EMADE: Caching Spring 2019

Samuel Huang, Eric Frankel, Anish Thite, Yash Shah, Benson Chau, Alex Gurung, William Li

Subteam Goals

1. Support all data types that EMADE supports.
2. Create APIs that gives the users options to use their own cache invalidation method suitable for their problem space
3. Provide stats as to how well cache is performing and benchmarking each instance (when running on a cluster)
4. Merge cache into master! (or image)

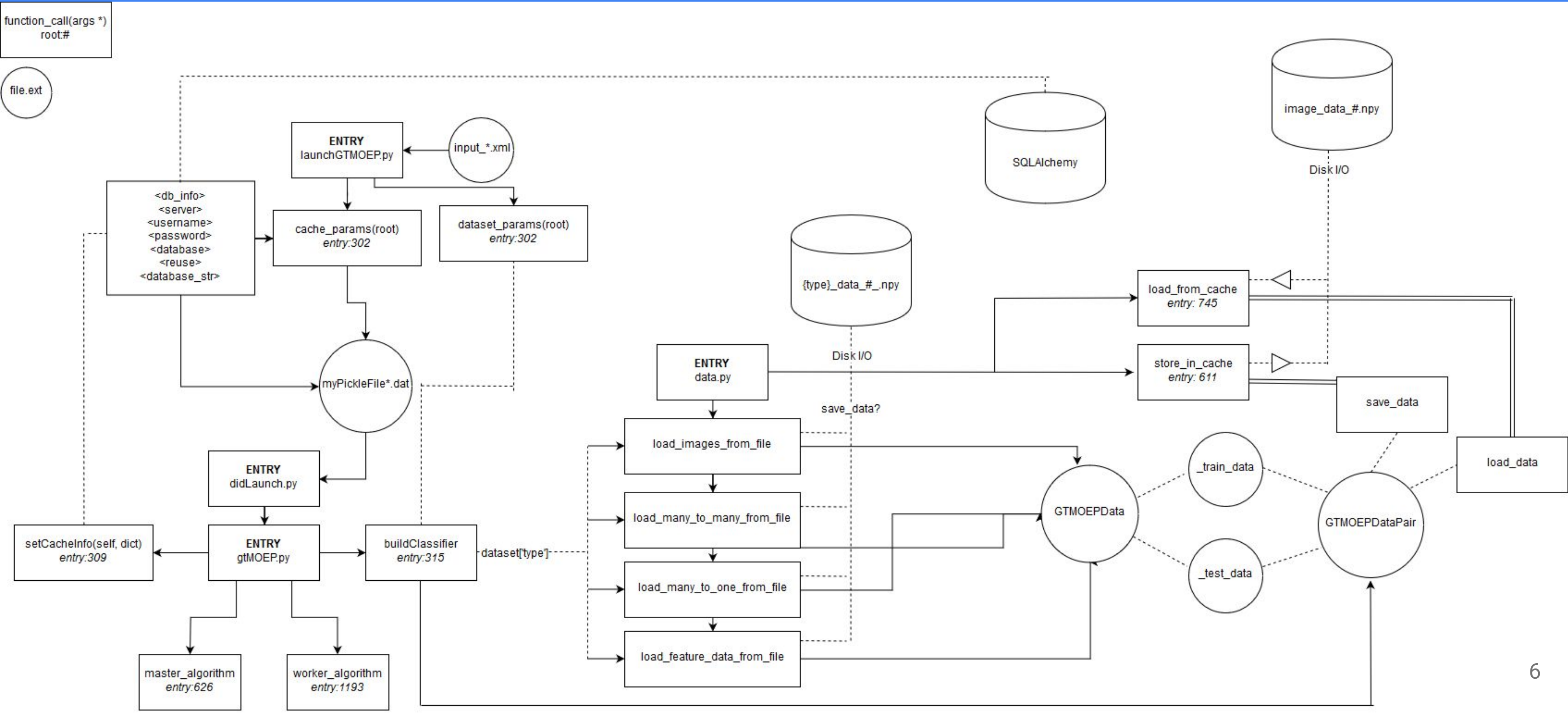
Overview

Cache Maintenance

Cache Maintenance

- Maintain the current iteration of the cache branch.
- Add documentation (in the source code or otherwise) for the cache branch.
- Fix and document bugs and other problems.

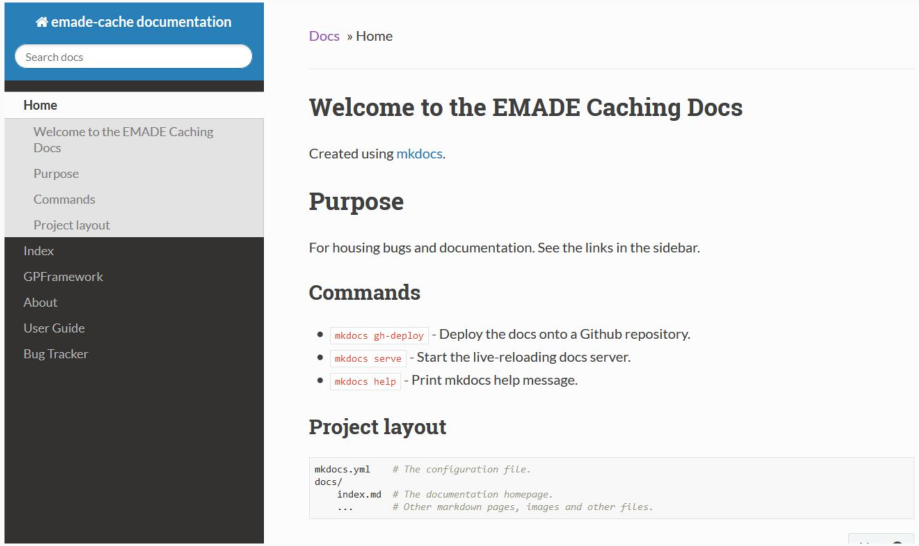
Caching: Current Workflow


function_call(args *) root:#
file.ext
ENTRY launchGTMOEP.py
input_*.xml
<db_info> <server> <username> <password> <database> <reuse> <database_str>
cache_params(root) entry:302
dataset_params(root) entry:302
myPickleFile*.dat
ENTRY didLaunch.py
setCacheInfo(self, dict) entry:309
ENTRY gtMOEP.py
buildClassifier entry:315
master_algorithm entry:626
worker_algorithm entry:1193
dataset['type']
SQLAlchemy
{type}_data_#_.npy
Disk I/O
save_data?
ENTRY data.py
load_images_from_file
load_many_to_many_from_file
load_many_to_one_from_file
load_feature_data_from_file
image_data_#.npy
Disk I/O
load_from_cache entry: 745
store_in_cache entry: 611
save_data
load_data
GTMOEPData
_train_data
_test_data
GTMOEPDataPair

Documentation Tool

- A small documentation tool via Python's mkdocs has been introduced.

- Meant to help keep track of bugs and be an external documentation resource.



Welcome to the EMADE Caching Docs

Created using mkdocs.

Purpose

For housing bugs and documentation. See the links in the sidebar.

Commands

- `mkdocs gh-deploy` - Deploy the docs onto a Github repository.
- `mkdocs serve` - Start the live-reloading docs server.
- `mkdocs help` - Print mkdocs help message.

Project layout

```
mkdocs.yml    # The configuration file.
docs/
    index.md  # The documentation homepage.
    ...       # Other markdown pages, images and other files.
```

The home page for the caching docs.

Dockerizing Cache

- Simplifies/standardizes process of running on all operating systems
  - Conda setup
  - Mysql setup
- GCP setup involves less tears
- Can be used for general EMADE as well
  - Dockerfile
  - Docker-compose.yml
- Separates source code from running code
- Updating is easier
  - environment.yml

Instructions

1. Clone repo
    a. Installation script
2. Docker-compose up
3. Docker-compose run emade “command”
4. Outputs stored in “/var/lib/docker/volumes/<container_id>”

Future Updates

- Build faster
  - Takes ~15 minutes to build currently
  - Caches after that
- Standardize outputs
  - Currently writes files to wherever the container is stored
  - Different per computer
- Fix conda version problems
  - error: numpy 1.15.4 is installed but numpy<=1.14.5,>=1.13.3 is required by {'tensorflow'}
- Mount /emade to the container, allows for updated source code to be reflected in the container
- Test on other operating systems

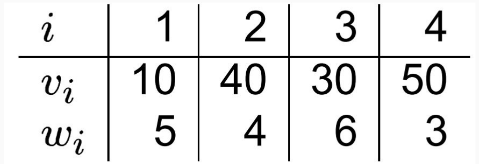
Cache Invalidation

The Problem

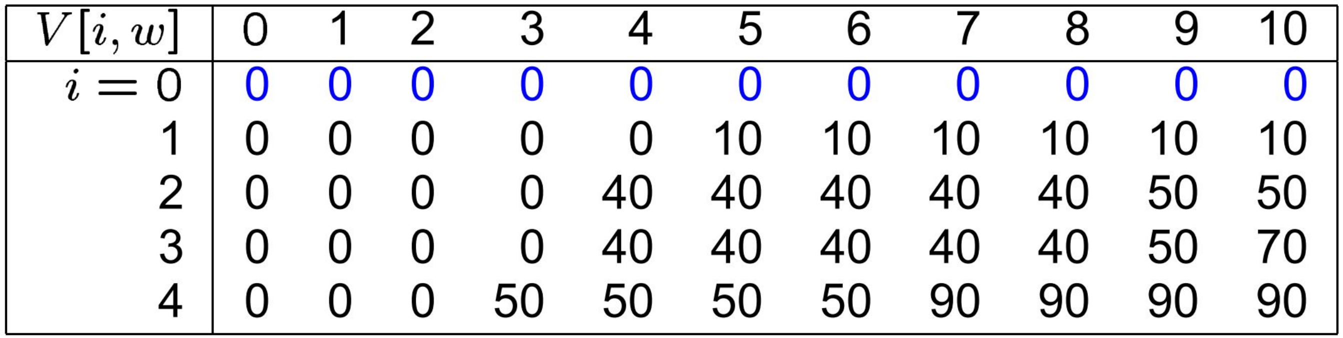
- Have max size our cache can take up
- Want to maximize benefit of subtrees we store
- Current system:
  - Order subtrees by benefit
  - Remove subtrees until under size cap
- Want a more optimal solution

Our Solution

- Dynamic Programming solution to the 0-1 Knapsack Problem
    - Construct matrix based on value of subtree and size would take up



i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3



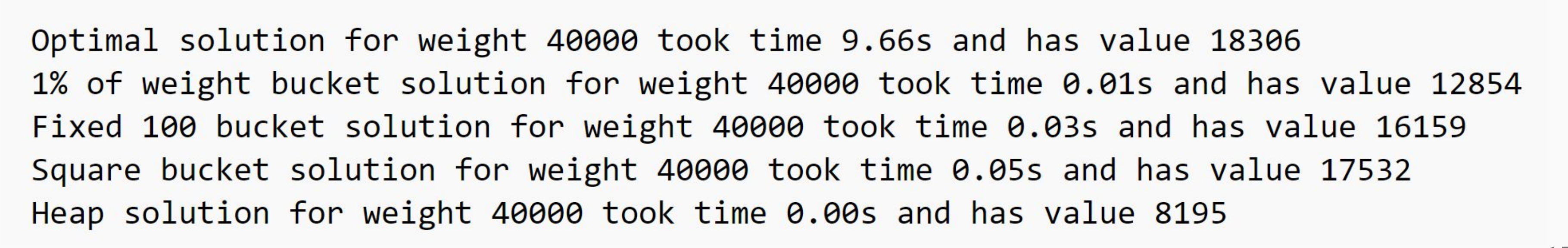
$V[i,w]$	0	1	2	3	4	5	6	7	8	9	10
$i = 0$	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10	10
2	0	0	0	0	40	40	40	40	40	50	50
3	0	0	0	0	40	40	40	40	40	50	70
4	0	0	0	50	50	50	50	90	90	90	90

Problems with Our Solution

- Very time ineffective: our default max weight is 10,000,000
- Time cost is large even when number of subtrees in cache is very small
- Heap solution is really really fast

Solutions to the Problems with Our Solution

- Determine if number of objects is significantly slower than weight
- Use buckets, or precision parameter
- Scale weights
- Benchmarking, and other tests



```
Optimal solution for weight 40000 took time 9.66s and has value 18306
1% of weight bucket solution for weight 40000 took time 0.01s and has value 12854
Fixed 100 bucket solution for weight 40000 took time 0.03s and has value 16159
Square bucket solution for weight 40000 took time 0.05s and has value 17532
Heap solution for weight 40000 took time 0.00s and has value 8195
```

Potential Optimization of Knapsack Solving

- Performance <-> Precision
  - Sacrifice precision
    - Approximation Algorithms
      - Greedy
      - Scaling and rounding
- Parallelism
  - Hypercubes
    - $O(mc/p + c^2)$
  - Irregular Mesh
    - $O(mc/p + c/w_{min})$

A. Goldman, D. Trystram / J. Parallel Distrib. Comput. 64 (2004) 1213–1222

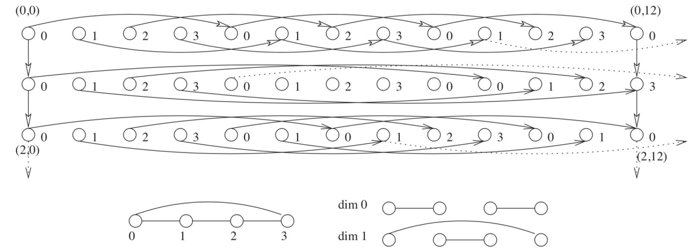
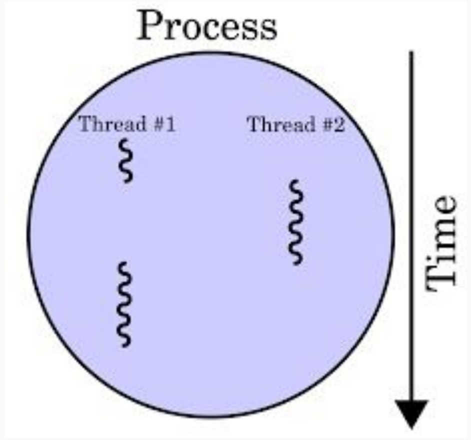


Fig. 3. Irregular mesh with weights 4, 9, 6, ... and $c \geqslant 13$ and its allocation on path P_k $(H(2)\square H(0))$.

Multithreading

- Memory Latency decreases
- Instruction miss rate <-> Data miss rate
- Data miss rate decreases due to locality
- Dynamic behavior interferes with instruction locality

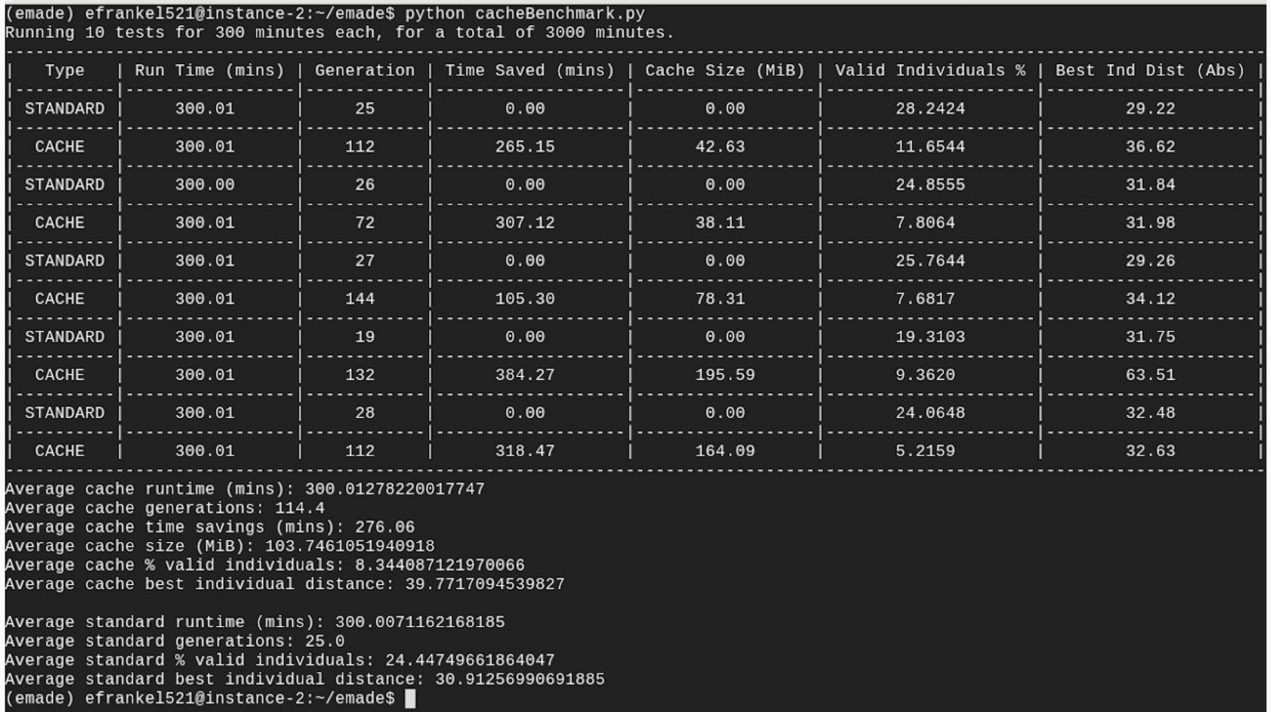


Benchmarking

Setup

- Created a script that automatically runs EMADE
  - Runs set number of tests, alternating between cache and no cache
  - Collects stats
- Ran 10, 5 hour runs (5 cache, 5 no cache) on GCP
- Future work
  - Command line options for configuration
  - Save output to disk (CSV)
  - Make it easy to add new metrics
  - Use any dataset
  - Run multiple tests in parallel on different instances

Script output



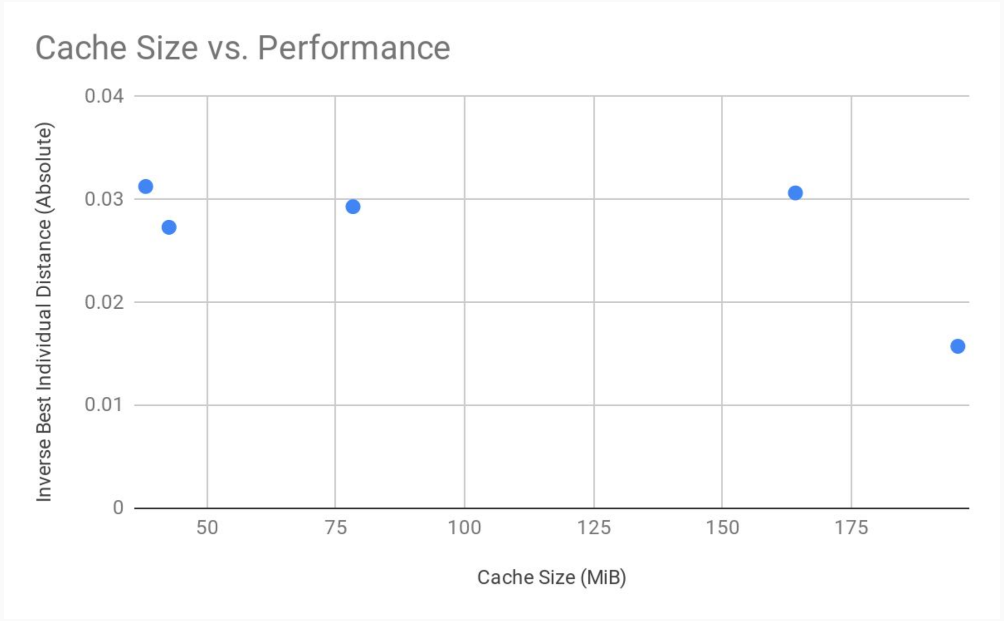
```
(emade) efrankel521@instance-2:~/emade$ python cacheBenchmark.py
Running 10 tests for 300 minutes each, for a total of 3000 minutes.
---------------------------------------------------------------------------------------------------------------------------------
|   Type   | Run Time (mins) | Generation | Time Saved (mins) | Cache Size (MiB) | Valid Individuals % | Best Ind Dist (Abs) |
|----------|-----------------|------------|-------------------|------------------|---------------------|---------------------|
| STANDARD |     300.01      |     25     |       0.00        |       0.00       |       28.2424       |        29.22        |
|----------|-----------------|------------|-------------------|------------------|---------------------|---------------------|
|  CACHE   |     300.01      |    112     |      265.15       |      42.63       |       11.6544       |        36.62        |
|----------|-----------------|------------|-------------------|------------------|---------------------|---------------------|
| STANDARD |     300.00      |     26     |       0.00        |       0.00       |       24.8555       |        31.84        |
|----------|-----------------|------------|-------------------|------------------|---------------------|---------------------|
|  CACHE   |     300.01      |     72     |      307.12       |      38.11       |       7.8064        |        31.98        |
|----------|-----------------|------------|-------------------|------------------|---------------------|---------------------|
| STANDARD |     300.01      |     27     |       0.00        |       0.00       |       25.7644       |        29.26        |
|----------|-----------------|------------|-------------------|------------------|---------------------|---------------------|
|  CACHE   |     300.01      |    144     |      105.30       |      78.31       |       7.6817        |        34.12        |
|----------|-----------------|------------|-------------------|------------------|---------------------|---------------------|
| STANDARD |     300.01      |     19     |       0.00        |       0.00       |       19.3103       |        31.75        |
|----------|-----------------|------------|-------------------|------------------|---------------------|---------------------|
|  CACHE   |     300.01      |    132     |      384.27       |      195.59      |       9.3620        |        63.51        |
|----------|-----------------|------------|-------------------|------------------|---------------------|---------------------|
| STANDARD |     300.01      |     28     |       0.00        |       0.00       |       24.0648       |        32.48        |
|----------|-----------------|------------|-------------------|------------------|---------------------|---------------------|
|  CACHE   |     300.01      |    112     |      318.47       |      164.09      |       5.2159        |        32.63        |
---------------------------------------------------------------------------------------------------------------------------------
Average cache runtime (mins): 300.01278220017747
Average cache generations: 114.4
Average cache time savings (mins): 276.06
Average cache size (MiB): 103.7461051940918
Average cache % valid individuals: 8.344087121970066
Average cache best individual distance: 39.7717094539827

Average standard runtime (mins): 300.0071162168185
Average standard generations: 25.0
Average standard % valid individuals: 24.44749661864047
Average standard best individual distance: 30.91256990691885
(emade) efrankel521@instance-2:~/emade$ █
```

Results (Averages)

Type	Generations	Time Saved (Minutes)	Cache Size (MiB)	Valid Individuals (%)	Best Individual Distance (Absolute)
Standard	25.00	N/A	N/A	24.45	30.91
Cache	114.40	276.06	103.75	8.34	39.77

Results (Graph)



Results (Full)

Type	Generations	Time Saved (Minutes)	Cache Size (MiB)	Valid Individuals (%)	Best Individual Distance (Absolute)
Standard	25	N/A	N/A	28.24	29.22
Cache	112	265.15	42.63	11.65	36.62
Standard	26	N/A	N/A	24.86	31.84
Cache	72	307.12	38.11	7.81	31.98
Standard	27	N/A	N/A	25.76	29.26
Cache	144	105.30	78.31	7.68	34.12
Standard	19	N/A	N/A	19.31	31.75
Cache	132	384.27	195.59	9.36	63.51
Standard	28	N/A	N/A	24.06	32.48
Cache	112	318.47	164.09	5.22	32.63

Final Remarks & Future Work

- Cache obviously has several problems.
  - Although we have much more in terms of documentation, many function calls are wrapped very closely together in such a way that it takes a long time to trace & debug.
  - Hardcoded paths remain an issue due to the above. Moreover, it doesn't work on everything, and other issues involving evaluation have just been found per the benchmark.
- Refactor old code
- Full documentation
- Cache Invalidation API
- Benchmarking for instances and clusters
- Continuous Integration with master