Appendix A

User Guide

A.1 Instructions

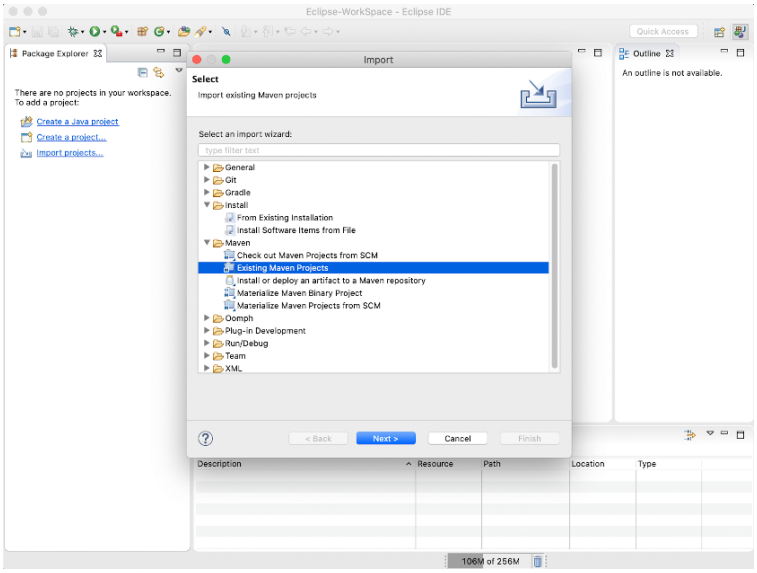
In order to to use the tool, the user needs to install:

- Eclipse (version: Eclipse for Java and DSL developers is recommended)
- Java 11
- Apache Ant
- Maven

Linux or macOS is recommended for the operating system, for Windows users, special configurations for Ant might be required, detail, please visit Apache Ant's official guide.

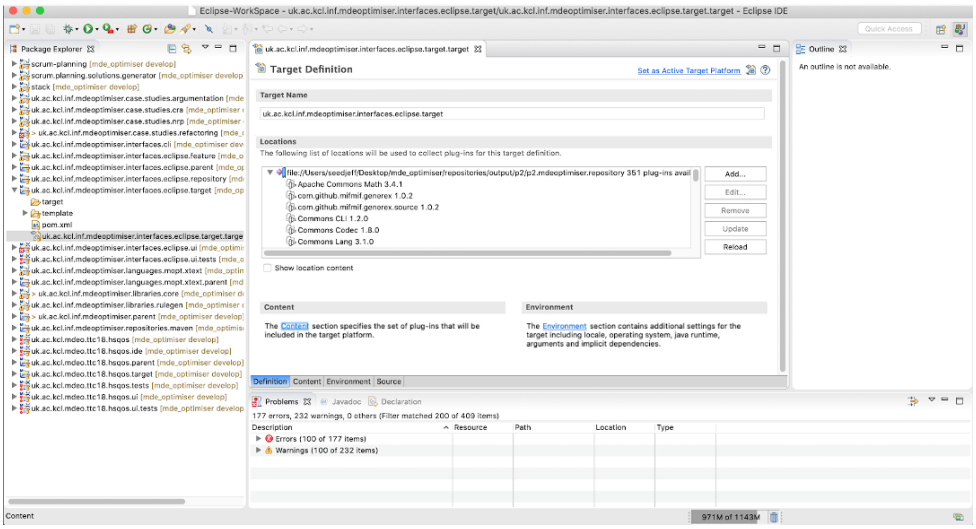
Then the user needs to go to the root folder of the tool and navigates to the "ant" folder under the "build" folder in the terminal, by "cd build/ant". Then enter "ant" in the terminal to compile the tool.

In order To launch the tool, the user first need to import the source code as a maven project into eclipse.



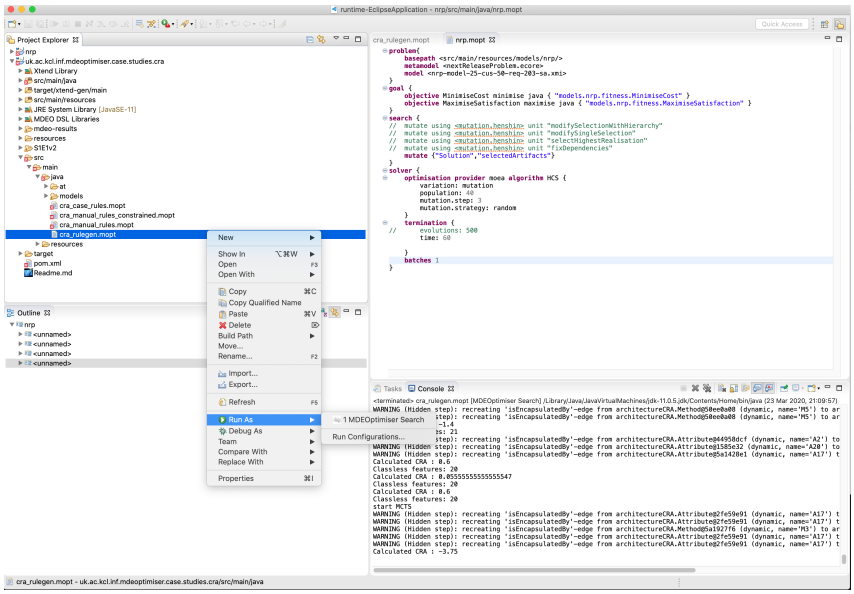
After, import complete, which is indicated by a progress bar on the bottom of the eclipse window as shown below.

Then the user need to open ”uk.ac.inf.mdeoptimiser.interface.eclipse.target”, and double click verb—uk.ac.inf.mdeoptimiser.interface.eclipse.target.target—, and set it as active target platform.



After this is complete, which is indicated by a progress bar similar to before. The user will now be able to launch the tool by right click ”uk.ac.inf.mdeoptimiser.interface.eclipse.ui” and run as an eclipse application. Ignore the warnings regarding ”uk.ac.inf.mdeoptimiser.interface.eclipse.ui.tests”.

Another eclipse window should appear, now the user can import case studies/problems as a maven project, and run the mopt file inside the case studies which defines the specifics of the problem as "MDEOptimiser Search" to solve them.



Algorithm for solving the problem can be specfied at where "HCS" is, after "algorithm" using the acronym of the algorithm:

- MCTS - Monte Carlo tree search
- HCS - Hill Climbing
- NSGAII - NSGAII

Appendix B

Source Code

”MoeaOptimisationSolution”, ”MoeaOptimisationProblem”, ”SolutionGenerator” and ”MoeaOptimisationAlgorithmProvider” are classes in the orignal MDEO where some alterations are made for the project, ”PSJ” is stated in the comments where the new codes are written.

Listings


```
1  package uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.problem
2
3  import org.moeaframework.core.Solution
4  import java.util.LinkedHashMap
5  import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.executor.
       SolutionGenerator
6
7  class MoeaOptimisationSolution extends Solution {
8
9    SolutionGenerator solutionGenerator
10
11   new(MoeaOptimisationSolution moeaOptimisationSolution){
12     super(1, moeaOptimisationSolution.numberOfObjectives,
     moeaOptimisationSolution.numberOfConstraints)
13     this.model = moeaOptimisationSolution.getModel
14     this.solutionGenerator = moeaOptimisationSolution.getSolutionGenerator
15   }
16
```

```
17  new(int numberOfObjectives, int numberOfConstraints) {
18    super(1, numberOfObjectives, numberOfConstraints)
19  }
20
21  new(SolutionGenerator solutionGenerator){
22    this(solutionGenerator.optimisationModel.goal.objectives.size(),
23      solutionGenerator.optimisationModel.goal.constraints.size()
24    )
25    this.solutionGenerator = solutionGenerator;
26    setModel(solutionGenerator.mutate(solutionGenerator.initialSolutions.head))
27  }
28
29  //PSJ: Generate the vanilla model as a soluion
30  new(SolutionGenerator solutionGenerator, int n){
31    this(solutionGenerator.optimisationModel.goal.objectives.size(),
32      solutionGenerator.optimisationModel.goal.constraints.size()
33    )
34    this.solutionGenerator = solutionGenerator;
35    setModel(solutionGenerator.initialSolutions.head)
36  }
37
38  override MoeaOptimisationSolution copy(){
39    new MoeaOptimisationSolution(this);
40  }
41
42  def uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.interpreter.
      guidance.Solution getModel(){
43    (getVariable(0) as MoeaOptimisationVariable).model
44  }
45
46  def void setModel(uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.
      interpreter.guidance.Solution model) {
47    setVariable(0, new MoeaOptimisationVariable(model, solutionGenerator))
48  }
49
50  def SolutionGenerator getSolutionGenerator(){
51    return this.solutionGenerator
52  }
53
54  def void setSolutionGenerator(SolutionGenerator solutionGenerator) {
55    this.solutionGenerator = solutionGenerator;
56  }
```

```
57
58  def LinkedHashMap<String, Double> getFormattedObjectives(){
59    val objectives = new LinkedHashMap<String, Double>()
60
61    solutionGenerator.optimisationModel.goal.objectives.forEach[objective ,
      index |
62      objectives.put(objective.objectiveName, this.objectives.get(index))
63    ]
64
65    return objectives
66  }
67
68  def LinkedHashMap<String, Double> getFormattedConstraints(){
69    val constraints = new LinkedHashMap<String, Double>()
70
71    solutionGenerator.optimisationModel.goal.constraints.forEach[constraint ,
      index |
72      constraints.put(constraint.constraintName, this.constraints.get(index))
73    ]
74
75    return constraints
76  }
77
78  override toString(){
79
80    val sb = new StringBuilder();
81
82    val objectives = getObjectives()
83    sb.append("[")
84
85    objectives.forEach[value,index|
86      sb.append(value)
87
88      if(index < objectives.size - 1){
89        sb.append(",")
90      }
91    ]
92    sb.append("]")
93
94    val constraints = getConstraints();
95
96    if(constraints.size > 0){
```

```
97        sb.append("[")
98
99        constraints.forEach[value, index|
100         sb.append(value)
101
102         if(index < constraints.size - 1){
103           sb.append(",")
104         }
105       ]
106
107       sb.append("]")
108     }
109
110
111     return sb.toString
112   }
113 }
```

Listing B.1: MoeaOptimisationSolution

```
1  package uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.problem
2
3  import java.util.List
4  import org.moeaframework.core.Solution
5  import org.moeaframework.problem.AbstractProblem
6  import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.executor.
       SolutionGenerator
7  import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.IGuidanceFunction
8  import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.interpreter.
       guidance.GuidanceFunctionsFactory
9  import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.interpreter.
       guidance.GuidanceFunctionAdapter
10
11 class MoeaOptimisationProblem extends AbstractProblem {
12
13   SolutionGenerator solutionGenerator
14
15   List<IGuidanceFunction> fitnessFunctions;
16   List<IGuidanceFunction> constraintFunctions;
17
18   new(int numberOfVariables, int numberOfObjectives, int numberOfConstraints) {
19     super(numberOfVariables, numberOfObjectives, numberOfConstraints)
20   }
```

```
21
22  new(SolutionGenerator solutionGenerator) {
23    // Number of variables is for now always one.
24    this(1, solutionGenerator.optimisationModel.goal.objectives.size(),
25      solutionGenerator.optimisationModel.goal.constraints.size())
26    this.solutionGenerator = solutionGenerator
27  }
28
29  def SolutionGenerator getSolutionGenerator() {
30    this.solutionGenerator
31  }
32
33  def getConstraintFunctions() {
34    if (this.constraintFunctions === null) {
35      setConstraintFunctions()
36    }
37
38    this.constraintFunctions
39  }
40
41  def setConstraintFunctions() {
42    if (constraintFunctions === null) {
43      this.constraintFunctions = solutionGenerator.optimisationModel.goal.
    constraints.map [ constraint |
44        new GuidanceFunctionsFactory().loadFunction(new GuidanceFunctionAdapter(
    constraint))
45      ]
46    }
47  }
48
49  def getFitnessFunctions() {
50    if (this.fitnessFunctions === null) {
51      setFitnessFunctions()
52    }
53
54    this.fitnessFunctions
55  }
56
57  def void setFitnessFunctions() {
58    if (fitnessFunctions === null) {
59      this.fitnessFunctions = solutionGenerator.optimisationModel.goal.
    objectives.map [ objective |
```

```
60          new GuidanceFunctionsFactory().loadFunction(new GuidanceFunctionAdapter(
    objective))
61        ]
62      }
63    }
64
65    override evaluate(Solution solution) {
66
67      // TODO if some constraints are the same as the objectives, they shoyuld be
    cached for the same model
68      val moeaSolution = solution as MoeaOptimisationSolution;
69
70      // Set objectives
71      getFitnessFunctions.forEach [ fitnessFunction, objectiveId |
72        moeaSolution.setObjective(objectiveId, fitnessFunction.computeFitness(
    moeaSolution.model))
73      ]
74
75      // Set Constraints
76      getConstraintFunctions.forEach [ constraintFunction, objectiveId |
77        moeaSolution.setConstraint(objectiveId, constraintFunction.computeFitness(
    moeaSolution.model))
78      ]
79
80    }
81
82    override newSolution() {
83      new MoeaOptimisationSolution(solutionGenerator)
84    }
85
86    //PSJ: create a new solution representing the starting model
87    def initialModelasSolution() {
88      new MoeaOptimisationSolution(solutionGenerator, 1);
89    }
90  }
```

Listing B.2: MoeaOptimisationProblem

```
1  package uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.executor
2
3  import java.util.Iterator
4  import java.util.List
5  import org.eclipse.emf.ecore.EPackage
```

```
import org.eclipse.emf.henshin.model.Unit
import uk.ac.kcl.inf.mdeoptimiser.languages.mopt.Optimisation
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.IModelProvider
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.interpreter.
    evolvers.parameters.EvolverParametersFactory
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.interpreter.
    guidance.Solution
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.interpreter.
    henshin.HenshinExecutor
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.operators.
    adaptation.MutationStepSizeStrategy
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.operators.
    crossover.CrossoverStrategy
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.operators.
    crossover.CrossoverStrategyFactory
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.operators.mutation
    .MutationStrategy
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.operators.mutation
    .MutationStrategyFactory

class SolutionGenerator {

  EPackage theMetamodel

  Optimisation optimisationModel

  IModelProvider initialModelProvider

  MutationStrategy mutationStrategy;
  CrossoverStrategy crossoverStrategy;

  HenshinExecutor henshinExecutor

  MutationStepSizeStrategy mutationStepSizeStrategy

  new(Optimisation model, List<Unit> breedingOperators, List<Unit>
    mutationOperators, IModelProvider modelProvider,
    EPackage metamodel) {
    this.optimisationModel = model
    this.initialModelProvider = modelProvider
    this.theMetamodel = metamodel;
    this.henshinExecutor = new HenshinExecutor(
```

```
39      new EvolverParametersFactory(model.search.evolvers),
40      mutationOperators,
41      breedingOperators,
42      model.solver
43    )
44  }
45
46  /**
47   * Set the configured mutation step size strategy to use when generating
48   * solutions from this solution generation using mutation.
49   *
50   * @param mutationStepSizeStrategy an instance of the configured strategy
51   */
52  def setMutationStepSizeStrategy(MutationStepSizeStrategy
      mutationStepSizeStrategy) {
53    this.mutationStepSizeStrategy = mutationStepSizeStrategy;
54  }
55
56  //PSJ: return component needed for mutation
57  def getHenshinExcutor() {
58    return this.henshinExecutor
59  }
60
61  // TODO IOC
62  def getMutationStrategy() {
63
64    if (this.mutationStrategy === null) {
65      this.mutationStrategy = new MutationStrategyFactory(henshinExecutor, this.
      mutationStepSizeStrategy,
66          optimisationModel.solver.algorithm).getStrategy()
67    }
68
69    return this.mutationStrategy
70
71  }
72
73  def getCrossoverStrategy() {
74
75    if (this.crossoverStrategy === null) {
76
77      //TODO- When adding the rule chain, implement the correct strategies here
78      this.crossoverStrategy = new CrossoverStrategyFactory(henshinExecutor).
```

```
    getStrategy(null);
79  }
80
81  return this.crossoverStrategy
82 }
83
84 /**
85  * This will produce a lazy iteration of possible initial solutions
86  * @return list of initial list of models
87  */
88 def Iterator<Solution> getInitialSolutions() {
89   initialModelProvider.initialModels(theMetamodel)
90 }
91
92 /**
93  * Returns the optimisation model to use inside the moea problem/solution
94  */
95 def Optimisation getOptimisationModel() {
96   return optimisationModel
97 }
98
99 /**
100  * Produces two offspring from the two parents provided in the parameter.
101  * @param parents a list of two parent models
102  * @returns a list of results offspring
103  */
104 def List<Solution> breed(List<Solution> parents) {
105   return this.getCrossoverStrategy.breed(parents);
106 }
107
108 /** Produce a new solution from the given one using one of the evolvers
    defined in the optimisation model.
109  * This will
110  *
111  * @param object solution candidate to be evolved
112  */
113 def Solution mutate(Solution model) {
114   return this.getMutationStrategy.mutate(model)
115 }
116
```

```
}
```

Listing B.3: SolutionGenerator

```
package uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.algorithms

import java.util.Properties
import org.moeaframework.algorithm.NSGAII
import org.moeaframework.core.Algorithm
import org.moeaframework.core.NondominatedSortingPopulation
import org.moeaframework.core.Problem
import org.moeaframework.core.Variation
import org.moeaframework.core.operator.GAVariation
import org.moeaframework.core.operator.RandomInitialization
import org.moeaframework.core.operator.TournamentSelection
import org.moeaframework.core.spi.AlgorithmProvider
import uk.ac.kcl.inf.mdeoptimiser.languages.mopt.Parameter
import uk.ac.kcl.inf.mdeoptimiser.languages.validation.algorithm.
    UnexpectedAlgorithmException
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.executor.
    SolutionGenerator
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.operators.
    MoeaOptimisationCrossoverVariation
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.operators.
    MoeaOptimisationMutationVariation
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.operators.
    MoeaProbabilisticVariation
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.PSJ.MCTS
import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.PSJ.HCS

class MoeaOptimisationAlgorithmProvider extends AlgorithmProvider {

  public Algorithm algorithm;

  override getAlgorithm(String algorithm, Properties properties, Problem problem
    ) {
    switch algorithm {
      case "NSGAII":
        this.algorithm = createNSGAII(problem, properties)
        // this.algorithm = createMCTS(problem, properties)
      case "MCTS":
        this.algorithm = createMCTS(problem, properties)
      case "HCS":
```

```
34          this.algorithm = createHCS(problem, properties )
35        default:
36          throw new UnexpectedAlgorithmException(algorithm)
37      }
38
39      return this.algorithm;
40    }
41
42    def Variation getVariation(Properties properties){
43
44      var algorithmVariation = new AlgorithmVariation(properties.get("
        variationType") as Parameter)
45
46      //Check if we have weighted genetic variation
47      //TODO: This needs to be refactored and fixed
48      if(algorithmVariation.isProbabilisticVariation){
49        val crossoverVariation = new MoeaOptimisationCrossoverVariation(properties
        .get("solutionGenerator") as SolutionGenerator)
50        val mutationVariation = new MoeaOptimisationMutationVariation(properties.
        get("solutionGenerator") as SolutionGenerator)
51
52        return new MoeaProbabilisticVariation(crossoverVariation,
        mutationVariation,
53          algorithmVariation.crossoverRate,
54          algorithmVariation.mutationRate)
55      }
56
57      //Check variation type is crossover with mutation
58      if(algorithmVariation.isGeneticVariation){
59        val crossoverVariation = new MoeaOptimisationCrossoverVariation(properties
        .get("solutionGenerator") as SolutionGenerator)
60        val mutationVariation = new MoeaOptimisationMutationVariation(properties.
        get("solutionGenerator") as SolutionGenerator)
61
62        return new GAVariation(crossoverVariation, mutationVariation)
63      }
64
65      //Check variation type is mutation
66      if(algorithmVariation.isMutationVariation){
67        return new MoeaOptimisationMutationVariation(properties.get("
        solutionGenerator") as SolutionGenerator)
68      }
```

```
69
70    //Must be crossover only then
71    return new MoeaOptimisationCrossoverVariation(properties.get("
      solutionGenerator") as SolutionGenerator)
72
73  }
74
75  def Algorithm createNSGAII(Problem problem, Properties properties) {
76    //Create an initial random population of population size
77    var initialization = new RandomInitialization(problem, properties.get("
      populationSize") as Integer)
78
79    var selection = new TournamentSelection(2);
80
81    new NSGAII(
82        problem,
83        new NondominatedSortingPopulation(),
84        null, // no archive
85        selection,
86        getVariation(properties),
87        initialization
88      );
89  }
90
91  def Algorithm createMCTS(Problem problem, Properties properties){
92    //Create an initial random population of population size
93    var initialization = new RandomInitialization(problem, properties.get("
      populationSize") as Integer)
94
95    var selection = new TournamentSelection(2);
96
97    new MCTS(
98        problem,
99        new NondominatedSortingPopulation(),
100       null, // no archive
101       selection,
102       getVariation(properties),
103       initialization
104     );
105
106 }
107
```

```
108
109   def Algorithm createHCS(Problem problem, Properties properties){
110     //Create an initial random population of population size
111     var initialization = new RandomInitialization(problem, properties.get("
      populationSize") as Integer)
112
113     var selection = new TournamentSelection(2);
114
115     new HCS(
116         problem,
117         new NondominatedSortingPopulation(),
118         null, // no archive
119         selection,
120         getVariation(properties),
121         initialization
122       );
123
124   }
125
126 //
127 //  def Algorithm createSPEA2(Problem problem, Properties properties) {
128 //
129 //    var initialization = new RandomInitialization(problem, properties.get("
    populationSize") as Integer)
130 //
131 //    new SPEA2(
132 //        problem,
133 //        initialization,
134 //        getVariation(properties),
135 //        properties.get("populationSize") as Integer,
136 //        1
137 //      );
138 //  }
139 //
140 //  def Algorithm createeMOEA(Problem problem, Properties properties) {
141 //
142 //    //Create an initial random population of population size
143 //    var initialization = new RandomInitialization(problem, properties.get("
    populationSize") as Integer)
144 //
145 //    var selection = new TournamentSelection(2);
146 //
```

```
147 //    new EpsilonMOEA(
148 //        problem,
149 //        new NondominatedSortingPopulation(),
150 //        //TODO This must be a user configurable parameter
151 //        new EpsilonBoxDominanceArchive(0.01),
152 //        selection,
153 //        getVariation(properties),
154 //        initialization
155 //      );
156 //  }
157
158 }
```

Listing B.4: MoeaOptimisationAlgorithmProvider

```
1  package uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.PSJ;
2  import org.moeaframework.core.Solution;
3  import org.sidiff.common.emf.access.path.axis.Parent;
4
5  public class Node {
6
7      protected Solution solution;
8      protected Node left = null;
9      protected Node right = null;
10     protected Node parent = null;
11     protected int visited = 0;
12     protected double gameValue = 0; // heuristic
13     protected int childrenVisited = 0;
14
15     Node(Solution solution) {
16         this.solution = solution;
17     }
18
19     public Node() {
20     }
21
22     public Node(Solution solution, Node left, Node right, Node parent) {
23         this.solution = solution;
24         this.left = left;
25         this.right = right;
26         this.parent = parent;
27     }
28
```

```
29      public Solution getSolution() {
30          return this.solution;
31      }
32
33      public void setSolution(Solution solution) {
34          this.solution = solution;
35      }
36
37      public void visited(){
38          visited++;
39      }
40
41      public int getChildrenVisited(){
42          return childrenVisited;
43      }
44      public void setChildrenVisited(int c){
45          childrenVisited = c;
46      }
47      public double setGameValue(double g){
48          gameValue = g;
49          return gameValue;
50      }
51
52      public double getGameValue(){
53          return gameValue;
54      }
55
56      public int getVisited(){
57          return visited;
58      }
59
60      public Node getLeft() {
61          return this.left;
62      }
63
64      public void setLeft(Node left) {
65          this.left = left;
66      }
67
68      public Node getRight() {
69          return this.right;
70      }
```

```
71
72      public void setRight(Node right) {
73          this.right = right;
74      }
75
76      public Node getParent() {
77          return this.parent;
78      }
79
80      public void setParent(Node parent) {
81          this.parent = parent;
82      }
83  }
```

Listing B.5: Node

```
1
2   package uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.PSJ;
3
4   import java.util.ArrayList;
5   import java.util.Arrays;
6   import java.util.LinkedList;
7   import java.util.List;
8   import java.util.stream.*;
9
10
11  import org.moeaframework.algorithm.AbstractEvolutionaryAlgorithm;
12  import org.moeaframework.core.EpsilonBoxDominanceArchive;
13  import org.moeaframework.core.EpsilonBoxEvolutionaryAlgorithm;
14  import org.moeaframework.core.Initialization;
15  import org.moeaframework.core.NondominatedSortingPopulation;
16  import org.moeaframework.core.Population;
17  import org.moeaframework.core.Problem;
18  import org.moeaframework.core.Selection;
19  import org.moeaframework.core.Solution;
20  import org.moeaframework.core.Variation;
21
22  import org.moeaframework.core.operator.TournamentSelection;
23
24  import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.problem.
        MoeaOptimisationProblem;
25  import org.eclipse.emf.ecore.EObject;
26  import java.lang.Math;
```

```
27
28
29
30 public class MCTS extends AbstractEvolutionaryAlgorithm implements
31     EpsilonBoxEvolutionaryAlgorithm {
32
33
34   private final Selection selection;
35
36
37   /**
38    * The variation operator.
39    */
40   private final Variation variation;
41
42   // private int counter;
43
44
45   private final TournamentSelection s;
46
47   private Node root;
48   private Node best;
49   private Node choice;
50
51   MoeaOptimisationProblem moeaProblem;
52
53   /**
54    * Parameters needed for MDEO to accept MCTS without adapter, however not
       necessary
55    *
56    * @param problem the problem being solved
57    * @param population the population used to store solutions
58    * @param archive the archive used to store the result; can be {@code null}
59    * @param selection the selection operator
60    * @param variation the variation operator
61    * @param initialization the initialization method
62    */
63   public MCTS(Problem problem, NondominatedSortingPopulation population,
64       EpsilonBoxDominanceArchive archive, Selection selection,
65       Variation variation, Initialization initialization) {
66     super(problem, population, archive, initialization);
67     this.moeaProblem = (MoeaOptimisationProblem) problem;
```

```
68      this.s = (TournamentSelection) selection;
69      this.selection = selection;
70      this.variation = variation;
71      root = new Node();
72
73    }
74
75    @Override
76    public void iterate() {
77
78      NondominatedSortingPopulation population = getPopulation();
79
80      if (root.solution == null) {
81        initialization();
82      }
83
84
85      //select the Node/solution to expand
86      selection();
87
88      expansionMain();
89
90      backpropagation();
91
92      //determine if the selected solution is the best
93      if(compareDomin(choice.getSolution(), best.getSolution()) == -1){
94        best = choice;
95      }
96
97      population.clear();
98      population.add(best.getSolution());
99
100
101     // //nrp output
102     // population.add(choice.getSolution());
103     // population.truncate(population.size());
104
105   }
106
107   public void initialization(){
108       root.setSolution( moeaProblem.initialModelasSolution());
109       best = root;
```

```
110   }
111
112   //Expand selected Node
113   public void expansionMain(){
114       Solution[] next = new Solution[variation.getArity()];
115       for (int i = 0; i < next.length; i++) {
116         next[i] = choice.getSolution();
117       }
118       Node left = new Node(expand(next)[0], null, null, choice);
119       Node right = new Node(expand(next)[0], null, null, choice);
120       left.setGameValue(heuristicEstimate(left.getSolution()));
121       right.setGameValue(heuristicEstimate(right.getSolution()));
122
123       choice.setLeft(left);
124       choice.setRight(right);
125
126   }
127
128   //-1 solution 1 dominates 2
129   public int compareDomin(Solution solution1, Solution solution2){
130     return s.getComparator().compare(solution1, solution2);
131   }
132
133
134   public boolean compareHeuristic(Solution solution1, Solution solution2){
135     return heuristicEstimate(solution1) > heuristicEstimate(solution2);
136   }
137
138   // estimate heuristic use fitnesses and unsatisfied constraints
139   public double heuristicEstimate(Solution solution){
140     evaluate(solution);
141     return ( -1.0*Arrays.stream(solution.getObjectives()).sum() )
142           - 0.5 *Arrays.stream(solution.getConstraints()).sum()
143           ;
144   }
145
146   //Selection Strategy2
147   public double selectionValue(Node node){
148
149     //Selection Strategy 2
150     if(node.getVisited() == 0){
151       return Double.POSITIVE_INFINITY;
```

```
152     }
153     else if(node.getChildrenVisited() == 0){
154       return Double.POSITIVE_INFINITY;
155     }
156     else{
157     return node.getGameValue() + 0.2*Math.sqrt( (Math.log((double) node.
        getVisited())) / (double) node.getChildrenVisited() )
158     ;
159
160     }
161   }
162
163   public double selectionValue1(Node node){
164
165   //Selection Strategy 1
166   if(node.getVisited() == 0){
167     return Double.POSITIVE_INFINITY;
168   }
169   else{
170   Node parent = node.getParent();
171   return node.getGameValue()
172       + 0.1*Math.sqrt( (Math.log((double) parent.getVisited())  ) / (double)
      node.getVisited());
173
174   }
175
176   }
177
178
179   // Create new Solutions based on parent
180   public Solution[] expand(Solution[] parent) {
181
182     //expansion Strategy 1
183     // Solution[] solutions = new Solution[variation.getArity()];
184
185     //  solutions = variation.evolve(parent);
186     //  evaluateAll(solutions);
187     // return solutions;
188
189
190     //expansion Strategy 2
191     Solution[] equal = null;
```

```
192     Solution[] solutions = new Solution[variation.getArity()];
193     for (int i = 0; i < 50; i++) {
194       solutions = variation.evolve(parent);
195       evaluateAll(solutions);
196       if(compareDomin(solutions[0], parent[0]) == -1){
197         return solutions;
198       }
199       else if(compareDomin(solutions[0], parent[0]) == 0){
200         equal = solutions;
201       }
202     }
203 
204     if(equal != null)
205       {return equal;}
206     else{
207       return variation.evolve(parent);
208       }
209   }
210 
211   // Update childrenVisted count/heuristics
212   public void backpropagation(){
213     Node back = choice;
214 
215     while(back != root && back.getParent() != root){
216       back = back.getParent();
217 
218       back.setChildrenVisited(back.getLeft().getChildrenVisited()+back.getLeft()
      .getVisited() + back.getRight().getChildrenVisited()+back.getRight().
      getVisited());
219 
220       //new heuristics update
221       double difference = ((back.getLeft().getGameValue()+back.getRight().
      getGameValue())/ 2.0) - back.getGameValue();
222       back.setGameValue(back.getGameValue() + difference);
223 
224       //old heuristic update
225       // back.setGameValue((back.getLeft().getGameValue()+back.getRight().
      getGameValue())/ 2.0 );
226 
227     }
228   }
229 
```

```
230   //select the next node to be expended
231   public void selection() {
232     Node select = root;
233     select.visited();
234     while (select.getRight() != null && select.getLeft() != null)  {
235 
236       Node left =  select.getLeft();
237       Node right = select.getRight();
238 
239       // System.out.println(selectionValue(left));
240       if(selectionValue(left)<selectionValue(right)){
241         select = right;
242       }
243       else{
244         select = left;
245       }
246       select.visited();
247     }
248 
249     choice = select;
250   }
251 
252 
253 
254   @Override
255   public EpsilonBoxDominanceArchive getArchive() {
256     return (EpsilonBoxDominanceArchive)super.getArchive();
257   }
258 
259 
260 
261   @Override
262   public NondominatedSortingPopulation getPopulation() {
263     return (NondominatedSortingPopulation)super.getPopulation();
264   }
265 
266 }
```

Listing B.6: MCTS

```
1 
2 package uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.PSJ;
3 
```

```
4  import java.util.ArrayList;
5  import java.util.Iterator;
6  import java.util.LinkedList;
7  import java.util.List;
8
9
10 import org.moeaframework.algorithm.AbstractEvolutionaryAlgorithm;
11 import org.moeaframework.core.EpsilonBoxDominanceArchive;
12 import org.moeaframework.core.EpsilonBoxEvolutionaryAlgorithm;
13 import org.moeaframework.core.Initialization;
14 import org.moeaframework.core.NondominatedSortingPopulation;
15 import org.moeaframework.core.PRNG;
16 import org.moeaframework.core.Population;
17 import org.moeaframework.core.Problem;
18 import org.moeaframework.core.Selection;
19 import org.moeaframework.core.Solution;
20 import org.moeaframework.core.Variation;
21 import org.moeaframework.core.comparator.ChainedComparator;
22 import org.moeaframework.core.comparator.CrowdingComparator;
23 import org.moeaframework.core.comparator.DominanceComparator;
24 import org.moeaframework.core.comparator.ParetoDominanceComparator;
25 import org.moeaframework.core.operator.TournamentSelection;
26 import java.lang.Math;
27
28 public class HCS extends AbstractEvolutionaryAlgorithm implements
29     EpsilonBoxEvolutionaryAlgorithm {
30
31
32   private final Selection selection;
33
34   /**
35    * The variation operator.
36    */
37   private final Variation variation;
38
39
40   private final TournamentSelection s;
41
42   /**
43    * Parameters needed for MDEO to accept HCS without adapter, however not
       necessary
44    *
```

```
45   * @param problem the problem being solved
46   * @param population the population used to store solutions
47   * @param archive the archive used to store the result; can be {@code null}
48   * @param selection the selection operator
49   * @param variation the variation operator
50   * @param initialization the initialization method
51   */
52
53  public HCS(Problem problem, NondominatedSortingPopulation population,
54      EpsilonBoxDominanceArchive archive, Selection selection,
55      Variation variation, Initialization initialization) {
56
57    super(problem, population, archive, initialization);
58    this.s = (TournamentSelection) selection;
59    this.selection = selection;
60    this.variation = variation;
61  }
62
63  @Override
64  public void iterate() {
65    NondominatedSortingPopulation population = getPopulation();
66    Iterator<Solution> itr = population.iterator();
67        Solution besSolution = (Solution) itr.next();
68    Solution[] next = new Solution[variation.getArity()];
69    for (int i = 0; i < next.length; i++) {
70      next[i] = besSolution;
71    }
72    besSolution = expand(next);
73    if(besSolution!= null){
74      population.clear();
75      population.add(besSolution);
76
77    // //nrp output
78    // population.add(besSolution);
79    // population.truncate(population.size());
80    }
81
82
83  }
84
85  // -1 if solution 1 dominated solution 2
86  public int compareDomin(Solution solution1, Solution solution2){
```

```
87      return s.getComparator().compare(solution1, solution2);
88    }
89
90    // Create new Solutions based on parent
91    public Solution expand(Solution[] parent) {
92      Solution equal = null;
93      Solution[] solutions = new Solution[variation.getArity()];
94      for (int i = 0; i < 50; i++) {
95        solutions = variation.evolve(parent);
96        evaluateAll(solutions);
97        if(compareDomin(solutions[0], parent[0]) == -1){
98          return solutions[0];
99        }
100       else if(compareDomin(solutions[0], parent[0]) == 0){
101         equal = solutions[0];
102       }
103     }
104     return equal;
105
106
107   }
108
109
110
111   @Override
112   public EpsilonBoxDominanceArchive getArchive() {
113     return (EpsilonBoxDominanceArchive)super.getArchive();
114   }
115
116
117
118   @Override
119   public NondominatedSortingPopulation getPopulation() {
120     return (NondominatedSortingPopulation)super.getPopulation();
121   }
122
123 }
```

Listing B.7: Hill Climbing

```
1 package uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.PSJ;
2
3 import org.moeaframework.core.Solution;
```

```
4  import org.sidiff.common.emf.access.path.axis.Parent;
5  import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.problem.
       MoeaOptimisationSolution;
6  import java.util.ArrayList;
7
8  public class UnrestrictedNode {
9
10     protected MoeaOptimisationSolution solution = null;
11     protected UnrestrictedNode parent = null;
12     protected int visited = 0;
13     protected double gameValue = 0; // heuristic
14     protected int childrenVisited = 0;
15     public UnrestrictedNode[] children = null;
16
17     UnrestrictedNode(MoeaOptimisationSolution solution) {
18         this.solution = solution;
19     }
20
21     public UnrestrictedNode() {
22     }
23
24
25     public void setChildren(UnrestrictedNode[] children){
26         this.children = children;
27     }
28
29     public UnrestrictedNode[] getChildren(){
30         return this.children;
31     }
32
33     public MoeaOptimisationSolution getSolution() {
34         return this.solution;
35     }
36
37     public void setSolution(MoeaOptimisationSolution solution) {
38         this.solution = solution;
39     }
40
41     public void visited(){
42         visited++;
43     }
44
```

```
45      public int getChildrenVisited(){
46          return childrenVisited;
47      }
48      public void setChildrenVisited(int c){
49          childrenVisited = c;
50      }
51      public double setGameValue(double g){
52          gameValue = g;
53          return gameValue;
54      }
55
56      public double getGameValue(){
57          return gameValue;
58      }
59
60      public int getVisited(){
61          return visited;
62      }
63
64      public UnrestrictedNode getParent() {
65          return this.parent;
66      }
67
68      public void setParent(UnrestrictedNode parent) {
69          this.parent = parent;
70      }
71  }
```

Listing B.8: UnrestrictedNode

```
1   package uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.PSJ;
2
3   import java.util.ArrayList;
4   import java.util.Arrays;
5   import java.util.LinkedList;
6   import java.util.List;
7   import java.util.stream.*;
8   import java.util.Iterator;
9
10  import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.executor.
        SolutionGenerator;
11
12  //required for implementing mutation strategy
```

```
13 import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.interpreter.
       henshin.HenshinExecutor;
14 import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.operators.
       adaptation.MutationStepSizeStrategy;
15 import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.operators.mutation
       .selection.RandomOperatorSelector;
16
17 import org.moeaframework.algorithm.AbstractEvolutionaryAlgorithm;
18 import org.moeaframework.core.EpsilonBoxDominanceArchive;
19 import org.moeaframework.core.EpsilonBoxEvolutionaryAlgorithm;
20 import org.moeaframework.core.Initialization;
21 import org.moeaframework.core.NondominatedSortingPopulation;
22 // import org.moeaframework.core.PRNG;
23 import org.moeaframework.core.Population;
24 import org.moeaframework.core.Problem;
25 import org.moeaframework.core.Selection;
26 import org.moeaframework.core.Solution;
27 import org.moeaframework.core.Variation;
28
29 import org.moeaframework.core.operator.TournamentSelection;
30
31 import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.problem.*;
32 import java.lang.Math;
33
34
35
36 public class UnrestrictedMCTS extends AbstractEvolutionaryAlgorithm implements
       EpsilonBoxEvolutionaryAlgorithm {
37
38
39   private final Selection selection;
40
41   private AllMatchingOneStepMutationStrategy mutation;
42
43   /**
44    * The variation operator.
45    */
46   private final Variation variation;
47
48   // private int counter;
49
50
51
```

```
52  private final TournamentSelection s;
53
54  private UnrestrictedNode root;
55  private UnrestrictedNode best;
56  private UnrestrictedNode choice;
57
58
59  MoeaOptimisationProblem moeaProblem;
60
61  /**
62   * Parameters needed for MDEO to accept MCTS without adapter, however not
     necessary
63   *
64   * @param problem the problem being solved
65   * @param population the population used to store solutions
66   * @param archive the archive used to store the result; can be {@code null}
67   * @param selection the selection operator
68   * @param variation the variation operator
69   * @param initialization the initialization method
70   */
71
72  public UnrestrictedMCTS(Problem problem, NondominatedSortingPopulation
     population,
73      EpsilonBoxDominanceArchive archive, Selection selection,
74      Variation variation, Initialization initialization) {
75    super(problem, population, archive, initialization);
76
77    this.moeaProblem = (MoeaOptimisationProblem) problem;
78
79    this.mutation = new AllMatchingOneStepMutationStrategy(moeaProblem.
     getSolutionGenerator().getHenshinExcutor(),
80                                new RandomOperatorSelector(moeaProblem.
     getSolutionGenerator().getHenshinExcutor()));
81
82    this.s = (TournamentSelection) selection;
83
84    this.selection = selection;
85
86    this.variation = variation;
87
88    root = new UnrestrictedNode();
89
```

```
90  }
91
92  @Override
93  public void iterate() {
94
95    NondominatedSortingPopulation population = getPopulation();
96
97    if (root.getSolution() == null) {
98
99      initialization();
100   }
101
102   //select the Node/solution to expand
103   selection();
104
105
106   expansionMain();
107
108
109   backpropagation();
110
111   //determine if the selected solution is the best
112   if(compareDomin((Solution) choice.getSolution(), (Solution) best.getSolution
      ()) == -1){
113     best = choice;
114   }
115   // if( heuristicEstimate((Solution) choice.getSolution()) >
      heuristicEstimate((Solution) best.getSolution())  ){
116   //  best = choice;
117   // }
118   population.clear();
119
120   population.add( (Solution) best.getSolution());
121
122
123   // //nrp output
124   // population.add(choice.getSolution());
125   // population.truncate(population.size());
126
127 }
128
129 public void initialization(){
```

```
130
131         root.setSolution( moeaProblem.initialModelasSolution());
132         root.setGameValue(heuristicEstimate((Solution) root.getSolution()));
133         choice = root;
134         best = root;
135     }
136
137     //Expand selected Node
138     public void expansionMain(){
139
140         choice.setChildren( expand(choice) );
141
142     }
143
144     //-1 solution 1 dominates 2
145     public int compareDomin(Solution solution1, Solution solution2){
146       return s.getComparator().compare(solution1, solution2);
147     }
148
149
150     public boolean compareHeuristic(Solution solution1, Solution solution2){
151       return heuristicEstimate(solution1) > heuristicEstimate(solution2);
152     }
153
154     // estimate heuristic use fitnesses and unsatisfied constraints
155     public double heuristicEstimate(Solution solution){
156       evaluate(solution);
157       return ( -1.0*Arrays.stream(solution.getObjectives()).sum() )
158             - Arrays.stream(solution.getConstraints()).sum()
159             ;
160     }
161
162     //Selection Strategy2
163     public double selectionValue2(UnrestrictedNode node){
164
165       //Selection Strategy 2
166       if(node.getVisited() == 0){
167         return Double.POSITIVE_INFINITY;
168       }
169       else if(node.getChildrenVisited() == 0){
170         return Double.POSITIVE_INFINITY;
171       }
```

```
172     else{
173     return node.getGameValue() + 0.2*Math.sqrt( (Math.log((double) node.
        getVisited())) / (double) node.getChildrenVisited() )
174     ;
175
176     }
177   }
178
179   public double selectionValue(UnrestrictedNode node){
180
181   //Selection Strategy 1
182   if(node.getVisited() == 0){
183     return Double.POSITIVE_INFINITY;
184   }
185   else{
186   UnrestrictedNode parent = node.getParent();
187   return node.getGameValue()
188       + 0.1*Math.sqrt( (Math.log((double) parent.getVisited())  ) / (double)
        node.getVisited());
189
190   }
191
192   }
193
194
195   public UnrestrictedNode[] expand(UnrestrictedNode parent){
196
197     MoeaOptimisationSolution parentSolution = parent.getSolution();
198     ArrayList<uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.interpreter
        .guidance.Solution> newModels = mutation.mutate(parentSolution.getModel());
199
200     UnrestrictedNode children[] = new UnrestrictedNode[newModels.size()];
201
202     for(int i = 0; i < newModels.size(); i++) {
203
204       MoeaOptimisationSolution copy = parentSolution.copy();
205       copy.setModel(newModels.get(i));
206       UnrestrictedNode child = new UnrestrictedNode();
207       child.setSolution(copy);
208       child.setParent(parent);
209       child.setGameValue(heuristicEstimate((Solution) copy ));
210       children[i] = child;
```

```
211     }
212
213     return children;
214   }
215
216   // Update childrenVisted count
217   public void backpropagation(){
218     UnrestrictedNode back = this.choice;
219
220     while(back != root && back.getParent() != root){
221       UnrestrictedNode[] children = back.getChildren();
222
223       // back.setChildrenVisited(back.getLeft().getChildrenVisited()+back.
      getLeft().getVisited() + back.getRight().getChildrenVisited()+back.getRight
      ().getVisited());
224       back.setChildrenVisited(childrenVisted(children));
225       // new heuristics update
226       // double difference = ( sumHeuristics(children) / (double) children.
      length) - back.getGameValue();
227       // back.setGameValue(back.getGameValue() + difference);
228
229       //old heuristic update
230       back.setGameValue(( sumHeuristics(children) / (double) children.length)  )
      ;
231
232       back = back.getParent();
233
234     }
235   }
236
237   public int childrenVisted(UnrestrictedNode[] children){
238     int visited = 0;
239         for(int i = 0; i < children.length; i++){
240             visited = visited + children[i].getVisited() + children[i].
      getChildrenVisited();
241         }
242     return visited;
243   }
244
245   public double sumHeuristics(UnrestrictedNode[] children){
246         double sum = 0;
247         for(int i = 0; i < children.length; i++){
```

```
248             sum += children[i].getGameValue();
249         }
250     return sum;
251     }
252
253
254
255   //select the next node to be expended
256   public void selection() {
257     UnrestrictedNode select = root;
258     select.visited();
259     while (select.getChildren() != null){
260        UnrestrictedNode[] children = select.getChildren();
261        UnrestrictedNode bestChild = children[0];
262        for(int i = 1; i < children.length; i++){
263          if( selectionValue( children[i] ) > selectionValue( bestChild ) ){
264            bestChild = children[i];
265          }
266        }
267        select = bestChild;
268        select.visited();
269     }
270
271
272     this.choice = select;
273   }
274
275
276
277   @Override
278   public EpsilonBoxDominanceArchive getArchive() {
279     return (EpsilonBoxDominanceArchive)super.getArchive();
280   }
281
282
283
284   @Override
285   public NondominatedSortingPopulation getPopulation() {
286     return (NondominatedSortingPopulation)super.getPopulation();
287   }
288
```

```
289 }
```

Listing B.9: UnrestrictedMCTS

```
1  package uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.PSJ
2
3  import java.util.HashMap
4  import java.util.Map
5  import java.util.ArrayList
6  import org.eclipse.emf.henshin.interpreter.impl.EGraphImpl
7  import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.interpreter.
       guidance.Solution
8  import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.interpreter.
       henshin.HenshinExecutor
9  import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.operators.mutation
       .selection.RandomOperatorSelector
10 import org.eclipse.emf.henshin.model.Unit
11 import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.moea.problem.
       MoeaOptimisationSolution
12
13 class AllMatchingOneStepMutationStrategy{
14
15   HenshinExecutor henshinExecutor
16
17   RandomOperatorSelector operatorSelectionDecorator;
18
19   new(HenshinExecutor henshinExecutor, RandomOperatorSelector
       randomOperatorSelector) {
20
21     this.henshinExecutor = henshinExecutor
22     this.operatorSelectionDecorator = randomOperatorSelector
23   }
24
25   def mutate(Solution model) {
26
27     var candidateSolution = new Solution(model)
28
29     //all solutions reachable by applying one step of transformation to the
       model
30     var allReachableSolutions = new ArrayList<Solution>
31
32     val graph = new EGraphImpl(candidateSolution.getModel)
33
```

```
  var transformations = applyOperators(candidateSolution, graph);


      while (transformations != null){
          candidateSolution.updateModel(graph.roots.head, transformations);
    allReachableSolutions.add(candidateSolution);
    candidateSolution = new Solution(model);
    transformations = applyOperators(candidateSolution, graph);
      }

  operatorSelectionDecorator.flushTriedOperators()

  return allReachableSolutions;
}

/* Apply transformations according to the configured step size.
 * @return a map of the ordered transformations applied in this step
 */
def Map<Integer, String> applyOperators(Solution candidateSolution, EGraphImpl
   egraph) {

      if(this.operatorSelectionDecorator.hasUntriedOperators == false){
          return null;
      }

  var stepTransformations = new HashMap<Integer, String>();

      // Run the mutation for one steps
    var Unit operator = null;
    var operatorApplied = false;

    do {
      operator = this.operatorSelectionDecorator.getNextOperator();

      if (henshinExecutor.operatorApplied(operator, egraph, candidateSolution)
  ) {
        stepTransformations.put(1, operator.name)
        operatorApplied = true;
        operator = null;
      }
    } while (!operatorApplied && operatorSelectionDecorator.
  hasUntriedOperators())
```

```
73
74
75      return stepTransformations;
76    }
77
78  }
```

Listing B.10: AllMatchingOneStepMutationStrategy

```
1  package uk.ac.kcl.inf.mdeoptimiser.libraries.core.tests
2
3  import com.google.inject.Injector
4  import com.google.inject.Key
5  import java.nio.file.Paths
6  import java.util.Date
7  import org.eclipse.xtext.testing.util.ParseHelper
8  import org.eclipse.xtext.testing.validation.ValidationTestHelper
9  import org.junit.jupiter.api.Assertions
10 import org.junit.jupiter.api.BeforeEach
11 import org.junit.jupiter.api.Test
12 import org.junit.jupiter.api.TestInfo
13 import uk.ac.kcl.inf.mdeoptimiser.languages.MoptStandaloneSetup
14 import uk.ac.kcl.inf.mdeoptimiser.languages.mopt.Optimisation
15 import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.
       OptimisationInterpreter
16 import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.output.MDEOBatch
17 import uk.ac.kcl.inf.mdeoptimiser.libraries.core.optimisation.output.
       MDEOResultsOutput
18 import org.junit.jupiter.api.Disabled
19
20 class AlgorithmTests {
21
22   Injector injector = new MoptStandaloneSetup().
       createInjectorAndDoEMFRegistration
23
24   ParseHelper<Optimisation> parseHelper
25   ValidationTestHelper validationHelper
26   String pathPrefix;
27
28   @BeforeEach
29   def void initialiseParserHelper() {
30     parseHelper = injector.getInstance(new Key<ParseHelper<Optimisation>>() {
31     })
```

```
32    validationHelper = new ValidationTestHelper()
33    pathPrefix = "gen/"
34  }
35
36  //MODEL A
37
38  @Test
39  def void craModelAMCTS(TestInfo testInfo) {
40
41    val model = parseHelper.parse('''
42      problem {
43        basepath <src/test/resources/models/cra/>
44        metamodel <architectureCRA.ecore>
45        model <TTC_InputRDG_D.xmi>
46      }
47      goal {
48        objective CRA maximise java { "models.moea.MaximiseCRA" }
49        constraint MinimiseClasslessFeatures java { "models.moea.
    MinimiseClasslessFeatures" }
50      }
51      search {
52        mutate using <craEvolvers.henshin> unit "createClass"
53        mutate using <craEvolvers.henshin> unit "assignFeature"
54        mutate using <craEvolvers.henshin> unit "moveFeature"
55        mutate using <craEvolvers.henshin> unit "deleteEmptyClass"
56      }
57      solver {
58        optimisation provider moea algorithm MCTS {
59          population: 40
60          variation: mutation
61          mutation.step: 1
62          mutation.strategy: random
63        }
64        termination {
65          evolutions: 500
66        }
67        batches 1
68      }
69    ''')
70
71    runTestSearch(model, testInfo)
72  }
```

```
73
74
75  @Test
76  def void craModelAHCS(TestInfo testInfo) {
77
78    val model = parseHelper.parse('''
79      problem {
80        basepath <src/test/resources/models/cra/>
81        metamodel <architectureCRA.ecore>
82        model <TTC_InputRDG_D.xmi>
83      }
84      goal {
85        refine metamodel {"Feature", "isEncapsulatedBy", 1, 1}
86        objective CRA maximise java { "models.moea.MaximiseCRA" }
87        constraint MinimiseClasslessFeatures java { "models.moea.
    MinimiseClasslessFeatures" }
88      }
89      search {
90        mutate {"Class"}
91      }
92      solver {
93        optimisation provider moea algorithm HCS {
94          population: 40
95          variation: mutation
96          mutation.step: 1
97          mutation.strategy: random
98        }
99        termination {
100         evolutions: 500
101       }
102       batches 1
103     }
104   ''')
105
106   runTestSearch(model, testInfo)
107 }
108
109
110 ////////////////////////////////
111 ////////////////////////////////
112 //NRP
113 ////////////////////////////////
```

```
114 ////////////////////////////
115
116 //NRP MODEL A
117
118 @Test
119 def void nrpModelAMCTS(TestInfo testInfo) {
120
121   val model = parseHelper.parse('''
122     problem {
123       basepath <src/test/resources/models/nrp/>
124       metamodel <nextReleaseProblem.ecore>
125       model <nrp-model-5-cus-25-req-63-sa.xmi>
126     }
127     goal {
128       objective MinimiseCost minimise java { "models.nrp.fitness.MinimiseCost"
    }
129       objective MaximiseSatisfaction maximise java { "models.nrp.fitness.
   MaximiseSatisfaction" }
130     }
131     search {
132       mutate using <mutation.henshin> unit "modifySelectionWithHierarchy"
133       mutate using <mutation.henshin> unit "modifySingleSelection"
134       mutate using <mutation.henshin> unit "selectHighestRealisation"
135       mutate using <mutation.henshin> unit "fixDependencies"
136     }
137     solver {
138       optimisation provider moea algorithm MCTS {
139         population: 40
140         variation: mutation
141         mutation.step: 1
142         mutation.strategy: random
143       }
144       termination {
145         evolutions: 500
146       }
147       batches 1
148     }
149   ''')
150
151   runTestSearch(model, testInfo)
152 }
153
```

```
154
155 @Test
156 def void nrpModelAHCS(TestInfo testInfo) {
157
158   val model = parseHelper.parse('''
159     problem {
160       basepath <src/test/resources/models/nrp/>
161       metamodel <nextReleaseProblem.ecore>
162       model <nrp-model-5-cus-25-req-63-sa.xmi>
163     }
164     goal {
165       objective MinimiseCost minimise java { "models.nrp.fitness.MinimiseCost" }
166       objective MaximiseSatisfaction maximise java { "models.nrp.fitness.MaximiseSatisfaction" }
167     }
168     search {
169       mutate using <mutation.henshin> unit "modifySelectionWithHierarchy"
170       mutate using <mutation.henshin> unit "modifySingleSelection"
171       mutate using <mutation.henshin> unit "selectHighestRealisation"
172       mutate using <mutation.henshin> unit "fixDependencies"
173     }
174     solver {
175       optimisation provider moea algorithm HCS {
176         population: 40
177         variation: mutation
178         mutation.step: 1
179         mutation.strategy: random
180       }
181       termination {
182         evolutions: 500
183       }
184       batches 1
185     }
186   ''')
187
188   runTestSearch(model, testInfo)
189 }
190
191
192
193
```

```
194
195 //NRP MODEL B
196
197 @Test
198 def void nrpModelBMCTS(TestInfo testInfo) {
199
200   val model = parseHelper.parse('''
201     problem {
202       basepath <src/test/resources/models/nrp/>
203       metamodel <nextReleaseProblem.ecore>
204       model <nrp-model-25-cus-50-req-203-sa.xmi>
205     }
206     goal {
207       objective MinimiseCost minimise java { "models.nrp.fitness.MinimiseCost"
    }
208       objective MaximiseSatisfaction maximise java { "models.nrp.fitness.
  MaximiseSatisfaction" }
209     }
210     search {
211       mutate using <mutation.henshin> unit "modifySelectionWithHierarchy"
212       mutate using <mutation.henshin> unit "modifySingleSelection"
213       mutate using <mutation.henshin> unit "selectHighestRealisation"
214       mutate using <mutation.henshin> unit "fixDependencies"
215     }
216     solver {
217       optimisation provider moea algorithm MCTS {
218         population: 40
219         variation: mutation
220         mutation.step: 1
221         mutation.strategy: random
222       }
223       termination {
224         time: 10
225       }
226       batches 1
227     }
228   ''')
229
230   runTestSearch(model, testInfo)
231 }
232
233
```

```
@Test
def void nrpModelBHCS(TestInfo testInfo) {

  val model = parseHelper.parse('''
    problem {
      basepath <src/test/resources/models/nrp/>
      metamodel <nextReleaseProblem.ecore>
      model <nrp-model-25-cus-50-req-203-sa.xmi>
    }
    goal {
      refine metamodel {"Solution", "selectedArtifacts", 1, -1}
      objective MinimiseCost minimise java { "models.nrp.fitness.MinimiseCost" }
      objective MaximiseSatisfaction maximise java { "models.nrp.fitness.MaximiseSatisfaction" }
    }
    search {
      mutate {"Solution","selectedArtifacts"}
    }
    solver {
      optimisation provider moea algorithm HCS {
        population: 40
        variation: mutation
        mutation.step: 1
        mutation.strategy: random
      }
      termination {
        time: 10
      }
      batches 1
    }
  ''')

  runTestSearch(model, testInfo)
}
```

```
274
275 //Mutation strategy repetitive
276
277 /**
278  * Helper method to run the MOPT configurations
279  *
280  * @param model instance of a parsed MOPT file
281  * @param testInfo instance of the running test function
282  */
283 private def void runTestSearch(Optimisation model, TestInfo testInfo) {
284       Assertions.assertNotNull(model)
285   validationHelper.assertNoErrors(model)
286
287   if (model !== null) {
288
289     val mdeoResultsOutput = new MDEOResultsOutput(
290       new Date(),
291       Paths.get(pathPrefix),
292       Paths.get(testInfo.testMethod.get.name),
293       model
294     );
295
296     var experimentId = 0;
297     do {
298
299       val startTime = System.nanoTime;
300       val optimisationInterpreter = new OptimisationInterpreter("", model);
301       val optimisationOutcome = optimisationInterpreter.start
302       val endTime = System.nanoTime;
303
304       val experimentDuration = (endTime - startTime) / 1000000
305
306       mdeoResultsOutput.logBatch(
307         new MDEOBatch(experimentId, experimentDuration, optimisationOutcome,
308           optimisationInterpreter.rulegenOperators))
309
310       experimentId++
311     } while (experimentId < model.solver.algorithmBatches);
312
313     mdeoResultsOutput.saveOutcome();
314   }
315 }
```

```
316 }
```

Listing B.11: AlgorithmTests