

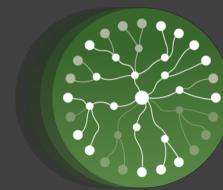


# EPOS Code Tutorial

Presented

by Thomas Asikis, [asikist@ethz.ch](mailto:asikist@ethz.ch)

Self-organizing Multi-agent Systems



<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.)

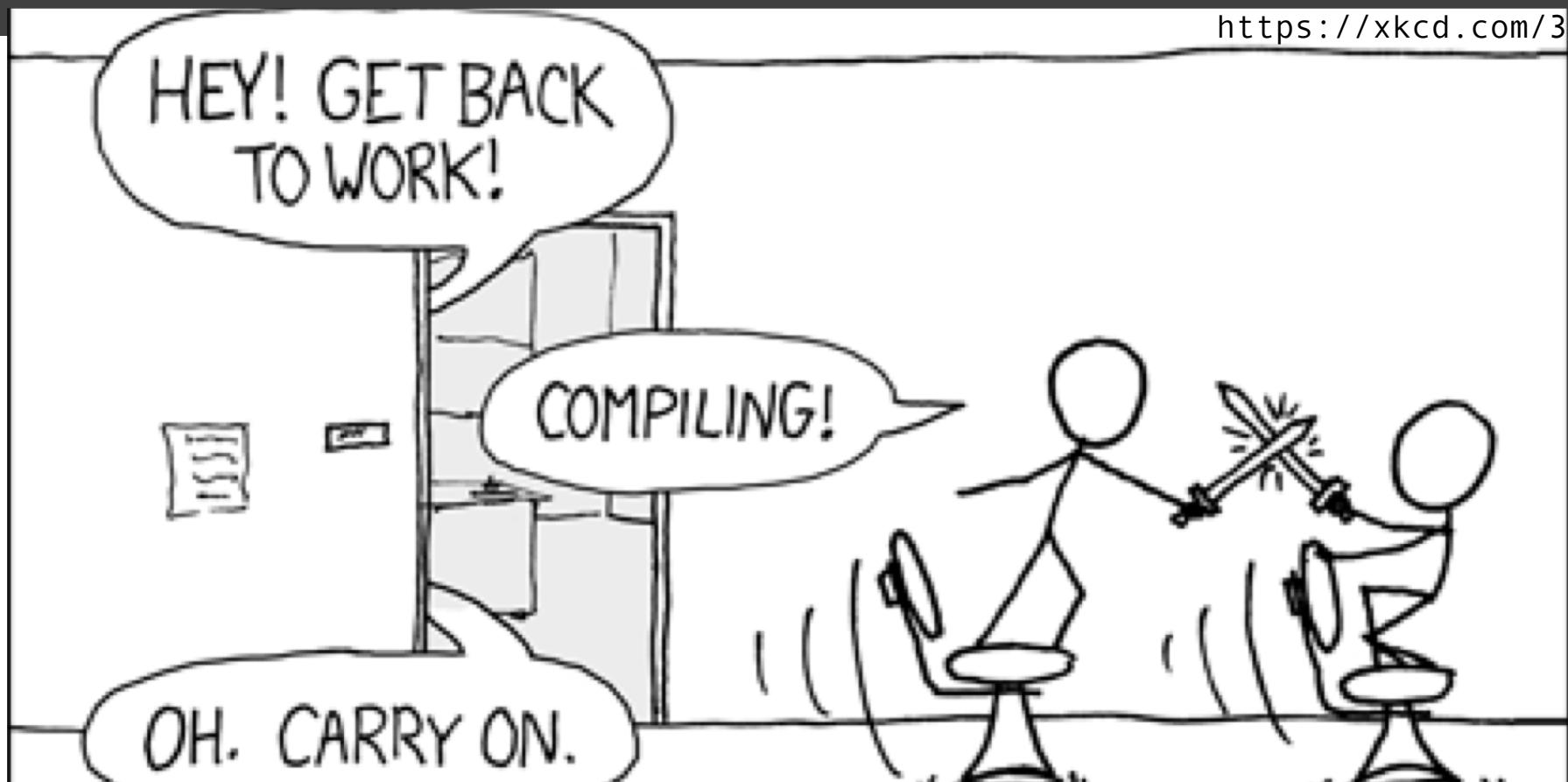
# Today's Presentation

Basic usage as black box: input/output hyperparams

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

Advanced:

- Extend EPOS code
- Design and implement new concepts



## 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:

plan score

0.2:0.1,0.2,0.3,0.4,-0.5

**important**  
**plans of same size**

plan values

# 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.
- $\text{planDim} < \text{goalDim} \rightarrow$  crop goal signal from beginning to  $\text{planDim}$
- $\text{planDim} > \text{goalDim} \rightarrow$  repeat goal signal vector till  $\text{planDim}$  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):

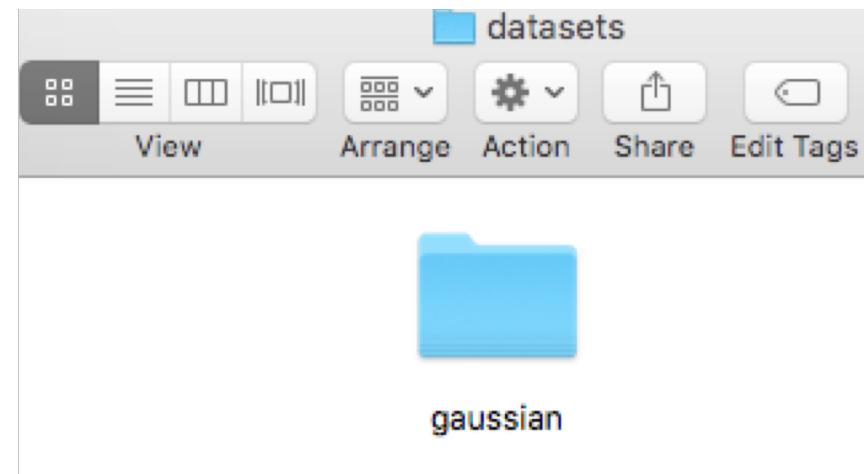
```
conf/epos.properties
```

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.

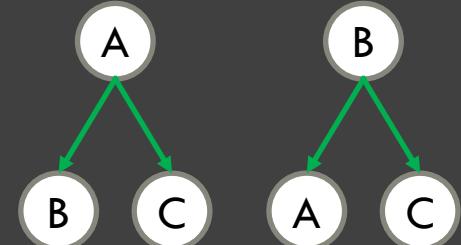
Several repetitions from random start → statistical significance

```
#### Shuffle seeds ####

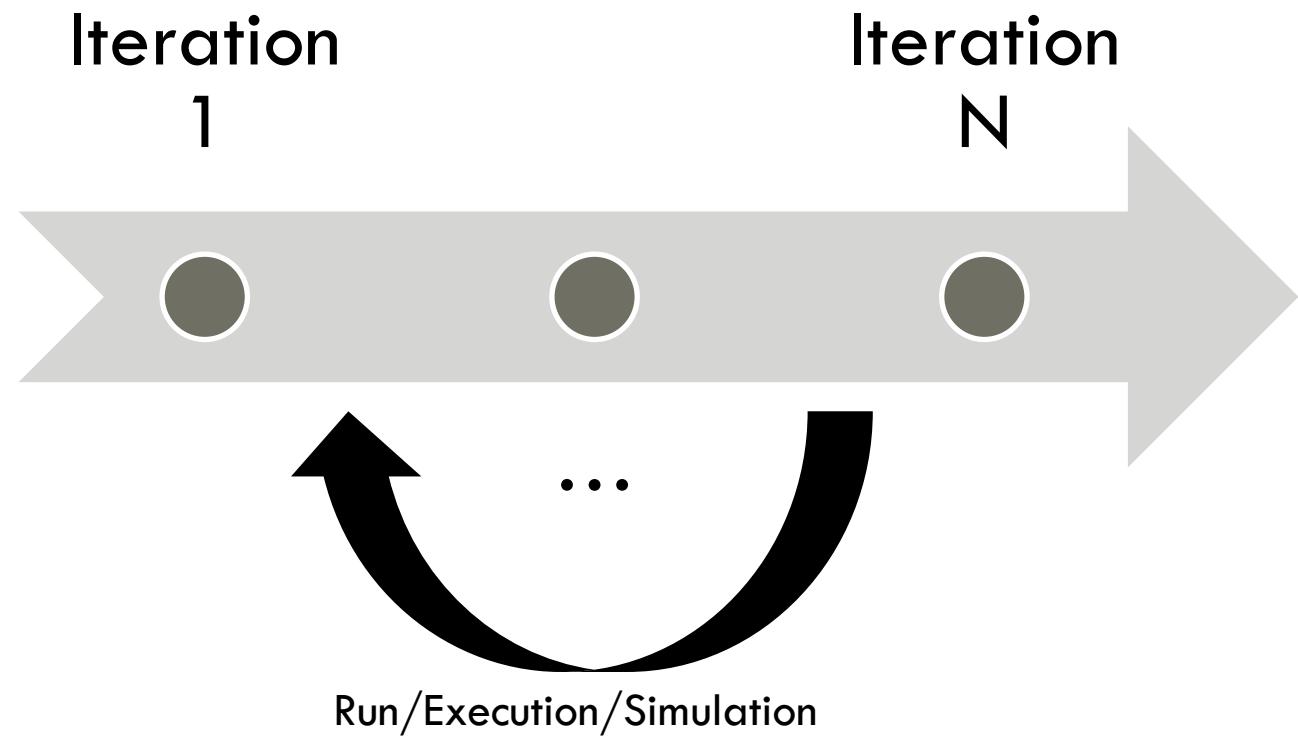
# number of simulations for whole epos experiments
# any integer > 0
numSimulations=5

# 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"
```

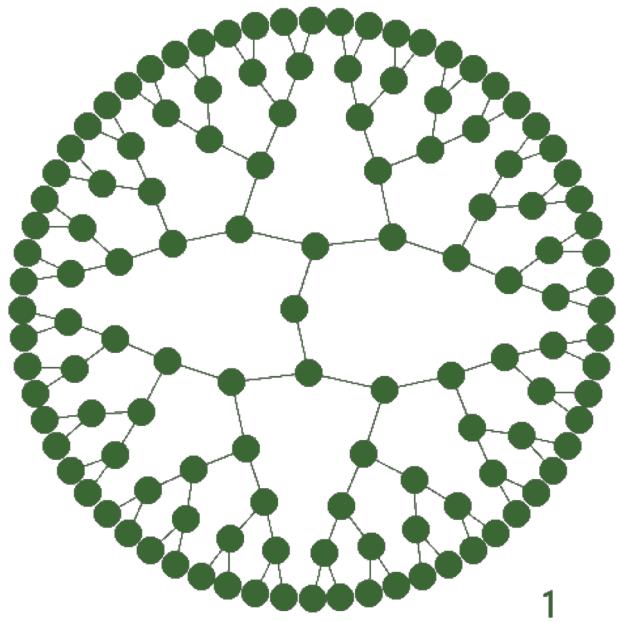
# Output

## 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

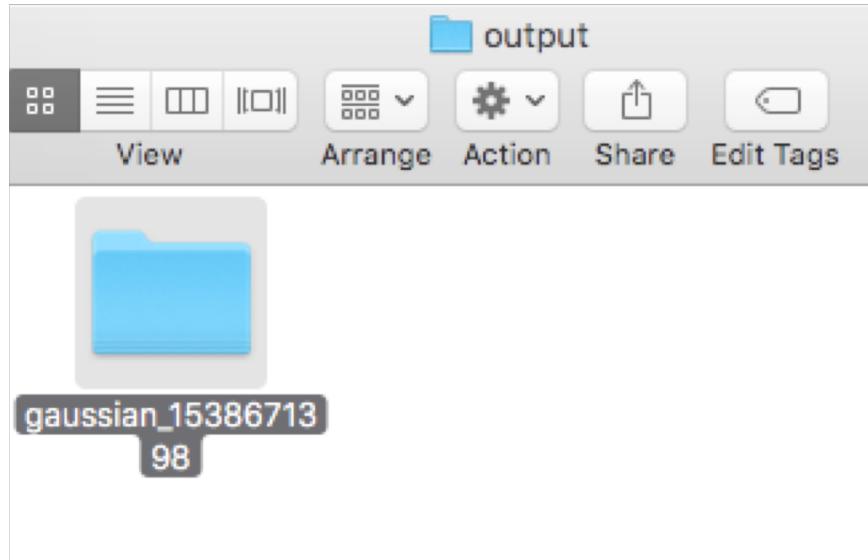
# A first Run

```
### 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
```

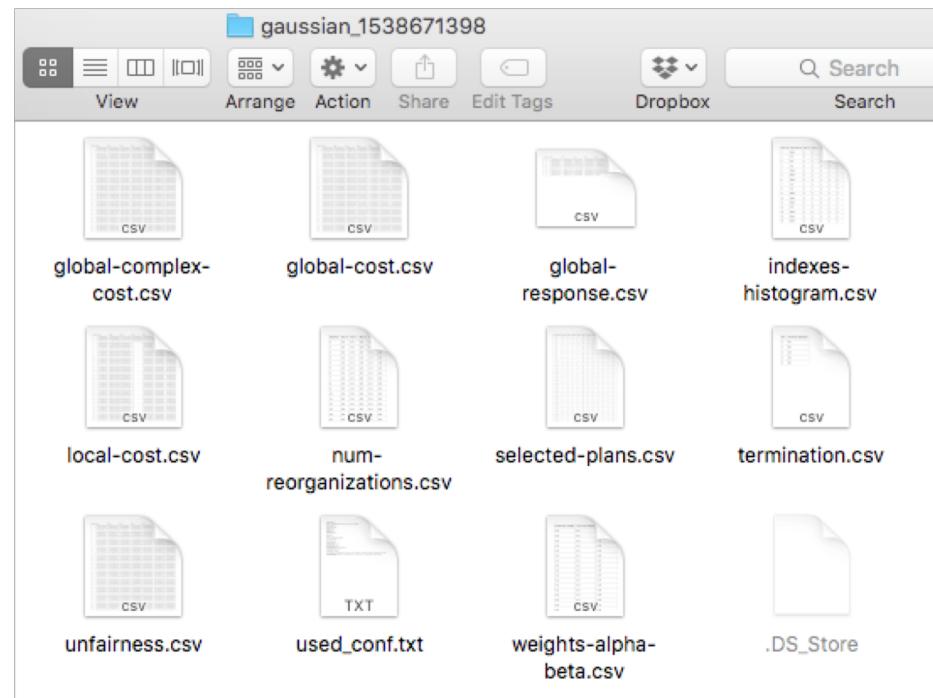
# Experiment Output

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.

Iteration index	Mean and st. deviation of cost/iteration over all simulations	Exact cost values on given iteration on given simulation. Controlled by numSimulations
Iteration,Mean,Stdev,Run-0,Run-1,Run-2,Run-3,Run-4		
0,6.059968011953338,0.605154483052378,6.27882118601477,6.725920874475804,5.412928752040584,6.609195202407165,5.2729740448283655		
1,4.6617281134323205,0.24804483477801084,4.616099255907369,5.053207086861703,4.501590486613892,4.802757307939723,4.334986429838915		
2,4.150807000167388,0.1403666018337861,4.062661582370829,4.222059522048272,4.105797655644052,4.3839036041588795,3.979612636614905		

# local-cost.csv

- Contains results from LocalCostMultiObjectiveLogger.

Iteration index	Mean and st. deviation of cost/iteration over all simulations	Exact cost values on given iteration on given simulation. Controlled by numSimulations
		Iteration,Mean,Stdev,Run-0,Run-1,Run-2,Run-3,Run-4
		0,6.50909090909091,0.6019252857180569,6.27272727272725,5.5454545454546,6.45454545454,7.27272727272725,7.0
		1,6.30909090909095,1.0532907896698391,5.27272727272725,5.72727272727275,6.72727272727275,5.6363636363637,8.1818181818182
		2,6.636363636363636,0.7670917494446927,5.27272727272725,6.909090909090909,7.272727272725,6.3636363636363,7.3636363636363

# unfairness.csv

- Contains results from UnfairnessLogger.

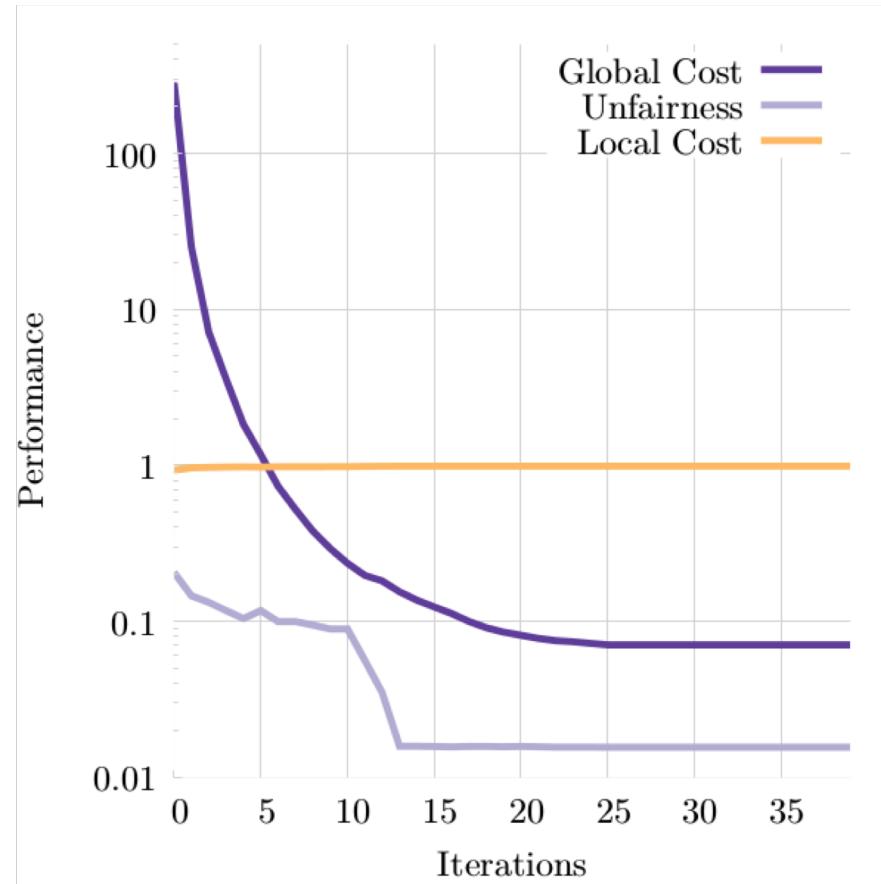
Iteration index	Mean and st. deviation of cost/iteration over all simulations	Exact cost values on given iteration on given simulation. Controlled by numSimulations
Iteration,Mean,Stdev,Run-0,Run-1,Run-2,Run-3,Run-4		
0,3.8809105747895294,0.25000010527111544,3.886828786623991,4.3141129155289715,3.916483496097822,3.5698461665585217,3.7172815091383407		
1,3.713206171232129,0.3906351209149316,3.768069413829583,4.158035136737917,4.047241689009465,3.5232404318320176,3.069444184751659		
2,3.5622883009127237,0.43524809494763994,3.71950409917796,3.6792920626996475,4.2231162361225225,2.9625764976000593,3.226952608963428		

# global-complex-cost.csv

- Contains results from GlobalComplexCostLogger.

Iteration index	Mean and st. deviation of cost/iteration over all simulations	Exact cost values on given iteration on given simulation. Controlled by numSimulations
		Iteration,Mean,Stdev,Run-0,Run-1,Run-2,Run-3,Run-4
		0,5.6241565245205765,0.5012549087015863,5.800422706136614,6.243559282686438,5.113639700852032,6.001325395237437,4.9618355376903605
		1,4.472023724992282,0.25453758780542807,4.446493287491812,4.874172696836946,4.410720727093007,4.546853932718182,4.0818779808214645
		2,4.033103260316454,0.11250762633790407,3.9940300857322555,4.113506030178547,4.129261371739746,4.099638182847116,3.8290806310846097

# Plotting Cost Outputs



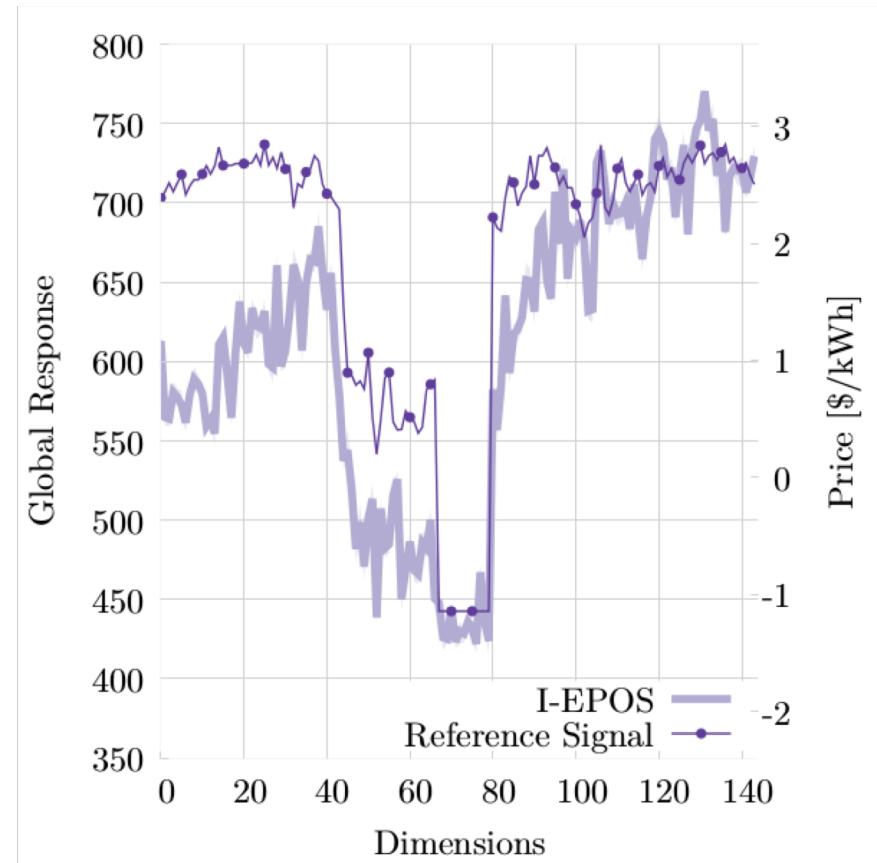
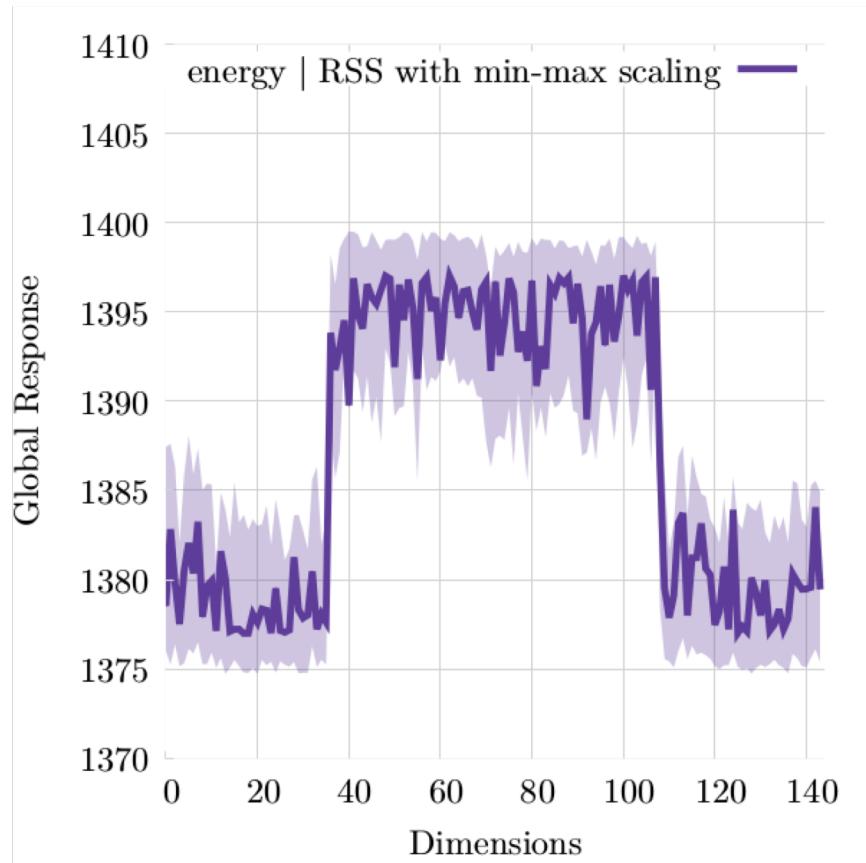
# global-response.csv

- Contains results from GlobalResponseVectorLogger.

Sim index	Iteration index	Run	Iteration	dim-0	dim-1	dim-2	dim-3
		0,0,-0.02175645	-1.768066418,1.088741702,4.799229653				
		0,1,0.695493182	-2.661674319,3.213883319,3.874797964				
		0,2,0.273525578	-1.961923932,-2.087565819,5.331557181				
		0,3,2.520007507	-0.739394543,0.319293763,4.73895452				
		0,4,2.520007507	-0.739394543,0.319293763,4.73895452				

Sum of all selected plans,  
each column a dimension of  
sum vector

# Plotting Global Response



# selected-plans.csv

- Contains results from SelectedPlanLogger.

Run	Iteration	agent-0	agent-1	agent-2	agent-3	agent-4	agent-5	agent-6
0,0,11,3,1,5,12,10,9								
0,1,1,3,1,5,12,10,8								
0,2,1,3,1,5,12,8,8								
0,3,1,3,1,7,12,8,8								

Diagram illustrating the structure of the selected-plans.csv file:

- The first row contains column headers: "Run", "Iteration", and "agent-0, agent-1, agent-2, agent-3, agent-4, agent-5, agent-6".
- Subsequent rows provide data for each run, iteration, and agent.
- Annotations with arrows point to specific columns:
  - "Sim index" points to the "Run" column.
  - "Iteration index" points to the "Iteration" column.
  - "The selected plan index from the agent" points to the "agent-0" column.

# indexes-histogram.csv

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

Plan ID	Plan Scores	Run-0	Run-1	Run-2	Run-3	Run-4
0,NaN,0,0,1,0,0						
1,0.0,3,1,1,1,1						
2,0.0,1,1,0,0,1						
3,0.0,2,1,0,1,0						

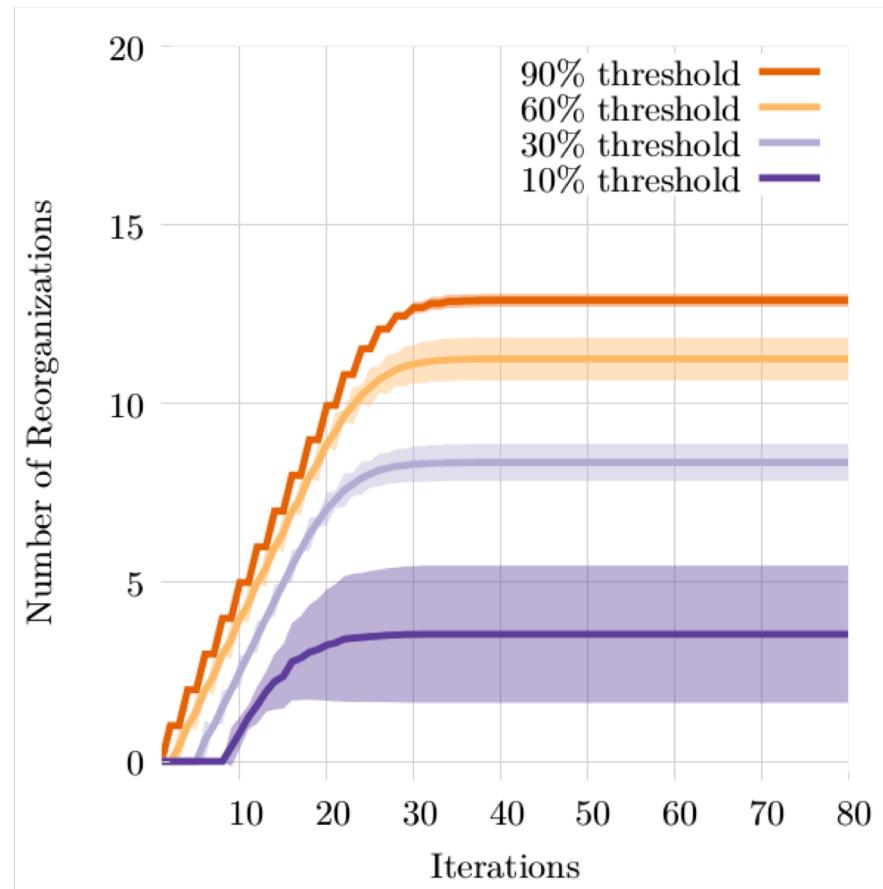
Plan index  
↑  
Plan scores from dataset files  
↑  
The frequency of selected plan index for simulation  
↑

# num-reorganizations.csv

- Contains results from ReorganizationLogger.

Iteration Index	Number of reorganization until given iteration for specific simulation
Iteration	[Run-0, Run-1, Run-2, Run-3, Run-4]
0,0.0,0.0,0.0,0.0,0.0	
1,0.0,0.0,0.0,0.0,0.0	
2,0.0,0.0,0.0,0.0,0.0	
3,1.0,1.0,1.0,1.0,1.0	

# Plotting Reorganizations



# termination.csv

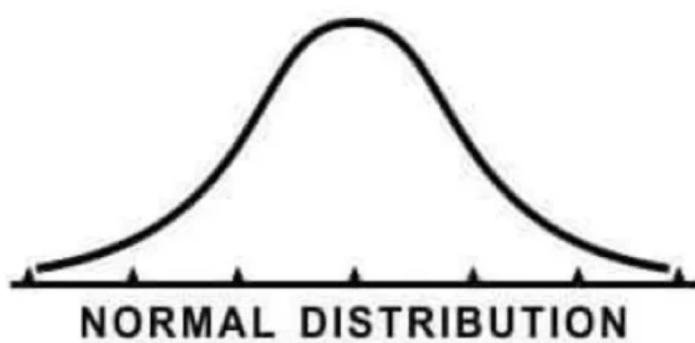
- Contains results from TerminationLogger.
- Terminal iteration → detection of convergence

Sim index	Index of last iteration in simulation before i-epos converges
0	40
1	40
2	32
3	32

# weights-alpha-beta.csv

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

Value of $\alpha$	Value of $\beta$	Value of $1 - \alpha - \beta$
Unfairness weight	Local cost weight	Global cost weight
0.2,0.0,0.8		
0.2,0.0,0.8		
0.2,0.0,0.8		
0.2,0.0,0.8		



## Dataset Creation and Experiment Evaluation

# Dataset Creation

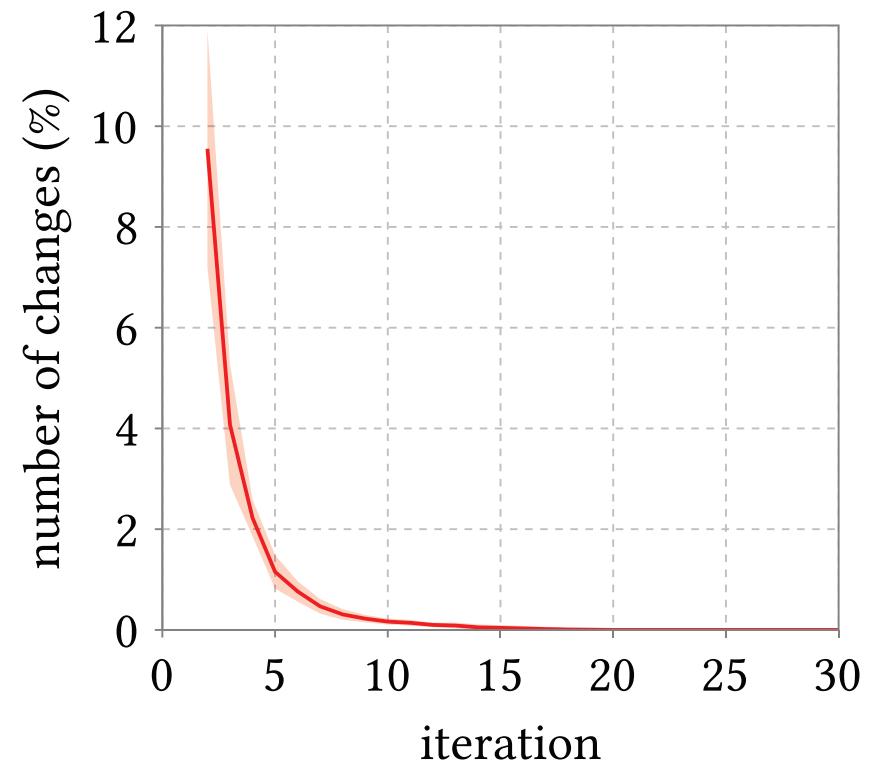
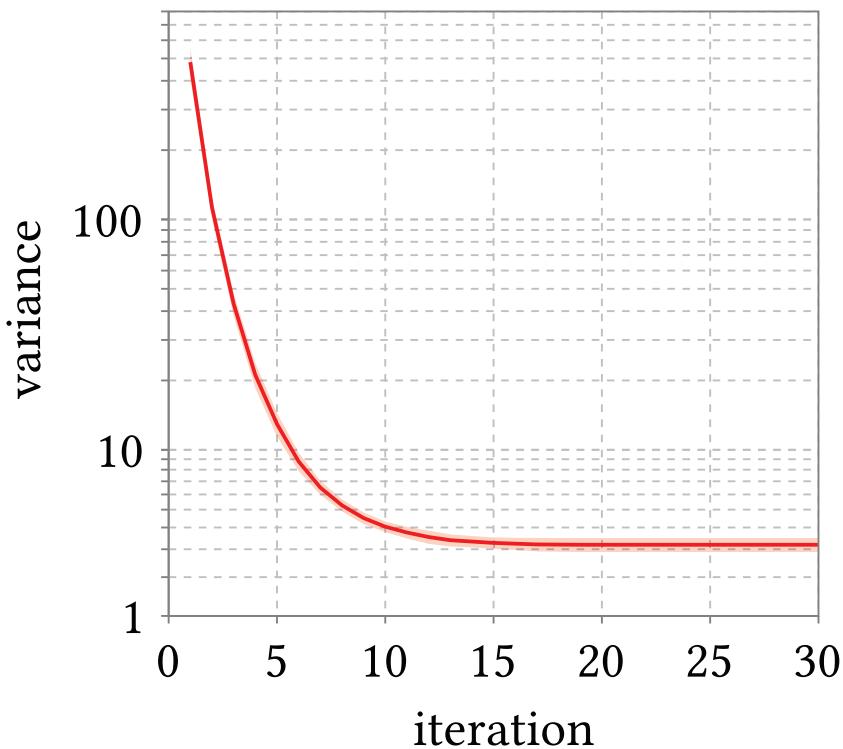
- File structure, agent and goal signal files should explicitly follow the provided guidelines.
- How do I make a plan? What does it mean?

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

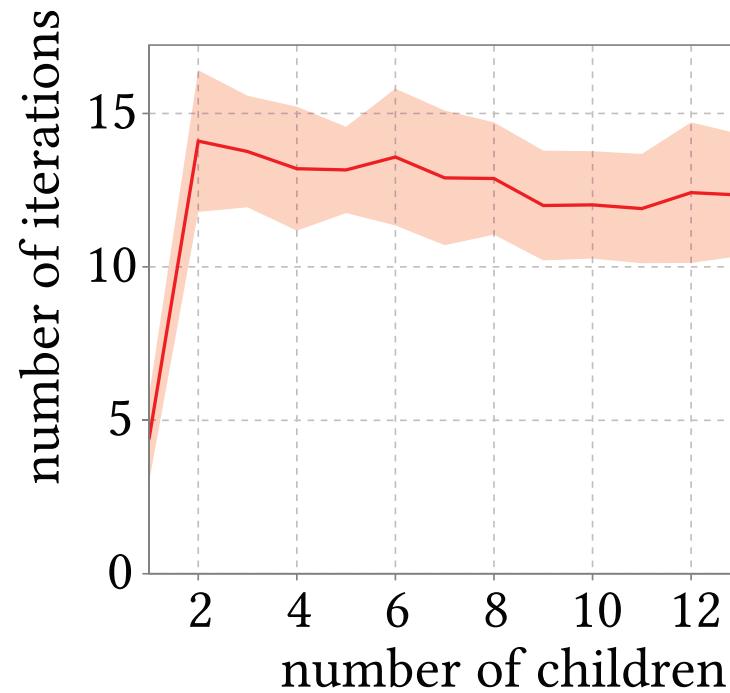
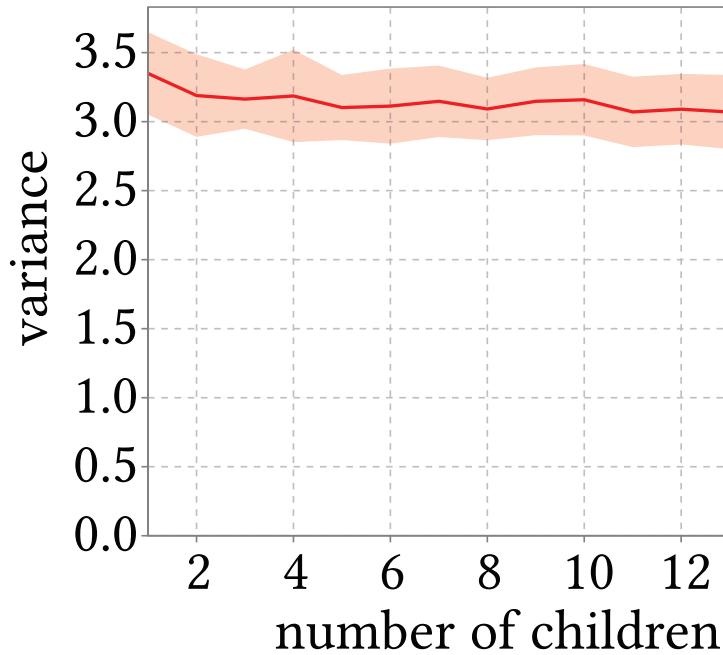
# Evaluation Criteria

- 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

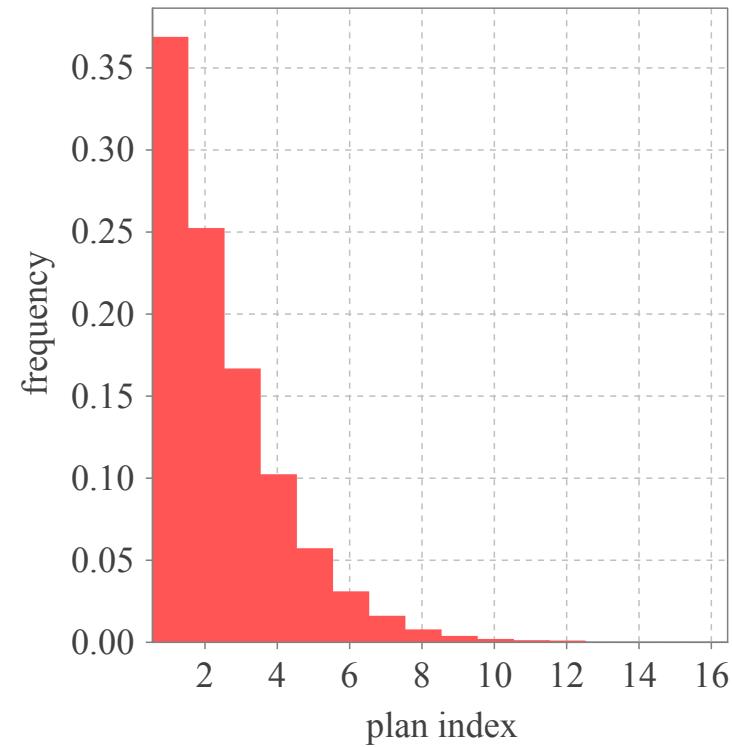
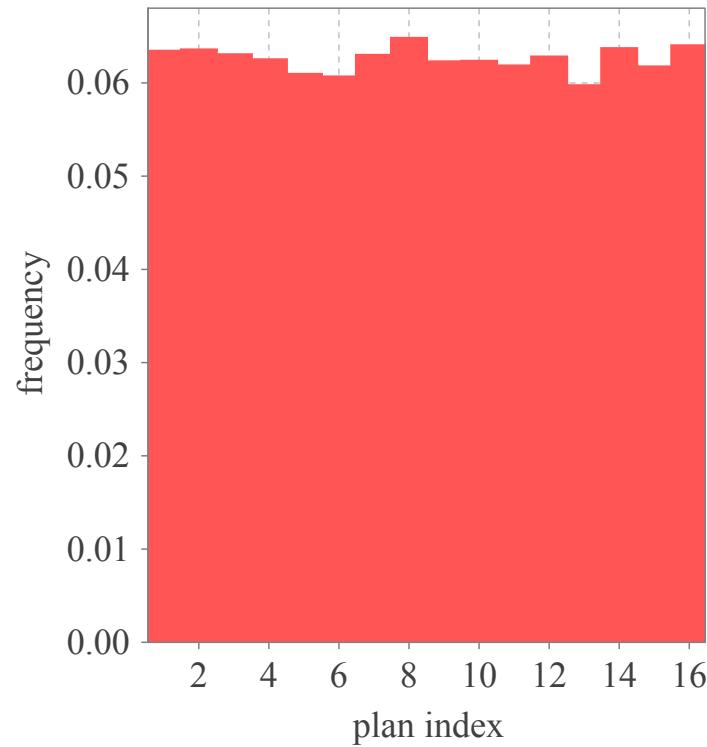
# Plots ~ Timeseries



# Plots ~ Varying Parameters



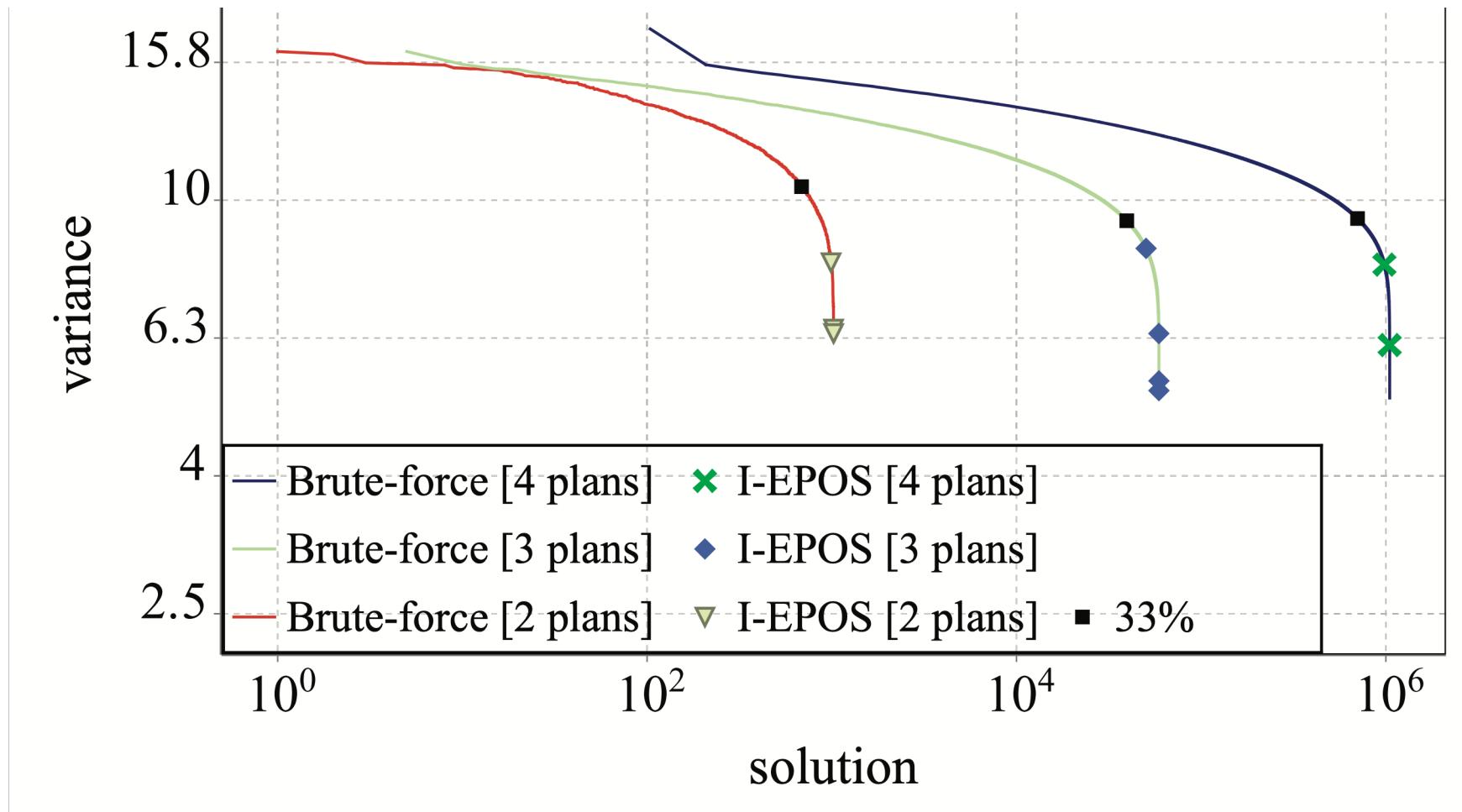
# Plots ~ Distributions



# 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 x

```
    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()`:

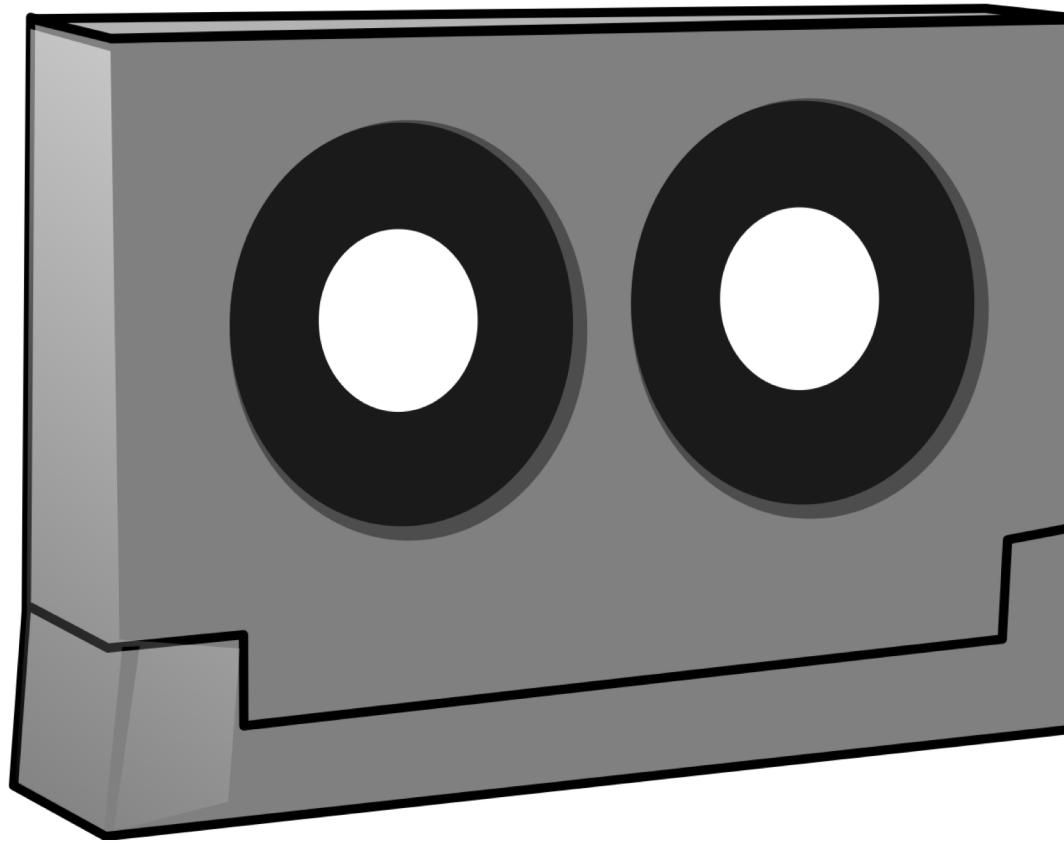
```
public abstract void print(MeasurementLog log){  
    Aggregate agg = log.getAggregateByEpochNumber(int epochNumber, Object tag1, Object tag2); // possible aggregates  
    grouped by tags  
    agg.getAverage();  
    //etc...  
}
```

# 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](mailto:asikist@ethz.ch)**  
**Jovan Nikolic / [jovan.nikolic@gess.ethz.ch](mailto:jovan.nikolic@gess.ethz.ch)**  
**Farzam / [farzam.fanitabasi@gess.ethz.ch](mailto:farzam.fanitabasi@gess.ethz.ch)**  
**Evangelos Pournaras / [epournaras@ethz.ch](mailto:epournaras@ethz.ch)**
- **Code extensions are welcome but not mandatory**

# Questions

?



## Backup Slides

Just in case

# Understanding Reorganization

