SIMULATED EVALUATION

This folder provides the sources and scripts to reproduce the results and data from the Simulated Evaluation in **Section VII.**

Folder Contents

- `README.md` - the Markdown source of this file that explains how to use.
- `bin` - contains all of the scripts that runs an exact or approximation algorithm.
- `python-libs` - contains necessary python libraries used by several python scripts.
- `export.sh` - exports the graphs used in publication.
- `gen.mk` - used by the parent directory to automatically run and generate a select approximation algorithm.
- `go.sh` - script for quickly generating an evaluation context.
- `makefile` - makefile used to generate graphs after an evaluation context is ran.

Step 1 - Environment Prerequisites

The following instructions have been tested on the Linux distribution **Ubuntu 22.04 LTS**. Several scripts rely on appropriate path varibles. The `Path` variable needs to be updated to include the fjcolo/eval/bin/ folder. Assuming the installation base directory is ${HOME}/fjcolo , updating the PATH variable can be done by:

```
export PATH=${PATH}:${HOME}/fjcolo/eval/bin
```

Similar to the `Path` variable the `PYTHONPATH` variable will also need to be updated to include the python-libs folder in the directory. This can be updated with:

```
export PYTHONPATH=${PYTHONPATH}:${HOME}/fjcolo/eval/python-libs
```

Step 2 - Software dependencies

The simulated evaluation uses a variety of scripts and packages to generate and run the simulations. The required software dependencies can be installed with:

```
sudo apt install make pip latexmk texlive-pictures texlive-font-utils
```

In addition to this several python packages are required and can be installed with:

```
pip install alive-progress matplotlib pandas
```

Step 3 - Running The Evaluation

The evaluation implementation is a set of python applications, many of them multi-threaded. Each of the applications participates in a pipeline of taking input and producing output for the next stage. Summarily, the **pipeline stages** are:

1. Generate an Evaluation Context
   `gen-context.py`

2. Create an acceptable set of tasks
   `gen-objects.py`
   `filter-tasks.py`

3. Execute each of the algorithms
   -- Exact Methods --
   `opt-task-no-colo.py` `opt-task-colo.py`
   -- Approximation Methods --
   `3-parm.py`
   `3-parm-hd.py`
   `2-gram.py`
   -- DAGOT Heuristics --
   `dag-m.py`
   `dag-lp.py`
   `dag-gp.py`

   For the Approximation Methods and DAGOT Heuristics, parallel versions of each program are provided with a -par.py suffix.
   EX: dag-lp.py has a parallel implementation dag-lp-par.py**

4. Coalate and Present the Results
   `tabulate.py`
   `graph.py`

Step 3 - Quick Start Alternative

Included in this directory is a sample script `go.sh` that will execute the pipeline given an Evaluation Context. An example invocation of this script is:

```
./go.sh -M 5 all-tiny.json all-tiny
```

This will generate the results for a "tiny" run using all of the algorthims in pipeline stage 3. Resulting graphs will be placed in the newly created all-tiny directory under all-tiny/graphs.

Provided Contexts

What follows are the evaluation contexts, the command line used to generate each of them, their purpose, and their execution using go.sh:

all-tiny.json

Verifies the evaluation pipeline is working correctly, limited to 5 cores.

1. Generate the context

   ```
   $ ./gen-context.py -S 3 -O 5 -R 10 -D 200 -B 10 -I 10 -t 50 -A 0 \
   -c all-tiny.json
   ```

2. Execute the pipeline given the context $ ./go.sh -M 5 all-tiny.json all-tiny

all-small.json

Provides representative results for all of the methods (exact and approximate), limited to 8 cores.

1. Generate the context

   ```
   $ ./gen-context.py -S 5 -O 15 -R 15 -D 500 -t 1000 -B 50 -I 50 -A 0 \
   -c all-small.json
   ```

2. Execute the pipeline

   ```
   $ ./go.sh -M 8 all-small.json all-small
   ```

approx.json

Extends the approximate results to a greater number of tasks and cores, excluding the exact methods, limited to 16 cores.

1. Generate the context

   ```
   $ ./gen-context.py -c approx.json
   ```

2. Execute the pipeline given the context

   ```
   $ ./go.sh -X -M 16 approx.json approx
   ```

   #

Parallelism Info

The exact methods (`opt-task-no-colo.py` and `opt-task-colo.py`) are inately parallel processes. When executed for a single task they will utilize as many core as are available on the system.

The approximation methods: `3-parm.py`, `3-parm-hd` , and `2-gram.py` are sequential processes. Parallel applications that perform the approximation methods are also provided, `3-parm-par.py` `3-parm-hd-par.py` and `2-gram-par.py`. They execute the approxmation algorithm over all tasks in parallel.

Time Requirements

The exact methods are intractable and are used primarily for illustrating their performance compared to the approximation algorithms given a small number of cores and less complex tasks. The "all-small.json" Evaluation Context has taken between 24 and 36 hours to complete on a Ryzen 3970X with 24 cores dedicated to the pipeline. Given the time requirements of the exact algorithms, the "all-small.json" and "all-tiny.json" evaluation contexts are intended for use with the exact algorithms.

Further, the complexity of the exact methods grows exponentially. The exponential base is the maximum number of cores a task may be assigned. If the pipeline exceeds the time budget for the machine it runs upon reduce the maximum number of cores used by the algorithm. This can be done by providing a smaller -M option to the go.sh script or the individual algorithms:

```
$ ./go.sh -M 10 all-tiny.json all-tiny
# Maxmimum of 10 cores
```

```
$ ./opt-task-no-colo.py -M 8 all-tiny/task/task.008.json
# Maximum of 8 cores
```

Relevant Contexts

To generate a set of contexts that associate with the context sets in the work specific arguments can be passed into go.sh . The full script commands can be found in the table below:

Context Name	Graph Name	Command
sete	Figure 7: Core Comparison for E	go.sh -M 5 -U 100 ctxs/all-m5.json group-e
setx	Figure 8: Core Comparison for X	go.sh -M 64 -U 500 -X ctxs/approx-m64-u500.json group-x