ETH zürich

EPOS Code Tutorial

Presented

by Thomas Asikis, asikist@ethz.ch

Self-organizing Multi-agent Systems

EPOS

http://epos-net.org/

Intro

A scripting and coding tutorial on how to use EPOS

- **Code and scripts can be found at:**
  https://github.com/epournaras/EPOS/tree/tutorial
- **Release can be found at:**
  https://github.com/epournaras/EPOS/releases/tag/0.0.1
- **Documentation is found at:**
  http://epos-net.org/i-epos/software/documentation
  Older Artifact with GUI can be found at:
- **http://epos-net.org/i-epos/**

Prerequisites

You need to have installed:

- Java 8 or higher

Knowledge in:

- Programming ~ Java
- Scripting ~ python, shell scripting (bash, power-shell etc.)

Today's Presentation

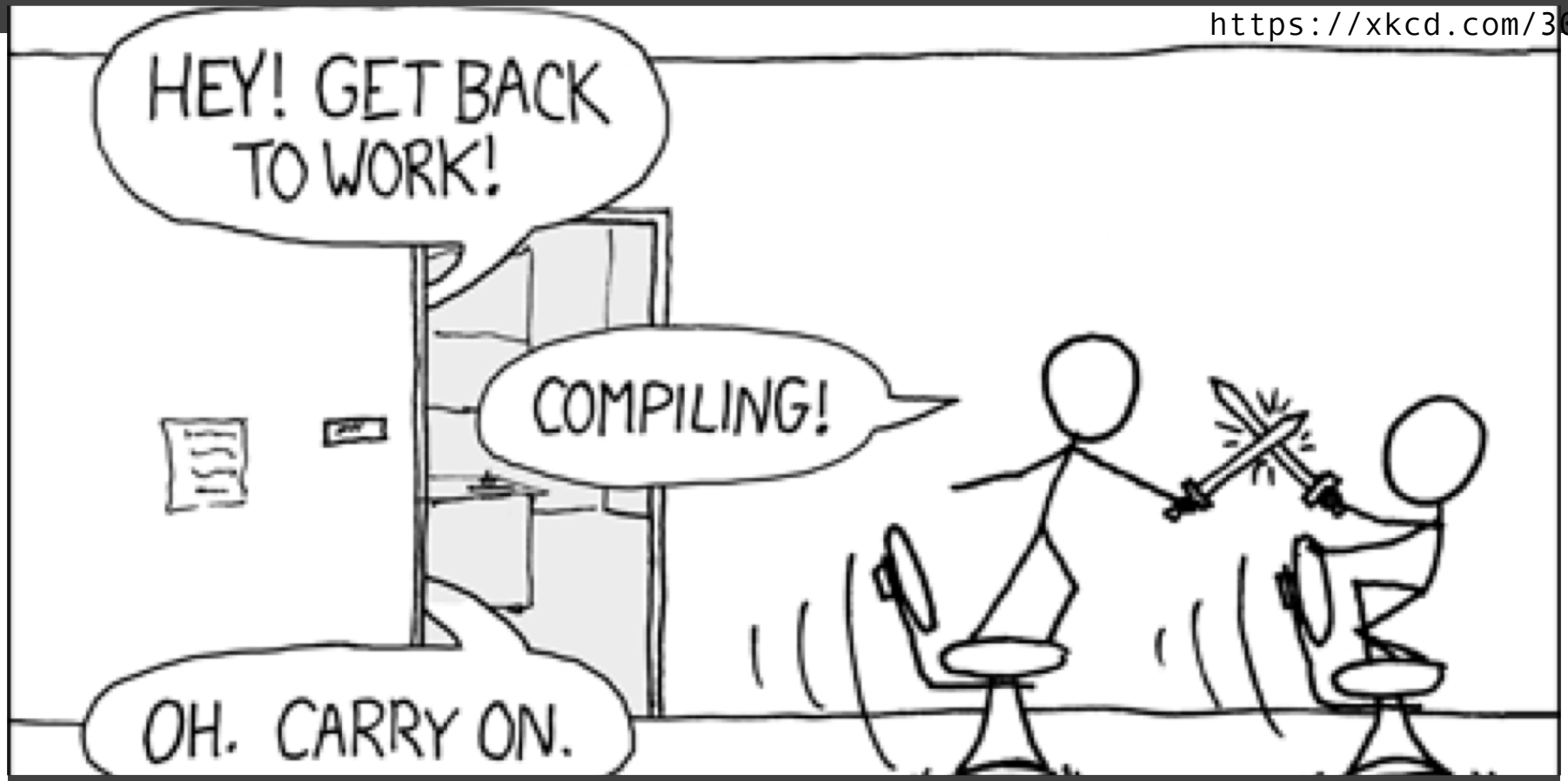
Basic usage as black box: input/output hyperparams

- Hyperparameters of EPOS
- Input and output data
- Processing and ploting results
- Ideas on analyzing and reasoning with results

Advanced:

- Extend EPOS code
- Design and implement new concepts

https://xkcd.com/303/



Setup

A guide on how to start using the scripts and the code

Executable

1. Download the EPOS-folder
2. Unzip and enter the resulting folder
3. The jar can be executed as:

```
java –jar epos-tutorial.jar #default config file is used
java –jar epos-tutorial.jar “path/to/conf” #custom config file
```

4. The configuration, dataset & output folders are respectively in:

```
datasets/
conf/
output/
```

Datasets

datasets

dataset_1

- agent_0.plans
- agent_1.plans
- ...
- default_goal_signal.target

Agent Plans File

- Always named like: agent_{index}.plans
- {index} is an integer, increasing incrementally by one.
- Avoid skipping indices, e.g:

```
agent_0.plans
agent_2.plans #no file named agent_1.plans
```

- Each line is a candidate plan for selection, formatted as:

Goal signal file

- Filetype suffix in filename is ".target"
- Incentive/Goal/Target/Reference signal
- Format is:

```
0.1,0.2,0.3,0.4,-0.5
```

Goal signal values

- The goal signal in EPOS has the same dimensions as the plan dimensions.
- `planDim < goalDim` → crop goal signal from beginning to planDim
- `planDim > goalDim` → repeat goal signal vector till planDim is reached

Configuration Files

Fixed configurations (we don't touch these files):

```
conf/log4j.properties
conf/measurement.conf
conf/protopeer.conf
```

EPOS parametrization configuration (encouraged to play w/ this):

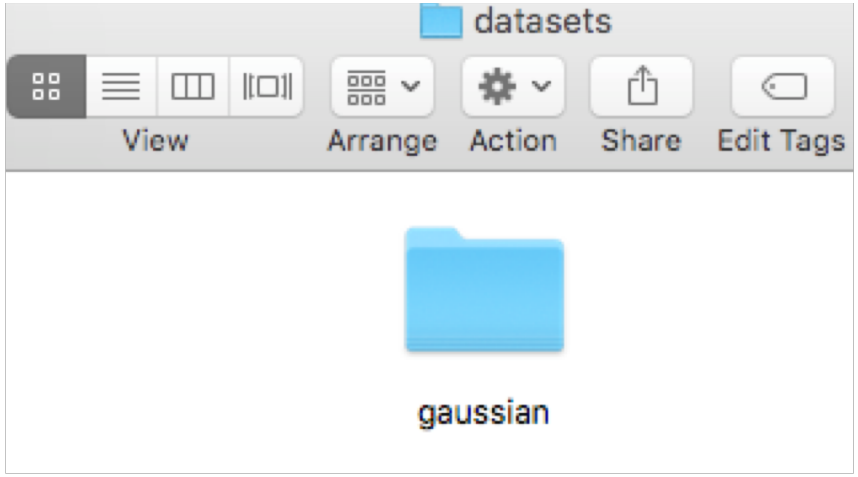
```
conf/epos.properties
```

Java-properties format: key=value pairs

spaces and quotes (“”) are removed

EPOS Properties ~ Dataset

```
### Dataset ###
#The folder name in the datasets path.
#Make sure it has no spaces, tabs or newlines
#(alphanum & underscore combination preferred)
dataset=gaussian
```



EPOS basic parameters ~ Setup

! Recall that:

- EPOS is an **iterative** algorithm
- It is based on a **tree structure:** parent & children nodes

```
### Basic epos parameters ###
# maximum number of iterations per simulation
# any integer > 0
numIterations=40

# number of children per node in the EPOS tree.
# the tree is always symmetric and balanced
# any integer > 0
numChildren=2
```

EPOS Basic Parameters ~ Dimensions

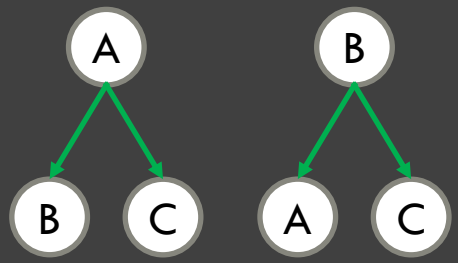
```
# number of agent used for the experiment
# if it is higher than available agents in dataset, the maximum available
agents will be used.
# any integer > 0
numAgents=1000


# number of plans used per agent
# if it is higher than the available plans per agent, the maximum number will
be used
# any integer > 0
numPlans=500


# number of plan dimensions used from the start of the plan.
# if more than available, the maximum available are used.
# any integer > 0
planDim=100
```

Shuffling

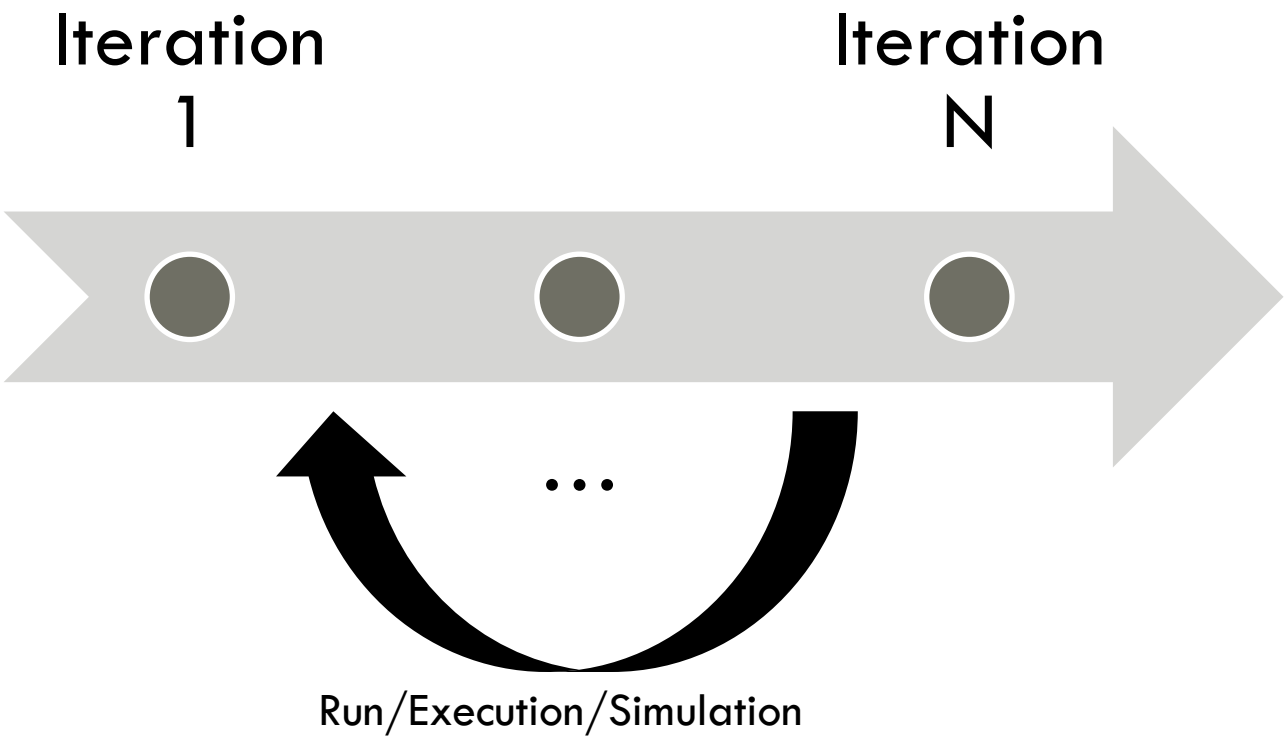
Randomness in agent order and position in tree.

Several repetitions from random start → statistical significance

```
### Shuffle seeds ###

# number of simulations for whole epos experiments
# any integer > 0
numSimulations=5

# initial agent structure before reorganization occurs
# any integer > 0
shuffle=0

# path to a file containing permutation of indices
# file structure: one column integer index in each row
shuffle_file="permutation.csv”
```



Difference of Runs and Iterations



Cost Parameters

Cost is calculated: $\alpha \cdot unfairnes + \beta \cdot localCost + (1 - \beta - \alpha) \cdot globalCost$
$localCost$: average of local cost function among users

```
### Weights of the global complex cost ###

# double from [0, 1], alpha + beta <= 1, unfairness
alpha=0.2

# double from [0, 1], alpha + beta <= 1, local objective
beta=0
# alpha*unfairness + beta*local_cost + (1-alpha-beta)*global_costs
```

Cost Functions

```
### Cost Functions ###
# Available cost function choices:
# NoGoal "MAX":max value,  "VAR":variance, "STD":standard deviation, "INDEX":
plan index value, "PREF": preference, "DISC": discomfort
# Goal Singal "SQR":square distance, "RMSE":root mean squared error,
"RSS":residual sum of squares, "DOT":dot product, "XCORR":cross correlation

# Suggested values : "XCORR", VAR", "RSS", "RMSE"
globalCostFunction="VAR”

# Suggested values : "INDEX"  "DISC", "PREF"
localCostFunction="INDEX"

# "MIN-MAX", "STD", "UNIT-LENGTH", only for "RSS" cost
scaling="MIN-MAX"
```

Goal Signal

```
### Goal Signal ###

# filepath of the file containing the vector to be used as goal signal
# goal signal dimension are expected to be same as planDim otherwise cropping or repetitive padding might occur
# if no path is provided, it will search for a ".target" file in the dataset folder
# if no file is found at all a zero valued goal signal will be generated
goalSignalPath=default
```

Reorganization Parameters

- Escape Local Optima (Advanced Feature)
- Statistical Significance
- Reorganization Logic like shuffling with a predefined strategy

```
### Reorganization strategy ###
# possible values: periodically, convergence, globalCostReduction, never.
strategy=periodically
# any integer > 0, if "periodically" strategy is chosen
periodically.reorganizationPeriod=3
# any positive integer > 0, if convergence strategy is chosen
convergence.memorizationOffset=5
# double from [0, 1]
globalCost.reductionThreshold=0.5
# any integer, keep same for reproducibility
strategy.reorganizationSeed=0
```

Logging

```
### Loggers ###
logger.GlobalCostLogger=true
logger.LocalCostMultiObjectiveLogger=true
logger.TerminationLogger=true
logger.SelectedPlanLogger=true
logger.GlobalResponseVectorLogger=true
logger.PlanFrequencyLogger=true
logger.UnfairnessLogger=true
logger.GlobalComplexCostLogger=true
logger.WeightsLogger=true
logger.ReorganizationLogger=true

# Code related logger for debugging and checks, please check here
# For experiments "SEVERE" is preferred
logLevel="SEVERE"
```

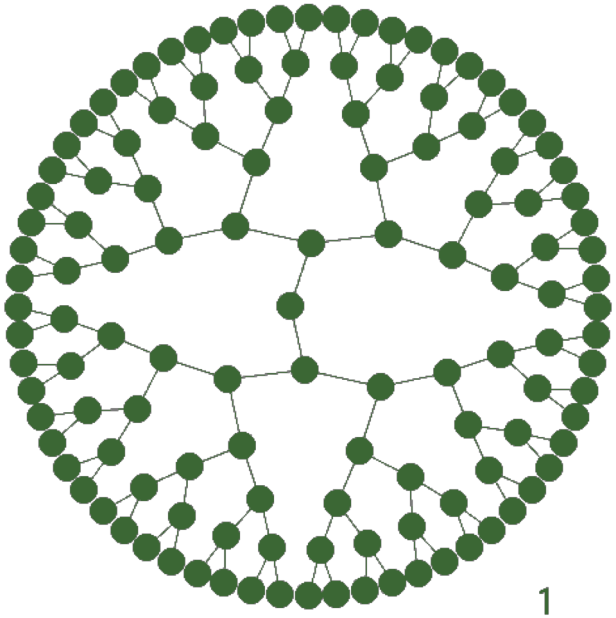
Output

Output folder:

- Folder is created after first run of the .jar
- contains a result subfolder for each jar run named like: {dataset_name}_{unix_timestamp}
- Each result subfolder contains:
  - Log output from the loggers in ".csv" format
  - A copy of the applied configuration for sanity checks named "used_conf.txt"

Log folder

- Created on runtime
- Contains debug logs
- Logs are usually empty



Running Experiments

How to run experiments

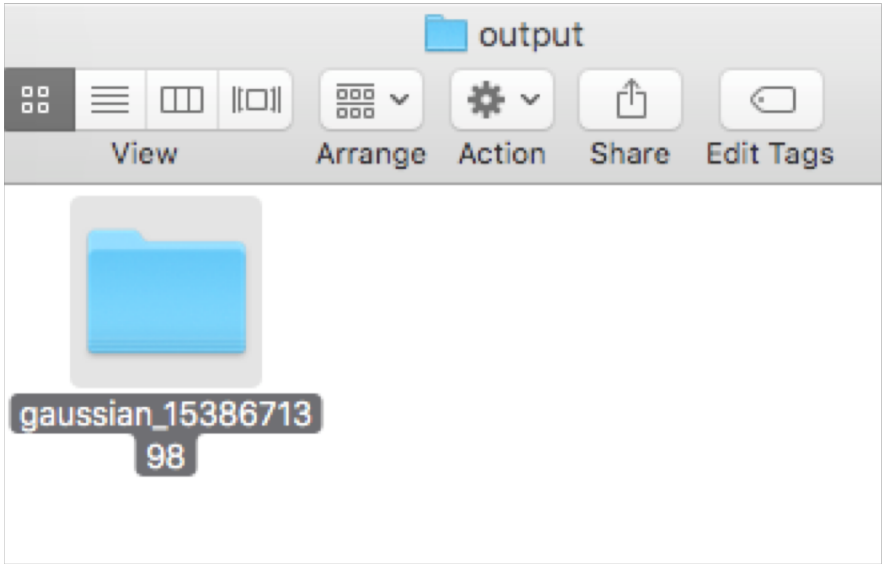
A first Run

```
### Dataset ###
dataset=gaussian
### Basic epos parameters ###
numSimulations=5
numIterations=40
numChildren=2
numAgents=1000
numPlans=500
planDim=100
### Shuffle seeds ###
shuffle=0
shuffle_file="permutation.csv"
### Weights of the global complex cost ###
alpha=0.2
beta=0
### Cost Functions ###
globalCostFunction="VAR"
localCostFunction="INDEX"
scaling="MIN-MAX"
### Goal Signal ###
goalSignalPath=default
### Reorganization strategy ###
strategy=periodically
periodically.reorganizationPeriod=3
convergence.memorizationOffset=5
globalCost.reductionThreshold=0.5
strategy.reorganizationSeed=0
### Loggers ###
logger.GlobalCostLogger=true
logger.LocalCostMultiObjectiveLogger=true
logger.TerminationLogger=true
logger.SelectedPlanLogger=true
logger.GlobalResponseVectorLogger=true
logger.PlanFrequencyLogger=true
logger.UnfairnessLogger=true
logger.GlobalComplexCostLogger=true
logger.WeightsLogger=true
logger.ReorganizationLogger=true
logLevel="SEVERE"
```

```
> release-0.0.1-example u$ ls
  conf datasets epos-tutorial.jar
> release-0.0.1-example u$ java –jar epos-tutorial.jar
  WARNING or INFO logs about config loading …
  Current config …
  Simulation 1
  IEPOS Finished! It took: 0 seconds.
  . . .
> release-0.0.1-example u$ ls
  conf datasets epos-tutorial.jar log output
```

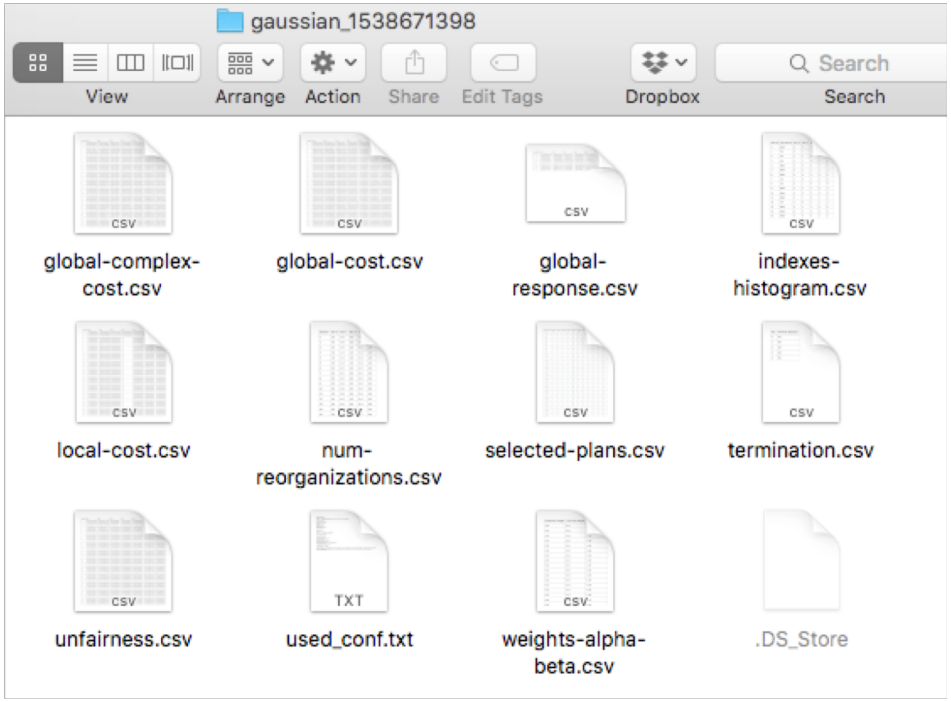
Experiment Output

Result folder is created in the "*output/*" folder:



Experiment Results Folder

The output of the experiments should look like:



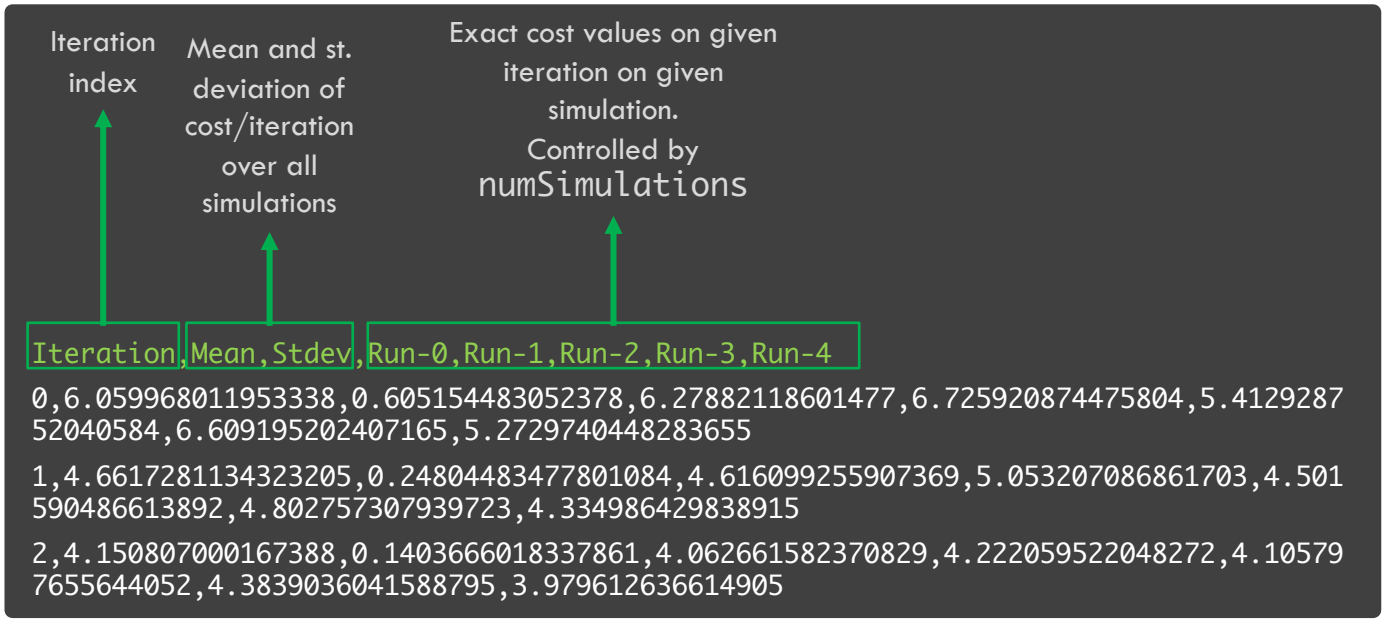
used_conf.txt

A file containing the configured parameters for the given experiment:

```
CONFIGURATION:
dataset = gaussian
output =
/Users/tadev/Repos/EPOS/output/gaussian_1538671398
==============
numSimulations = 5
dataset = gaussian
numAgents = 11
numPlans = 16
planDim = 100
numIterations = 40
numChildren = 2
--------------
. . .
```

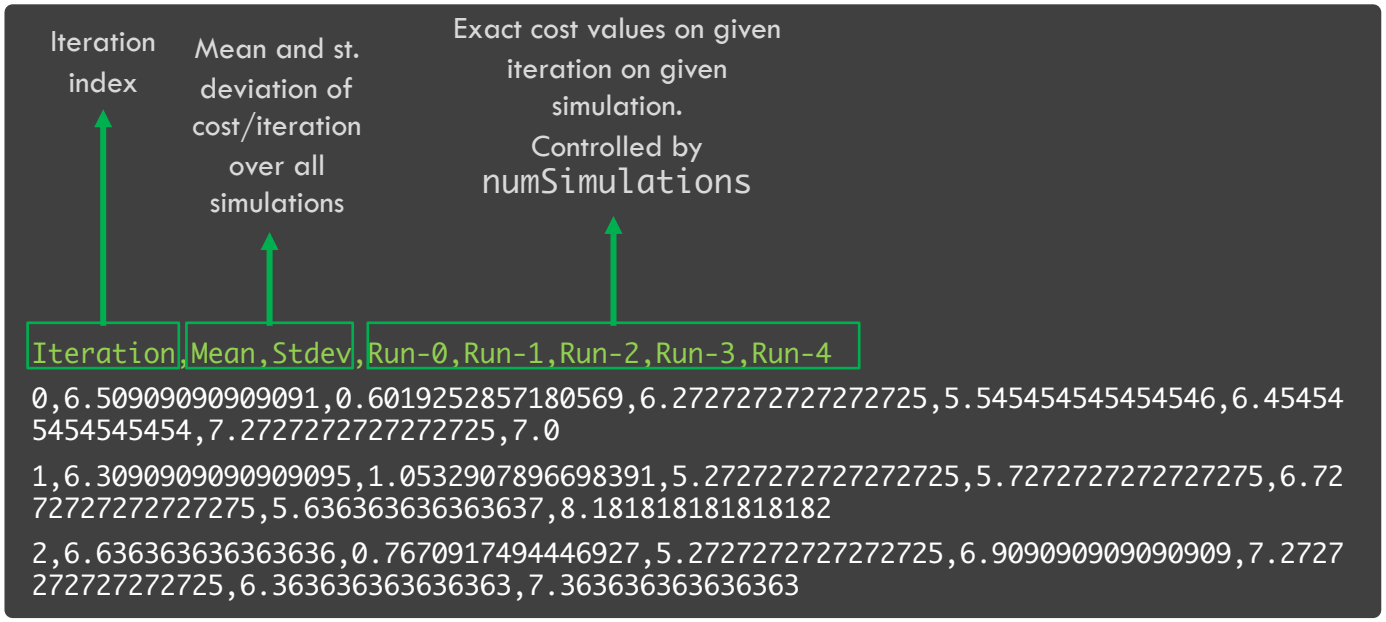
global-cost.csv

- Contains results from `GlobalCostLogger`.



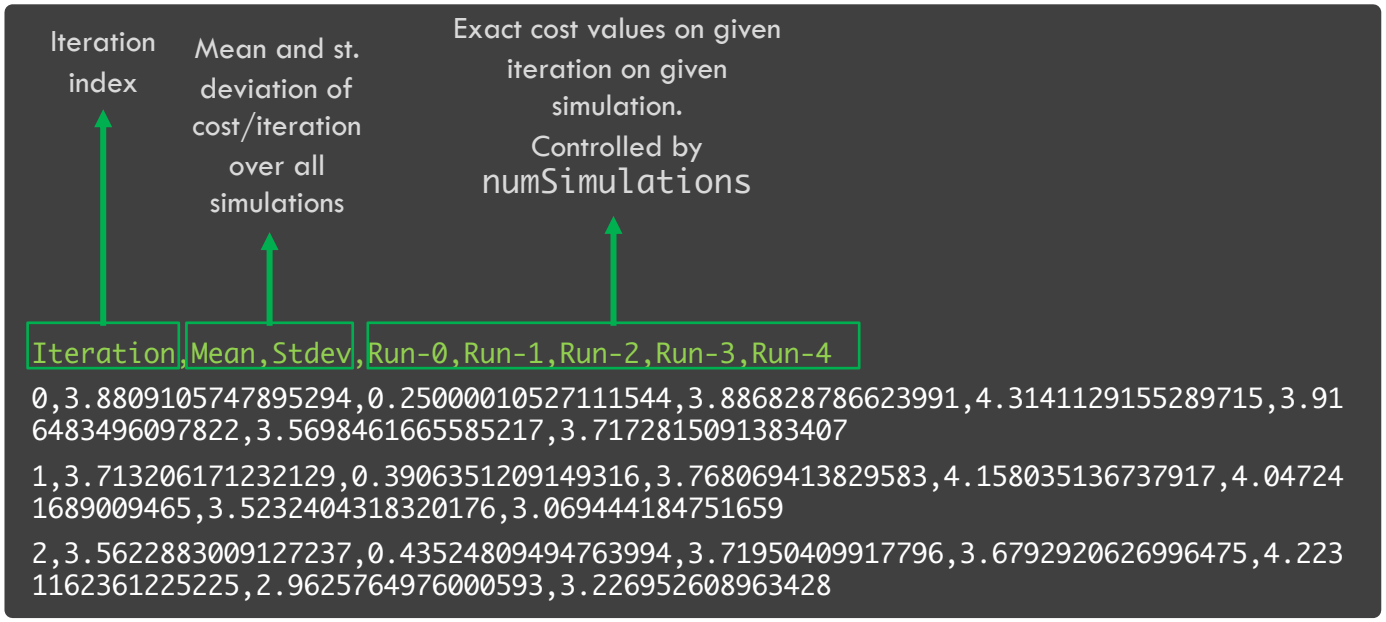
local-cost.csv

- Contains results from `LocalCostMultiObjectiveLogger`.



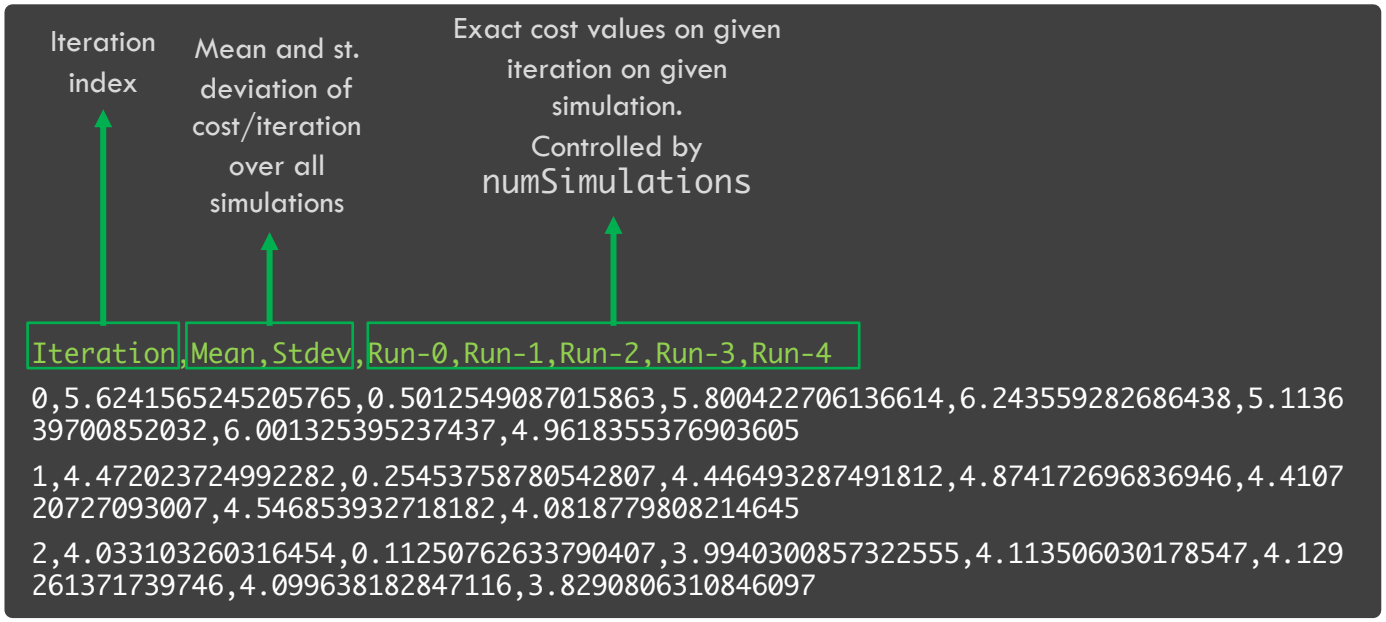
unfairness.csv

- Contains results from `UnfairnessLogger`.

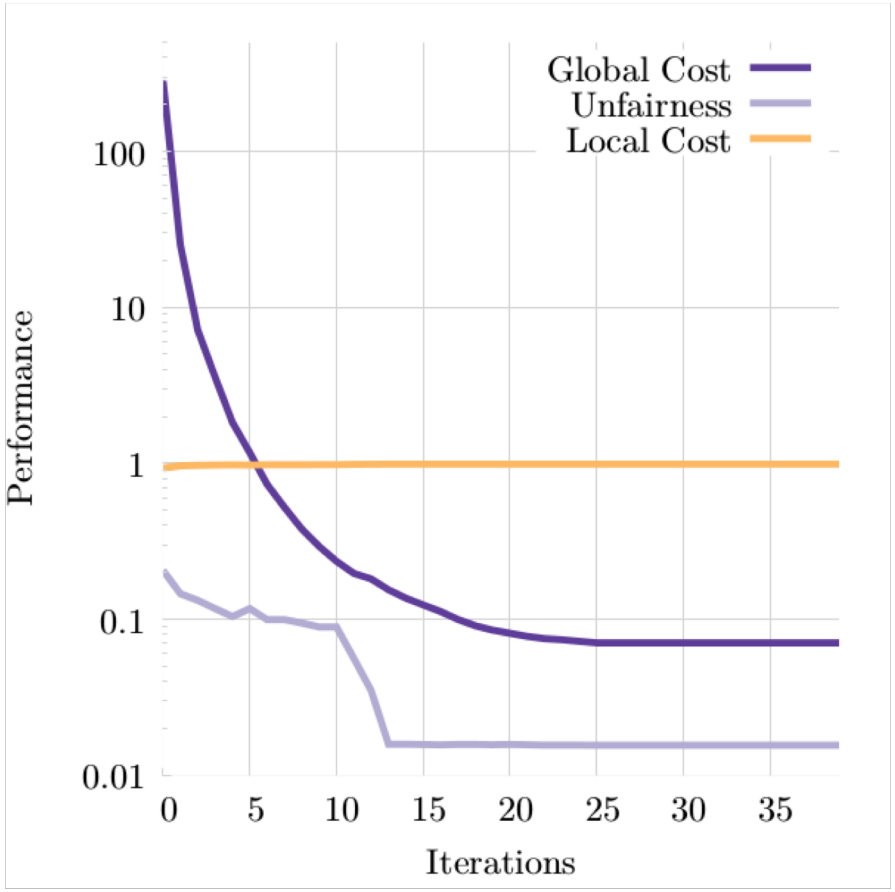


global-complex-cost.csv

- Contains results from `GlobalComplexCostLogger`.

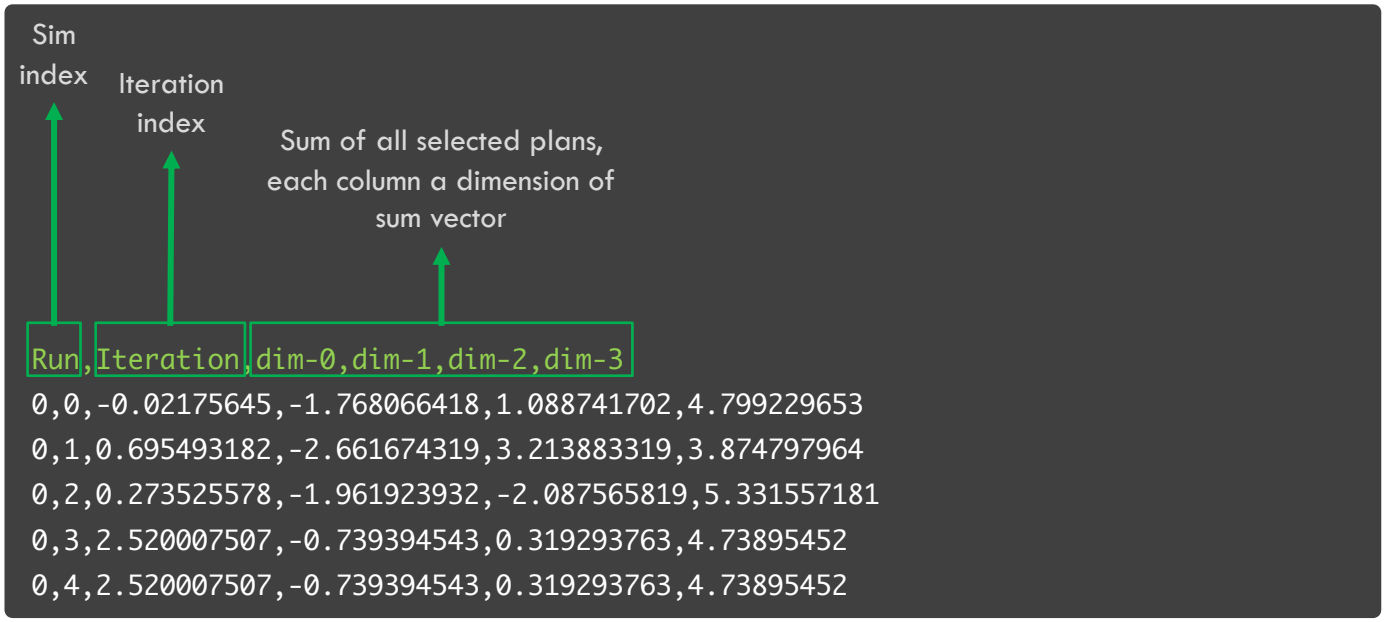


Plotting Cost Outputs

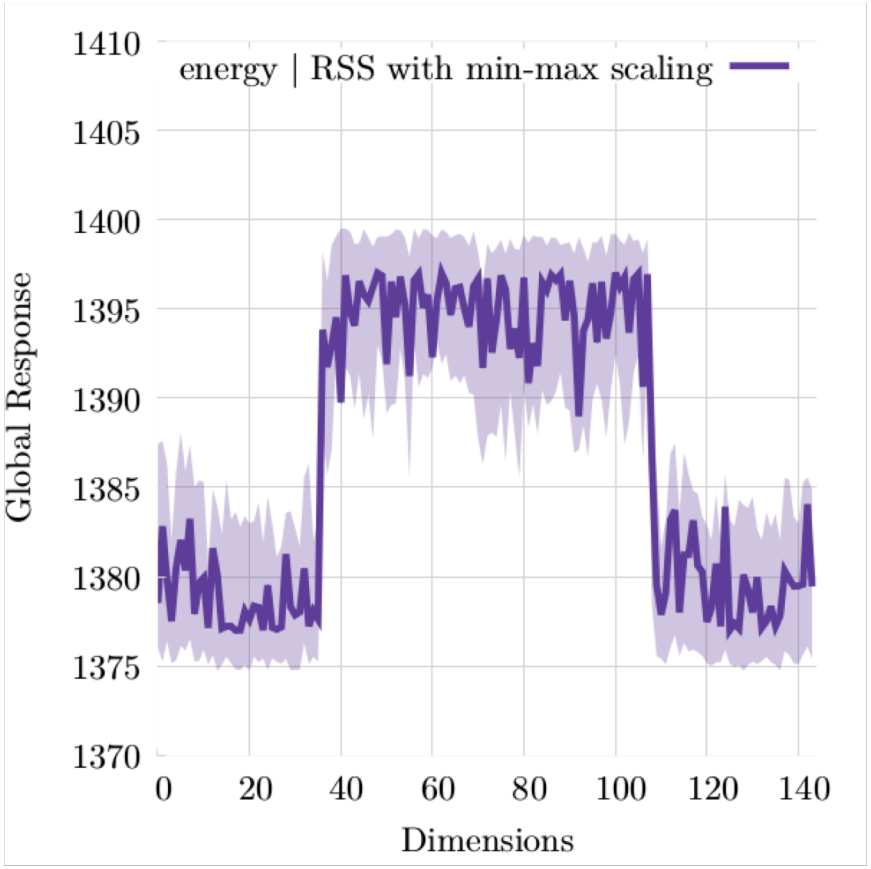


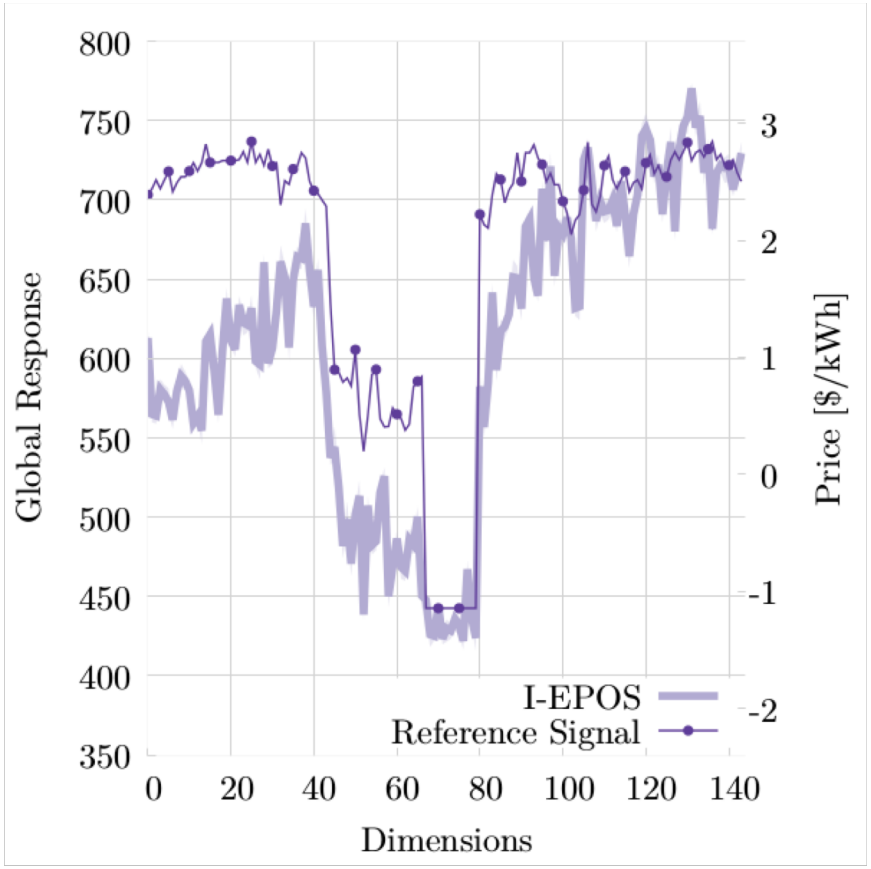
global-response.csv

- Contains results from `GlobalResponseVectorLogger`.



Plotting Global Response



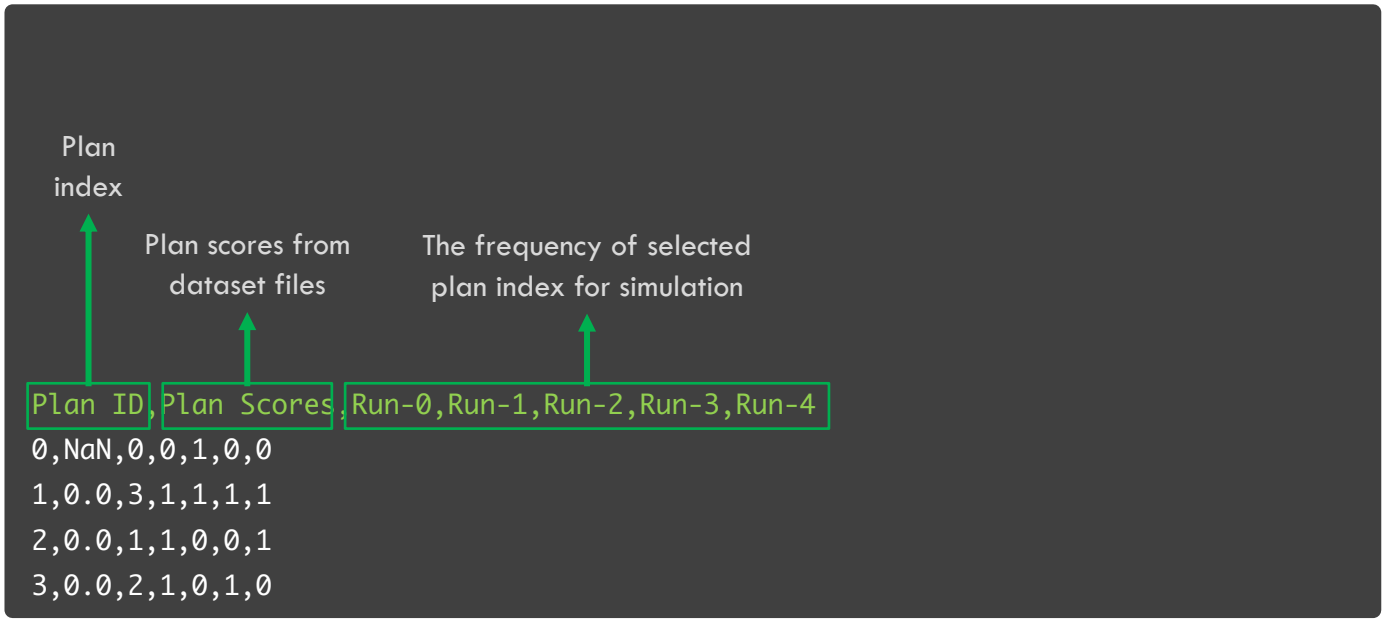


selected-plans.csv

- Contains results from `SelectedPlanLogger`.

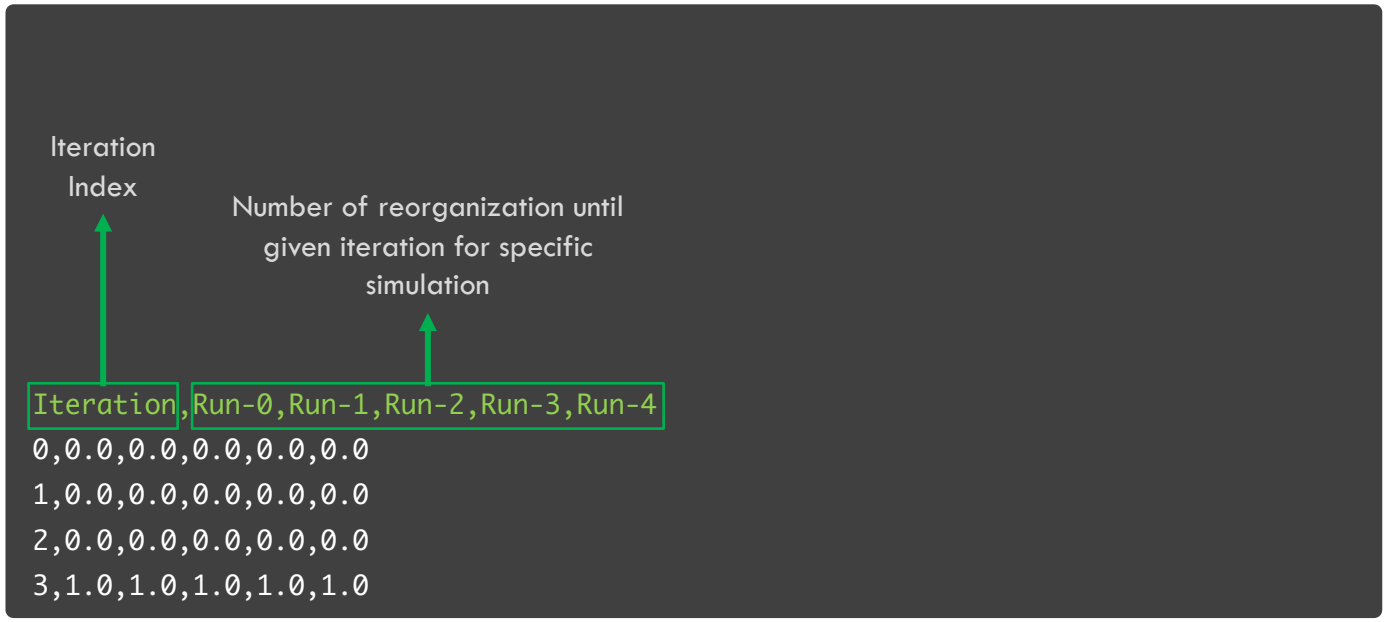
indexes-histogram.csv

- Contains results from `PlanFrequencyLogger`.
- Plan id and score might not make sense for all datasets

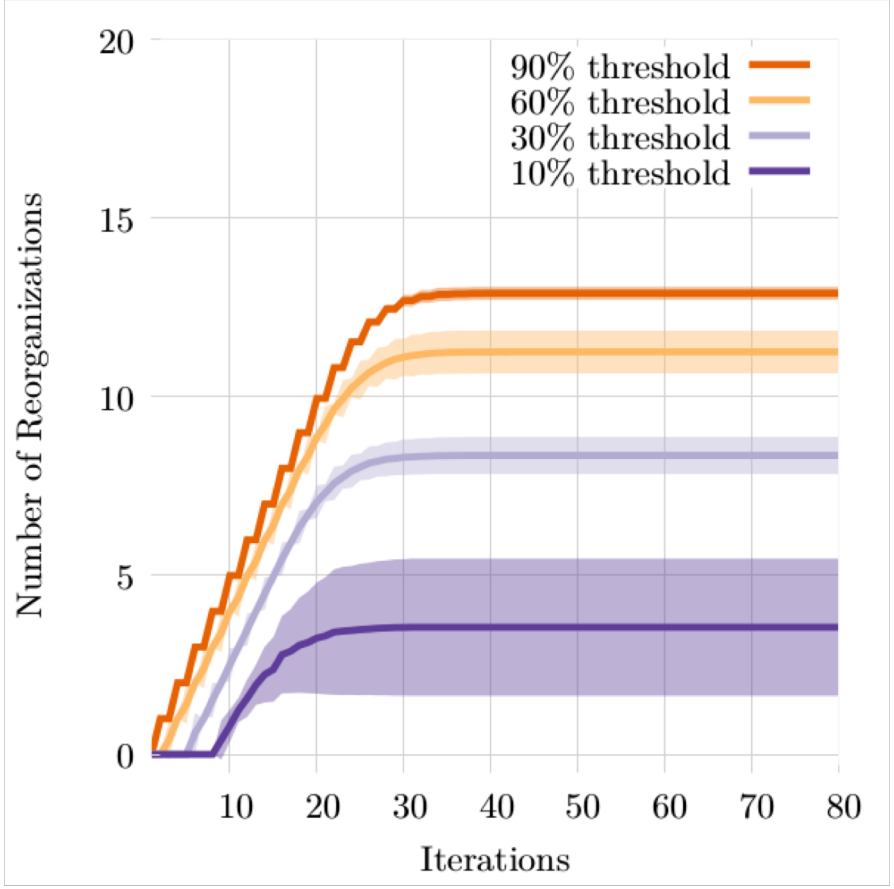


num-reorganizations.csv

- Contains results from `ReorganizationLogger`.

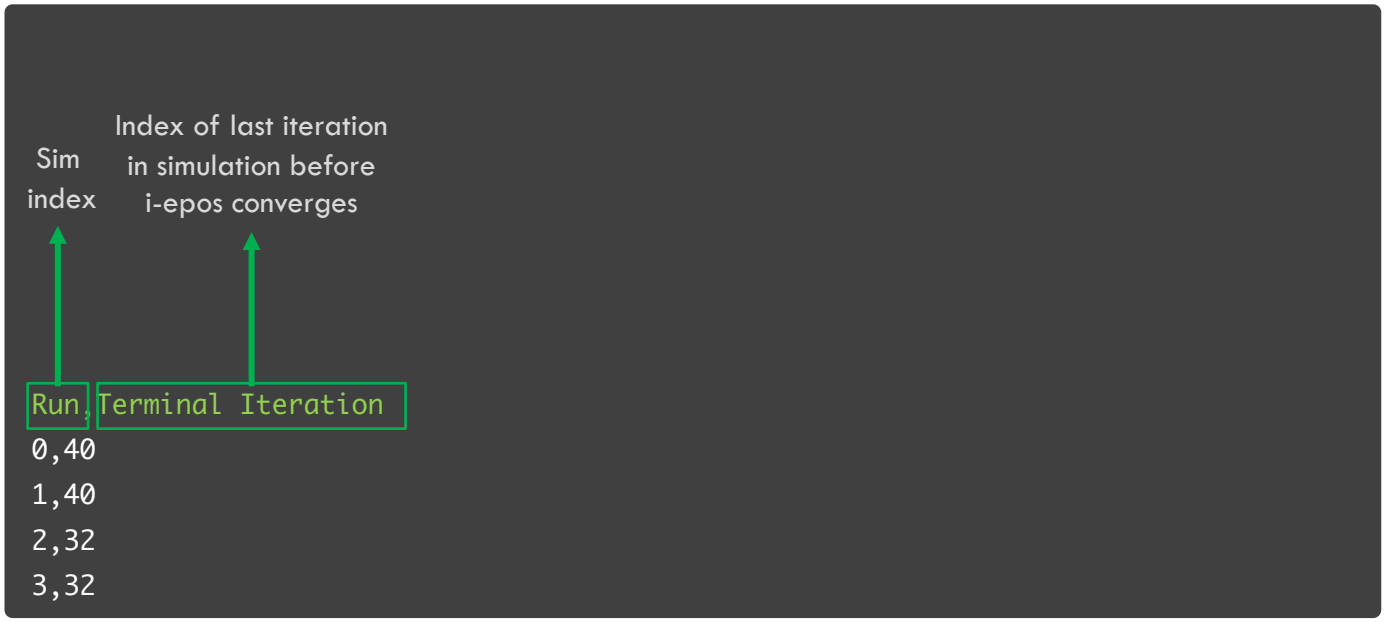


Plotting Reorganizations



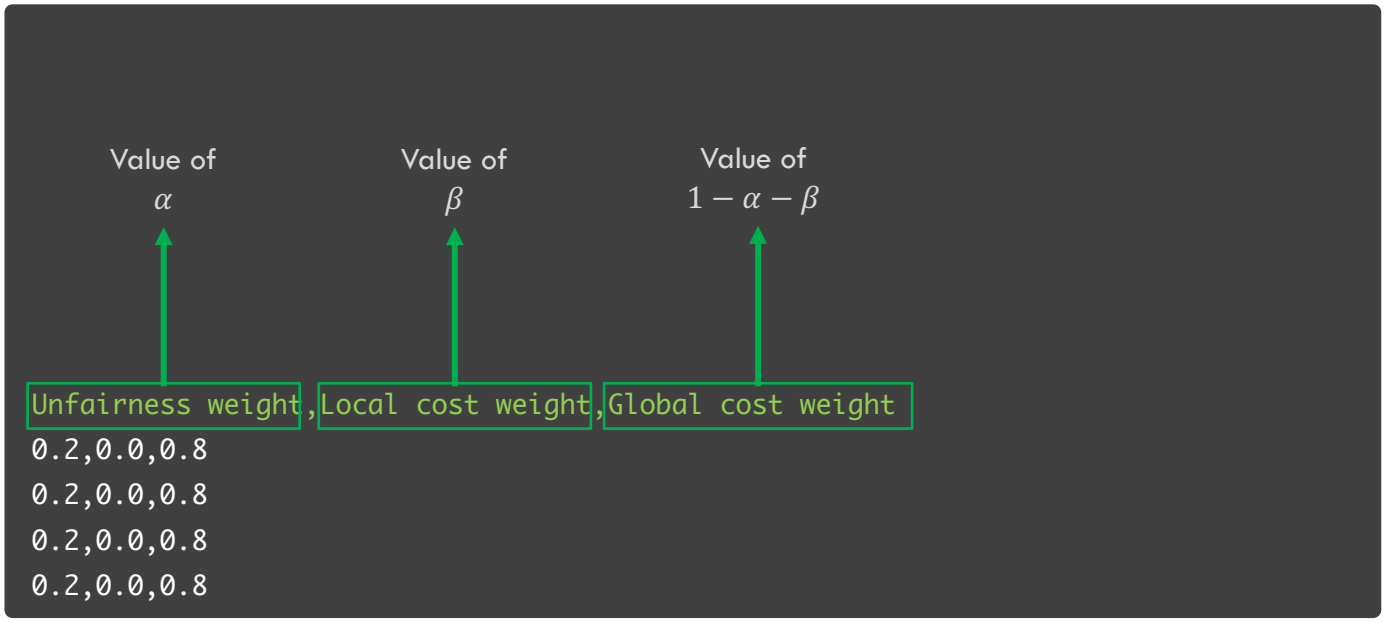
termination.csv

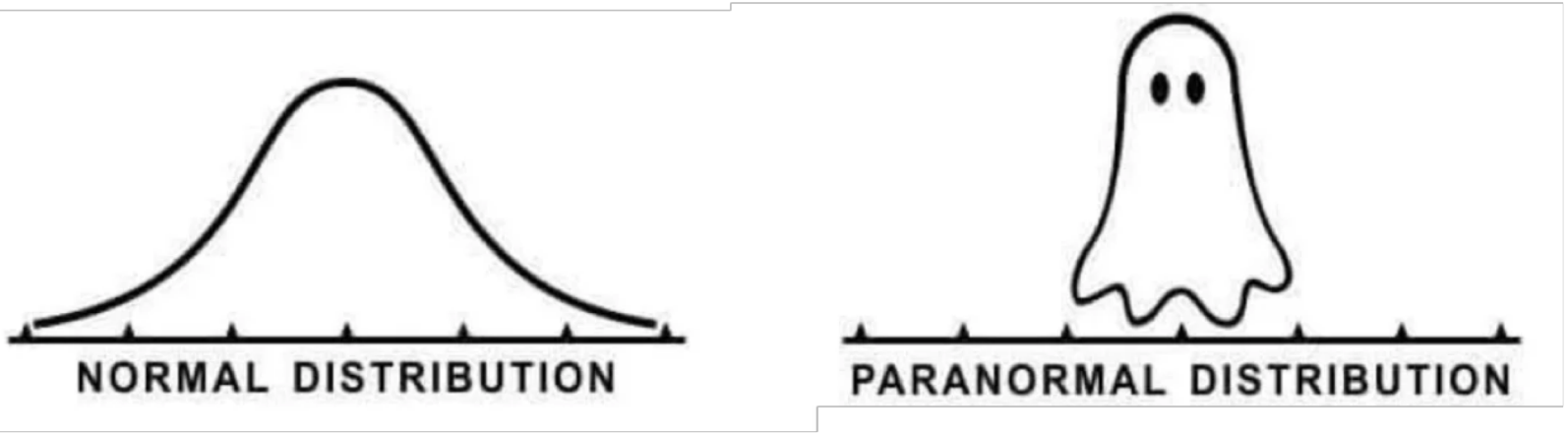
- Contains results from `TerminationLogger`.
- Terminal iteration → detection of convergence



weights-alpha-beta.csv

- Contains results from `WeightsLogger`.
- Custom implementations with weight changing strategies.





Dataset Creation and Experiment Evaluation

Dataset Creation

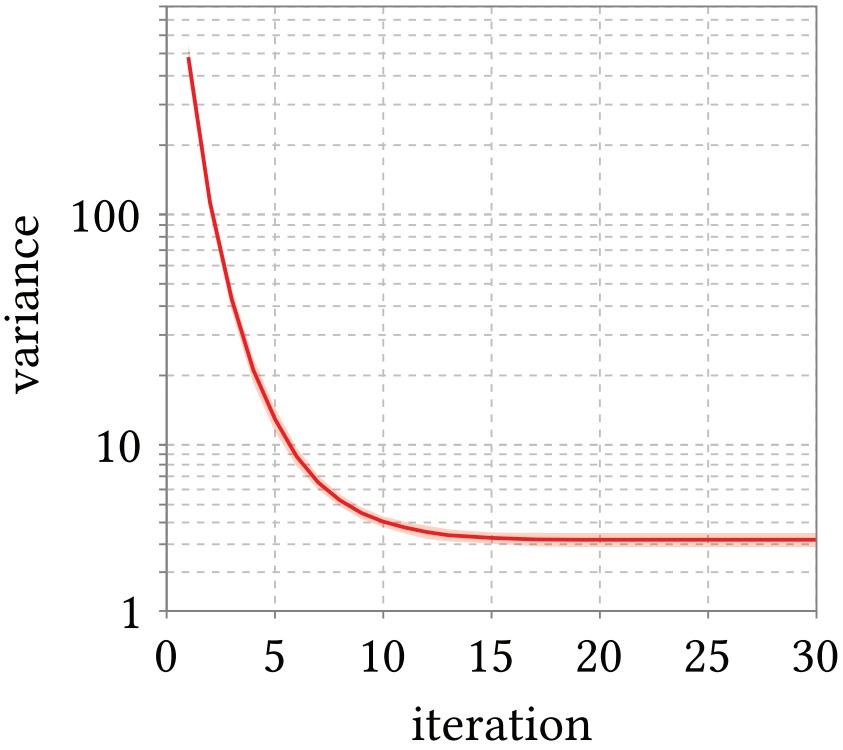
- File stucture, agent and goal signal files should explicitly follow the provided guidelines.
- How do I make a plan? What does it mean?

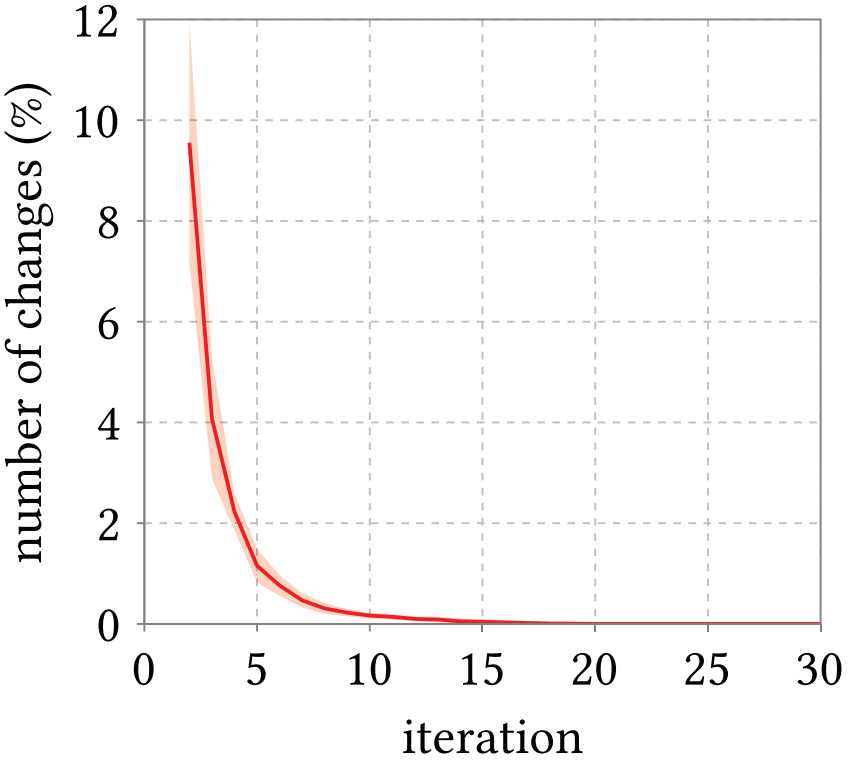
Scenario	Plan Values	Dimensions	Plan Score
Power grid	power consumption	Time	Discomfort
Bicycle Sharing	Incoming/outgoing bicycles	Bike stations	Preferred
Electrical Vehicles	Power battery charge	Time	Preference

Evaluation Criteria

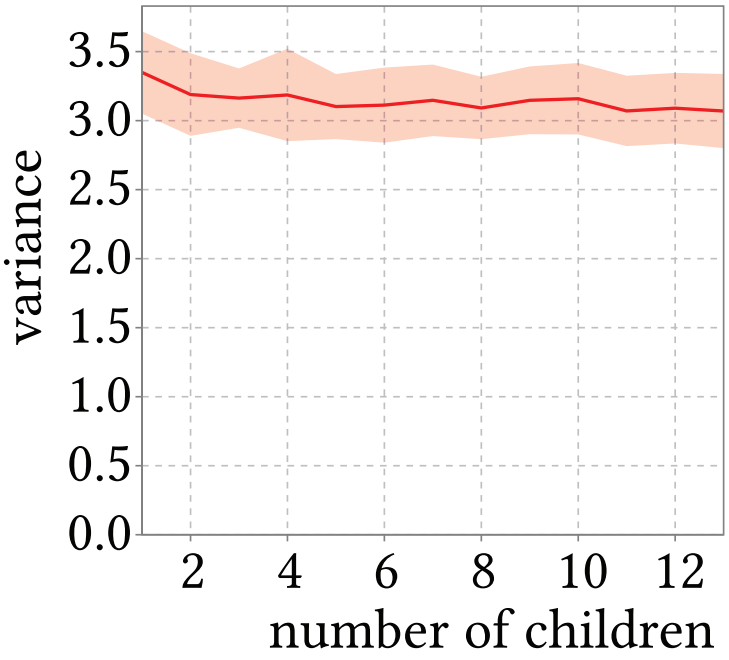
- Repeat multiple experiments with different seeds for statistical significance
- Expected behavior: usually the average between simulations
- Possible divergence: usually the variance between simulations
- Outlier and extreme behaviors: values usually outside confidence intervals

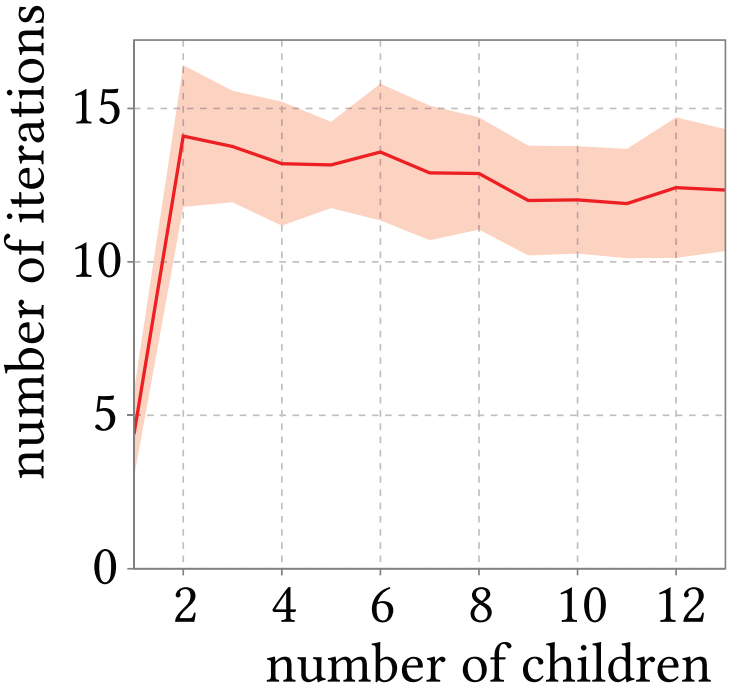
Plots ~ Timeseries



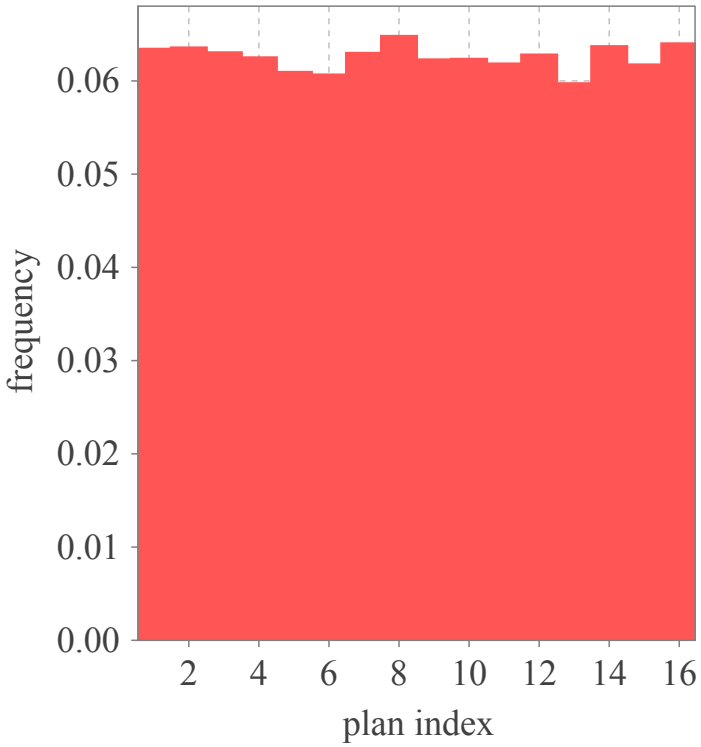


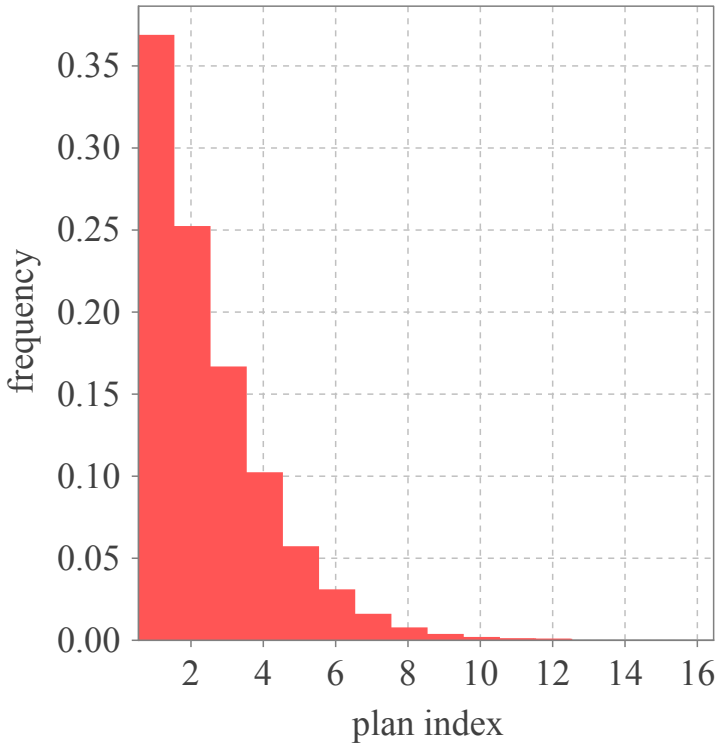
Plots ~ Varying Parameters





Plots ~ Distributions

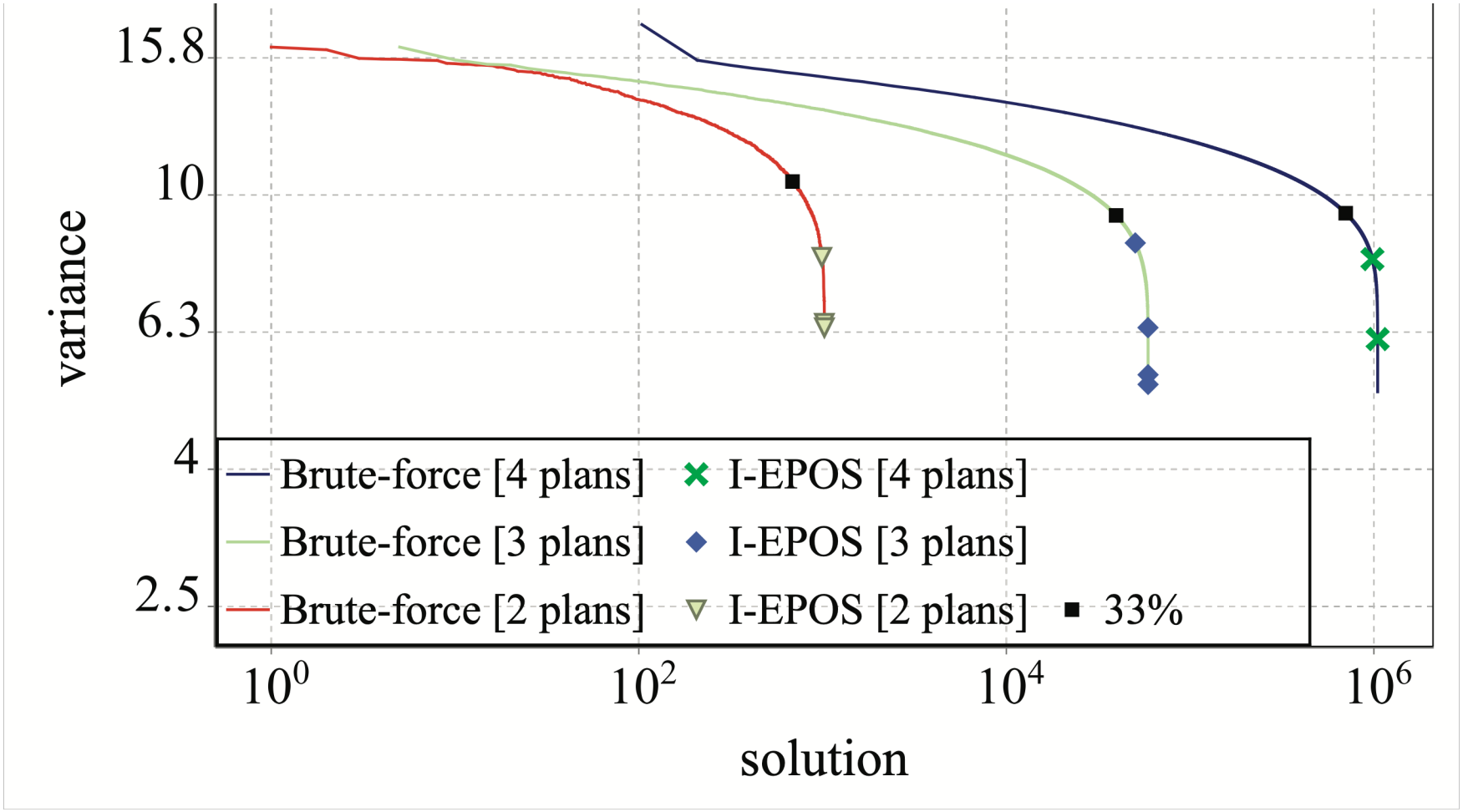




Optimality

- Evaluate via comparison to other plan selection algorithms
- Common: Random plan selection
- Select agent plans randomly and calculate all costs
- Compare improvement to average or best random plan selection by using EPOS plan selection
- For low number of agents and possible plan selections, random can be replaced by exhaustive search
- Evaluation: Place epos solution on the sorted exhaustive search cost.
- Or compare with: setting $\alpha = 0, \beta = 1$ creates a greedy selection, minimizing the local cost function (optimizing plan score)

Optimality plots



ating.java ×

```
    final CompletableFuture<Integer> answer =
            CompletableFuture.completedFuture(42);

    final int fortyTwo = answer.get();  //does not block
}

/**
 * Built-in thread pool
 */
@Test
public void supplyAsync() throws Exception {
    final CompletableFuture<String> java =
            CompletableFuture.supplyAsync(() ->
                    client.mostRecentQuestionAbout( tag: "java")
            );
    log.debug("Found: '{}'", java.get());
}
```

Code Extensions

Setup the Project Code

Prerequisites for coding:

- jdk1.8.x_xx or higher
- Maven 3.3 or higher
- Git

```
> ~/Desktop u$ git clone -b tutorial --single-branch https://github.com/epournaras/EPOS
> ~/Desktop u$ cd EPOS
> ~/Desktop u$ mvn clean install –U #or where the project pom.xml is
```

Now you can open the project from the folder of the pom.xml with any IDE as Maven project, and it should work fine!

Experiment Class

```
package experiment;


/**
 * @author Jovan N., Thomas Asikis
 *
 */
public class ReorganizationExperiment {

            public static void runSimulation(int numChildren, // number of children for each middle node
                                             int numIterations, // total number of iterations to run for
                                             int numAgents, // total number of nodes in the network
                                             Function<Integer, Agent> createAgent, // lambda expression that creates an agent
                                             Configuration config) . . .
```

Cost Functions

```
public interface PlanCostFunction<V extends
DataType<V>> {
    public double calcCost(Plan<V> plan);
    public abstract String getLabel();
}
```

```
public class SqrDistCostFunction implements PlanCostFunction <Vector> {
    private Vector target;
    public void setCostVector(Vector target) {
        this.target = target;
    }
    @Override
    public double calcCost(Vector vector) {
        Vector v = vector.cloneThis();
        v.subtract(target);
        v.pow(2);
        return v.sum();
    }
    @Override
    public String getLabel() {
       return "SQR";
    }
 }
```

Loggers

```
public abstract class AgentLogger<A extends Agent> implements Cloneable {
    int run;
    public void setRun(int run) {
        this.run = run;
    }
    public abstract void init(A agent); //first by agent
    public abstract void log(MeasurementLog log, int epoch, A agent); // at every iteration/epoch to log
    public abstract void print(MeasurementLog log); //in the end to print on console or to a file

    @Override
    public AgentLogger<A> clone() // should be already implemented
}
```

MeasurementLog & Aggregates

- Belongs to protopeer* codebase which is available upon request
- MeasurementLog: usually in logger.log():

```
public abstract void log(MeasurementLog log, int epoch, A agent){
  double value = agent.getSomeDoubleValue(); // the value to log
  Comparable tag2 = this.getClass().getName(); // an identifier to group the values by, e.g. all the values of this
logger
  Comparable tag1 = agent.getSimulationID(); // an identifier to group the values by, e.g. for this simulation id
  log.log(epoch, Comparable tag1, tag2, value); // the log call with input types: log(int, Object, Object, double)
}
```

- Aggregate: usually in logger.print():

```
public abstract void print(MeasurementLog log){
  Aggregate agg = log.getAggregateByEpochNumber(int epochNumber, Object tag1, Object tag2); // possible aggregates
grouped by tags
  agg.getAverage();
  //etc...
}
```

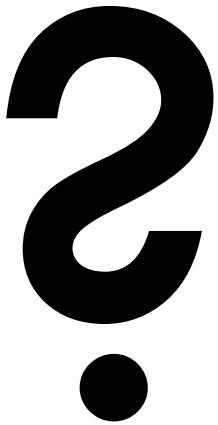
EPOS

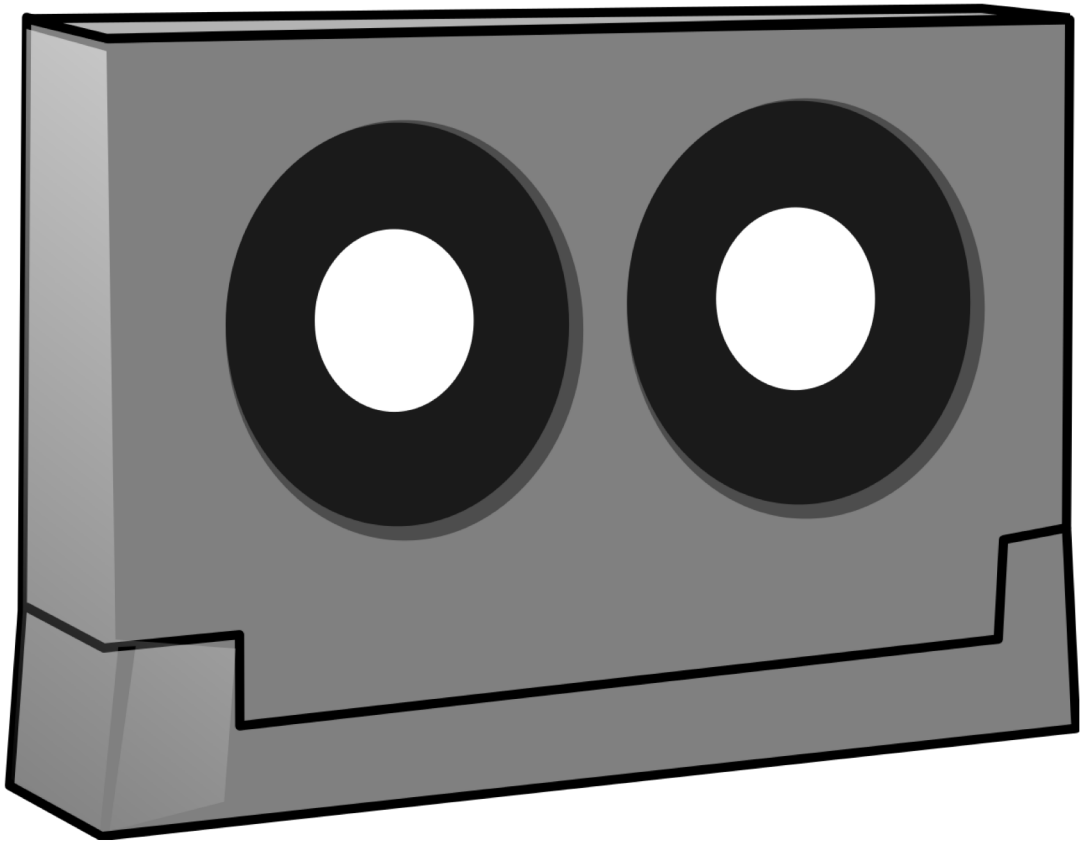
* W. Galuba, K. Aberer, Z. Despotovic and W. Kellerer, "ProtoPeer: Distributed systems prototyping toolkit," *2009 IEEE* http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5284513&isnumber=5284489

Final Remarks

- **Documentation:** http://epos-net.org/i-epos/software/documentation
- **Slack:** https://self-org-multi-agent.slack.com/
- **Development support in slack/email:**
  thomasA / asikist@ethz.ch
  Jovan Nikolic / jovan.nikolic@gess.ethz.ch
  Farzam / farzam.fanitabasi@gess.ethz.ch
  Evangelos Pournaras / epournaras@ethz.ch
- **Code extensions are welcome but not mandatory**

Questions

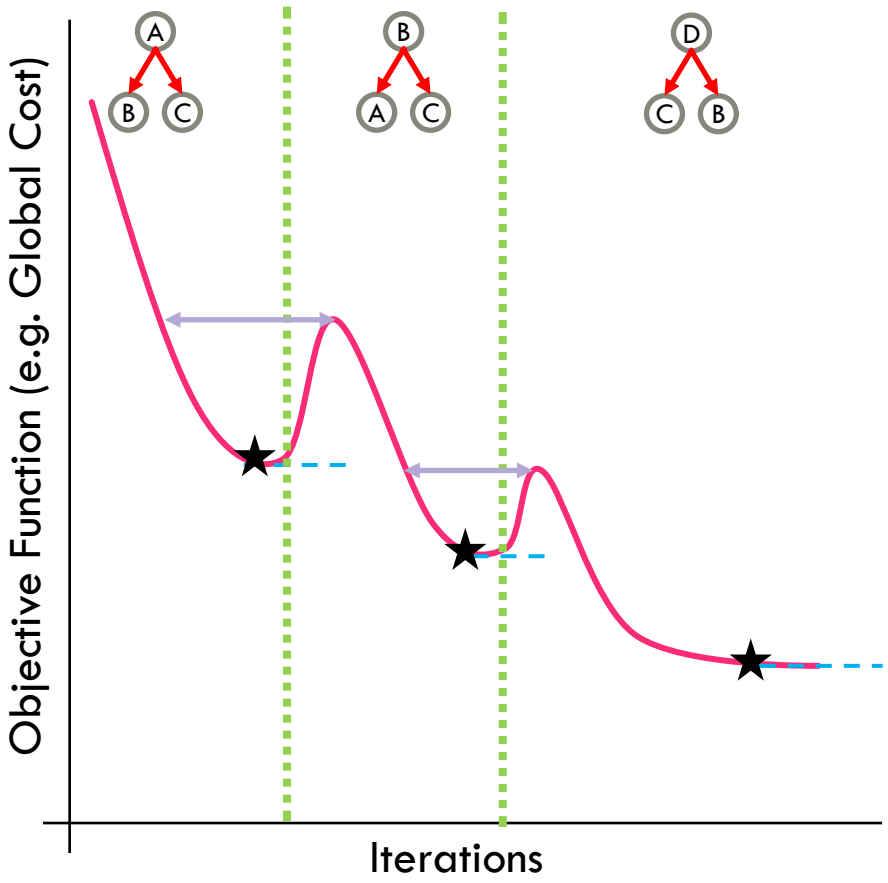




Backup Slides

Just in case

Understanding Reorganization



Symbol	Meaning
★	Local Optimum
- - -	Convergence
⋮	Reorganization of agent tree
↔	Revert to agents choices on iteration