TRAPPIST:
A toolkit for comparative analysis and visualization of genomic regions

Geraldine A. Van der Auwera, PhD

https://github.com/gglobster/trappist
TRAPPIST: A toolkit for comparative analysis and visualization of genomic regions

Geraldine A. Van der Auwera, PhD
Shankar Ambady

https://github.com/gglobster/trappist
Evolution = 4 bn years of forking without version tracking

... and you thought legacy Fortran code was a pain
Two distinct issues

- **Getting** the code from the repository (living beings)
- **Reverse-engineering** the code (zero documentation!)

Extraction, sequencing, assembly

Experimentation, mutagenesis + (comparative) sequence analysis
Top issue for “getting”

NGS is outscaling Moore’s Law

Lincoln Stein via C. Titus Brown @PyCon 2011
Top issue for “getting”

NGS is outscaling Moore’s Law

Lincoln Stein via C. Titus Brown @PyCon 2011
Evolving process of rev-eng

- No genomes ➔ entirely experimental
  - Make random mutants, trace back effect to gene of interest
- One genome ➔ some predictive filtering
  - Design mutants, long iterative process
- Many related genomes ➔ much better predictive filtering
  - Nature’s mutants, drastically reduced iterative process
Typical analysis process

All done through separate GUIs ➞ poor batching, no automation, no chaining
The servers can be accessed with scripts*, and there are awesome libraries that provide wrappers, data structures etc.

But here’s the rub…

* (there are a few GUI pipeline apps but usability is an issue)</diplomatic>
“What’s a command line?”
TRAPPIST

Totally Rad Analysis Pipelines Python Super Tool
TRAPPIST

Totally Rad Analysis Pipelines Python Super Tool
TRAPPIST

 Totally Rad Analysis Pipelines Python Super Tool
Do it manually?

Exhibit B: Lazy Postdoc (me)

Yeah I'll get right on that.
Long story short

Collection of one-time scripts

Toolkit

Full-featured application

DNA-based OS?
Fundamental requirements

- Design, assembly and modification of pipelines / workflows
- Automated execution, parameter / output versioning, provenance data bundling, interactive visualization
What do you want to do today?

Tip: You can switch mode whenever you want.

Build workflows

Analyze data

☐ Remember this choice for next time and skip this dialog?
Basic requirements
- intuitive
- flexible
- extensible

+ Pre-assembled workflows / pipelines

+ Hooks for external / roll-your-own functions
What workflow do you want to work with?

Tip: You can open other workflows whenever you want.

- Create blank workflow
- Import workflow from file
- Clone existing workflow
- Edit existing workflow

Once a workflow is opened, components or groups of components can be copied and pasted into other workflows. This way users can “recycle” part of an existing workflow without having to rebuild everything and reconnect all the inputs and outputs.
TRAPPIST provides discrete task components for every step of analysis:

- Initial inputs selection
- Data processing steps (existing algorithms)
- Graphical output

Component I/O relies on matching ports with data object classes
- Forces validation of data type/format (not up to user)
Component representation

In the component viewer, the log box displays the component name and a one-line description.

- Input A: description and link to documentation
- Input B: description and link to documentation
- Input C: description and link to documentation
- Parameters:
  - List of defaults
  - Link to default parameter set editor
  - Link to documentation
- Output A: description and link to documentation
- Output B: description and link to documentation

In the component viewer, the log box displays the full method description.
Interacting with components

INSTRUCTION: add another component

After dragging a component to the work area, the user can get an overview of its specifications. The next step is to add another component to connect to.

Mouseover each to view input specs

Mouseover each to view output specs

Mouseover to view full text description of method or click to open component in the viewer

Mouseover to view parameters or click to open parameter set in editor

Component 1 added to workflow
To connect components, the user first selects the upstream component and clicks on an output port. The selected component is highlighted in a different color, as is the selected output port. The next step is to select another component.
Connecting components

The user has selected another component, which is now also highlighted. Compatible input ports are highlighted in green while incompatible ports are highlighted in red. In addition, a text cue is provided for color-blind users (‘O' for green, ‘X' for red, replacing the port letter).

Incompatible ports may be made compatible by using adapters that can split, join or transform data structures. It would be awesome to have the app automatically suggest adapters where appropriate but it might be a bit complicated.
Connecting components

The two components are now connected output to input.

The user must continue in this fashion until all inputs are filled. However, it is not necessary to utilize all outputs.
Execution system

- Progressive / modular / dependency-aware

- Parameter set versioning linked to output versioning

  Users more likely to try various parameters to test assumptions
Flow control rule

- Component ports have “fill” status

- If all its inputs are filled, component is OK to execute
  - add component to execution queue

I’m OK to go!
Database architecture

Workflow 1
- Central DB
- Staging DB
- Execution DB

Workflow 2
- Central DB
- Staging DB
- Execution DB
DB schema + data dump sufficient to fully describe a workflow
Enforcing good practices

- Provenance bundle including:
  - Workflow schema
  - Parameter sets
  - Code version info

- Executable papers!
- Reproducibility!
- Science!
Interactive visualization

Im in ur computer

makin ur graphix
U haz questions?

O RLY?