The Computer Scientist's Guide to Galaxy

An evaluation of the Galaxy workflow system's ability to address the challenges of bioinformatics

Peter van Heusden
student number: 45764980

Friday 30th January, 2015

My abstract

Contents

List of Figures

1. Introduction and Motivation

Bioinformatics is the discipline of solving problems in biology and medicine using computational resources (databases, analysis programs, etc). Within this field biological sequence analysis (BSA)[1] covers those analyses that "infer biological information from sequence alone" [Dur98, p. 1]. As far back as 1998, Richard Durbin and his co-authors recognised that the development of widely available and accessible genome sequencing would lead to a great acceleration in the accumulation of biological knowledge. Data from the US National Human Genome Research Institute shows that between 2000 and 2012 the cost of sequencing deoxyribonucleic acid (DNA) has decreased at an exponential rate [Nat12], and this cost decline has seen a concomitant increase in the amount of genomic sequence data from which biological knowledge can be inferred [CKN10]. Raw DNA sequence data does, however, equate to biological knowledge. This data needs to be analysed, a process that combines human skill (the ability to translate biological questions into computational procedures) with computational infrastructure. With the cost of computing infrastructure steadily dropping (although not at a pace as fast as the cost of DNA sequencing drops), the human element in analysing biological data will tend to dominate the cost of producing knowledge in bioinformatics to an ever greater degree, unless innovations in the work process of BSA keep pace with the explosion in available data (and the implosion of sequencing costs).

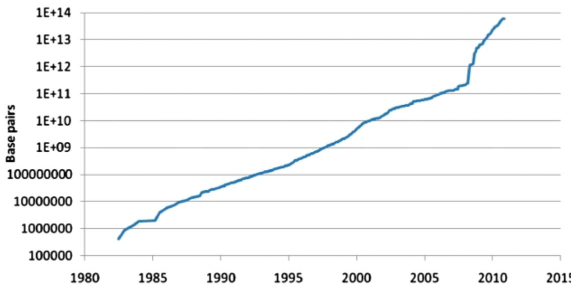


Figure 1.1.: Growth of sequence content of public databases, from [CKN10]

In addition to the problem of cost structure in bioinformatics, some experts in the field suggest that there are simply not enough bioinformaticists to keep up with the data deluge. Mabey et al [MA05] suggest that "the rate of growth of [bioinformatics], and its cross-disciplinary nature, has created a problem: while there are many trained biologists and computer scientists, there are few computer-literate biologists or biology-literate computer scientists." To put this shortage in perspective, at the South African National Bioinformatics Institute (SANBI), South Africa's largest bioinformatics centre, student enrollment grew from 8 in 2001 to 31 in 2008, an approximately

[1]While the objects of study in bioinformatics range beyond biological sequences, this research focuses on BSA since it makes up such a large part of day to day bioinformatics workloads. In Stevens et al [SGBB01]'s (albeit dated) survey of tasks in bioinformatics, 254 out of 315 activities undertaken by the respondents were clearly BSA.

fourfold increase [Sou10]. During the same period, the amount of data in public sequence databases grew approximately 100-fold. In addition, the growth in both the amount of data that requires analysis, and the computational power used to analyse such data, inevitably leads to requirements for more complex data analyses. What might once have been sequential analysis run by hand on a single server, with results stored in a researcher's folder on disk might now become parallel analysis run on a cluster or grid of computers, with results stored in some form of database. This requirement for increased sophistication in the computational side of BSA puts further pressure on bioinformaticists, many of whom have a background in life sciences and limited computer science training. [2] The sudden on reliance on computation for BSA has resulted in what some have called a *informatics crisis* for researchers in the field [GNTT10], and this in turn has spurred a search for ways to make computing more accessible for researchers.

Scientific workflow management systems have been proposed as a solution to make computing resources accessible to research scientists in their question to analyse ever larger datasets. [DGST09] A scientific workflow management system (SciWMS) is a software tool that allows for the orchestration of the workflows that describe the procedures of scientific data analysis. These procedures typically involve moving data to or from computing resources (i.e. staging), launching and managing computation, and managing the output of computation, either storing it in a data store for future reference, or passing it on to the next stage in analysis. This research will focus on one particular SciWMS: Galaxy.

Since 2009, the "Pipelines" collaboration between SANBI [Pip11] and researchers at the University of Cape Town has been investigating tools that can be used to rapidly assemble workflows for BSA.[3] In the context of this research the Galaxy platform [GRH+05] was chosen as a platform for rapid development of BSA workflows. Experience with this framework and with the broader problem of enabling non-programmers[4] to perform BSA inspired the topic of this research. The objective of this research is to evaluate Galaxy as a scientific workflow management system for biological sequence analysis.

This evaluation will proceed in two parts. In chapter 2 SciWMS will be examined in detail, to understand how they might make computation accessible to non-programmers and how their expressive power and features relate to tasks in bioinformatics. Building on the existing literature and case studies of real world bioinformatics problem solving, a set of criteria for evaluating SciWMSs will be introduced. Then, in chapter 3 Galaxy will be introduced and evaluated in terms of the criteria introduced

[2]A survey I conducted of 16 randomly selected postgraduate students at SANBI revealed that 11 (69%) of them had been life sciences students as undergrads, 3 (19%) of them came from mathematical and statistical science backgrounds, and only 2 (12%) had been exposed to computer science prior to adopting bioinformatics as a field of study.

[3]The name of the group derives from the term "sequence analysis pipeline", where pipeline is synonymous with workflow. More information about the "Pipelines" group can be found at the Pipelines website.

[4]In this context non-programmers refers to users that are neither experts in application programming languages such as Java and C++, nor are they experts in using scripting languages such as Perl and Python

in chapter 2. Features provided by and absent in Galaxy will be highlighted. Finally, conclusions will be drawn about Galaxy's strengths and weaknesses, with reference to the central problem of making the authoring and execution of analyses more efficient and accessible. Directions for possible future work will be also be highlighted.

2. Introduction to scientific workflow systems

The notion of a workflow originated in the study of business processes and the 1995 Workflow Management Coalition's reference model describes a workflow in terms of "procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal" [Hol95]. According to Abbott et al, the need for workflow management in business was driven by the challenges of coordinating work taking place on large numbers of desktop computers interconnected in a network [AS94]. Workflows in business are realised using a workflow management system that defines, manages and executes workflows on computational resources [YB05].

While the concept of workflows was first applied to business processes, there are now many SciWMSs. These aim to model and enact workflows that "[support] large-scale, complex, yet fault-tolerant and maintainable scientific processes" [YGN09b]. Castro et al describe workflows as analogous to laboratory protocols, that is allowing scientists to provide "rich descriptions of their experiments (materials and methods) so they can be easily replicated" [CTGR05]. Yildiz et al [YGN09a] note that scientific workflows are distinguished from business workflows by being"dataflow oriented": whereas business workflows focus on the set of activities that take place to enact a business process, scientific workflows focus on the flow of data between tasks in an analytical process.

If SciWMSs in general, and Galaxy in particular, are to help solve the *informatics crisis* in bioinformatics, two questions need to be answered: Firstly, can SciWMSs make computation more accessible for non-programmers? Abouelhoda suggest that SciWMSs provide an "easy-to-use metaphor that can be used for designing and executing bioinformatics applications". Empirical research on visual programming languages (VPLs)[1] done by Blackwell et al suggests that metaphor, on its own, is not a significant factor in guiding people's understanding of how to compose or understand program logic [BG99]. Instead they suggest [BWGP01] that Green et al's model of thirteen cognitive dimensions of programming environments should be used to judge the accessibility of a VPL.

In addition Blackwell et al point to the role that a large library of reuseable components play in the expressive power of programming environments[2] and also the importance of choosing a paradigm that "maps" to the user's understanding of their problem domain. McPhillips et al suggest that since scientific computation involves

[1]Most SciWMSs use some form of graph-based VPL to compose workflows [YB05].

[2]Both the Galaxy ([GRH$^+$05] and [Nek11]) and Taverna [HWS$^+$06] developers highlight the range of pre-existing components available for use in their SciWMSs

successive steps of analysis of collections of data, the dataflow paradigm[3] is appropriate for SciWMSs. They propose eight desirable characteristics that assist in the adoption of SciWMS by scientist who are not programmers. Green et al and McPhillips et al's schemas are introduced in section 2.1 and they are used to evaluate Galaxy in section 3.2.

There is a second perspective from which SciWMSs and Galaxy must be considered if they are to be useful for BSA. How can we know if these systems can efficiently express the necessary computational constructs necessary to undertake BSA? Stevens et al [SGBB01] and Castro et al [CTGR05] have provided a theoretical model of tasks encountered in bioinformatics. This model is discussed in Section 2.2.1. In order to validate this model and provide more practical understanding of the tasks undertaken in BSA, three cases studies are presented in section 2.2.2 that present real world examples of how problems in BSA have been solved. Having established an understanding of what is involved in BSA, a framework for evaluating SciWMSs will be presented. Van der Aalst et al and have identified a set of workflow patterns, where patterns are "abstraction[s] from a concrete form which [keep] recurring in specific nonarbitrary contexts" [vDATHKB03]. Yildiz et al [YGN09b] have applied van der Aalst et al's insights to the study of SciWMS. These patterns will be considered in the context of the model of computation in BSA in section 2.2.3, and the extent to which Galaxy supports these patterns will be discussed in section 3.3. In addition to the evaluation in terms of workflow patterns, Deelman et al have proposed a framework for evaluating SciWMS according to four stages of "workflow lifecycle". This schema will be introduced in section 2.2.4 and Galaxy will be examined according to this schema in section 3.4.

2.1. Evaluation the usability of scientific workflow systems

In the introduction to this research I made the point that the wealth of available sequencing resources are not match by a similar wealth of skilled bioinformatics programmers. It follows, then, that a SciWMS for bioinformatics and more particularly BSA needs to be usable by non-programmers[4]. While several authors (for example Yu et al [YB05], Altinas et al [ABJ$^+$04] and Oinn et al [OAF$^+$04]) have illustrated the expressive power of SciWMS[5], the literature on usability is of SciWMS is sparse. One

[3]The dataflow paradigm is embodied by a set of programming languages and environments where computation is expressed in terms of the flow of data between computing steps. See for example [GT79] and [LP95]. This paradigm is contrasted with "control flow", where computation is expressed in terms of the flow of control in a program. Lee et al [LP95] suggest that whereas in the control flow paradigm programming is imperative, that is, the job of the user is to tell the computer what to do at each step, the dataflow paradigm emphasises declarative design, where the user specifies inputs and outputs but does not necessarily specify the detail of how computation is executed.

[4]Usability is always a question of *who* is trying to use a system for *what*, and in thi research the *who* is bioinformaticists who are non-programmers, and the *what* is BSA.

[5]This expressive power compares favourably to the use of scripting languages for tool integration. Scripting languages such as Python and Perl are well established in bioinformatics and BSA, where

Name	**Description**
Clarity (CLR)	Make it easy to create self-explanatory workflows.
Well-Formedness (WFV)	Make it easier to design well-formed and valid workflows.
Predictablity (PRE)	Make it easy to see what a workflow will do without running it.
Recordability (REC)	Make it easy to see what a workflow did do when it ran.
Reportability (REP)	Make it easy to see if a workflow result makes sense scientifically.
Reuseability (REU)	Make it easy to design new workflows from existing ones.
Scientific Data Modelling (SDM)	Provide first-class support for modelling scientific data.
Automatic Optimisation (OPT)	Systems should take responsibility for optimising performance.

Table 2.1.: Desirable features in a SciWMS according to McPhillips et al [MBZL09]

welcome exception is McPhillips et al [MBZL09], where a set of desirable features of SciWMSs are presented. The eight SciWMS desiderata that McPhillips present are focussed on making it easier for non-programmers to create, understand and share efficient workflows, and they are summarised in table 2.1.

In addition to McPhillips et al's SciWMS desiderata, it is also useful to look at SciWMSs as programming environments. Yu et al [YB05] point out that most SciWMSs allow users to compose workflows using graphical editing of some kind. The SciWMS of interest in this research, Galaxy, presents a workflow as a directed acyclic graph (DAG) that describes the dataflow between individual workflow steps. The research done by Green et al [GP96] on the usability of visual programming languages is thus relevant to examining the usability of many SciWMSs and of Galaxy in particular. Green et al [GP96] present thirteen 'cognitive dimensions' that can be used to discuss the cognitive features of programming environments, that is, the extent to which programming environments help (or hinder) the process of a human trying to compose a correct computer program. The term 'cognitive dimensions'[6] denotes a "broad-brush analysis" of the relationship between structures of a programming environment and the "mental processing" of the environment's users. These dimensions are summarised in table 2.2, along with the SciWMS features from McPhillips et al [MBZL09] that relate to the dimensions.

As can be seen from the table, there are correspondences between seven of McPhillips

they are used for file format conversion and the execution of workflows. For an early example of advocacy for scripting languages in bioinformatics, see [Ste96].

[6]Cognitive theories deal with the working of the mind, and include models of motivation, information processing, memory, perception and so on.

et al's desired features and the cognitive dimensions proposed by Green et al. In particular:

- Clarity (CLR) refers to the degree to which a workflow is self-explanatory. Three of Green et al's dimensions relate to this feature: abstraction gradient, closeness of mapping, diffuseness, role-expressiveness and consistency.
  - An abstraction is a "grouping of elements to be treated as one entity" [GP96]. Green et al note that abstractions can increase the readability of a language, and thus increase clarity. On the other hand, if abstractions are required to specify a program, they force the user to investigate and understand these abstractions in order to understand the conceptual structure of the program[7]. The ability to move between different levels of abstraction in a programming environment leads to a *abstraction gradient*. Green et al suggest that it is useful if programming environments provide some abstractions that closely map to the user's understanding of the problem they are trying to solve, and they also suggest that an accessible abstraction gradient should be provided so that abstractions can be employed in a program in order to preserve clarity.
  - Closeness of mapping refers to the ease with which concepts in the user's "problem world" can be mapped to concepts in the "program world". Green et al note that VPLs are potential more effective at preserving this close mapping than textual programming languages, because VPLs can dispense with textual syntax (e.g. the matching braces '{' and '}' that define a block of code in the C programming language). A program specification that maps closely to the conceptual model of the workflow that it enacts enhances clarity.
  - Programming languages differ in how terse they are, in other words, how many symbols or how much space is needed to express a program. Green et al call this property diffuseness. In VPLs, diffuseness can also be impacted by the presentation capabilities of the programming environment, for example does the environment help lay out the program effectively, and is a zoom feature available? The diffuseness of a program impacts on its clarity,

[7]Green et al classify programming languages as *abstraction-hating*, *abstraction-tolerant* and *abstraction-hungry*. Abstraction-hating languages do not support abstraction. For example, flowcharts use a fixed set of elements with no support for grouping modules into abstractions. Abstraction-tolerant languages allow programming without abstractions, but permit forms of abstraction to be used in program. This group includes languages such as Perl [CfWO12], Python [Pyt12] and C [ISO11]. The starting level of abstraction in abstraction-tolerant languages varies. Some, like Python, require no abstraction: a set of bare statements is a valid program. Others, such as C, impose some minimal abstraction: in C each program requires a main function.

Finally, abstraction-hungry languages for the programmer to feed the language new abstractions. Smalltalk [GR83] and Java [Ora11] both require that programs are classes, thus requiring that the programmer at least knows how to create a class in order to create a program. Abstraction-hungry languages thus have a high starting level of abstraction that might be daunting to inexperienced programmers.

  because a program that cannot be taken in "all at once" is more difficult to understand.

  - Programs are made up out of a variety of constructs that have different roles. Clarity is enhanced by clearly denoting the role of different program elements, for example in their visual appearance or some secondary attribute like colour. This attribute of a programming environment is called "role expressiveness" by Green et al.
  - While consistency in a programming language is hard to define, for Green et al consistency is related to guessability: when a person knows part of a language's structure, how much of the rest can be successfully guessed. They state the VPLs often have good consistency in part because their syntax is simple. Consistency can also be aided by the "role expressiveness" of a language. A consistent language is also a clear language, and thus consistency in a VPL can aid in the clarity of SciWMS specifications.

- Well-Formedness (WFV) refers to features of a SciWMS that make it easy to design well-formed and valid workflows. WFV relates very closely to Green et al's "error-proneness" dimension.
  - Green et al distinguish between two types of error. A *slip* is something that you "didn't mean to do", in other words an accidental mistake. A *design error* is an error at the level of conceptualising a program: with such an error you did what you meant to do, but the program did not achieve the intended result.

    Green et al call a programming environment "error-prone" if it has features that encourage slips, and they suggest that many textual programming languages are thus error-prone. Similarly, McPhillips et al call a SciWMS well-formed if all parts of the workflow contribute to the result of a run, in other words, if there are no *slips* in the workflow specification. Both McPhillips et al and Green et al highlight the role of type declaration and checking in eliminating slips (or making a workflow well-formed), showing that the two concepts, error-proneness and well-formedness, are essentially talking about the same thing.

- Predictability (PRE) means making it easy to see what a workflow does without running it. There is some superficial similarity between this desirable feature and clarity (CLR), but the key difference between the two is that clarity relates to the specification of the workflow, typically using a VPL, and predictability relates to the enactment of a workflow. In this regard there is little direct correlation between predictability and Green et al's cognitive dimensions, except in the case of *hard mental operations*.
  - A hard mental operation is one whose effect is not easy to visualise. Green et al point out that they use this is a matter of notation, not semantics: it is not that what is being done is complex, it is that the meaning of the

notation is not easy to visualise. Secondly, a hard mental operation gets even harder to understand if two or more are combined.

In the world of SciWMSs, the use of shims can lead to hard mental operations. A shim is a workflow step that whose only purpose is to do some kind of format conversion to prepare data for analysis at a later workflow step. McPhillips et al suggest that workflow steps should describe the type fragments that operate on and produce (read and write scope) in order to clarify the semantics of workflow operation. Their aim is to allow the user to see what operation the step is performing, but the effect is also to help avoid hard mental operations.

This mapping is intended to provide a synthesis between the two approaches The final desireable feature that McPhillips et al mention, "Automatic Optimisation", does not fit within the cognitive features schema as it is a feature of the SciWMS ability to map from workflow description to executable resources. As such it is not strictly a criterion of usability, although of course the ability to hand over optimisation to the SciWMS does reduce the burden on the user. The way that these two schemata (McPhillips et al and Green et al's) over does, however, show that they can be used together, and as a result in the discussion in section 3.2 I will largely use McPhillips et al's desirable features and supplement them with relevant cognitive dimensions.

As has been mentioned before, there is unfortunately little published work on considering the usablity.

Cognitive dimension	**Description**	**Associated SciWMS features**
Abstraction gradient	What are the minimum and maximum levels of abstraction?	Clarity (CLR)
Closeness of mapping	How close is the mapping between the problem world and the program world? What 'programming games' need to be learned to map the problem into the programming environment?	Clarity (CLR), Scientific Data Modelling (SDM)
Consistency	When some of the language has been learned, how much of the rest can be inferred?	Clarity (CLR)
Diffuseness	How many symbols or graphic entities are required to express a meaning?	Clarity (CLR)
Error-proneness	Does the design of the notation induce 'careless mistakes'?	Well-Formedness (WFM)
Hard mental operations	Are there places where the use needs to resort to fingers or pencilled annotation to keep track of what is happening?	Predictablity (PRE), Recordability (REC)
Hidden dependencies	Is every dependency overtly indicated in both directions? Is the indication perceptual or only symbolic?	
Premature commitment	Do programmers have to make decisions before they have the information they need?	
Progressive evaluation	Can a partially-complete program be executed to obtain feedback on 'how am I doing'?	
Role-expressiveness	Can the reader see how each components of a program relates to the whole?	Clarity (CLR)
Secondary notation	Can programmers use layout, colour or other cues to convey extra meaning, above and beyond the 'official' semantics of the language?	
Viscosity	How much effort is required to perform a single change?	Reusability (REU)
Visibility	Is every part of the code simultaneously visible (assuming a large enough display), or is it at least possible to juxtapose any two parts side-by-side at will? If the code is dispersed, is it at least possible to know in what order to read it?	Predictability (PRE)

Table 2.2.: Green et al cognitive dimensions and McPhillips et al SciWMS desiderata

2.2. Expressing bioinformatics analyses using scientific workflow management systems

ADD STUFF

2.2.1. A model of bioinformatics tasks

In the introduction bioinformatics was defined as "the discipline of solving problems in biology and medicine using computational resources". Stevens et al [SGBB01] provide a classification of tasks in bioinformatics, a classification that they undertake to explore "the range of tasks that need to be supported and the components needed" in what they call a general query system. In the field of BSA that interests us, a query system is a system that is able to answer questions that seek to "infer biological knowledge from sequence alone". While we present case studies that illustrate sample workflows in BSA, Stevens et al's model provides a more general framework within which to discuss the capabilities of a SciWMS. The authors propose that the "general query system" can be built out of the set of six components that are listed in Table 2.3.

These components describe the flow of data (where data include sets of data objects and sets of parameters) through a series of transformations and selection operations. So for example, a sequence similarity search will take two sets of data input objects, the database and the query sequences, and a set of query parameters (for example, that the search is a protein versus protein search). The search itself is a transformation, where each query sequence is transformed into a (possibly empty) set of matches against the database. This set of matches might then be sent to a filter, that will apply a set of parameters that describe which matches are worth displaying to the end user. A key distinction between a transform and a filter is that a transform alters the type of a data object [8] whereas a filter selects a subset of a collection but does not alter the type of the data in any way. In this way the question "which of these sequences match a known sequence in the database" can be answered. Stevens et al's model is thus a dataflow model and thus it provides a fairly straightforward set of requirements for how a SciWMS for BSA should be able to address dataflow issues: support collections of data and parameters, provide for transforms and filters, and fork the dataflow to enable concurrent processing [9].

In their work on designing the GPIPE SciWMS, Castro et al [CTGR05] draw on Stevens et al's classification and enrich it with three new operators. Where Stevens et al call their model a "query system", Castro et al draw on the "task-flow" model where a set of tasks (individual analytic methods) pass state between themselves in order to enact a complex bioinformatics analysis. This model allows them to address control flow issues that are not considered in Stevens et al's schema. These three operators are iteration, recursion and suspension / resumption (see Table 2.4).

Combining Stevens et al's schema with Castro et al's control flow operators, we see that a bioinformatics SciWMSs must be able to take input collections of data and sets of parameters and feed these into transform and filter tasks. These tasks must be able to iterate across the collection, or repeat the task with different sets of parameter, or do both. There might be one or multiple outputs from a transform, and

[8] Here "type" refers to the semantics of a data object, not its syntax. Thus, for example, a tool that selected a subset of a sequence is a transform, even though both input and output are biological sequences.

[9] Stevens et al mention forks but not joins in their model. Presumably a join could simply be implemented as a transform that took multiple data sets as inputs and concatenated them to provide a single input set.

Name	Description
collections	collections of either data objects or parameters
filters	an operation that takes as input a data collection, a restriction collection (e.g. keywords to filter by) and a projection (e.g. which attributes of a data item you want returned)
transformers	an operation that takes a data collection (and optionally a collection of parameters) and transforms the items in the data collection according to a defined operation. For example, the set of parameters could describe subsequences to extract from an input set of sequences
transformer-filters	a composition of a transformer and a filter. Stevens et al. elevate this to the level of a fundamental component because this composition of tasks is encountered so often in bioinformatics
forks	unconditional or conditional forks in the analysis workflow.

Table 2.3.: Components used in addressing bioinformatics tasks, according to Stevens et al [SGBB01]

each of these outputs might require filtering. The possibility to fork dataflows or use control flow operators like iteration or recursion means that there is scope for multiple processes to be executing concurrently. Finally, the SciWMS must be able to suspend and resume workflow execution. These criteria set a "lower bound" in terms of what must a SciWMS for bioinformatics must support.

Of course, this model requires validation by examining its application to real world problems in bioinformatics. The next section will outline three case studies, and illustrate how the model drawn from Stevens et al [SGBB01] and Castro et al's [CTGR05] work is, in fact, adequate to describe the tasks undertaken in bioinformatics analysis.

2.2.2. Case studies of bioinformatics problem solving

Two case studies of real-world bioinformatics analyses will now be presented. These case studies have been chosen both because they illustrate real-world problems in the field, and because they present a variety of problems, including the need to iterate across data collections and apply different parameters to the analysis of a single data collection (i.e. explore a "parameter space"). In addition, they illustrate how bioinformatics workflows can be enacted in a scripting languages (Perl). By contrast, a third case study (presented in Section **??** TODO: provide label for Alecia's section) will use Galaxy to enact a bioinformatics workflow.

Name	Description
iteration	the application of a transformer or transformer-filter over multiple sets of inputs. A special case here is iteration with a blank transformer that simply generates a number of replicas of the input.
recursion	repetitive application of a single transform to a single collection, but with a range of parameters. This allows the exploration of a "parameter space".
suspension / resumption	the capacity to stop and re-start execution of a workflow.

Table 2.4.: Control flow operators described in [CTGR05].

Case study 1: Mining bacterial genomes for virulence genes

The first case study is based on the work of Dr Sumir Panji[10] [Pan09], and relate to the search for the genes that harmful bacteria use to produce "virulence factors", the molecules that allow bacteria to invade and live inside a host organism. Bacteria are involved in a constant "arms race" with the host organisms that they invade: as the host's immune system adapts to foreign bacteria, the bacteria in turn need to evolve to evade or suppress the immune system. Since virulence factors are manufactured in the bacteria using genes, in order for the bacteria to evolve, they need to change their genes. Drawing on earlier work, Dr Panji decided to try and detect the rate at which different genes within a single strain of bacteria were evolving, reasoning that the host versus bacteria "arms race" would put pressure on genes that encoded for virulence factors ("virulence genes") to evolve faster than the other genes. After these fast-evolving potential virulence genes were identified, they could be examined to see whether they really did play a role in increasing the virulence of the organism, and possibly they could provide clues for new medicines to be developed.

In order to search for these potential virulence genes, Dr Panji implemented an analysis workflow that identified corresponding genes contained in two bacterial genomes to one another and output a list of genes that were evolving faster than the average. This workflow was implemented as a set of Perl [CfWO12] scripts that call various bioinformatics tools, and the steps in the workflow were executed by the user manually enacting them on the command line.

Panji's workflow is schematically described in their Ph.D thesis in the form of a diagram (Figure 3.3). Further knowledge of the workflow was gained by interviewing the author directly and by examining the Perl scripts used to enact the workflow. The Perl scripts that make up this workflow are available in Appendix A1 of [Pan09]. Finally, part of the workflow was also re-implemented using the Galaxy SciWMS and some Python **??** scripts. Diagrams of this re-implemented workflow is included in

[10]Presented in his Doctoral thesis.

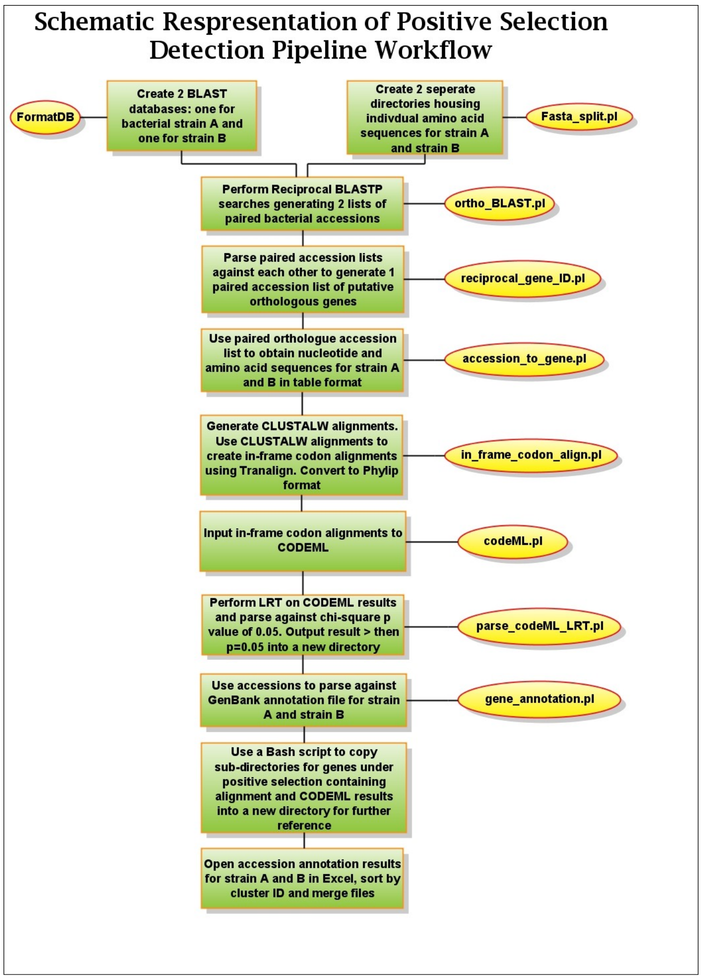

Schematic Respresentation of Positive Selection Detection Pipeline Workflow
FormatDB
Create 2 BLAST databases: one for bacterial strain A and one for strain B
Create 2 seperate directories housing indivdual amino acid sequences for strain A and strain B
Fasta_split.pl
Perform Reciprocal BLASTP searches generating 2 lists of paired bacterial accessions
ortho_BLAST.pl
Parse paired accession lists against each other to generate 1 paired accession list of putative orthologous genes
reciprocal_gene_ID.pl
Use paired orthologue accession list to obtain nucleotide and amino acid sequences for strain A and B in table format
accession_to_gene.pl
Generate CLUSTALW alignments. Use CLUSTALW alignments to create in-frame codon alignments using Tranalign. Convert to Phylip format
in_frame_codon_align.pl
Input in-frame codon alignments to CODEML
codeML.pl
Perform LRT on CODEML results and parse against chi-square p value of 0.05. Output result > then p=0.05 into a new directory
parse_codeML_LRT.pl
Use accessions to parse against GenBank annotation file for strain A and strain B
gene_annotation.pl
Use a Bash script to copy sub-directories for genes under positive selection containing alignment and CODEML results into a new directory for further reference
Open accession annotation results for strain A and B in Excel, sort by cluster ID and merge files

Figure 2.1.: Identifying positively selected genes in bacterial genomes

Appendix A.

As can be seen from Figure 3.3, Panji's pipeline consists, as per Stevens et al's schema, of a sequence of transforms and filters. For example, genomic sequence from bacterial strains A and B are transformed into BLAST format databases. These BLAST databases are used in order to find pairs of genes that are present in both strains. These pairs, known as orthologs, are the raw material that for the comparisons that occur later in the workflow. In essence, using the BLAST tool and the BLAST format database, the collection of proteins (i.e. amino acid sequences) is transformed into a set of sequence identifier (i.e. accession number) pairs that describe orthologous genes.

The workflow proceeds as a sequential set of transformations: genomic sequence is transformed into sequence ids, sequence ids are transformed into nucleotide and protein sequence pairs, the protein sequence pairs are transformed into protein to protein alignments, these are then combined with the nucleotide sequence pairs to transform these into a set of aligned nucleotide sequences, and finally a score is calculated (using the **CodeML** tool from the PAML **??** phylogenetics toolkit). The set of sequence pairs is filtered using this score to extract only those that show signs of evolving faster than average (i.e. "positive selection"). In other words, the entire workflow can be described using Stevens et al's notions of collections, transforms and filters.

Turning to Castro et al's operators, we can observe that the workflow involves multiple instances of iteration over a collection. For example the step in Dr Panji's implementation that does the BLAST search of the nucleotide sequences against from one strain against the other (implemented in the script **ortho_BLAST.pl**) involves iterating over the collection of nucleotide sequences in a strain and running BLAST for each of these. Dr Panji implemented this by calling the BLAST tool with the complete collection of query sequences as input. While BLAST does have some support for operating in parallel, it is restricted to executing on a single machine, and thus Dr Panji's implementation of this step cannot efficiently make use of a computing cluster. A SciWMS that made the iteration operation available to the user could provide some more sophisticated way of executing this step in parallel.

In conclusion, Dr Panji implemented a workflow to try and discover "virulence genes" in bacteria. This workflow was implemented using a collection of Perl scripts that were executed in sequence by the user. Examination of this workflow showed that it conformed to Stevens et al's collection, transform and filter description of bioinformatics tasks. Furthermore, Dr Panji's workflow used the iteration operation as described in Castro et al. Examination of this workflow confirms that Stevens et al and Castro et al's models do indeed describe real world bioinformatics.

Case study 2: Exploring the impact of k-mer length on assembly quality

The DNA sequence of living organisms is In the first decade of the 21st century DNA sequencing has been revolutionised by a set of technologies that are collectively known as "next generation sequencing" (NGS)[SJ08]. Like previous sequencing technologies, NGS is only able to directly sequence small fragments of DNA. In order to determine the correct sequence of a large DNA sequence, many fragments (known as "reads") are sequenced and the results are combined in a computational process that is known as assembly. Whereas older technologies produced runs of tens or hundreds of thousands of sequence fragments, NGS technologies produce millions of sequence fragments (known as "reads") in a single day. This huge growth in the number of reads that need to be assembled has led to a series of innovations in assembly algorithms. Many newer assembly approaches compute a graph that describes overlapping sequences (known as a de Bruijn graph) and then "walk" the graph in order to find areas of best overlap. [CPT11] In order to speed up assembly, assembly algorithms designed for working with NGS reads preprocess the input data by converting each read into a collection of fixed-length subsequences, known as *k-mers. A k-mer might, for example, be a string* *of 20 DNA bases. Once a k-mer size is chosen, a database is built up that notes which k-mers are present in which reads. If two reads share one or more k-mers, then we know that they have some DNA sequence in common and might be assembled together to form a longer sequence fragment.*

The output of the assembly process is a collection of longer sequence fragments, known as contigs (short for contiguous sequence). In order to compare the quality of different assembler programs or algorithms, or the impact of different assembly parameters, the contig lengths of the final output sequence set is compared. Often contig lengths are summarised into a statistical measure known as a $N50$ *score [Bur08].* [11]*.*

Stanley Mbandi-Kimbung, a doctoral student at SANBI, was approached to describe their workflow for assessing the quality of assemblies produced by the ***velvet*** *[ZB08] and* ***oases*** *[SZVB12] tools. The first of these,* ***velvet****, is an assembler that uses a de Bruijn graph based approach to compute the correct assembly of a set of genome fragments. The* ***oases*** *tool then post-processes the output of* ***velvet*** *to generate the final assembly.*

Mbandi's workflow repeatedly runs ***velvet*** *while varying the k-mer length parameter, while keeping all other input to the assembly process constant. This workflow is thus an example of a "parameter sweep", where a parameter space for a tool is explored to try and discover the optimal parameter set. Mbandi implemented this workflow in the form of a Perl script (available in Appendix B)*[12] *that operates according to the logic show in 2.2. For simplicity's sake the* ***oases*** *step has been left out since that does nothing besides post-processing the assembly produced by velvet.*

The workflow takes as input a set of sequence reads from a DNA sequencing experiment (in practice Mbandi used a set of about 10 million reads from the fungus Venturia inequalis*) and a parameter range describing a set of k-mer lengths. For each k-mer length, the* ***velveth*** *program is run to build up information on which k-mers exist and in which sequences they are found. Then the* ***velvetg*** *program is run to assemble the reads using a de Bruijn graph, and produce (amongst others) a file containing assembled contigs. Finally the contig file is analysed to produce a* $N50$ *value and the* $N50$ *value and its associated k-mer parameter are stored.*

While Mbandi's workflow iterates through the parameter set, the workflow conforms with the van der Aalst et al's "multiple instances with a priori knowledge" workflow pattern, since each the number of sub-workflows is known at the outside, and one could execute the ***velvet*** *tool for each k-mer value in parallel, resulting in the workflow as showing in Figure 2.3. An analysis of Mbandi's workflow in terms of data flow instead of control flow naturally elicits this parallelism. While enacting the workflow resource constraints should be borne in mind (as in Naidu's workflow discussed above) since even on Mbandi's relatively small set of reads the* ***velvet*** *tool uses approximately 1.5 GB of RAM during its execution.*

An attempt was made to implement this workflow in Galaxy, and in the process it was discovered that the ***VelvetOptimiser.pl*** *tool (and the corresponding tool for* ***oases****)*

[11]The $N50$ measure describes the length N for which 50% of the bases in the assembly are in contigs shorter than N

[12]Recent versions of **velvet** include a script named **VelvetOptimiser.pl** that implements a similar workflow.

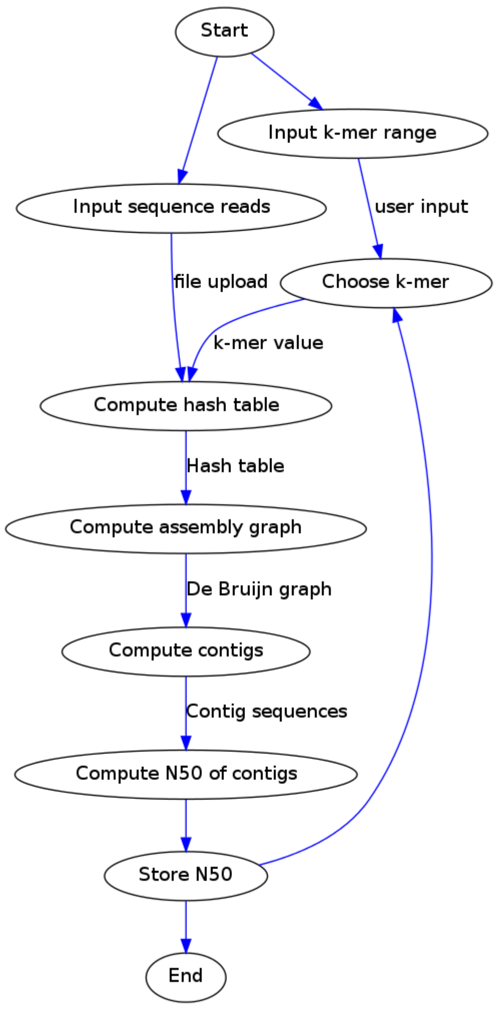


Figure 2.2.: Mbandi's parameter sweep of the velvet assembler

has been incorporated into a tool available in the Galaxy Toolshed. Galaxy itself is, however, unsuitable for implementing a parameter sweep because when a workflow is composed in Galaxy the parameters used are fixed at Workflow composition time, and at execution time only the dataset can be varied. In addition, as mentioned before, sub-workflows are not supported and these, together with that Castro et al call "recursion" are necessary for implementing a parameter sweep. Thus to implement a workflow such a Mbandi's in Galaxy would instead mean to implement the workflow in its entirety outside of Galaxy and simply invoke the resulting implementation as a single tool.

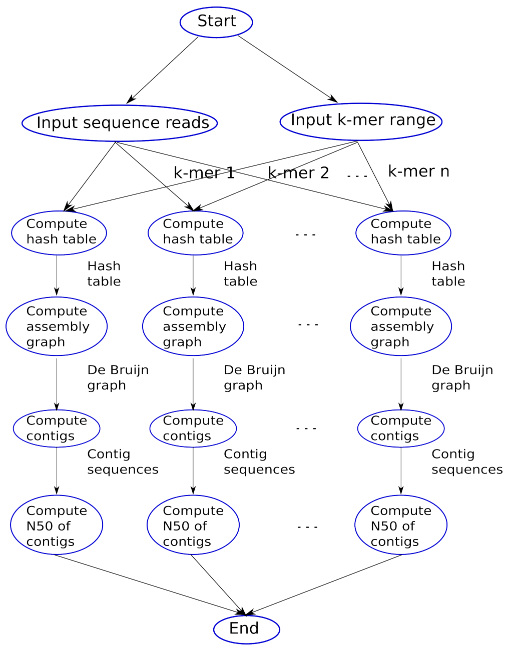


Figure 2.3.: Velvet parameter sweep implemented in parallel

2.2.3. Scientific workflow patterns for bioinformatics

2.2.4. Evaluating the features of scientific workflow management systems

3. Galaxy as a scientific workflow management system

3.1. Introduction to Galaxy

3.2. Useability analysis of the Galaxy SciWMS

3.3. Support for workflow patterns in Galaxy

3.4. Evaluation of the features of the Galaxy SciWMS

DON'T READ BEYOND THIS POINT.

The Galaxy framework is an application server that has been promoted as a tool to make computing more accessible for life science researchers [GNTT10], and in fact headlines its website with the text "Data intensive biology for everyone" [The12]. Galaxy is maintained by a cross-institutional collaboration that goes by the name "The Galaxy Team"[1] *It is open source software and may be installed by downloading the source code distribution from the project's website. As mentioned previously, Galaxy was chosen as a software platform for BSA by researchers at SANBI and the University of Cape Town. This choice was largely motivated by the ease with which existing bioinformatics tools could be executed within Galaxy.*[2]

The following sections will describe the analysis and workflow components of the Galaxy framework, the procedure for incorporating new bioinformatics tools in Galaxy and some details of the internal architecture of Galaxy.

3.5. The Galaxy user interface

Galaxy is an applications server, written in Python, that combines the ability to execute bioinformatics tasks with a workflow composition and execution system[3]*. It evolved out of a much simple web interface that allowed scientists to gather and manipulate data*

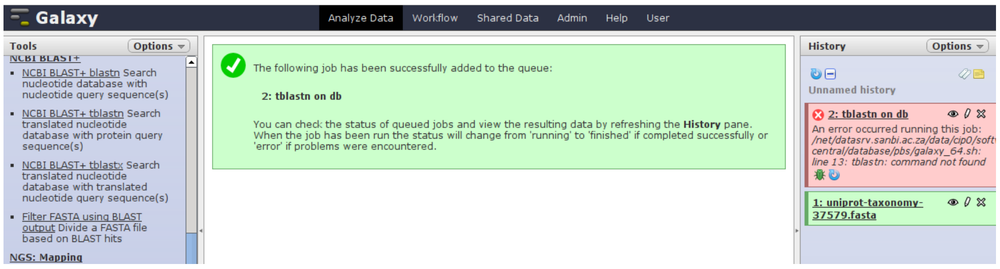
[1]The Galaxy Team is drawn from staff and students at the Centre for Comparative Genomics and Bioinformatics at Penn State University and the Computational Biology group at Emory University [The12]. Both of these institutions are in the United States of America.

[2]The other system evaluated by the "Pipelines" group was Taverna [HWS$^+$06], but this system was not chosen because it requires all tools to be presented as web services.

[3]Galaxy is maintained by the Galaxy Team that is comprised of members from the BX group at Penn State University and the Biology, Mathematics and Computer Science departments at Emory University in the United States of America.

from different sources, with all results being stored in a history system (for a description of this early Galaxy see [GRH⁺05] and [Nek11]). This early version of Galaxy introduced the tool abstraction, where the user is presented with a collection of tools and the details of how and where the tool is executed are hidden. The tool execution and history system has been retained in the current Galaxy but it is supplemented by a graphical workflow composer and a system for sharing "libraries" of data between users.

When a user navigates to the Galaxy web application (either running locally or on a remote server) in their web browser they are presented with the "Analyze Data" pane (Figure 3.1). On the left side of this pane is a list of tools, organised in groups. On the right side of the pane is a history list, describing the datasets available in the current history. The central area is occupied by a form describing either the tool being considered or the history item being filled. The user imports data into their history, from a biological data, files they upload, or items in the Galaxy installation's shared data library. They can then execute tool such as analyses or workflows, with the results being stored in their history. Galaxy maintains provenance information for each dataset in the history, storing information about the tool, the tool version, and the input data that was involved in producing each dataset. This information is important for other scientists who might want to reproduce the dataset.



NCBI BLAST+

- NCBI BLAST+ blastn Search nucleotide database with nucleotide query sequence(s)
- NCBI BLAST+ tblastn Search translated nucleotide database with protein query sequence(s)
- NCBI BLAST+ tblastx Search translated nucleotide database with translated nucleotide query sequence(s)
- Filter FASTA using BLAST output Divide a FASTA file based on BLAST hits

The following job has been successfully added to the queue:

2: tblastn on db

You can check the status of queued jobs and view the resulting data by refreshing the **History** pane. When the job has been run the status will change from 'running' to 'finished' if completed successfully or 'error' if problems were encountered.

Unnamed history

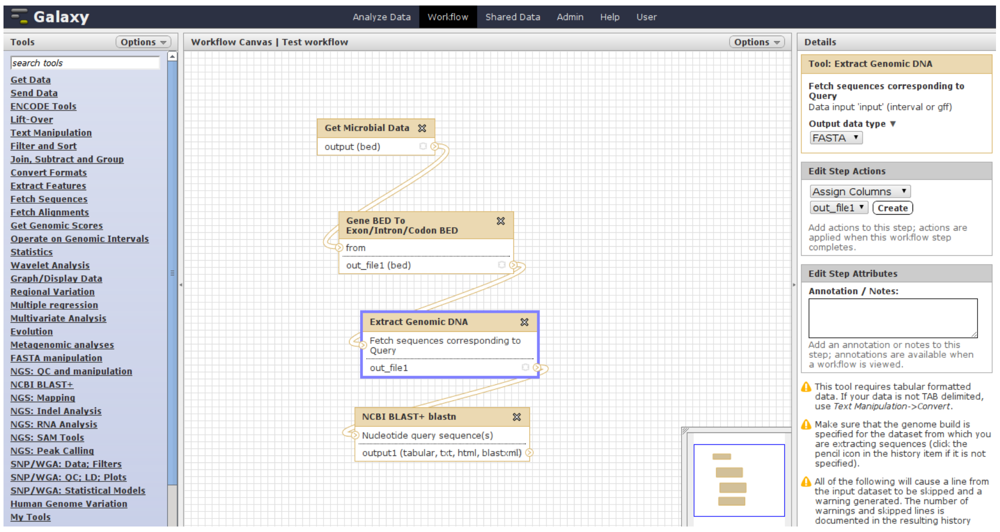
2: tblastn on db
An error occurred running this job: */net/datasrv.sanbi.ac.za/data/cip0/soft central/database/pbs/galaxy_64.sh: line 13: tblastn: command not found*

1: uniprot-taxonomy-37579.fasta

Figure 3.1.: The Galaxy "Analyze Data" view

The set of tools available to Galaxy users is configured by XML files that describe the input and output data, parameters and (typically) the command line tool that needs to be executed in order to accomplish a particular analysis tool. These XML files can either be authored locally or they may be imported from the Galaxy tool shed [The11] by the Galaxy installation's administrator. Input and output data is tagged with format information to ensure that the input dataset for a tool are in a format that the underlying tool can parse.

Individual tools can be combined into workflows in the "Workflow" pane of Galaxy. Here the existing workflows are shown, and new workflows can be created in a workflow editor. The workflow editor (show in Figure 3.2) allows the creation and editing of workflows using a visual data flow language. Workflows can also be instantiated from the contents of a history in the "Analyze Data" pane, allowing for a kind of "programming by example". The right side of the workflow editor allows the parameters for a tool in the workflow to be set, and some post-execution actions to be added (such as



Fetch sequences corresponding to Query
Data input 'input' (interval or gff)

Add actions to this step; actions are applied when this workflow step completes.

Add an annotation or notes to this step; annotations are available when a workflow is viewed.

⚠ This tool requires tabular formatted data. If your data is not TAB delimited, use *Text Manipulation->Convert*.

⚠ Make sure that the genome build is specified for the dataset from which you are extracting sequences (click the pencil icon in the history item if it is not specified).

⚠ All of the following will cause a line from the input dataset to be skipped and a warning generated. The number of warnings and skipped lines is documented in the resulting history

Figure 3.2.: The Galaxy "Workflow" pane showing the workflow composer

renaming the output dataset, since otherwise datasets have names such as "Filter text on dataset 3").

Workflows can be shared with other Galaxy users (users that are local to the same Galaxy installation) and published to the Galaxy public workflow repository (part of the Galaxy tool shed). Workflow descriptions can also be downloaded (in JSON format) and then uploaded to another Galaxy instance. [4]

3.6. Using Galaxy

Three students (one doing their post-doctoral studies, and two doing their doctorates) at SANBI were approached to solicit real-world examples of workflows in life sciences research. All three students are engaged in BSA of a diverse collection of species, including plants, fungi and bacteria. Of these students, two were implementing workflows as a collection of Perl and Python scripts, with the workflow being manually enacted by the user. The last was using Galaxy as their platform for workflow construction. Through interviews conducted with these students their workflows were documented. This documentation was supplemented in one case with a workflow description drawn from the student's doctoral thesis. In addition, the workflow described in [BTS+07] was examined. All collected workflows were described in terms of the workflow patterns identified by [vDATHKB03] and the task descriptions provided by [SGBB01] and [CTGR05].

[4]This export and import facility is somewhat limited in that the workflow description does not include the XML files for the tools in the workflow, and thus those need to be transported between Galaxy installations independently.

A literature search was conducted, starting from the keywords "scientific workflow". Key articles were identified, with a focus on papers that reviewed the field. In this regard [YB05] and [DGST09] were highlighted as particularly important works. The pool of articles was expanded through examining the references cited in the key articles and through examining papers that cited these key articles. A second literature search was conducted with keywords "galaxy bioinformatics" to identify papers referencing the Galaxy workflow system.

The Galaxy system was then described in terms of the classifications proposed by [YB05] and [DGST09]. Drawing from the literature on workflow system, and the data collected about real world bioinformatics workflows, Galaxy was evaluated in terms of its ability to express workflow patterns, the abstractions it supports and its ability to enact real world bioinformatics workflows.

In terms of mapping these concepts to ones identified in the domain of workflow systems, we can identify which of van der Aalst et al's [vDATHKB03] workflow patterns are necessary to implement a bioinformatics workflow. These are:

- *Pattern 1: sequence — enabling an activity in a workflow after the completion of another activity in the same process.*
- *Pattern 2: parallel split — a point in the workflow where a single thread of control splits into multiple threads of control that can be executed in parallel.*
- *Pattern 3: synchronisation — a point in the workflow where multiple parallel subprocesses converge into a single thread of control.*
- *Pattern 4: exclusive choice — a point in a workflow where, based on a decision or workflow control data, one of several branches is chosen.*
- *Pattern 14: multiple instances with a priory knowledge — where an activity is enabled multiple times. This describes both the iteration operation, where the multiple workflow instances are created, one for each element in the set of inputs, and it also describes the exploration of a "parameter space" if the elements of the parameter space are known at some time before the instances need to be created.*
- *Pattern 19: cancel activity — an activity thread is disabled. This is necessary to implement suspension / resumption.*
- *Pattern 20: cancel case — a workflow instance is removed completely. While this is not strictly necessary to implement the tasks described above, the ability for a user to remove a workflow instance from execution is very useful if problems become apparent during the enactment of a workflow.*

Thus of the twenty workflow patterns that van der Aalst et al. describe, only seven should be necessary to implement workflows in bioinformatics.

3.7. Case studies

Before I discuss the particular capabilities of Galaxy and consider scientific workflow systems in a more general sense, I will illustrate the challenges encountered in BSA through a detailed description of three real world bioinformatics workflows. These workflows were chosen because, in my experience, they represent typical examples of problems faced by life scientists doing bioinformatics, and thus examining the workflows the Galaxy system's ability to enact them provides a starting point for a more detailed examination of the capabilities of the system.

3.7.1. Case study 1: Mining bacterial genomes for virulence genes

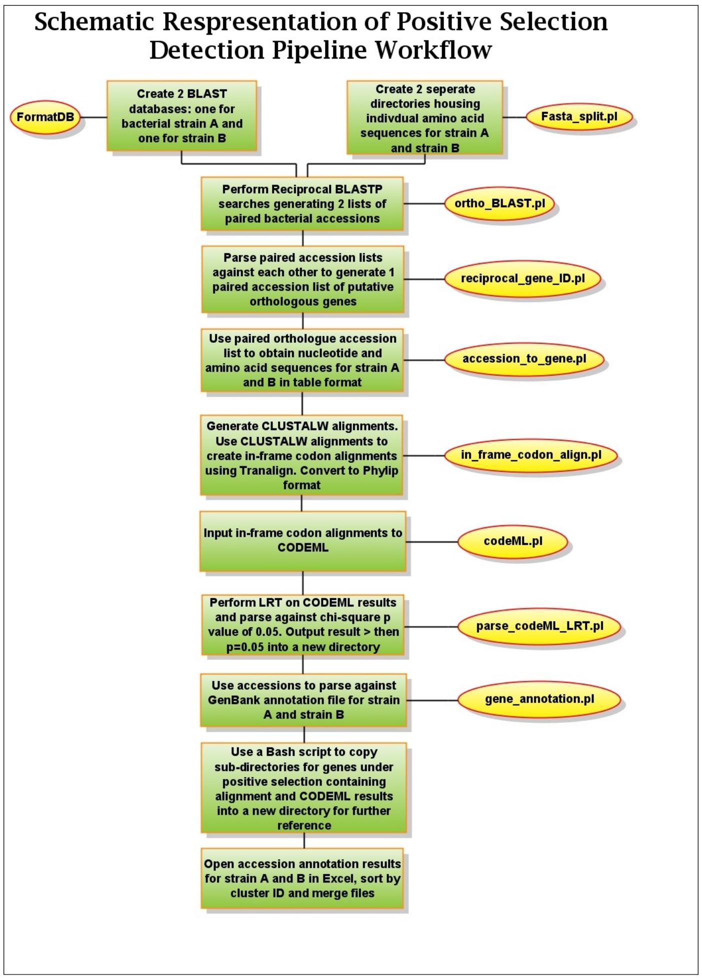
Pathogens that invade the human body engage in a continual "arms race" with the human immune system. Part of this "arms race" is pathogenic bacteria evolving new virulence genes (genes that encode for virulence factors) to defeat the human immune system's defences. Building upon earlier work, Panji [Pan09] investigated bioinformatics techniques to identify genes in bacteria that appear to be evolving at a rate higher than the genome as a whole. This analysis was the first step in investigating whether the rate of evolution of a gene might be an indicator of its involvement in the virulence of an organism. Panji implemented an analysis workflow that compared the genes contained against two bacterial genomes to one another and output a list of genes that showed signs of positive selection. This workflow was implemented as a set of Perl scripts that call various bioinformatics tools, and the steps in the workflow were executed by the user manually enacting them on the command line.

Panji's workflow is schematically described in their Ph.D thesis in the form of a diagram (Figure 3.3). Further knowledge of the workflow was gained by interviewing the author directly and by examining the Perl scripts used to enact the workflow. The Perl scripts that make up this workflow are available in Appendix A1 of [Pan09].

The workflow was the partially re-implemented in Galaxy (up until the LRT implemented by ***parse_codeML_LRT.pl****) in order to get a better understanding both of the tasks involved in the workflow, and Galaxy's adequacy for performing them. Time constraints prevented a full re-implementation of the workflow, but the portion that was implemented was adequate to gain a sufficient understanding both of the workflow and of Galaxy's ability to implement it, especially since the Galaxy implementation resulted in a workflow that could answer the basic question answered by Panji's workflow, that is, given the annotated genomic sequence of two strains of bacteria, which genes are experiencing positive selection?*

*In order to implement the pipeline in Galaxy, the baseline Galaxy code was retrieved from `https://bitbucket.org/galaxy/galaxy-dist` and installed. Five tools needed to be added to the baseline Galaxy system, two of these (****sed_wrapper*** *and* ***uniqueline****) were installed using the "Admin" panel of Galaxy from the Galaxy Toolshed, and three of these were added by myself by writing the necessary code and XML tool description files and adding these to the baseline Galaxy install.*

After experimentally enacting parts of the workflow manually (using the "Analyze Data" panel of Galaxy) a workflow consisting of thirty steps was composed using the



Schematic Respresentation of Positive Selection Detection Pipeline Workflow
FormatDB
Create 2 BLAST databases: one for bacterial strain A and one for strain B
Create 2 seperate directories housing indivdual amino acid sequences for strain A and strain B
Fasta_split.pl
Perform Reciprocal BLASTP searches generating 2 lists of paired bacterial accessions
ortho_BLAST.pl
Parse paired accession lists against each other to generate 1 paired accession list of putative orthologous genes
reciprocal_gene_ID.pl
Use paired orthologue accession list to obtain nucleotide and amino acid sequences for strain A and B in table format
accession_to_gene.pl
Generate CLUSTALW alignments. Use CLUSTALW alignments to create in-frame codon alignments using Tranalign. Convert to Phylip format
in_frame_codon_align.pl
Input in-frame codon alignments to CODEML
codeML.pl
Perform LRT on CODEML results and parse against chi-square p value of 0.05. Output result > then p=0.05 into a new directory
parse_codeML_LRT.pl
Use accessions to parse against GenBank annotation file for strain A and strain B
gene_annotation.pl
Use a Bash script to copy sub-directories for genes under positive selection containing alignment and CODEML results into a new directory for further reference
Open accession annotation results for strain A and B in Excel, sort by cluster ID and merge files

Figure 3.3.: Identifying positively selected genes in bacterial genomes

Galaxy workflow editor (in the the "Workflow") panel. A portion of the resultant workflow is show in Figure 3.4. As can be seen from that figure, Galaxy workflows resemble flowcharts, however, whereas flowcharts describe the flow of control in a program, the Galaxy workflow describes the flow of data between analyses (this distinction is important, as the expressive power of flowcharts has been strongly criticised, e.g. in [Bro95], but it is not clear that the limitations of flowcharts also apply to dataflow diagrams). In order to accommodate the entire workflow in a graphical representation, the textual representation of the workflow was converted into a graph that could be processed by ***graphviz****, shown as Figure A.1. Every node in this graph is a dataset, and every edge is an analysis (in other words a transformer or a filter). A second graph that shows*

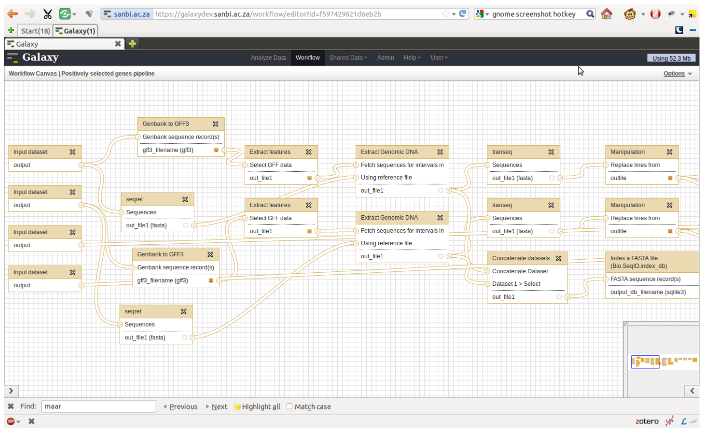


Figure 3.4.: Positively selected genes workflow in the Galaxy workflow editor

the same workflow but with analyses as nodes, is available in Appendix A.

In the workflow depicted, steps that are labelled A and B, and the ones labelled C and D, are equivalent. In the first case (A and B) a Genbank format file that contains genomic sequence combined with annotated features on that sequence is transformed into two separate datasets: a FASTA format file containing only the genomic sequence and a Generic Feature Format (GFF3) description of the genomic features. This description of features is then filtered to extract only the gene features, and these are used, in combination with the genomic sequences, to generate a collection of gene sequences (labelled "nucleotide FASTA" for strain 1 and strain 2). The genomic sequences are translated to amino acids to yield the proteins that correspond with the known genes (datasets labelled "protein FASTA"). These are then compared against one another, using the BLAST sequence homology search tool to find matching pairs of genes (i.e. orthologous[5] genes). The steps labelled A and B are roughly equivalent to Panji's ***ortho_BLAST.pl*** *script. Meanwhile the workflow has forked off processes that combine the sets of genomic and protein sequences to generate indexed sequence databases (that will be used at later steps in the analysis).*

The BLAST homology search outputs results in tabular format, where each row consists of a pair of matching sequences and associated information. These matches are filtered to find hits with at least 50% similarity, and a projection is done to extract only the columns containing the sequence identifiers. Finally the results of the two searches are combined and duplicates are removed. This results in a collection of set of orthologous genes, where each gene is identified by a pair of sequence identifiers (known as "accession numbers"). Having identified orthologous genes, the workflow now needs to test each pair of orthologs to see if the differences between their sequences indicate positively selected mutations. This is done in Panji's workflow using the scripts ***accession_to_gene.pl***, ***in_frame_codon_align.pl***, ***codeML.pl***

[5]In genomics two genes are orthologous if they are share an origin in a common ancestor.

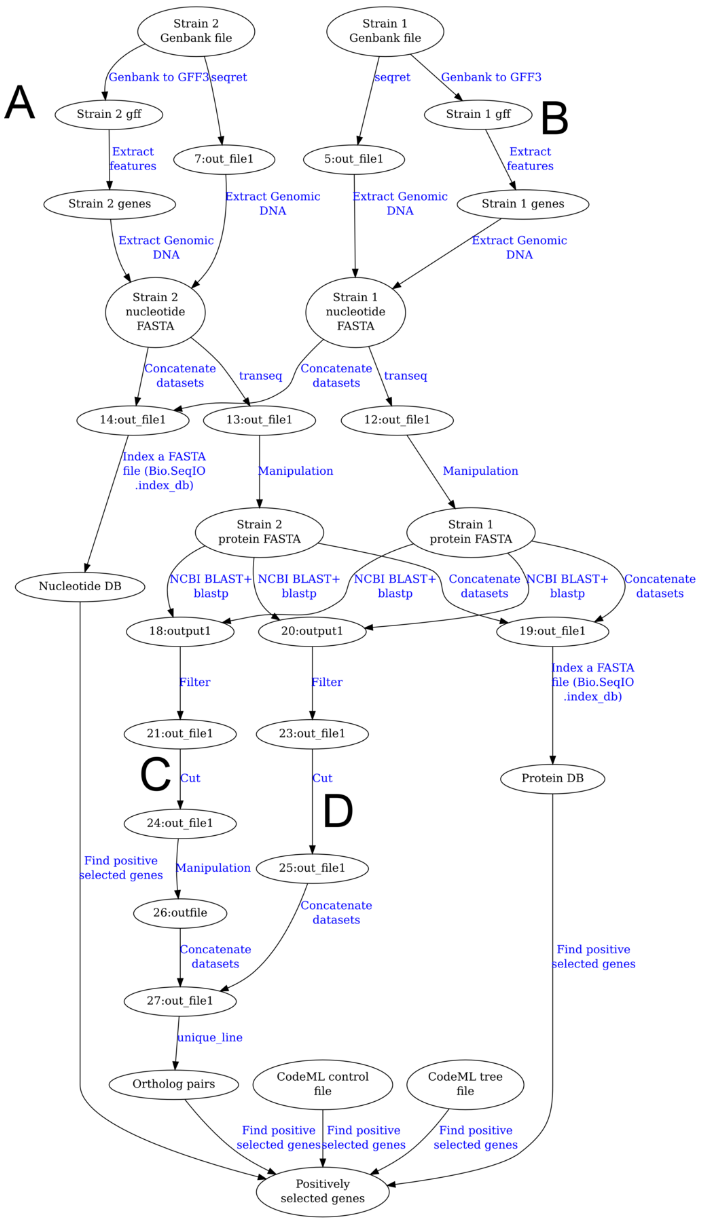


Figure 3.5.: Positively selected genes workflow

*and **parse_codeML_LRT.pl**. Using tools from the EMBOSS [RLB+00], ClustalW [LBB+07] and PAML [Yan07] software suites, a Log-Ratio Test is performed to find ortholog pairs that are positively selected (with a 95% confidence level, using the chi-*

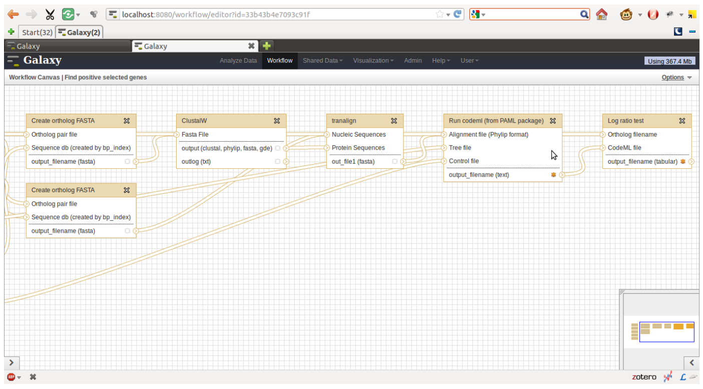


Figure 3.6.: Subworkflow for doing Log-Ratio Test on pair of gene sequences

squared distribution). A workflow analogous to Panji's was constructed in Galaxy (see Figure 3.6), but here Galaxy's limitations proved fatal: Galaxy does not support the iteration operation, and thus it is not possible to apply a workflow iteratively across a data collection. The sub-workflow was thus re-implemented as a Python script (the item labelled "Positively selected genes" in Figure A.1) and then added to Galaxy as a tool that could take the outputs of the preceding workflow steps.

While Panji's workflow executes tools in an entirely sequential fashion, the growth in multi-core CPUs and the use of computing clusters ([Ste08]) means that very often iterative workflows in bioinformatics are executed in parallel, for example by splitting the input dataset into partitions that can be executed independently. Since at least the 1970s it has been recognised that dataflow specifications enable parallelism [GT79], and Galaxy supports this pattern of task parallelism in the tool definition language (using the undocumented ¡parallelism¿ tag), and this capability was used to add support for parallel execution to the sub-workflow, splitting the input dataset into eight parts that could be analysed in parallel. Since parallelism is specified in the definition of a tool, and since the tool definition is de facto static, Galaxy's support for parallel execution is neither responsive to the size of the dataset (an eight way split might, for example, not make sense for a dataset of ten items) nor to resource constraints in the execution environment (in a shared compute cluster, the degree of parallel execution needs to be balanced by the need to share computing resources between many users).

Finally, it is worth noting that Galaxy executes each task independently. When tasks are executed locally, this might not be much of a concern, but in many computing environments the process of setting up and tearing down execution units (for example jobs on a compute cluster or grid) involves overhead. Thus in the steps marked C in Figure A.1, dataset 18 is first filtered, then the output of that operation (dataset 21) is transformed through a projection that selects a number of columns (yielding dataset 24) and finally a further text manipulation is performed to yield dataset 26. Such a sequential sub-workflow could be transformed (via "edge contraction", i.e. the combination of analyses steps that operate on a chain of datasets) into a single analysis *that could be executed as a single unit.*

In conclusion, Panji's workflow could be implemented in Galaxy, and fairly rapidly. Doing so involved adding tools to Galaxy, some of which were available via the Galaxy Toolshed and thus could be installed without leaving the Galaxy web-based user interface (although doing the installation required administrative privileges that typically would not be available to ordinary users), but three tools had to be added by writing Galaxy tool descriptors in XML, and editing Galaxy's configuration files. The shift in idiom between the Galaxy web-based environment and the command line environment required to add tools to Galaxy is quite significant. In addition, one of the tools added was an implementation of a sub-workflow, and thus would probably not be reuseable outside the context of the current workflow. Galaxy's lack of support for workflow iteration and sub-workflows forced this form of implementation. In addition, while parallel iteration of sub-workflows is somewhat supported by the (as yet undocumented) parallelism tag in the tool descriptor XML, the degree of parallelism can not be chosen at runtime. Finally, executing each step of the workflow as its own task leads to inefficiencies that might be avoidable with a less naive workflow execution engine.

3.7.2. Case study 2: Mapping methylated regions in a bacterial genome

Deoxyribonucleic acid (DNA) is the molecule that contains the genetic instructions in the cells of every living thing. This molecule is made out of repeating sub-units called "bases". In general there are four types of DNA base, adenine, thymine, guanine and cytosine. The combination of these bases describes the sequence of amino acids that make up proteins. The process of creating a protein sequence from a DNA sequence involves a sub-process called transcription. Transcription in cells is regulated, thereby "switching" genes on and off, and altering the production of proteins. One mechanism involved in transcription regulation is methylation, where a methyl group is added to the cytosine base. Detecting methylation in a genome is important for understanding gene regulation and thus the molecular biology of an organisation. Alecia Naidu, a doctoral student at SANBI is studying this process in bacteria and they have implemented their analysis workflow in Galaxy. They were approached to document their workflow via a series of interviews.

In order to detect methylation, DNA is treated with bisulfite. This converts unmethylated cytosine basis to uracil, but methylated cytosine bases are unchanged. Thus, methylated bases can be detected by comparing the sequence of bases in bisulfite treated DNA with that of untreated DNA from the sample biological sample. An image of the workflow that Naidu produced is included as Figure 3.7. The details of most of the analysis steps are blurred out since this work is, as yet, unpublished. In general, however, the workflow involves transforming a set of input nucleotide reads from the sequencing of a bisulfite treated genome into a set of predicted methylation sites by combining the read dataset with a reference genome dataset. In other words, it complies with the transform and filter task schema that Stevens et al outlined. The author has had some experience with producing workflows using the command line and scripting languages, and their assessment of Galaxy is not positive. They compared the process of doing *bioinformatics in Galaxy to "doing bioinformatics using blinkers", and in particular noted the fact that the tool wrappers of Galaxy do not always expose the full set of command line switches that some tools support. This is an understandable omission given the vast number of options available for some tools (for example, the* ***tranalign*** *tool from the EMBOSS tool suite has 51 different command line switches).*

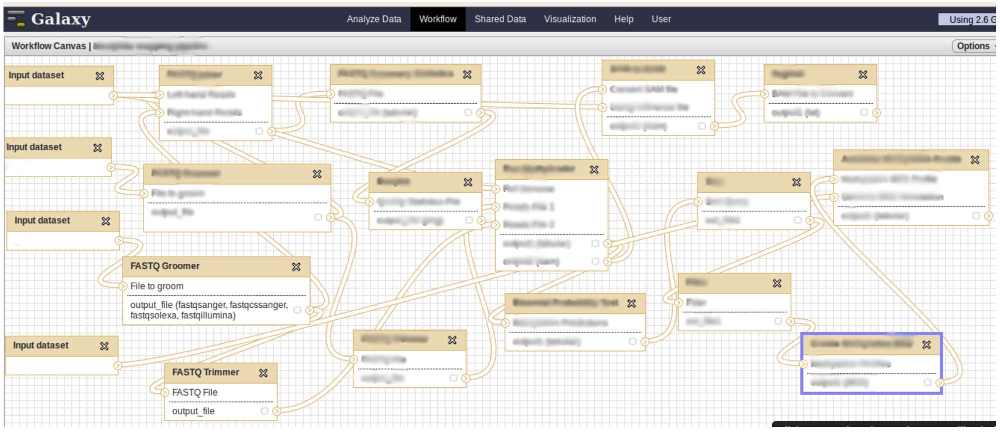


Figure 3.7.: Methylation analysis workflow (details blurred out)

In addition, Naidu noted that each dataset is different, and that this made the details of the workflow to some extent dependent on the characteristics of the input data. For example, the accession number that is used to identify a nucleotide or amino acid sequence might sometimes also contain further information in its formatting, thus a tool might need to act differently depending on the formatting of this sequence identifier. This kind of complexity requires that the workflow user has detailed knowledge of the bioinformatics applications used in the workflow and thereby the ability to hide complicity using the workflow abstraction is reduced.

In addition, the Galaxy workflow abstraction does not allow the user to specify the resource requirements of analysis steps. In a cluster or grid environment this is typically something that is specified at job submission time, and might be used to allocate particular tasks to different nodes in a computing environment. This was a concern to Naidu because the analysis that their are enacting is resource intensive, requiring a computer with more than 15 GB of RAM in order to process. In addition to being resource intensive during processing, Naidu's analysis involves considerable grow in data storage requirements. Their test dataset is 360 MB in size and by the end of the analysis some 2.6 GB of disk space is used by all the datasets involved in the workflow. In Figure 3.7, the FASTQ Groomer steps simply transforms an input dataset from one input format to another for use in later analysis, yet the duplicate dataset is retained. The next step, the FASTQ trimmer, transforms its input by performing "quality trimming", but once again the input dataset is retained in its entirety.

Finally, Galaxy does not support the suspension and resumption of workflows, thus *a failure in any one of the workflow's steps would require re-executing the workflow in its entirety. This is different from the experience of command line bioinformatics workflows where analysis steps can be manually re-enacted as needed. Thus, while Naidu's workflow is a fairly straightforward transform and filter operation, they have experienced considerable frustration in implementing it in Galaxy and thus complain that the Galaxy environment "doesn't do what it promises".*

3.8. The features of Galaxy as a workflow system

In the introduction to this research report I cited the suggestion that BSA faces an "informatics crisis" characterised by the accumulation of data without a concomitant increase in analytical capacity. This "informatics crisis" is not a phenomenon unique to bioinformatics and as a result scientific workflow systems have been widely used outside of the field. Yu et al mention their use in fields as varied as astronomy, geophysics and bioinformatics. [YB05] The case studies detailed above, together with Stevens and Castro's more theoretical overview, shows the elements involved in the challenge of bioinformatics. Approaching the question of Galaxy's ability to contribute as a tool for enacting bioinformatics workflows, I will now consider the capabilities of scientific workflow systems in a more general sense. Both Deelman et al [DGST09] and Yu et al [YB05] have proposed classifications of workflow features, and I will now illustrate how these features relate to both the concrete bioinformatics workflows and the feature set exhibited by Galaxy. Deelman et al suggest that workflow systems should be be classified according to their support for:

1. *Workflow composition*
2. *Mapping workflow specifications to executable resources*
3. *Workflow execution*
4. *Support for recording data provenance*
5. *Interoperability with other workflow systems*

3.8.1. Workflow composition

Workflows in Galaxy are composed using a graphical tool in what is essentially a graphical dataflow language. Tools are selected from the sidebar, parameters are configured and input and output datasets are represented as ports that can be chained to other tools. Each dataset has an associated format and this information is used to decide which ports can be connected together. The data format is also used when deciding how to display the data in the "Analyze Data" pane. Each dataset also has associated metadata that describes it, for example the genome it is associated with. In terms of the workflow structures that Yu et al [YB05] consider, Galaxy workflows are Directed Acyclic Graphs (DAGs) that support parallelism (the output of a single tool can be input to more than one downstream tool) but not choice between dataflow paths.

Deelman et al [DGST09] suggests that abstraction is necessary to manage the complexity of graphical representations of workflows. This point is reinforced by Blackwell et al in their survey of the cognitive features of visual programming languages [BWGP01], where they speak of an "abstraction gradient". What levels of abstraction are possible? The Galaxy workflow composer presents two levels of abstraction: the view of the entire workflow, and the detailed view of the parameters of a single tool. As the case studies have shown, there are a number of common tasks is bioinformatics workflows that Galaxy does not support, and in some cases (such as the duplicated sections of the workflow illustrated in Figure A.1) this leads to increased complexity.

Deelman et al and Altintas et al [ABB$^+$05] makes three suggestions for abstraction in workflows: nesting workflows (i.e. the ability to call sub-workflows), the ability to hide "shim services", and abstracting patterns in workflows into "abstract actors". The ability to call sub-workflows is the most straightforward requirement that Galaxy lacks. Workflow patterns such as iteration using multiple instances, and parameter sweeps both require sub-workflows to be implemented, as is illustrated by the obstacles encountered in implementing case studies one and three in Galaxy.

Shim services (or tasks) are workflow elements that simply reformat data (in other words they make changes that are simply syntactic). The manipulations that happen to datasets 13 and 12 in the positively selected genes workflow from case study one (Figure A.1) are examples of shims. These simply trim off an extraneous piece of text that is added to the sequence identifier by transeq. Similarly the FASTQ groomer used in Naidu's workflow (Figure 3.7) simply transforms a FASTQ format file between two different variants of the same sequence format. Galaxy allows the workflow designer to hide the outputs of workflow tasks. This facilites hiding shim tasks and reduces the complexity of the set of datasets presented to the user when a workflow completes. aAt the workflow composition stage it might, however, be useful to be able to hide these shims in order to reduce graphical complexity. This is somewhat related to encapsulating with sub- workflows, in the sense that a tool and a format altering shim might be conceptually combined into a sub-workflow that appears as a single item in the final workflow.

Finally, Altintas et al suggest that concrete tasks in workflows can sometimes be replaced by abstract actors. This corresponds to the abstract pole on Yu et al's workflow model axis. While Galaxy abstracts away the details of how tasks are executed (thus the same workflow could execute on a single computer and also on a compute grid), the notion that a task itself could be a "place holder" is a useful abstraction. Thus, for example, the framework for a parameter sweep evaluation of an assembly tool could be retained and reused with different concrete assembly tools. As workflows are shared and reused this kind of abstraction could reduce the proliferation of workflows that are essentially similar and only differ in a single detail.

The abstractions listed above allow for workflows to be considered from a higher level of abstraction. Another dimension of workflow composition involves the details of the tasks that workflows are composed out of. Currently in Galaxy adding or modifying a tool requires leaving the Galaxy environment and editing XML format text files, or alternately adding the tool from the Galaxy tool shed. There is no way to edit the definition of a tool from within the Galaxy web interface. The NAIAD HPC Web workflow *system [Tar11] (itself built upon the Mobyle [NMM+ 09] workflow system) solves this problem by providing a web form based tool builder. The Galaxy tool definition language (partially documented in [The11]) is, however, more complex than the one used in HPC Web, thus it might not be straightforward to implement an equivalent tool builder.*

Blackwell et al have suggested that the availability of a well-stocked library of reuseable components elevates the semantic level of a visual programming language. This is certainly true for Galaxy, as the base system comes with more than 300 tools already available, and the Galaxy tool shed provides access to almost 2000 more. A large number of powerful operations are immediately available for inclusion into Galaxy workflows. These tools are complemented with nearly 200 data format definitions. In a sense Galaxy comes "batteries included": immediately after installation it is possible to use it for real world work.

Deelman et al suggest that handling of parameters should be considered when surveying workflow system features. The Galaxy workflow system treats datasets as input and output parameters, but they are the only parameters supported in workflow composition. The blocks that represent tools used in workflows have no ports for settings parameters (such as k-mer length for sequence assembly), something that prevents certain kinds of workflows (for example parameter sweeps) from being implemented. The abstract actor proposal that Altintas et al suggest would require a further extension of the Galaxy system, so that tools themselves can be passed as parameters to workflows.

Finally data flow in Galaxy is structured by matching syntax of outputs and inputs. Stein [Ste08] has suggested that semantic information could be used to guide the construction of workflows. Yu et al and Deelman et al also take up this theme, suggesting that workflow construction might even be partially or fully automated. (An early example of research in this area was the TAMBIS system [SBB+ 00] and the topic is also explored in Gil et al [GDB+ 04]). While the full semantic description of tools is too much to ask for at this point, Galaxy does make use of some semantic information such as the fact that certain data types describe genomic intervals. The new parallelism tool configuration also makes use of the fact that certain data formats in fact describe collections that can be analysed in parallel.

3.8.2. Mapping workflows to resources

Mapping involves translating the abstract workflow description composed by the user into a concrete execution plan. In this regard Deelman et al differentiate between task-based and service-based workflows. In service-based workflows, mapping involves binding to services (often, as in the case of Taverna [HWS+ 06], web services), where services might be chosen based on suitability and quality of service issues. On the other hand, task-based workflow systems map workflow steps to particular executable resources. Clearly Galaxy is a task-based workflow, and implements mapping through "job runners". Executable resources in Galaxy are referred to as tools and the system maintains a registry of tool definitions that, in turn, are specified in a XML based tool description language. The tool registry is loaded at Galaxy startup, and to add new items the Galaxy application needs to be restarted (although, once loaded, the tool defi-

nition for individual tools can be updated without restarting the entire Galaxy application). Job runners translate a request to invoke a particular tool into the concrete steps required to execute that tool. Galaxy supports local execution (on the same machine as the Galaxy application is running) and execution via a number of batch-queueing systems. The Galaxy administrator can configure the association between individual tools and job runners as well as a default job runner for the Galaxy installation.

Galaxy jobs are scheduled in a centralised fashion, with the scheduler communicating with job runners through a job queue. The Galaxy web interface translates user actions into job specifications that are placed on the queue queue, and job runners (each with a configurable number of threads of execution) consume jobs from the job queue and manage the execution of the tools specified in the jobs. In the case where Galaxy interfaces with a batch-queue system the limitations of the Galaxy mapping system becomes apparent, since while modern batch-queueing systems possess sophisticated schedulers that can route jobs to appropriate computing resources based on characteristics such as user role, expected memory utilisation or execution duration, Galaxy provides no such information to the batch-queueing system. Instead, all jobs are scheduled by a single system user with no resource information supplied. Other workflow systems, such as Triana [CGH+06], allow workflow specifications to be annotated with information that can be used in scheduling decisions, such as which elements are compute intensive. Another approach, used by the ICENI workflow system [MYA+04], is to use some form of metric to try and estimate resource usage based on input data characteristics.

Finally, workflow mapping offers the possibility of workflow optimisation. Deelman et al mentions sophisticating optimisation of task-based workflows carried up by Pegasus [DSS+05] and Askalon [FPD+05]. These involve analysing the DAG that describes a workflow and applying operations such as task clustering. Galaxy does not perform any such optimisations, with the result that even shim tasks are scheduled independently of the analyses they bind together. Since Galaxy generates an internal DAG representation of the workflow, adding an optimiser to the system should be quite feasible and this would have significant performance advantages.

3.8.3. Workflow execution

Once a workflow has been translated into a concrete specification of tasks to be executed, the actual execution of tasks needs to be managed. Galaxy manages this through the communication between the Galaxy scheduler and the job runners. Task execution involves not just running tools, but also the operation that surround the execution of a tool. For example data might need to be transferred to the location of execution (stageing in) and results downloaded after execution (staging out), authentication might need to be managed (for example, in order to access resources on a compute grid) and the tool might require particular environment variable to be set. While some Galaxy job runners do support stageing in and stageing out, in general Galaxy assumes that both the Galaxy application server and the tools it executes will have access to a shared filesystem space (typically shared using NFS). Galaxy users authenticate to the Galaxy web application, and there is experimental support for mapping users on the Galaxy web app to users on the system that Galaxy is executing on. There is clearly much *scope for improvement in how Galaxy manages tool execution, as the management of the complexities involved in this domain seems somewhat ad-hoc at present.*

In addition to the details of how to execute a task, a workflow system is often faced with the failure of a particular task execution. This failure might be permanent, for example in the case where a tool is incorrectly implemented, or transient, for example where a resource such as disk space is temporarily exhausted. As workflows increase in complexity, failure of an component becomes increasingly likely, and in their study of fault tolerance in scientific workflows, Kandaswamy et al [KMR08] found that of 165 weather forecasting workflows run on the US TeraGrid in one month, 131 encountered a task failure. Utilising a fault recovery service, 93 of these failed workflows were successfully recovered, reducing a workflow failure rate from 79% to 23%. Bowers et al propose certain patterns of fault recovery (in other words exception handling) that can be applied to the Kepler workflow system [BLNC06]. At present Galaxy does not have any fault tolerance features, and the failure of a single task leads to the abort of the entire workflow downstream from that task. Tasks that do not depend on the failed task continue to run, thus a workflow might exhibit partial success. The concern with fault tolerance lies in a space that where workflow execution and workflow mapping overlap to some extent, since one response to failure (for example of a compute node) might be to reconsider the mapping between an abstract workflow and its concrete execution plan. Yu et al refer to this capability as dynamic planning, and as Bowers et al's work shows, it is an area where concern with the flow of control appears even if the workflow is specified in terms of data flow.

One aspect of control flow is that is related to fault tolerance is pause and restart capability. A user might identify and remedy the cause of a failure, and wish to restart a workflow from a particular step onwards. Deelman et al mention that Askalon supports workflow checkpointing to facilitate such a restart. Galaxy, however, does not support any notion of workflow pausing or restarting. In essence, if control in a workflow can be imagined as a set of tokens that passes from task to task, Galaxy does not support any mechanism for the user to manipulate such control tokens. The result is that workflow execution in Galaxy can be a frustrating fragile process.

3.8.4. Support for recording data provenance

Data provenance is a record of the history of the creation of a data object. In "traditional" bioinformatics workflows, like Panji's positive gene selection workflow, datasets are produced without any information being captured about how they are produced, except for the documentation produced by workflow authors. Goecks et al point out the weakness of this approach, citing a study showing that less than half of a particular set of microarray experiments could be reproduced, with missing data and data provenance information being a large factor in the inability to reproduce experimental result. Galaxy addresses this problem by recording provenance information about datasets with each item produced. Information about inputs, outputs, tools used, tool XML version and tool parameters is stored, and can be shared between users on a Galaxy instance or exported as a file. The Galaxy system can also be used as a kind of content management system for scientific publications, where Galaxy tools, workflows and data library *items can be embedded in a web page. As far as I know, Galaxy is the only platform that allows for this kind of publication.*

Apart from data provenance, Deelman et al note that it is useful to track the history of changes to a workflow and that transformation of a workflow (for instance by workflow optimising by the scheduler) could impact on the provenance information recorded. Galaxy does not support workflow versioning, nor does it transform workflows in any significant way, except for optionally hiding datasets in the user's history. Besides ensuring reproducible research, stored data provenance information could be used to reduce the disk space usage of a workflow. As another example, a workflow implemented at SANBI for the analysis of novel tuberculosis strains generates some 60 GB of data for each strain of the bacteria analysed. Only a small fraction of this is input data and analysis results, the bulk is intermediate datasets used within the workflow. With sequencing now under way for hundreds and maybe even thousands of strains of Mycobacterium tuberculosis, the ability to identify which datasets may safely be erased (and possibly recomputed at a later date if necessary) could save terabytes of disk space. At present, however, Galaxy does not itself erase any data, not even datasets marked "deleted" in the user's history. Site administrators are advised by the Galaxy developers[6] to clean up their disk space by periodically running a "clean up" script that examines the Galaxy database and deletes datasets that are marked as deleted. If data provenance information were extended to include information about the resource cost of computing a dataset and about which datasets needed to be retained as analysis results, the Galaxy provenance data system could be a powerful tool for disk usage management.

3.8.5. Workflow interoperability

A bioinformaticist searching for a workflow system is faced with a dizzying array of choices. Searching the web for bioinformatics workflow systems reveals Galaxy [GRH+05], Taverna [OAF+04], Kepler [ABJ+04], Triana [CGH+06], Bioopera [BPSA02], Mobyle [NMM+09], BiosFlow [XH09], [OCG+10] and bpipe [SPO12], amongst others. Each of these workflow systems uses a different format for storing and exporing its workflow descriptions and there are often semantic differences between the systems, for example Taverna's service-based structure compared to Galaxy's task-based one. It is unrealistic (at this stage, at least) to expect workflow system designers to decide on a single language for specifying scientific workflows, but some work has been done on making Galaxy interoperate with other workflow systems. In particular, Wang et al [WBS+09] and Abouelhoda et al [AAG10] have attempted integration between Galaxy and web services based workflow systems.[7] Wang et al implement this by creating a tool that converts Web Service Definition Language (WSDL) descriptions into Galaxy tool and datatype specifications. Similarly, Galaxy exposes a XML-RPC based API that allows workflows in Galaxy to be called by external programs. This API is still, however, experimental and undocumented (except through code examples).

[6]See Purge Histories and Datasets in the Galaxy documentation.

[7]In the terminology of the Workflow Management Coalition, both forms of integration involve cooperation between different workflow systems, i.e. Level 2 interoperation [Wor99].

Abouelhoda's approach is on a higher level of abstraction, in that they present an abstract workflow specification that is converted into sub-workflows that are executed either by Galaxy or Taverna[8]*. The choice of which workflow system to execute sub-workflows on is made based on the workflow pattern exhibited by the sub-workflow. This form of integration grasp that workflow interoperability is not simply a matter of building interfaces between workflow systems, since interoperability needs to recognise the semantic differences between workflow systems. For example, a workflow system that is service-based requires different mapping logic (as a choice needs to be made between different instances of the same service specification) to a task-based one.*

3.9. Conclusion

Bioinformatics is currently characterised by a disconnect between the vast growth in repositories of biological data and the very slow growth in the capacity to analyse this data. The main constraint on the growth of analytical capacity is a human constraint: the prevalent paradigm of bioinformatics workflow construction requires programming skills on the part of the analyst and these skills are not widely available. In addition, the majority of bioinformaticists come from a life sciences background where training in programming is not common, thus the field is constantly accepting scientists who are not experts at orchestrating computation. This situation has been described as an "informatics crisis" [GNTT10] and has spurred the search for tools that allow scientists to "[ask] complex questions of the genome without programming" [Woo10].

One early approach to making bioinformatics tools more accessible to non programmers was to provide web interfaces for what would otherwise be command line tools. Any number of ad-hoc web wrappers exist for individual tools, but projects such as Pise [Gil02] and wEMBOSS [SC05] extended this approach by providing a way to automatically generate web based user interfaces from tool descriptions. Such tool interfaces were combined with authentication and dataset management functions to provide web portals for bioinformatics. In its earliest incarnation Galaxy [Nek11] was an example of such a portal, combining interfaces to analysis tools with the ability to import data from biological databases such as the UCSC Genome Browser. From the start Galaxy emphasised storing data provenance information in users' histories and this feature even gave its name to the Galaxy user interface (the History User Interface [GRH^{+}05]). More recent incarnations of Galaxy retain these features and the system is a powerful portal for providing web based user interfaces to bioinformatics tools, executing these tools and visualising results. As a bioinformatics portal Galaxy is powerful, with the Galaxy baseline system providing interfaces to some 300 tools, and the Galaxy tool shed allowing the installation of almost 2000 additional tool interfaces.

The user history metaphor used in Galaxy lends itself naturally to the notion of workflows, where a workflow describes the chaining of a number of different analysis tasks to build a more complex overall analysis. Stevens et al [SGBB01] and Castro et al [CTGR05] have categorised the tasks involved in bioinformatics and their classification

[8]The result is a tool dubbed Tavaxy.

suggests that workflows are a natural metaphor for answering complex problems bioinformatics. These classifications in turn seem to be expressed naturally in the form of visual dataflow languages. Dataflow computing describes computational tasks in terms of the flow of data between instances of computation [GT79] as opposed to the more common paradigm in programming that describes the flow of control in a program.

Dataflow computing provides an explicit description of the combination of data and tools that needs to be present at the moment of computation. As Gostelow [GT79] showed as long ago as 1979, dataflow description lends itself quite naturally to composing parallel execution plans that can be mapped to resources on the many-core compute clusters, grids and clouds available to scientists. Visual dataflow languages also have a long history, with Hils [Hil92] providing an early overview and Blackwell et al [BWGP01] discussing cognitive factors that affect their expressive power.

Thus, to recap, workflow systems that allow the user to compose workflows in the form of visual dataflow descriptions provide a powerful tool to ask complex questions about biological data without resorting to programming. Galaxy is one amongst a number of workflow systems that allow users to compose workflows in this manner. The workflow composition and enactment components of Galaxy are, however, limited in ways that make doing real-world bioinformatics in the system challenging.

Firstly, in terms of workflow composition, the Galaxy workflow system does not allow for a smooth "abstraction gradient" that would allow users to switch between high level overview of workflow tasks and drilling down into the details of workflow task definition. Tool definition in Galaxy happens outside of the web user interface, thus one is forced to be familiar with the XML tool declaration language in order to extend the set of tools that Galaxy supports. On the other side of the scale sub-workflows are not supported so workflows can not be encapsulated and reused within other workflows.

Castro et al [CTGR05] specifies two operations in bioinformatics that operate across collections. On the one hand what they call iteration operates across a collection of data items, applying the same operation to each one. On the other hand what they call recursion operates across a set of input parameters, executing a parameter sweep where the these parameters are used to alter an operation that is repeatedly executed on the same dataset. Both of these constructs require the support in a workflow system for bioinformatics, and neither is supported. Deelman et al [DGST09] suggests that such constructs can be well supported by combining control flow and dataflow descriptions in workflow composition. Bowers et al [BLNC06] concur with Deelman et al and provide a model of where a workflow can consist of dataflow description on a high level but incorporate control flow elements in a lower level description of workflow tasks. Elizondo et al [VEL10] propose that instead of providing this description on the task level, the necessary constructs can be expressed using a suitable library of component connectors. Whichever route is chosen, to make Galaxy useful for general purpose use, one of these solutions needs to be adopted. In addition, Altintas et al [ABB$^+$05] propose a further level of abstraction: tasks in a workflow might be place holders, allowing the same abstract workflow description to be reused with the place holder replaced at run-time with one or another concrete analysis task. For example, a general workflow for mapping novel sequence reads to a genome can be produced, and the choice of which mapping tool to use can be left for a moment close to run-time.

Moving along the abstraction gradient could also allow semantically uninteresting elements of a workflow, such as data format conversion shims, to be hidden in a high level workflow description. In addition to this navigation of several levels of abstraction, the Galaxy workflow system needs to be extended in terms of how the system maps workflow descriptions to execution plans. At the moment Galaxy does this in a naive manner, simply queuing up jobs for job runners to execute. There is no scope for expressing resource requirements such as the amount of memory or the duration of runtime that a task might require. Enhancing Galaxy workflow description with a way of expressing resource requirements would allow more detailed execution plans to be prepared. In addition, since Galaxy is aware of the workflow description in its entirety prior to workflow execution, the Galaxy scheduler could be enhanced to optimise execution steps, grouping steps together that share datasets, for example.

Once the Galaxy scheduler has transformed a workflow description into an execution plan, execution is effected through job runners that provide an interface to various batch-queue systems, as well as allowing tools to be executed directly on the local system. While Galaxy does an impressive job in abstracting away the differences in how analyses are executed, and thereby hiding the tricky details of computation from its users, the complexities of tool execution, in particular the stageing in and out of data and managing user authentication to computational resources is handled in a fairly ad-hoc manner. Some of this is no doubt due ot the experimental nature of these features of Galaxy, but in event it is an area that could do with improvement.

Finally, a more significant weakness is Galaxy's lack of support for fault tolerance and recovery. If a task within a Galaxy workflow fails to execute correctly, there is no way to restart part of a workflow. As workflows grow in complexity, the chance for task failure increases, and some form of fault recovery becomes imperative. This might take the form of exception handling, manual retry of failed tasks or a fault recovery service that automatically retries failed components. Manual fault recovery is probably the simplest idea to implement, but this would require making explicit the manipulation of control flow in the workflow, possibly through a way to stop and start particular workflow steps individually. An alternative is to generate a series of checkpoints as the workflow executes and allow the user to re-execute the workflow from a particular checkpoint.

In spite of these weaknesses, Galaxy is widely used for real world bioinformatics, with the public Galaxy server (`https://main.g2.bx.psu.edu/`) processing over 130 000 jobs per month and accepting more than one terabyte of data uploads every week. [Nek11] As the examination of the cases studies in my research make clear, however, Galaxy does not yet provide an environment capable of replacing the command line and scripting workflow style in use in bioinformatics. This is unfortunately as some of the features of Galaxy, for example, its handling of data provenance, are a significant advance over the shell script and command line style of bioinformatics. Galaxy's limitations are not, however, fundamental limitations of scientific workflow systems, and a survey of the field of scientific workflow systems suggests that they might provide a powerful tool to address the "informatics crisis". Certainly as biological data accumulates there is pressure to both reach the same answers faster (thus we cannot afford to spend years on sequence assembly problems anymore) and to ask more com-

plex questions of the data. In the absence of a sudden increase in scientists who are also programmers, tools that allow non programmers to ask questions of biology data are crucial. Galaxy in its incarnation as a web portal and naive workflow composition and execution environment is already one such tool, and the limitations identified in this research clearly don't stop Galaxy being used. Instead they could be seen as presenting an agenda for further research, a set of tasks the bioinformatics programmer community can adopt whose pay off is likely to be substantial.

Bibliography

[AAG10] *Mohamed Abouelhoda, Shady Alaa, and Moustafa Ghanem. Meta-workflows: pattern-based interoperability between galaxy and taverna.* Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science*, page 2:1–2:8, 2010. ACM ID: 1833400.*

[ABB$^+$05] *Ilkay Altintas, Adam Birnbaum, Kim Baldridge, Wibke Sudholt, Mark Miller, Celine Amoreira, Yohann Potier, and Bertram Ludaescher. A framework for the design and reuse of grid workflows. In Pilar Herrero, María Pérez, and Víctor Robles, editors,* Scientific Applications of Grid Computing*, volume 3458 of* Lecture Notes in Computer Science*, pages 295–299. Springer Berlin / Heidelberg, 2005.*

[ABJ$^+$04] *I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: an extensible system for design and execution of scientific workflows. In* 16th International Conference on Scientific and Statistical Database Management, 2004. Proceedings*, pages 423–424. IEEE, June 2004.*

[AS94] *Kenneth R. Abbott and Sunil K. Sarin. Experiences with workflow management: issues for the next generation. In* Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, page 113–120, New York, NY, USA, 1994. ACM.*

[BG99] *A.F. Blackwell and T.R.G. Green. Does metaphor increase visual language usability? In* Visual Languages, 1999. Proceedings. 1999 IEEE Symposium on*, pages 246 –253, 1999.*

[BLNC06] *S. Bowers, B. Ludascher, A. H. H. Ngu, and T. Critchlow. Enabling ScientificWorkflow reuse through structured composition of dataflow and control-flow. In* Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on*, page 70, 2006.*

[BPSA02] *W. Bausch, C. Pautasso, R. Schaeppi, and G. Alonso. Bioopera: Cluster-aware computing. In* Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*, page 99–106, 2002.*

[Bro95] *Frederick Phillips Brooks.* The Mythical Man-Month: Essays on Software Engineering. *Addison-Wesley Publishing Company, 1995.*

[BTS+07] *Daniel Blankenberg, James Taylor, Ian Schenck, Jianbin He, Yi Zhang, Matthew Ghent, Narayanan Veeraraghavan, Istvan Albert, Webb Miller, Kateryna D Makova, Ross C Hardison, and Anton Nekrutenko. A framework for collaborative analysis of ENCODE data: making large-scale analyses biologist-friendly.* Genome Research*, 17(6):960–964, June 2007. PMID: 17568012.*

[Bur08] *Joshua Burton. N50 - ArachneWiki. http://www.broadinstitute.org/crd/wiki/index.php/N50, 2008.*

[BWGP01] *Alan Blackwell, Kirsten Whitley, Judith Good, and Marian Petre. Cognitive factors in programming with diagrams.* Artificial Intelligence Review*, 15(1/2):95–114, 2001.*

[CfWO12] *Tom Christiansen, brian d foy, Larry Wall, and Jon Orwant.* Programming Perl: Unmatched Power for Text Processing and Scripting*. O'Reilly Media, Inc., April 2012.*

[CGH+06] *David Churches, Gabor Gombas, Andrew Harrison, Jason Maassen, Craig Robinson, Matthew Shields, Ian Taylor, and Ian Wang. Programming scientific and distributed workflow with triana services.* Concurrency and Computation: Practice and Experience*, 18(10):1021–1037, 2006.*

[CKN10] *Guy Cochrane, Ilene Karsch-Mizrachi, and Yasukazu Nakamura. The international nucleotide sequence database collaboration.* Nucleic Acids Research*, November 2010.*

[CPT11] *Phillip E. C. Compeau, Pavel A. Pevzner, and Glenn Tesler. How to apply de bruijn graphs to genome assembly.* Nature Biotechnology*, 29(11):987–991, 2011.*

[CTGR05] *A. G. Castro, S. Thoraval, L. Garcia, and M. Ragan. Workflows in bioinformatics: meta-analysis and prototype implementation of a workflow generator.* BMC bioinformatics*, 6(1):87, 2005.*

[DGST09] *Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-science: An overview of workflow system features and capabilities.* Future Generation Computer Systems*, 25(5):528–540, May 2009.*

[DSS+05] *E. Deelman, G. Singh, M. H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, and J. Good. Pegasus: A framework for mapping complex scientific workflows onto distributed systems.* Scientific Programming*, 13(3):219–237, 2005.*

[Dur98] *Richard Durbin.* Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, April 1998.*

[FPD+05] *T. Fahringer, R. Prodan, Rubing Duan, F. Nerieri, S. Podlipnig, Jun Qin, M. Siddiqui, Hong-Linh Truong, A. Villazon, and M. Wieczorek. ASKALON: a grid application development and computing environment. In* Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, GRID '05, page 122–131, Washington, DC, USA, 2005. IEEE Computer Society.*

[GDB+04] *Y. Gil, E. Deelman, J. Blythe, C. Kesselman, and H. Tangmunarunkit. Artificial intelligence and grids: Workflow planning and beyond.* Intelligent Systems, IEEE*, 19(1):26–33, 2004.*

[Gil02] *Don Gilbert. Pise: Software for building bioinformatics webs.* Briefings in Bioinformatics*, 3(4):405–409, December 2002.*

[GNTT10] *J. Goecks, A. Nekrutenko, J. Taylor, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences.* Genome Biol*, 11(8), 2010.*

[GP96] *T. R. G. Green and M. Petre. Usability analysis of visual programming environments: a 'cognitive dimensions' framework.* JOURNAL OF VISUAL LANGUAGES AND COMPUTING*, 7:131–174, 1996.*

[GR83] *Adele Goldberg and David Robson.* Smalltalk-80: the language and its implementation*. Addison-Wesley, 1983.*

[GRH+05] *Belinda Giardine, Cathy Riemer, Ross C Hardison, Richard Burhans, Laura Elnitski, Prachi Shah, Yi Zhang, Daniel Blankenberg, Istvan Albert, James Taylor, Webb Miller, W James Kent, and Anton Nekrutenko. Galaxy: a platform for interactive large-scale genome analysis.* Genome Research*, 15(10):1451–1455, October 2005. PMID: 16169926.*

[GT79] *Kim P. Gostelow and Robert E. Thomas. A view of dataflow. In* Managing Requirements Knowledge, International Workshop on*, volume 0, page 629, Los Alamitos, CA, USA, 1979. IEEE Computer Society.*

[Hil92] *Daniel D Hils. Visual languages and computing survey: Data flow visual programming languages.* Journal of Visual Languages & Computing*, 3(1):69–101, March 1992.*

[Hol95] *D. Hollingsworth. Workflow management coalition: The workflow reference model.* Document Number TC00-1003*, (1.1), 1995.*

[HWS+06] *D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M.R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services.* Nucleic acids research*, 34(suppl 2):W729, 2006.*

[ISO11] *ISO JTC 1/SC 22. C programming language specification. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=57853, 2011.*

[KMR08] *G. Kandaswamy, A. Mandal, and D. A Reed. Fault tolerance and recovery of scientific workflows on computational grids. In* 8th IEEE International Symposium on Cluster Computing and the Grid, 2008. CCGRID '08*, pages 777–782. IEEE, May 2008.*

[LBB+07] *M A Larkin, G Blackshields, N P Brown, R Chenna, P A McGettigan, H McWilliam, F Valentin, I M Wallace, A Wilm, R Lopez, J D Thompson, T J Gibson, and D G Higgins. Clustal w and clustal x version 2.0.* Bioinformatics (Oxford, England)*, 23(21):2947–2948, November 2007. PMID: 17846036.*

[LP95] *E.A. Lee and T.M. Parks. Dataflow process networks.* Proceedings of the IEEE*, 83(5):773 –801, May 1995.*

[MA05] *Jane E Mabey and Teresa K Attwood. Report on the EMBER project – a european multimedia bioinformatics educational resource. http://www.bioscience.heacademy.ac.uk/journal/vol6/beej-6-4.aspx, 2005.*

[MBZL09] *Timothy McPhillips, Shawn Bowers, Daniel Zinn, and Bertram Ludäscher. Scientific workflow design for mere mortals.* Future Generation Computer Systems*, 25(5):541–551, May 2009.*

[MYA+04] *S. McGough, L. Young, A. Afzal, S. Newhouse, and J. Darlington. Workflow enactment in ICENI. In* UK e-Science All Hands Meeting*, page 894–900. Citeseer, 2004.*

[Nat12] *National Human Genome Research Institute. DNA sequencing costs. http://www.genome.gov/sequencingcosts/, 2012.*

[Nek11] *Anton Nekrutenko. Introduction to galaxy, May 2011.*

[NMM+09] *Bertrand Néron, Hervé Ménager, Corinne Maufrais, Nicolas Joly, Julien Maupetit, Sébastien Letort, Sébastien Carrere, Pierre Tuffery, and Catherine Letondal. Mobyle: A new full web bioinformatics framework.* Bioinformatics*, 25(22):3005–3011, November 2009.*

[OAF+04] *T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, M.R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows.* Bioinformatics*, 2004.*

[OCG+10] *Joshua Orvis, Jonathan Crabtree, Kevin Galens, Aaron Gussman, Jason M. Inman, Eduardo Lee, Sreenath Nampally, David Riley, Jaideep P. Sundaram, Victor Felix, Brett Whitty, Anup Mahurkar, Jennifer Wortman, Owen White, and Samuel V. Angiuoli. Ergatis:*

A web interface and scalable software system for bioinformatics workflows. Bioinformatics, *26(12):1488–1492, June 2010.*

[Ora11] *Oracle. JSR 336: Java(TM) SE 7 release contents. http://www.jcp.org/en/jsr/detail?id=336, 2011.*

[Pan09] *Sumir Panji.* Identification of Bacterial Pathogenic Gene Classes Subject to Diversifying Selection. *Ph.D, University of the Western Cape, Bellville, South Africa, 2009.*

[Pip11] *Pipelines working group. Bio-Analysis pipelines. https://pipelines.sanbi.ac.za/index.php/Main_Page, 2011.*

[Pyt12] *Python Software Foundation. The python language reference. http://docs.python.org/reference/, 2012.*

[RLB$^+$00] *P. Rice, I. Longden, A. Bleasby, et al. EMBOSS: the european molecular biology open software suite.* Trends in genetics, *16(6):276–277, 2000.*

[SBB$^+$00] *Robert Stevens, Patricia Baker, Sean Bechhofer, Gary Ng, Alex Jacoby, Norman W. Paton, Carole A. Goble, and Andy Brass. TAMBIS: transparent access to multiple bioinformatics information sources.* Bioinformatics, *16(2):184–186, February 2000.*

[SC05] *Martín Sarachu and Marc Colet. wEMBOSS: a web interface for EMBOSS.* Bioinformatics, *21(4):540–541, February 2005.*

[SGBB01] *Robert Stevens, Carole Goble, Patricia Baker, and Andy Brass. A classification of tasks in bioinformatics.* Bioinformatics, *17(2):180–188, February 2001.*

[SJ08] *Jay Shendure and Hanlee Ji. Next-generation DNA sequencing.* Nature Biotechnology, *26(10):1135–1145, 2008.*

[Sou10] *South African National Bioinformatics Institute. South african national bioinformatics institute 2010 annual report, 2010.*

[SPO12] *Simon P. Sadedin, Bernard Pope, and Alicia Oshlack. Bpipe: A tool for running and managing bioinformatics pipelines.* Bioinformatics, *April 2012.*

[Ste96] *Lincoln D. Stein. How perl saved the human genome project. http://www.bioperl.org/wiki/How_Perl_saved_human_genome, 1996.*

[Ste08] *L.D. Stein. Towards a cyberinfrastructure for the biological sciences: progress, visions and challenges.* Nature Reviews Genetics, *9(9):678–688, 2008.*

[SZVB12] *Marcel H. Schulz, Daniel R. Zerbino, Martin Vingron, and Ewan Birney. Oases: Robust de novo RNA-Seq assembly across the dynamic range of expression levels.* Bioinformatics*, February 2012.*

[Tar11] *Michael Tartakovsky. Cyber infrastructure approach for the next-generation sequencing at the national institute of allergy and infectious diseases, December 2011.*

[The11] *The Galaxy Team. Galaxy tool shed. http://toolshed.g2.bx.psu.edu/, 2011.*

[The12] *The Galaxy Team. The galaxy project: Online bioinformatics analysis for everyone. http://galaxy.psu.edu/, 2012.*

[vDATHKB03] *W. M.P van Der Aalst, A. H.M Ter Hofstede, B. Kiepuszewski, and A. P Barros. Workflow patterns.* Distributed and parallel databases*, 14(1):5–51, 2003.*

[VEL10] *Perla Velasco Elizondo and Kung-Kiu Lau. A catalogue of component connectors to support development with reuse.* Journal of Systems and Software*, 83(7):1165–1178, July 2010.*

[WBS+09] *Rui Wang, Douglas Brewer, Shefali Shastri, Srikalyan Swayampakula, John A. Miller, Eileen T. Kraemer, and Jessica C. Kissinger. Adapting the galaxy bioinformatics tool to support semantic web service composition. In* Proceedings of the 2009 Congress on Services - I*, SERVICES '09, page 283–290, Washington, DC, USA, 2009. IEEE Computer Society.*

[Woo10] *P. Woollard. Asking complex questions of the genome without programming.* Methods Mol. Biol*, page 39–52, 2010.*

[Wor99] *Workflow Management Coalition. Workflow management coalition workflow standard - interoperability abstract specification, 1999.*

[XH09] *Qing-Wei Xu and Yu Huang. BiosFlow - a bioinformatics workflow platform based on semantic web technology. In* BioMedical Information Engineering, 2009. FBIE 2009. International Conference on Future*, pages 37 –40, December 2009.*

[Yan07] *Ziheng Yang. PAML 4: Phylogenetic analysis by maximum likelihood.* Molecular Biology and Evolution*, 24(8):1586–1591, August 2007.*

[YB05] *J Yu and R Buyya. A taxonomy of workflow management systems for grid computing.* Journal of Grid Computing*, 3(3):171–200, 2005.*

[YGN09a] *U. Yildiz, A. Guabtni, and A.H.H. Ngu. Business versus scientific workflows: A comparative study. In* Services - I, 2009 World Conference on*, pages 340 –343, July 2009.*

[YGN09b] *Ustun Yildiz, Adnene Guabtni, and Anne H. H. Ngu. Towards scientific workflow patterns. In* Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, WORKS '09, page 13:1–13:10, New York, NY, USA, 2009. ACM.*

[ZB08] *Daniel Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read assembly using de bruijn graphs.* Genome Research*, January 2008.*

Appendices

A. Positive selection workflow

Figure A.1 illustrates Dr Panji's workflow with datasets as nodes in the graph, in other words this is a dataflow description of the workflow. This diagram serves to expand the discussion in 2.2.2 of the workflow as a series of transformations of data collections. Datasets form the nodes of the diagram, and analyses are the edges that "carry" data from one collection to the next.

By contrast A.2 shows the workflow with analyses as nodes and datasets as edges. In other words, this is a control flow diagram of the dataset. This view is in some ways similar to the way that Galaxy displays a workflow, in that the nodes are analyses and the edges are datasets that flow from one analysis to the next.

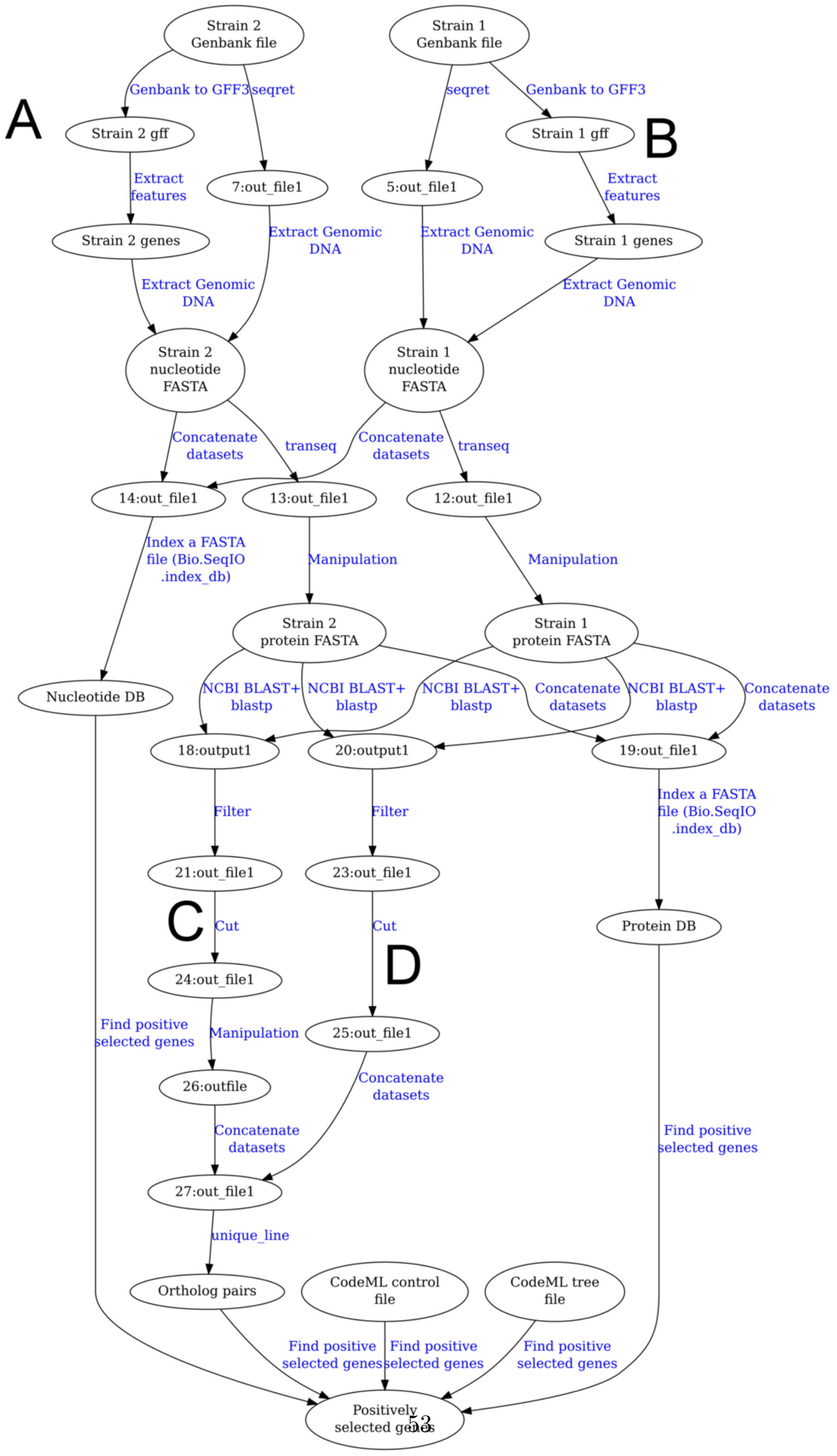


Figure A.1.: Positively selected genes workflow with analyses as nodes

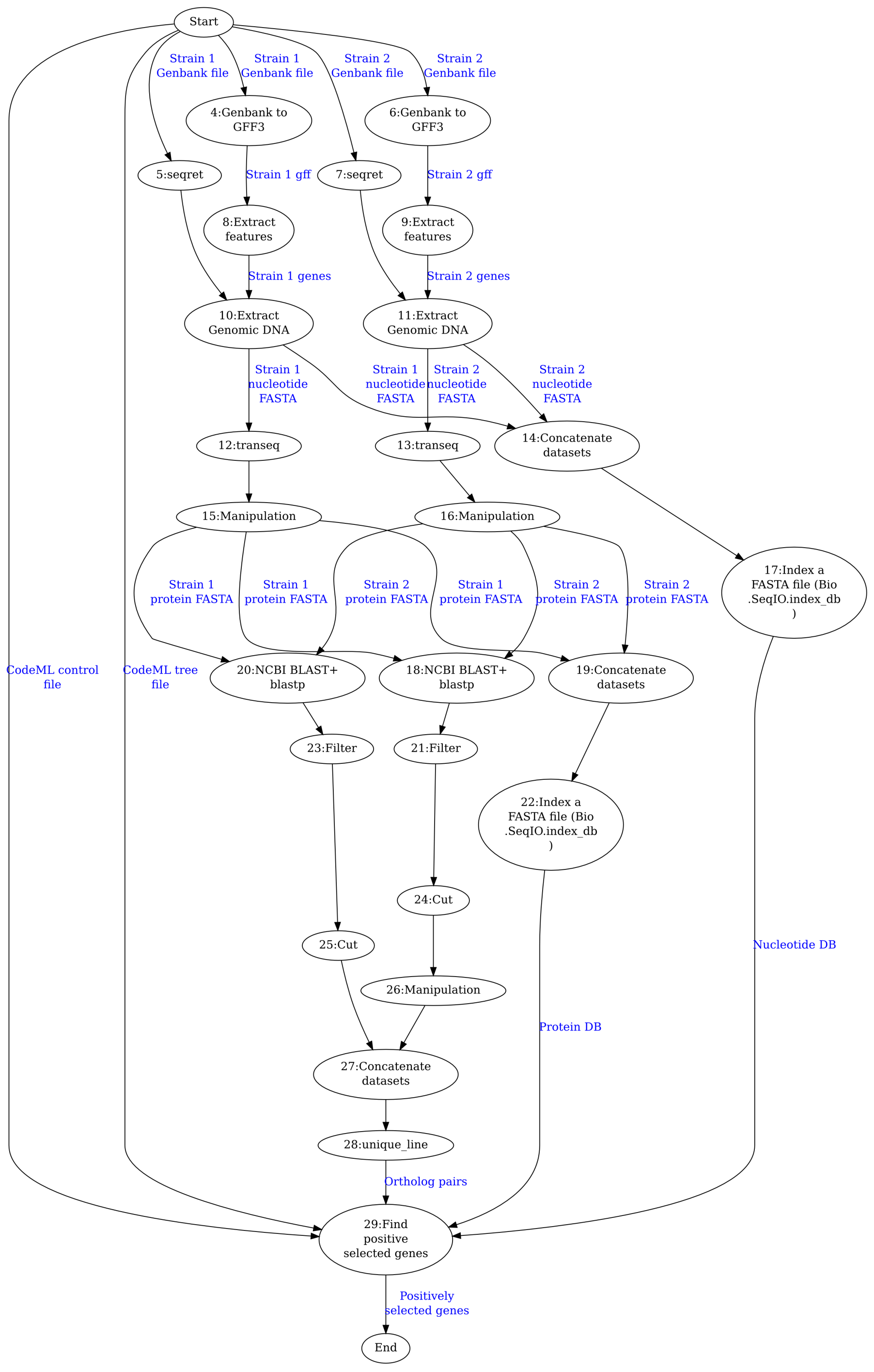


Figure A.2.: Positively selected genes workflow with analyses as nodes

B. Velvet assembly parameter sweep

*Stanley Mbandi-Kimung's workflow for exploring the **velvet** and **oases** parameter space.*

```
#!/usr/bin/perl

# AUTHOR:Mbandi 2011
# LAST REVISED: 2012
# The Regents of christoffels_lab
#South African National Bioinformatics Institute

#Runs velveth followed by oases for multiple values of k and calculates N50 sta
#File must be in fastA format else change -fasta to -fastq for fastQ files
use strict;
use warnings;

if (@ARGV!= 5) {
        print STDERR "usage: perl symbolic.velveth.pl shuffle.file.fa single.fi
        print STDERR "kmer2 must be and odd number\n";
        exit(1)
}
my $time = time;
my $shuffle = $ARGV[0];
my $single = $ARGV[1];
my $klow = $ARGV[2];
my $khigh = $ARGV[3];
my $stats = $ARGV[4];

if($khigh < $klow) {
        print STDERR "$khigh must be greater than $klow\n";
        exit(1)
}
$khigh++;

#initiates velveth for preparing files required by velvetg
system "velveth dir $klow,$khigh,2 -fasta -shortPaired $shuffle -short $single"

for(my $i=$klow; $i < $khigh; $i +=2) {
        #de bruign creation
        system"velvetg dir_$i  -unused_reads yes -read_trkg yes -scaffolding no
        #-min_contig_lgth 100

        #transcript reconstruction
        system "oases dir_$i -amos_file yes -unused_reads yes -scaffolding no -
        #-ins_length 300

        #assigns directory contig.fa file associated with  each k iteration
        my $contig = "./dir_$i/transcripts.fa";
        my ($len,$total)=(0,0);
        my @x;
        open (F,  $contig) || die "$contig does not exist\n";
        open (E, " >>$stats") || die "$stats does not exist\n";
        while(<F>){
                if(/^[\>\@]/){
                        if($len>0){
                                $total+=$len;
                                push @x,$len;
                        }
                        $len=0;
                }
                else{
                        s/\s//g;
                        $len+=length($_);
                }
        }
        if ($len>0){
                $total+=$len;
                push @x,$len;
        }
        @x=sort{$b<=>$a} @x;
        my ($count,$half)=(0,0);
        for (my $j=0;$j<@x;$j++){
                $count+=$x[$j];
                if (($count>=$total/2)&&($half==0)){
                        print E "$contig\tN50: $x[$j]\n";
                        $half=$x[$j]
                }
        }
        close F;
        close E;
}
$time = time - $time;

if($time < 60) {
        print  "Total␣time␣elapsed:␣$time␣secs.\n";
}
elsif (($time >= 60) && ($time < 3600)) {
        $time = $time/60;
        print  "Total␣time␣elapsed:␣$time␣mins.\n";
}
else {
        $time = $time/3600;
        print  "Total␣time␣elapse:␣$time␣hours.\n";
}
```