345429651e
Created categorical adaptive parameter for process stringProcess testing [0, null, false]
executor >  local (2)
executor >  local (3)
executor >  local (3)
[99/cc7f8d] process > stringProcess [100%] 3 of 3, failed: 2, retries: 2
Try adaptive parameter adaptParam0 with value 0
Try adaptive parameter adaptParam0 with value null
Try adaptive parameter adaptParam0 with value false

Adaptive parameter found correct value: adaptParam0 with value false
[58/c00d9c] NOTE: Process `stringProcess` terminated with an error exit status (255) -- Execution
     is retried (1)
[98/cdf8f5] NOTE: Process `stringProcess` terminated with an error exit status (255) -- Execution
     is retried (2)


real    0m4,609s
user    0m7,399s
sys     0m1,135s
```

Listing 6: STDOUT of toy example experiment 3 (categorical parameter)

Appendix Section B

BLAST Workflow script

```
params.query = "$baseDir/data/sample.fa"
params.db = "$baseDir/blast-db/pdb/tiny"
params.out = "result.txt"
params.chunkSize = 100

db_name = file(params.db).name
db_dir = file(params.db).parent


workflow {
        /*
        * Create a channel emitting the given query fasta file(s).
        * Split the file into chunks containing as many sequences as defined by the parameter '
            chunkSize'.
        * Finally, assign the resulting channel to the variable 'ch_fasta'
        */
        Channel
        .fromPath(params.query)
        .splitFasta(by: params.chunkSize, file:true)
        .set { ch_fasta }

        adaptparam1 = adaptparam_float(forProcess="blast", from=1e-50, to=1e-35, maxSteps=100,
            withSearchStrategy="sorted", startAt=1e-50)

        /*
        * Execute a BLAST job for each chunk emitted by the 'ch_fasta' channel
        * and emit the resulting BLAST matches.
        */
        ch_hits = blast(ch_fasta, db_dir, adaptparam1)

        /*
        * Each time a file emitted by the 'blast' process, an extract job is executed,
        * producing a file containing the matching sequences.
        */
        ch_sequences = extract(ch_hits, db_dir)

        /*
        * Collect all the sequences files into a single file
        * and print the resulting file contents when complete.
        */
        ch_sequences
        .collectFile(name: params.out)
        .view { file -> "matching sequences:\n ${file.text}" }
}


process blast {
        input:
        path 'query.fa'
        path db
        val eValue

        output:
        path 'top_hits'

        """
        ../../../ncbi-blast-2.14.1+/blast-bin/blastp -db $db/$db_name -query query.fa -outfmt 6 -
            evalue $eValue > blast_result

        cat blast_result | head -n 10 | cut -f 2 > top_hits
        HIT_COUNT=\$(wc -l < top_hits)

        if [ \$HIT_COUNT -lt 5 ]; then
        exit -1
        fi
        """
}


process extract {
        input:
        path 'top_hits'
        path db

        output:
        path 'sequences'

        """
        ../../../ncbi-blast-2.14.1+/blast-bin/blastdbcmd -db $db/$db_name -entry_batch top_hits |
            head -n 10 > sequences
        """
}
```

Listing 7: Log output of toy example experiment 1

Log Output of BLAST Workflow with Manually Set E-Value, Unsuccessful Run

```
$ time ./launch.sh run workflows/blast.nf
N E X T F L O W  ~  version 23.06.0-edge
Launching `workflows/blast.nf` [friendly_varahamihira] DSL2 - revision: fc3ecb5ee9
executor >  local (1)
[bc/32add5] process > blast (1) [100%] 1 of 1, failed: 1
[-        ] process > extract   -
ERROR ~ Error executing process > 'blast (1)'

Caused by:
Process `blast (1)` terminated with an error exit status (255)

Command executed:

../../../ncbi-blast-2.14.1+/blast-bin/blastp -db pdb/tiny -query query.fa -outfmt 6 -evalue 1E-50 >
     blast_result

cat blast_result | head -n 10 | cut -f 2 > top_hits
HIT_COUNT=$(wc -l < top_hits)

if [ $HIT_COUNT -lt 5 ]; then
exit -1
fi

Command exit status:
255

Command output:
(empty)

Work dir:
/vol/fob-vol1/mi15/freudeto/BA/bachelor-thesis/work/bc/32add51a5861db6bef0f5c8f866bbc

Tip: view the complete command output by changing to the process work dir and entering the command
     `cat .command.out`

-- Check '.nextflow.log' file for details


real    0m5,528s
user    0m8,022s
sys     0m0,894s
```

Listing 8: STDOUT of unsucessful BLAST Workflow execution with manually set e-Value to 1E-50

Log Output of BLAST Workflow with Manually Set E-Value, Successful Run

```
$ time ./launch.sh run workflows/blast.nf
N E X T F L O W  ~  version 23.06.0-edge
Launching `workflows/blast.nf` [intergalactic_moriondo] DSL2 - revision: db6c77fa12
executor >  local (2)
[58/16878d] process > blast (1)   [100%] 1 of 1
[aa/a8dd3d] process > extract (1) [100%] 1 of 1
matching sequences:
>1ABO:B
MNDPNLFVALYDFVASGDNTLSITKGEKLRVLGYNHNGEWCEAQTKNGQGWVPSNYITPVNS
>1ABO:A
MNDPNLFVALYDFVASGDNTLSITKGEKLRVLGYNHNGEWCEAQTKNGQGWVPSNYITPVNS
>1YCS:B
PEITGQVSLPPGKRTNLRKTGSERIAHGMRVKFNPLPLALLLDSSLEGEFDLVQRIIYEVDDPSLPNDEGITALHNAVCA
GHTEIVKFLVQFGVNVNAADSDGWTPLHCAASCNNVQVCKFLVESGAAVFAMTYSDMQTAADKCEEMEEGYTQCSQFLYG
VQEKMGIMNKGVIYALWDYEPQNDDELPMKEGDCMTIIHREDEDEIEWWWARLNDKEGYVPRNLLGLYPRIKPRQRSLA
>1PHT:A
MSAEGYQYRALYDYKKEREEDIDLHLGDILTVNKGSLVALGFSDGQEARPEEIGWLNGYNETTGERGDFPGTYVEYIGRK



real    0m5,791s
user    0m7,470s
sys     0m1,232s
```

Listing 9: STDOUT of succesfully executed BLAST Workflow with manually set e-Value to 6.060606060606155E-37

Log Output of BLAST Workflow with Adaptive Parameter Execution

```
$ time ./launch.sh run workflows/blast_adapt.nf
N E X T F L O W  ~  version 23.06.0-edge
Launching `workflows/blast_adapt.nf` [compassionate_marconi] DSL2 - revision: 3eb11db2a8
Created adaptive parameter(float) for process blast testing from 1.0E-50 to 1.0E-35 in 100 steps
    starting at 1.0E-50
executor >  local (8)
[06/894327] process > blast (1)   [100%] 7 of 7, failed: 6, retries: 6
[f2/c59b42] process > extract (1) [100%] 1 of 1
Try adaptive parameter adaptParam0 with value 1.0E-50
Try adaptive parameter adaptParam0 with value 1.0101010101011092E-37
Try adaptive parameter adaptParam0 with value 2.020202020202118E-37
Try adaptive parameter adaptParam0 with value 3.030303030303127E-37
Try adaptive parameter adaptParam0 with value 4.040404040404137E-37
Try adaptive parameter adaptParam0 with value 5.050505050505146E-37
Try adaptive parameter adaptParam0 with value 6.060606060606155E-37
Adaptive parameter found correct value: adaptParam0 with value 6.060606060606155E-37
matching sequences:
>1ABO:B
MNDPNLFVALYDFVASGDNTLSITKGEKLRVLGYNHNGEWCEAQTKNGQGWVPSNYITPVNS
>1ABO:A
MNDPNLFVALYDFVASGDNTLSITKGEKLRVLGYNHNGEWCEAQTKNGQGWVPSNYITPVNS
>1YCS:B
PEITGQVSLPPGKRTNLRKTGSERIAHGMRVKFNPLPLALLLDSSLEGEFDLVQRIIYEVDDPSLPNDEGITALHNAVCA
GHTEIVKFLVQFGVNVNAADSDGWTPLHCAASCNNVQVCKFLVESGAAVFAMTYSDMQTAADKCEEMEEGYTQCSQFLYG
VQEKMGIMNKGVIYALWDYEPQNDDELPMKEGDCMTIIHREDEDEIEWWWARLNDKEGYVPRNLLGLYPRIKPRQRSLA
>1PHT:A
MSAEGYQYRALYDYKKEREEDIDLHLGDILTVNKGSLVALGFSDGQEARPEEIGWLNGYNETTGERGDFPGTYVEYIGRK

[db/fef5e0] NOTE: Process `blast (1)` terminated with an error exit status (255) -- Execution is
    retried (1)
[b4/a76868] NOTE: Process `blast (1)` terminated with an error exit status (255) -- Execution is
    retried (2)
[b7/82d27d] NOTE: Process `blast (1)` terminated with an error exit status (255) -- Execution is
    retried (3)
[0b/1b6864] NOTE: Process `blast (1)` terminated with an error exit status (255) -- Execution is
    retried (4)
[2c/c8e6f5] NOTE: Process `blast (1)` terminated with an error exit status (255) -- Execution is
    retried (5)
[6a/8ea910] NOTE: Process `blast (1)` terminated with an error exit status (255) -- Execution is
    retried (6)


real    0m10,773s
user    0m8,160s
sys     0m1,855s
```

Listing 10: STDOUT of blast workflow experiment, with adaptive paramter

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird. Berlin, den

30. Oktober 2023 [illegible]