Scheduler Project

Roseanna Ferguson
&
Joy Tolia

December 1, 2018

Contents

1 Overview

1.1 Summary

The objectives of Scheduler are

- To improve efficiency of the code
- Encourage reuse of code
- Simplify the process of iterating through key parameters

Firstly, the user breaks down a process (referred to as a *recipe*) into a series of components. This information together with parameters are combined in a yaml file. By using pre defined dependencies Scheduler calculates the order in which components can be run and which components can be run in parallel. Finally, Scheduler uses serialisation and checks if a component is required to be re-run. The key functions are summarized below. Code is available on: `https://github.com/joy-rosie/generic.git`

1.2 Metadata

- Reads data from a yaml file. It structures the data into a list of components.
- Substitutes named variables and creates components for loops of parameters.
- Joins together information from *recipes* and *components* to create a list of components with all information needed.

1.3 Dependency Tree

- Given we have all the information about the components and their dependencies, we can create the dependency tree.
- Starting from the root, we see which components are final components and recursively find the dependent components.

1.4 Collapsing Tree and Check for Changes

- Given the dependency tree, we then go top down and merge components with the same inputs and parameters.
- We also check for changes in parameters as if we are rerunning the schedule then we do not want to run components which have had no changes.
- But when we do need to rerun a component, we need to make sure we rerun all the dependent components.

1.5 Building Schedule

- Given the dependency tree, we start from the components with no inputs and put them on the first level.
- Next we look at the components dependent on the components on the first level and put them on the second level and so on.
- All components on the same level can be run in parallel.

1.6 Component

- This is a parent class which holds wrapper functions for components.
- Example of one wrapper function is when we run a component, we will need to load all the input data from the dependent components.
- All components will be inheriting from the parent class.
- Components will be group by a *component type*. Each component type can have multiple *implementations*. For example, `price_data` may be a component type with the implementations `Quandl`, `Bloomberg` and `Reuters`.

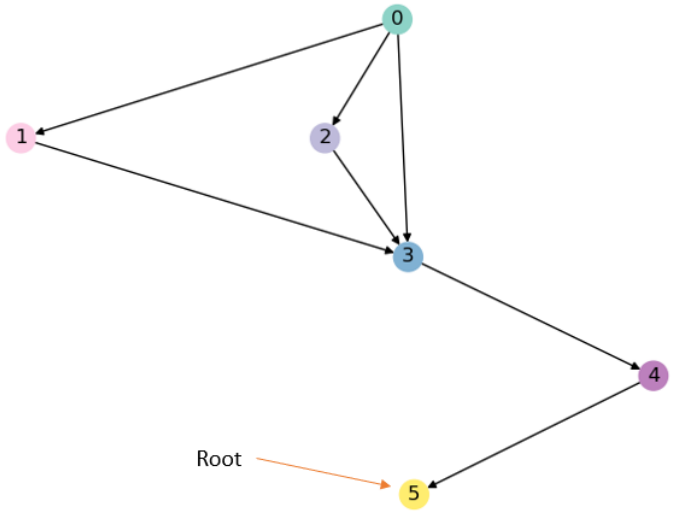
2 Examples

There are two examples included of how Scheduler can be implemented.

- **Machine Learning:** Scheduler is used to compare six different machine learning models used to categorize data.
- **Back Testing:** Scheduler is used to back test two different strategies on the same price data

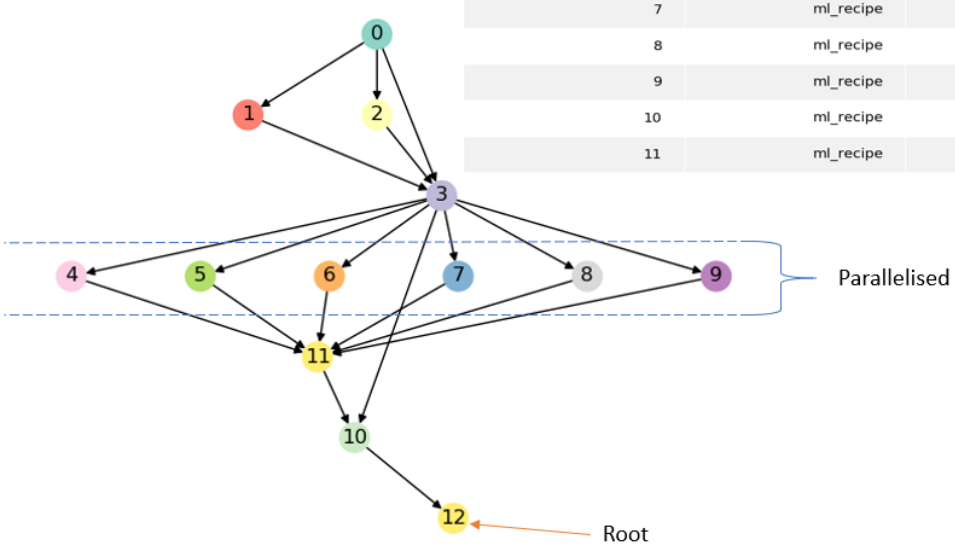
3 Simple Schedule

index	recipe_label	component_type	implementation	component_label
0	ml_recipe_single	data_loader	CsvReader	data_load_all
1	ml_recipe_single	data_summariser	SimpleStats	data_stats
2	ml_recipe_single	data_visualiser	GraphDistributions	data_plot_dist
3	ml_recipe_single	data_setup	SplitValidation	data_split
4	ml_recipe_single	ml_modeller	SklearnValidation	ml_model_validation



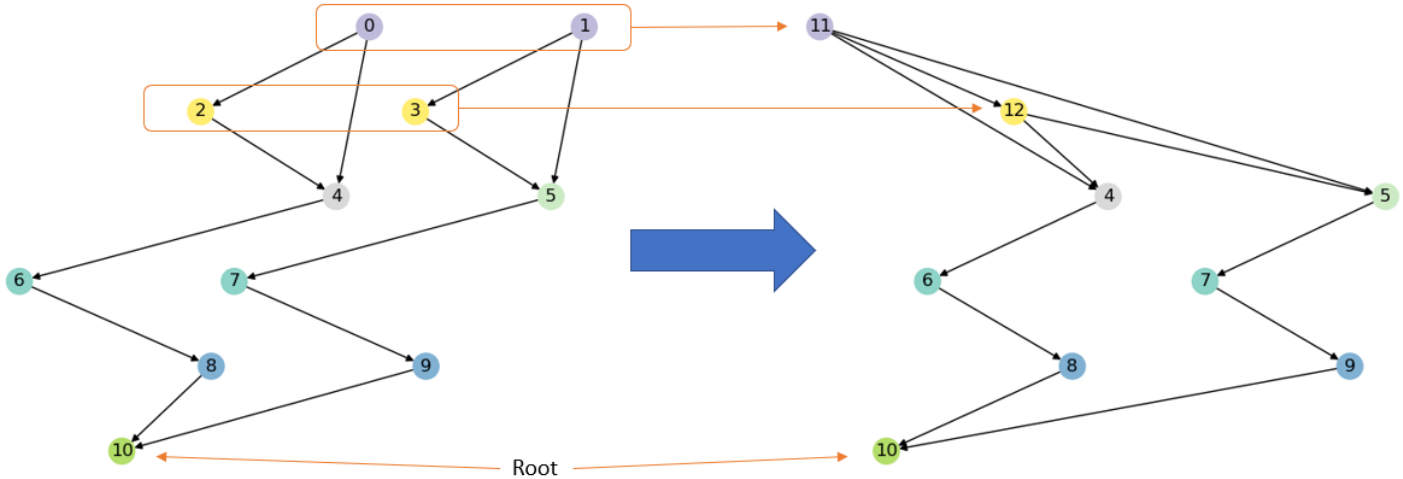
4 Schedule with Loops

index	recipe_label	component_type	implementation	component_label
0	ml_recipe	data_loader	CsvReader	data_load_all
1	ml_recipe	data_summariser	SimpleStats	data_stats
2	ml_recipe	data_visualiser	GraphDistributions	data_plot_dist
3	ml_recipe	data_setup	SplitValidation	data_split
4	ml_recipe	ml_modeller	SklearnCV	ml_model_MODEL_LR
5	ml_recipe	ml_modeller	SklearnCV	ml_model_MODEL_LDA
6	ml_recipe	ml_modeller	SklearnCV	ml_model_MODEL_KNN
7	ml_recipe	ml_modeller	SklearnCV	ml_model_MODEL_CART
8	ml_recipe	ml_modeller	SklearnCV	ml_model_MODEL_NB
9	ml_recipe	ml_modeller	SklearnCV	ml_model_MODEL_SVM
10	ml_recipe	ml_modeller	SklearnValidation	ml_model_validation
11	ml_recipe	model_aggregator	AggregatorSklearn	model_combine



5 Collapsing Schedule

index	recipe_label	component_type	implementation	component_label
0	momentum	price_data	Quandl	price_component
1	reversion	price_data	Quandl	price_component
2	momentum	vol	Arch	vol_component
3	reversion	vol	Arch	vol_component
4	momentum	signal	Momentum	signal_momentum
5	reversion	signal	Reversion	signal_reversion
6	momentum	accounting	Default	accounting_component
7	reversion	accounting	Default	accounting_component
8	momentum	report	ReportSimple	report_component
9	reversion	report	ReportSimple	report_component



6 Timing Schedule

Type	Time taken (seconds)	Cheating (seconds)
Serial	6	126
Parallel	21	42