Bioinformatics Advance Access published December 6, 2012

nestly— a framework for running software with nested parameter choices and aggregating results

Connor O. McCoy [1,*], Aaron Gallagher [1], Noah G. Hoffman [2] and Frederick A. Matsen [1*]

[1]Program in Computational Biology, Fred Hutchinson Cancer Research Center, Seattle, WA 98109
[2]Department of Laboratory Medicine, University of Washington, Seattle, WA 98195

Associate Editor: Prof. Alfonso Valencia

ABSTRACT
Summary: The execution of a software application or pipeline using various combinations of parameters and inputs is a common task in bioinformatics. In the absence of a specialized tool to organize, streamline, and formalize this process, scientists must write frequently complex scripts to perform these tasks.

We present `nestly`, a Python package to facilitate running tools with nested combinations of parameters and inputs. `nestly` provides three components: first, a module to build nested directory structures corresponding to choices of parameters. Second, the `nestrun` script to run a given command using each set of parameter choices. Third, the `nestagg` script to aggregate results of the individual runs into a CSV file, as well as support for more complex aggregation. We also include a module for easily specifying nested dependencies for the SCons build tool, enabling incremental builds.

Availability and implementation:
Source, documentation, and tutorial examples are available at http://github.com/fhcrc/nestly. `nestly` can be installed from the Python Package Index via `pip`; it is open source (MIT license).
Contact: cmccoy@fhcrc.org or matsen@fhcrc.org

1 INTRODUCTION

Many types of bioinformatics analyses involve running software or pipelines with a combination of a number of different parameters and inputs. For example, one may want to use an established algorithm such as BLAST with many combinations of different parameters, or benchmark the accuracy and speed of a new algorithm for all possible settings for a number of parameters. While a number of tools are available for streamlining bioinformatics pipelines (Goodstadt, 2010; Köster and Rahmann, 2012; Sadedin, S.P. *et al.*, 2012), these tools emphasize running a dataset through a workflow with a static parameter regime.

The task of running software with a variety of settings typically involves enumeration of parameters, execution of a command for each combination, and aggregation of results. Despite the ubiquity of this sort of task in bioinformatics, we are only aware of software packages that address it for specific bioinformatics problems (Darriba *et al.*, 2012; Hoekman *et al.*, 2012). The most common solution is to write a custom script to perform the tasks, which presents difficulties: isolating execution, running commands in parallel, and merging results with associated parameters can be non-trivial. Managing these tasks in one-off code creates additional work for researchers, introduces potential for error, and hinders reproducibility.

We have designed a Python software package, `nestly`, the purpose of which is to streamline and automate these operations.

*to whom correspondence should be addressed

2 IMPLEMENTATION

`nestly` streamlines, organizes, and automates running programs with combinatorial parameter choices. Users write a script using `nestly`'s Python API for generating parameter choices. This API makes expressing combinatorial choices trivial; arbitrary logic may be used for more complex cases. Once parameters have been selected, `nestly` organizes choices in a nested directory structure, providing an isolated environment in which to run bioinformatics software for each combination. Parameter choices are recorded in JSON files in leaf directories.

The `nestrun` tool facilitates running a command for each parameter combination in parallel. Commands are executed within a shell, using a simple syntax for parameter substitution. The user can therefore create a direct call to a bioinformatics program for each combination, or a script written in an arbitrary language.

For the common case of programs with comma- and tab-delimited outputs, the `nestagg` tool aggregates results from the nested directory structure, appending the parameters used to generate the results to each row.

For incremental builds, we also provide SCons integration allowing concise specification of nested dependencies. The `nestly` SCons module provides methods for adding a target for every combination of parameters, and aggregating across these targets once they have been created or updated. This feature has been particularly useful for avoiding the repeated execution of time consuming upstream steps when developing downstream components of an analysis.

3 EXAMPLE OF USE

3.1 Enumerating combinations

As an example we use `nestly` to set up a comparison of two implementations of an algorithm to prune leaves from a phylogenetic tree to minimize an objective function (Matsen, F.A. *et al.*, 2012). The "full" implementation solves the problem exactly, while the "pam" implementation uses a heuristic. We wish to run both algorithms on several 1000-leaf trees, varying the number of leaves pruned from the trees ("k").

To set up a nested directory structure:

```
import glob
from nestly import Nest, stripext

n = Nest()
n.add('algorithm', ['full', 'pam'])
n.add('tree', glob.glob('/trees/*.tre'),
      label_func=stripext)
n.add('k', [50, 100, 250, 500, 750, 900, 950])
n.build('runs')
```

This will do the following. It will create a `runs` directory, in which there will be two subdirectories, `full` and `pam`. Within each of these there will be directories labeled with the name of the tree. Within each of those, there will be directories for each value of "k", e.g. `50`, `100`, etc. Within each of those, there will be a JSON file `control.json`, which contains the parameter values corresponding to the directory hierarchy.

This example exhibits nestly on a parameter "Cartesian product;" online examples show how to use it to build a more complex subset.

3.2 Running a command

The JSON files thus created then serve as inputs to `nestrun` for template substitution, for example:

```
nestrun -d runs
  --template='rppr min_adcl_tree {tree}
              --algorithm {algorithm}
              --all-adcls-file adcls.csv
              --leaves {k}'
```

This command runs `rppr min_adcl_tree` in all of the tip directories with the appropriate values for each parameter, for example substituting 100 in place of {`k`} above. The results we're interested in (`--all-adcls-file`) will be written to a file named `adcls.csv` in each directory. `nestrun` can run a user-specified number of jobs in parallel.

3.3 Aggregating results

Given a list of JSON files and the name of the delimited files to combine, `nestagg` merges the results together into a single file, adding columns that label the parameter choices used when generating those results. The call:

```
nestagg delim adcls.csv -d runs -o all_adcls.csv
```

will aggregate data from all of the CSV files named `adcl.csv`, one for each control file under `runs`, appending columns containing the values of algorithm, tree, and k from the control file.

For more general cases, a simple framework is available for defining custom aggregations.

3.4 SCons integration

SCons (http://scons.org) is a build system implemented in Python that is analogous to `make`, in that an invocation only runs what is necessary to fulfill specified dependencies. Our `nestly` integration makes it easy to define targets for each parameter combination that also become part of the control dictionary.

For example, by including the code from Section 3.1 in an `SConstruct` file, we can add our command (abbreviated here):

```
from nestly.scons import SConsWrap
w = SConsWrap(nest, 'build')
@w.add_target()
def min_adcl(d, control):
  # set SCons ‘action‘ to template as done
  # above using values from ‘control‘
  return Command(os.path.join(d, 'adcl.csv'),
                 c['tree'], action)
```

This will execute the same combination of commands as before, but within the incremental build framework of the SCons build tool. The result of this step is then available via the control dictionary for future steps. The SCons integration also has facilities for aggregating results, described in the `nestly` documentation.

4 CONCLUSION

`nestly` is a simple yet powerful framework for running software with combinatorial choices of parameters and aggregating results of those runs. It has complete documentation and a suite of examples.

ACKNOWLEDGEMENTS

We thank Brian Hodges for contributing to the first version of `nestrun`, and to Brad Chapman and two anonymous reviewers for valuable comments on this manuscript.

Funding: A.G., N.H., and F.M are supported in part by NIH R01 HG005966-01; C.M. is supported in part by NIH R01 AI038518.

Conflict of Interest: none declared.

REFERENCES

Darriba, D., Taboada, G., Doallo, R., and Posada, D. (2012). jModelTest 2: more models, new heuristics and parallel computing. *Nature Methods*, **9**(8), 772–772.

Goodstadt, L. (2010). Ruffus: a lightweight python library for computational pipelines. *Bioinformatics*, **26**(21), 2778–2779.

Hoekman, B., Breitling, R., Suits, F., Bischoff, R., and Horvatovich, P. (2012). msCompare: a framework for quantitative analysis of label-free LC-MS data for comparative biomarker studies. *Molecular & Cellular Proteomics*.

Köster, J. and Rahmann, S. (2012). Snakemake–a scalable bioinformatics workflow engine. *Bioinformatics*.

Matsen, F.A. *et al.* (2012). Minimizing the average distance to a closest leaf in a phylogenetic tree. *Arxiv preprint arXiv:1205.6867*.

Sadedin, S.P. *et al.* (2012). Bpipe: a tool for running and managing bioinformatics pipelines. *Bioinformatics*, **28**(11), 1525–1526.