Linking AnyLogic and HeuristicLab

Based on: https://dev.heuristiclab.com/trac.fcgi/wiki/Documentation/Howto/OptimizeAnyLogicModels

Introduction

We assume the reader is familiar with xj technologies AnyLogic and has basic understanding of the Java programming language.

This howto is written as a tutorial by taking one of the sample models that ships with AnyLogic and add connectors to optimise its parameters with HeuristicLab algorithms. If you go through this tutorial you will hopefully get an idea on how to solve your specific problem and implement the connector in your model.

AnyLogic Supply Chain model

The model which we are going to optimise with HeuristicLab comes preinstalled with AnyLogic and is called *Supply Chain*. You can open it either on the *Welcome page* after starting AnyLogic or by choosing "Sample Models" from the Help menu. After you opened the model you can try to run the Simulation. There is also an *Optimization Experiment* in this model which uses the proprietary optimisation package OptQuest.

Optimisation Problem

The optimisation goal is to minimise the sum of all costs in the supply chain while satisfying a certain service level for the end user. Each echelon in the chain uses an (s,S) ordering policy. In total we have 6 decision variables, one target variable and one constraint on the output.

Preparing the Simulation Model in AnyLogic

First save the model in a new location. Click "Save As..." in the File menu. For the sake of simplicity just put "Supply Chain" in the box that asks for a model name and leave everything else. That way it will save it in a folder called "Models" in your user's directory.

Now download the HL3 External Evaluation Java library to a place that you remember (e.g. your Downloads directory; you should not download it into the model folder already). Then switch to AnyLogic again and click the model icon in the project view (should be called "Supply Chain" if you followed the model name suggestion above). In the properties window of this model in the section "Jar files and class folders required to build the model:" click the add button. Choose the file you just downloaded, make sure "Import to model folder is ticked" and click "Finish" to add the library. Save the model.

Now we need to add a new experiment which we can run to serve as our evaluation function. The best type of experiment for this task is a "Parameters Variation" experiment, so choose this and choose a name for it, e.g. "HeuristicLabOptimizationExperiment". Leave everything else as it is and click "Finish". Now we have created a new experiment with which we can optimise the parameters.

In the "Properties" view of this experiment go to the "General" tab and choose "Freeform" instead of "Varied in range" and put a large number in the box asking for the "Number of runs", e.g.: 1,000,000,000. Then type in each field in the column "Expression" the same word as you see on the general property page of the *Simulation* experiment. So instead of the first number 20 you type *sLoRetailer*, then you replace the 80 with *SHiRetailer*, then the next 20 you replace with *sLoWholesaler*, then *SHiWholesaler*, then *sLoFactory*, and finally *SHiFactory*. Click save.

Now in the upper right hand corner of this property page is a button called "Create default UI", click that now and you have some user interface elements added to this experiment.

Next we need to go to the "Advanced" section of the properties and configure the experiment loop: perform the parameter retrieval, run the simulation, send the quality back, reset the model and continue with another attempt of retrieving parameters.

The following code samples need to be entered into the respective boxes.

Imports

```
import java.io.*;
import java.text.*;
import com.heuristiclab.problems.externalevaluation.*;
import com.heuristiclab.problems.externalevaluation.ExternalEvaluationMessages.*;
```

Additional class code

```
int replications;
double quality;
com.heuristiclab.problems.externalevaluation.PollService commDriver;
SolutionMessage currentSolution = null;

private void getMessage() {
  currentSolution = commDriver.getSolution();
  if (currentSolution.getIntegerArrayVarsCount() > 0) {
    SolutionMessage.IntegerArrayVariable vector = currentSolution.getIntegerArrayVars(0);
    sLoRetailer = vector.getData(0);
    SHiRetailer = sLoRetailer + vector.getData(1);
    sLoWholesaler = vector.getData(2);
    SHiWholesaler = sLoWholesaler + vector.getData(3);
    sLoFactory = vector.getData(4);
    SHiFactory = sLoFactory + vector.getData(5);
  }
}
```

We have defined a method *getMessage()* that calls our library to get the next message from HeuristicLab and extract the parameters.

Initial experiment setup

```
commDriver = new com.heuristiclab.problems.externalevaluation.PollService(new ServerSocketListenerFactory(2112),1);
commDriver.start();
```

The number **2112** is important as this will be the TCP port that our simulation model will be listening on. This must be reachable by HeuristicLab through a network connection (yes this means you can run HeuristicLab on a different computer than your simulation model). You can of course choose another port (basically any number from 1 to 65535, but some numbers like 80 might be used by another application already).

Before each experiment run

```
quality = 0;
replications = 0;

getMessage();
```

The call to *getMessage()* will block, that means the simulation will wait here, until we have received a new set of parameters.

Before simulation run

leave this empty

After simulation run

```
double penalty = 10000 * (1 + root.histWaitingTime.max());
if (root.histWaitingTime.max() < 0.01) quality += root.meanDailyCost();
else quality += root.meanDailyCost() + penalty;
replications++;
```

This calls the cost function in the Main object called *meanDailyCost()* in this case. This is the target that we want to optimise (note that we add an increasing penalty term if the customer waiting time is above a certain level).

After iteration

```
try {
  commDriver.sendQuality(currentSolution, quality / replications);
} catch (IOException e) { }
quality = 0;
replications = 0;

getMessage();
```

This sends the mean quality of the (possible) replications back to HeuristicLab, resets the variables and receives a new message with which to perform the next simulation runs. We are done here. Save the model. You can now start the HeuristicLabOptimizationExperiement, although of course it won't do anything but block until it receives parameters.

Preparing the Optimisation Problem in HeuristicLab 3.3

In HeuristicLab we need to define the problem and choose an algorithm with which we can optimise it. The problem is also attached to this page, if you're impatient go down to attachments and download the .hl file.

Defining the problem

Click on the "New item" button in the toolbar (just below the File menu) and look for External Evaluation Problem. Name the problem "AnyLogic Supply Chain Problem". You can also double click the information icon right of the name box to edit the description.

In the Parameters box you'll see a number of different parameters. We need to configure the client that hosts the simulation and the solution encoding that we're using.

Client

Click on the parameter called "Clients" to see the list of clients. For each simulation model that participates in your experiment you have to add a client to this list. So this means you can optimise your parameters faster if you run the simulation on two separate computers (note however, that you need to use the "Parallel Engine" in your algorithm). Per default only one client is configured. Click on EvaluationServiceClient and you'll see another box with parameters. Among those is one called "Channel" which is currently null, meaning that it has no value defined. Click on Channel and then click on the pencil icon to give it a value. Choose the EvaluationTCPChannel by double clicking it in the emerging dialog. Now type the IP Address of the computer where the simulation will run ("127.0.0.1" in case it will be run on the same computer) and enter the port number (remember we chose "2112" above). Then we're done with the client.

Next we'll configure the parameter Encoding. Click on this parameter to get its list of parameters.

Encoding

Define the problem's encoding in that you click on the pencil icon and select the IntegerVectorEncoding from the corresponding namespace. Click OK in the dialog and change the encoding's parameters:

- IntegerVector.Bounds: Set first column to "0" and second column to "100" - this will be lower and upper bound for every dimension (you can also specify one line for each dimension)
- IntegerVector.Length: Set to "6"

So we're done with this parameter. The next one is Maximization which we'll not need to touch as it is already false (we're minimising costs).

You can now save your problem to a file.

Choosing an Algorithm

Next we create a new Genetic Algorithm by clicking the New Item button in the toolbar again. This time create a new genetic algorithm. In the problem tab click to open the problem and select the file that you just created.

Go to the parameters tab and select a mutation operator, e.g. "UniformOnePositionManipulator".

Optimising the model

Now it's time, we're ready to start the optimisation. First switch back to AnyLogic and start the HeuristicLabOptimizationExperiment. Click on the button "Run Experiment" to set it into listening mode.

Go to HeuristicLab and click the green arrow at the bottom to start the algorithm.

Now in the back the algorithm attempts a TCP connection to the given IP Address and port and exchanges parameters with the simulation model. It will receive quality values and use these to determine the fitness of its solutions.