I-EPOS Manual

Peter Pilgerstorfer

December 17, 2016

1 Introduction

1.1 Install and setup

TODO: netbeans/eclipse instructions TODO: add libraries

1.2 Execute the sample simulation

TODO: run GUI TODO: run from code: SimpleExperiment.main

2 Architecture

TODO: software architecture

3 Use cases

3.1 Configure the simulation

Let's take a look at the provided sample experiment SimpleExperiment. It contains a main function that specifies all parameters, starts the simulation and presents the results.

Dataset

Initially the dataset is specified. Note that the Dataset object is not required for the algorithm, but it simplifies the configuration. Technically all we need is a way to get a list of possible plans for each agent. The Dataset interface provides this functionality via Dataset.getPlans(int agentIdx). There are two classes of datasets implemented:

- FileVectorDataset is a dataset that is read from disk. Only the dataset folder has to be specified. The example datasets are located in the directory '¡project dir¿/input-data'. Section **??** describes the input format for this kind of dataset.

- GaussianDataset is a generated dataset where every plan is a vector drawn from a gaussian distribution. The parameters specify the number and dimensionality of the plans as well as mean and standard deviation of the distribution.

The number of agents can be set arbitrarily. However, when using a FileVectorDataset, the number of agents has to be below FileVectorDataset.getNumAgents().

Cost functions

The global cost function describes what we want to minimize globally. The local cost function describes what each agent wants to minimize locally. lambda is the tradeoff between global and local minimization. lambda = 0 means only global cost is minimized, lambda = 1 means only local cost is minimized. For global cost functions we can choose any implementation of the interface CostFunction in general. However, for gradient descent based algorithms we need an instance of DifferentiableCostFunction. A list of possible cost functions is as follows.

- DotCostFunction minimizes the dot product of a given vector with the global response. See Section .2.1 how to read a vector from a file. Be aware that the dimensionality of the vector has to match the dimensionality of the dataset. An example use case for this cost function is minimizing monetary cost. The agent plans contain the amount of resources they consume, and the vector passed to DotCostFunction describes the price for each resource. As this is a linear cost function, I-EPOS always finds the optimal value in the first iteration.

- SqrDistCostFunction minimizes the (squared) distance of the global response to a given vector. See Section .2.1 how to read a vector from a file. Be aware that the dimensionality of the vector has to match the dimensionality of the dataset. This cost function tries to make the global response as similar to the provided target vector as possible.

- VarCostFunction minimizes the variance of the global response. Therefore it can be used to stabilize resource consumption over time, or for load balancing applications.

- StdDevCostFunction minimizes the standard deviation of the global response. For I-EPOS there is no difference between minimizing standard deviation and minimizing variance, as the functions share the same minima.

- MaxCostFunction (non-differentiable) minimizes the maximum value of the global response. This function is useful for peak reduction.

double lambda = 0.1; DifferentiableCostFunction globalCostFunc = new VarCostFunction(); PlanCostFunction localCostFunc = new DiscomfortCostFunction();

// network int numChildren = 2; // +- of using this topology

// algorithm int numIterations = 20; PlanSelector¡IeposAgent¡Vector¿, Vector¿ planSelector = new IeposPlanSelector();

// loggers LoggingProvider¡IeposAgent¡Vector¿¿ loggingProvider = new LoggingProvider¡¿(); loggingProvider.add(new GlobalCostLogger()); loggingProvider.add(new LocalCostLogger()); loggingProvider.add(new TerminationLogger()); loggingProvider.add(new JFreeChartLogger()); loggingProvider.add(new GraphLogger¡¿(GraphLogger.Type.Change, null));

// start experiment new SimpleExperiment().run(numChildren, numIterations, numAgents, agentIdx -¿ List¡Plan¡Vector¿¿ possiblePlans = dataset.getPlans(agentIdx); AgentLoggingProvider agentLP = loggingProvider.getAgentLoggingProvider(agentIdx, 0);

IeposAgent newAgent = new IeposAgent(numIterations, possiblePlans, globalCostFunc, localCostFunc, agentLP, random.nextLong()); newAgent.setLambda(lambda); newAgent.setPlanSelector(planSelector); return newAgent;);

loggingProvider.print();

FileVectorDataset -¿ getNumAgents()

3.2 Write a new optimization function

3.3 Add a new dataset

.1 Glossary

.2 Utility functions

.2.1 Reading a vector

A vector can be read from a file via VectorIO.readVector(File vectorFile). The file is assumed to be text file, containing a comma-separated list of double values that make up the vector.