AutoFlow: an easy way to build workflows

Pedro Seoane, Rosario Carmona, Rocío Bautista, Darío Guerrero-Fernández y M. Gonzalo Claros

Plataforma Andaluza de Bioinformática & Dpto de Biología Molecular y Bioquímica, Universidad de Málaga, 29071 Málaga (Spain)

Abstract. Many bioinformatics tasks require the use of different software, making workflows a current need in this research field. There are workflow builders that usually try to simplify the interface disregarding a complex use. This may lead to a non-scalability limitation, or the dependence on the facilities available. Here it is presented AutoFlow, a workflow builder that can handle most computer systems. It has been developed in Ruby and accepts any kind of software that can even use very specific resources (such as GPU or FPGA). AutoFlow has been designed to automatically launch tasks to the queue system. It can then handle big workflows that can overflow the maximum execution time of the queue system provided that each individual task can be finished within the maximum execution time. AutoFlow has been implemented with iterative task capability. Other interesting capability is an environment variable system that allows the persistence of certain data for all tasks. This allows the data transfer from a task to the next task and so on, enabling the inclusion of decisions that can affect downstream tasks. AutoFlow includes tools to monitor task status, graphic representations of workflows, file searching and timing. Two case-of use are presented to illustrate AutoFlow capabilities: one workflow for assembling and annotation of several libraries of Roche 454 sequences and another workflow for RNA-seq analysis.

1 Introduction

The high-throughput sequencing technologies produce a large amount of data that require the development of large and complex workflows with lots of instructions. For example, genome sequencing generates large files of reads that must be pre-processed, assembled, verified, and then annotated. Typically, assembly and annotation can be performed using various programs and parameters to obtain different results that require further reconciliation or selection. Some way of automation of this repetitive task will be beneficial. Moreover, when selection is required, downstream tools are depending on this decision, providing different results. Therefore, a workflow or a pipeline is designed to encompass all the programs and parameters used to get the final result, and allows the user to save time and efforts by not having to launch the different tasks on his own.

Pipelining tasks allows to automatise the use different software tools, where the output data of a tool is used as input for other tools, or different input files must converge to provide a single, final result. Nowadays, many platforms are able to build and execute workflows, such as Ergatis[1], Kepler[2], Triana[3], or Dyscovery Net[4], although the most widely used are Galaxy[5], Taverna[6] and their fusion, known as Tavaxy[7]. Both platforms are based on web services that users can combine to design and execute customised workflows. The above platforms have been designed to simplify the creation and execution of workflows, so that users without computing skills can use them. Among the most noticeable features, these platforms have a graphical user interface (GUI) that enables the user to design and execute workflows. Nevertheless, the usability is limiting the flexibility, and complex workflows or workflows with new software are no easy to handle. Furthermore, the user has not control on used resources and this can diminish the workflow performance.

Here it is presented AutoFlow, a Ruby-based workflow manager that allows building any desired workflow or pipeline. It is self-contained and the only dependences are GNUplot and dot[8]. It enables execution control by the user. It allows the design of dynamic workflows that can take decisions about the data while the workflow is running. Its main goal is to simplify repetitive tasks while removing limitations inherent to other workflow tools. On the other hand, it requires that users must have computing skills and knowledge about the software to be used.

2 Methods

2.1 Implementation of tasks

AutoFlow is a ruby gem that has been developed on Ruby 1.9 and SLES Linux. It uses a template script where every task is described with all its attributes and afterwards it is launched to the queue system (in this case, we use a module sentence of SLURM queue system, but other queue systems can be implemented). Each task is identified by a unique tag, which will be used for reference purposes and graphical representation. The general structure of a task is written as:

```
listing){
          module load software
          ?
          ls folder
}
```

where the first line is the task name or tag (i.e. *listing*) finishing with a ')' character. This tag will be used to identify the task in graphical representations (see below). The task definition, written between '{}', has two parts separated by a line starting by the control character '?'. The lines before '?' serve to initialise the environment, and the lines after '?' are the commands to be executed (i.e. *ls folder*). The first word of the first command is used as reference for the output storage. Since the sintax is bash-based, any software or platform based on command lines, such as Matlab, C, Ruby or Python, can be used.

Task iteration, changing only some parameter, can be easily done writing:

```
listing_[user;system;folder]){
        module load software
        ?
        ls (*)
}
```

where the parameter succession within brackets separated by semicolons (*[user;system;folder]*) serves to create a new task for each parameter. This is done replacing the '(*)' tag by each parameter. This is very useful for software benchmarkings and searching the best parameters in a program with a particular dataset.

When a task is depending on finishing a previous task, it can be declared as follows:

```
listing){
        module load software
        ?
        ls folder > temp
}

show){
        module load software
        ?
        cat listing)/temp # it will not be launched until 'temp' is finished
}
```

The string variables implemented in AutoFlow are helpful in, for example, task decisions. They are alphanumeric strings that begin with '$' or '@' characters (i.e. *$sentence* in the following example).

```
message){
        module load software
        ?
        echo $sentence
}
```

The variables must be declared in the command line that launches the workflow, allowing to the user to change parameters in the workflow without modifying the template. Therefore, the workflow is completely independent of the used data. '$' character indicates that the variable will not be changed along the workflow. On the contrary, variables beginning with '@' can be modified. The '@' variables are very useful to transfer certain data between tasks and it helps in making-decisions.

2.2 Launching workflows

Workflows can be launched as follows:

```
Autoflow -w template
```

where -w indicates that *template* is the input data. This is the only mandatory parameter.

Before launching a new workflow, it must be checked to find errors and inconsistencies. The command line option *--graph* generates a graphical representation of the workflow, where the tasks are the nodes and the relations are the dependencies. The representation can be semantic (Fig. 1 and Fig. 2A) or structural

(Fig. 2B): in the semantic representation, the tasks are represented with their tags, allowing to the user to interpret the workflow easily. The structural representation uses the main command name of every task. These plots show which programs are used in the workflow and where the data are saved by each task. As a result, relation inconsistencies are becoming apparent, facilitating the identification and fixing by the user.

Another command line option, --*verbose* that generates a list of tasks with their attributes on command line, can also be helpful in debugging.

When a template is launched, AutoFlow generates a job for every task in the queue system. To do so, a folder per task is created with the name of their main command (as in the structural representation, Fig. 2B). All the folders are saved within the default folder *exec* where AutoFlow is running. Within each folder, a bash file is created with all necessary information for the queue system. AutoFlow replaces all key characters by their values and all the dependencies by absolute paths. Then, each bash file is sent to the queue system and AutoFlow takes its job ID that is used to control dependencies (if any) and the task launch timing. AutoFlow ends his work when the last task is queued.

Three additional information state files are created: (1) a log file containing the start and the end of each task; (2) a file containing the relations between tag task and where has it been saved; and (3) a file where all dynamic variables (that begins with '@' character) are defined. This file is loaded by all the task and it is created only if there is set a dynamic variable in the template.

2.3 Experimental data

Two dataset have been used in this work to illustrate AutoFlow capabilities: (i) for Case 1, a 454 dataset of two different tissues: A (372 750 reads) and B (429 909 reads), and (ii) for Case 2, an Illumina dataset of two different conditions (control and treatment) with three experimental replicates each one (experimental replicates have 2 457 983, 2 866 872 and 988 173 reads, and controls have ≈ 3 000 000 reads).

3 Results

3.1 Case 1: sequence assembly and annotation

The 454 dataset has been used to obtain three different assemblies with interconnected tasks: one for each tissue and another one with the whole dataset. The three assemblies can be obtained with the same template (http://www.scbi.uma.es/web/pedro/assembly_annotation.txt). This workflow was launched as follows:

```
Autoflow -w assembly_annotation -s -c 100 -u '-1' -t '5-
00:00:00'
```

The *-c* parameter says to the program how many cores must be used by each task. Previously, the user must replace the number of cores parameter in a command by the '*[cpu]*' string instead when he writes the template. *-s* says to Autoflow that tasks can use different nodes and *-u '-1'* that queue system must assign the best number of nodes to do the job. Besides, *-t* indicates the limit of time per task (in this case 5 days) .

The process described in Fig 1 allows the evaluation of the assembly quality based on the number of unigenes with orthologue, the number of unique unigenes, and the number of unique full length unigenes (Table 1). The manual inspection of Table 1 allows the selection of column "All-kmer29" as the better assembly, and these unigenes can be submitted to a full annotation with Sma3 [13]. However, based on these results, a new decision task was included (results not shown) to automatically select the best assembly (All-kmer29) that will be directed to Sma3 annotation that provided 34 971 annotated sequences. Sma3s lacks parallelisation capabilities but can be executed taking advantage of array jobs. Although AutoFlow cannot handle array jobs as such, it can be included within a wrapped easily created with SCBI_MapReduce[14] to confer parallelization capabilities to Sma3s. In future workflows, a wrapped Sma3 can be included in AutoFlow workflows.

In conclusion, this case-of-use of AutoFlow takes advantage of two features of AutoFlow: (1) its capability for using results generated in previous executions and (2) the capability of taking decisions when the workflow is running.

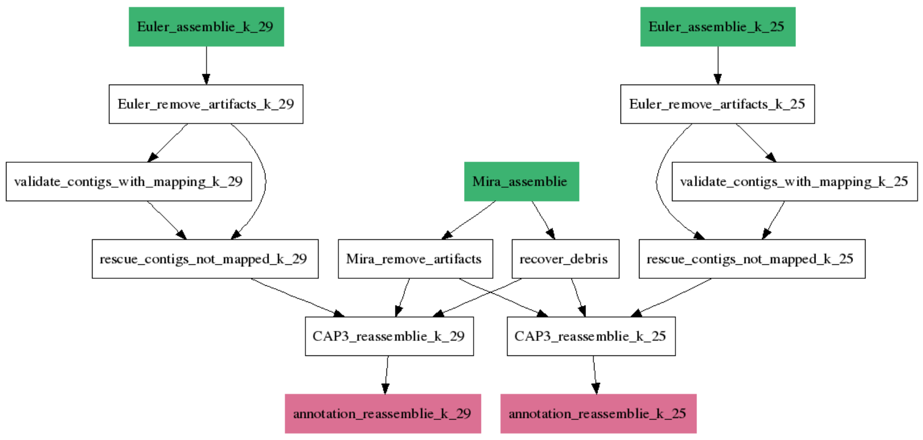

Euler_assemblie_k_29
Euler_assemblie_k_25
Euler_remove_artifacts_k_29
Euler_remove_artifacts_k_25
validate_contigs_with_mapping_k_29
Mira_assemblie
validate_contigs_with_mapping_k_25
rescue_contigs_not_mapped_k_29
Mira_remove_artifacts
recover_debris
rescue_contigs_not_mapped_k_25
CAP3_reassemblie_k_29
CAP3_reassemblie_k_25
annotation_reassemblie_k_29
annotation_reassemblie_k_25

Fig. 1. Semantic representation of Case 1 workflow. Green boxes correspond to starting tasks, and magenta boxes are the finishing tasks. Right and left branches of the workflow differ on the assembly k-mer used by Euler[10]. Central branch correspond to an assembly using Mira[11]. Initial assemblies are analysed to recover putative useful «debris» and to remove artefactual contigs using Full-LengtherNext and Bowtie[12], respectively. CAP3[9] is used to reconcile Euler and Mira verified contigs, and the resulting supercontigs are analysed using Ful-LengtherNext to obtain data of Table 1.

Table 1. Key parameters of Case 1 workflow

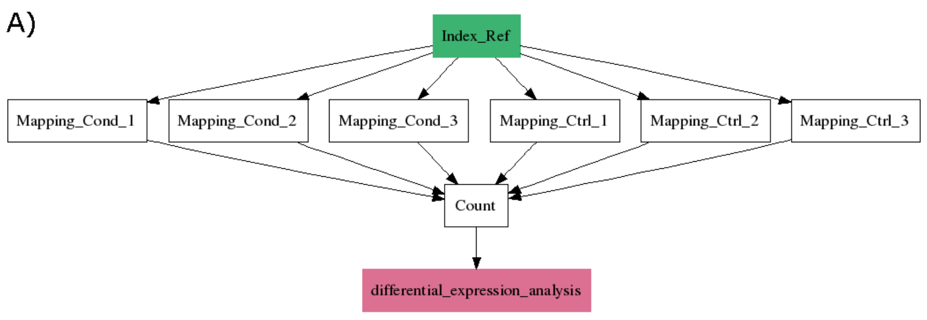
	Tissue A		Tissue B		All	
	kmer 25	kmer 29	kmer 25	kmer 29	kmer 25	kmer 29
Unigenes	37283	37235	14178	14191	44837	44858
Unique unigenes	13775	13786	5962	5955	15328	15307
Unique full-length unigenes	3882	3942	1512	1504	4736	4765

3.2 Case 2: RNA-seq analyses

This workflow (Fig. 2; http://www.scbi.uma.es/web/pedro/expression_analysis.txt) is a simple example that takes advantage of iterative capabilities of AutoFlow. The launching command was:

```
Autoflow -w template -c 4 -V '$evalue=p-value'
```

This command line has the new parameter *-V* that is used by the user for set internal variables to desired values. The same workflow with the same data has been launched three times, setting the static variable *$evalue* to 0.01, 0.05, and 0.1. The *$evalue* variable is the cut-off *P*-value that uses our in-house pipeline for analysing RNA-seq data to determine the statistic significance. The results were 2 780, 2 937 and 2 995 differentially expressed genes, respectively. As expected, the lower the *P*-value, the greater the number of genes. More sophisticated, iterative analyses with AutoFlow could help in the determination of the best *P*-value threshold without the need of human intervention.



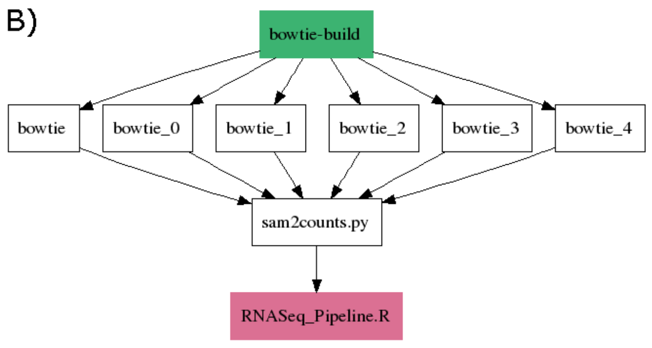


Fig. 2. Semantic (A) and structural (B) representation of the RNA-seq analysis workflow of Case 2. The workflow first creates a mapping reference index (*Index_ref*), then makes the mapping with the control and conditions using Bowtie2 (*Mapping_Cond_x* and *Mapping_Ctrl_x*), then makes the stats (*Count*) using the Python script sam2counts.py, and finally makes the differential expression analysis (*differential_expression_analysis*) with our in-house RNASeq_Pipeline.R script.

4 Discussion

AutoFlow is an easy-to-use and efficient workflow manager for researchers with programming skills but without expertise in workflow design. This tool can manage large and complex workflows, or can simplify repetitive workflows taking advantage of the automatic management of iterative tasks.

The parameters and steps of workflow of Case 1 was fine-tuned using different input data (results not shown). This demonstrates that the same workflow can be used with different input data, allowing its re-use for several different experiments. Moreover, the same workflow can be shared among laboratories provided that they have the same software and queue system. Therefore, when several researches in a project need to face data from different sources (laboratories) or organisms (animals, plants, microorganisms...), they can use the same AutoFlow workflow changing or tuning some parametres or programs.

As is shown in Figs. 1 and 2, and since AutoFlow is not limited by any database or ontology, it allows the incorporation of any desired software in a workflow, provided that the software works as a command line tool. Tools programmed in different languages (C, C++, Ruby, Perl, Python, R, Java, etc.) can be combined without any problem.

Finally, since AutoFlow is using bash for building the tasks and the environment variables system, it contains the flexibility and power of command lines and scripts. Consequently, the workflows can be dynamic and can take decisions when tasks are on-going.

References

1. Orvis J, Crabtree J, Galens K, Gussman A, Inman JM, Lee E, Nampally S, Riley D, Sundaram JP, Felix V, Whitty B, Mahurkar A, Wortman J, White O, Angiuoli SV: Ergatis: A web interface and scalable software system for bioinformatics workflows. Bioinformatics 26 (12). (2010) 1488-1492
2. Ludäscher B, Altintas I, Berkley C. D H: Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience 13(10) (2006) 1039–1065
3. Taylor I, Shields M, Wang I, Harrison A: The Triana workflow environment: Architecture and Applications. In: Workflows for e-Science, Springer (2007) 320–339.
4. Ghanem M, Curcin V, Wendel P, Guo Y. Building and using analytical workflows in discovery net. In: Data mining on the Grid, John Wiley and Sons (2008).
5. Hull D., Wolstencroft K., Stevens R., Goble C., Pocock M. R., Li P., Oinn T.: Taverna: a tool for building and running workflows of services. Nucl. Acids Res. 34 (suppl 2). (2006) W729-W732
6. Goecks J., Nekrutenko A., Taylor J. and The Galaxy Team: Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biology 11(8). (2010) R86
7. Abouelhoda M., Issa S. A., Ghanem M.: Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support. BMC Bioinformatics 13 (2012) 77
8. Ellson J. , Gansner E. R., Koutsofios E., North S. C., Woodhull G.: Graphviz and dynagraph – static and dynamic graph drawing tools. In: Graph Drawing Software. (2004) 127-148
9. Huang, X. and Madan, A.: CAP3: A DNA sequence assembly program. Genome Res 9. (1999) 868-877.
10. Pevzner PA., Tang H., Waterman MS.: An Eulerian path approach to DNA fragment assembly. PNAS 18(17) (2001) 9748–9753
11. Chevreux, B., Wetter, T. and Suhai, S.: Genome sequence assembly using trace signals and additional sequence information. In: Computer Science and Biology: Proceedings of the German Conference on Bioinformatics. (1999) 45-56.
12. Langmead B., Salzberg S.: Fast gapped-read alignment with Bowtie 2. Nature Methods 9. (2012) 357-359.
13. Muñoz-Mérida A, Viguera E., Claros M. G., Trelles O., Pérez-Pulido A.J.: Sma3s: a three-step modular annotator for large sequence datasets. DNA Research. In press. (2014)
14. Guerrero-Fernández D., Falgueras J., and Claros M. G.: SCBI_MapReduce, a New Ruby Task-Farm Skeleton for Automated Parallelisation and Distribution in Chunks of Sequences: The Implementation of a Boosted Blast+. Computational Biology Journal 2013 (2013) ID 707540