Cutting And Packing Algorithms Research Framework

http://caparf.googlecode.com

28 December 2009, Ufa SATU

Denis Nazarov

Cutting and packing algorithms developement problems

- Algorithms details are often omitted in articles
- It's hard to get/generate test instances proposed by other authors
- Different system configurations for testing
- Small number of review articles with algorithms comparisions

Cutting and Packing Algorithms Research Framework

CAPARF consists of:

1. Problems definition (for each type of problems)
2. Algorithms for solving particular type of problems or even a number of types (including lower/upper bounds, exact methods)
3. Test instances (raw data and generators)

CAPARF provides:

1. Convenient «testing scenario» model
2. Various reports generators (for example, results for article in TeX or presentation)
3. Easy way to develop your own algorithms using existing methods and compare them with well-known reference algorithms by other authors

CAPARF benefits

1. CAPARF contains a lot of algorithms and test instances for cutting and packing problems
2. CAPARF is an OpenSource project hosted at http://code.google.com under GPLv3 licence.
3. CAPARF is a cross-platform framework since it is implemented in Java
4. CAPARF forces unification: all algorithms for the same type of problem have the same interface

Problem definition in CAPARF

Each problem definitin consists of the following «fundamental types»:

- `Input` — input data for the problem,
- `Output` — output data for the problem
- `OutputVerdict` — result of output data verification

`OutputVerifier` is defined for triple of `<Input,Output,OutputVerdict>`, which is responsible for output data verification

Example 1: simple algorithm

```java
package com.googlecode.caparf.examples;

import java.util.ArrayList;
import java.util.List;

import com.googlecode.caparf.framework.base.Algorithm;
import com.googlecode.caparf.framework.spp2d.Input;
import com.googlecode.caparf.framework.spp2d.Output;

public class OneLineSample extends Algorithm<Input, Output> {
  @Override
  public Output solve(Input input) {
    List<Output.Point2D> placement =
        new ArrayList<Output.Point2D>(input.getRectangles().size());
    int currentHeight = 0;
    for (Input.Rectangle rect : input.getRectangles()) {
      // Place current rectangle on the top of the previous rectangle
      Output.Point2D lowerLeftPoint = new Output.Point2D();
      lowerLeftPoint.x = 0;
      lowerLeftPoint.y = currentHeight;
      placement.add(lowerLeftPoint);
      // Add height of the rectangle to the current height
      currentHeight += rect.height;
    }
    return new Output(placement);
  }
}
```

Example 2: testing scenario

```java
// Create testing scenario instance
Scenario<Input, Output, OutputVerdict> scenario =
    new Scenario<Input, Output, OutputVerdict>();

// Add algorithms to scenario
scenario.addAlgorithms(
    new SimpleFit(ItemOrder.NEXT_ITEM, PlacementStrategy.DEFAULT),
    new SimpleFit(ItemOrder.NEXT_ITEM, PlacementStrategy.SHIFT_RIGHTMOST_ITEM),
    new SimpleFit(ItemOrder.FIRST_FIT, PlacementStrategy.DEFAULT),
    new SimpleFit(ItemOrder.FIRST_FIT, PlacementStrategy.SHIFT_RIGHTMOST_ITEM));

// Create suite of Berkey and Wang inputs and add it to scenario
InputSuite<Input> berkeyWangSuite = new InputSuite<Input>()
    .addAll(BerkeyWangGenerator.getReferenceInstances(Type.CLASS_I))
    .addAll(BerkeyWangGenerator.getReferenceInstances(Type.CLASS_II))
    .addAll(BerkeyWangGenerator.getReferenceInstances(Type.CLASS_III))
    .addAll(BerkeyWangGenerator.getReferenceInstances(Type.CLASS_IV))
    .addAll(BerkeyWangGenerator.getReferenceInstances(Type.CLASS_V))
    .addAll(BerkeyWangGenerator.getReferenceInstances(Type.CLASS_VI));
scenario.addInputSuite(berkeyWangSuite);

// Set output verifier for scenario
scenario.setVerifier(new OutputVerifier());

// Run scenario
CaparfCore<Input, Output, OutputVerdict> invoker =
    new CaparfCore<Input, Output, OutputVerdict>();
invoker.run(scenario);
```

Contribution

One can contribute to CAPARF developement by:

1. Adding implemention of «good» algorithm described in some published article
2. Adding new problem type definition
3. Adding or suggesting improvements/fixes in CAPARF core

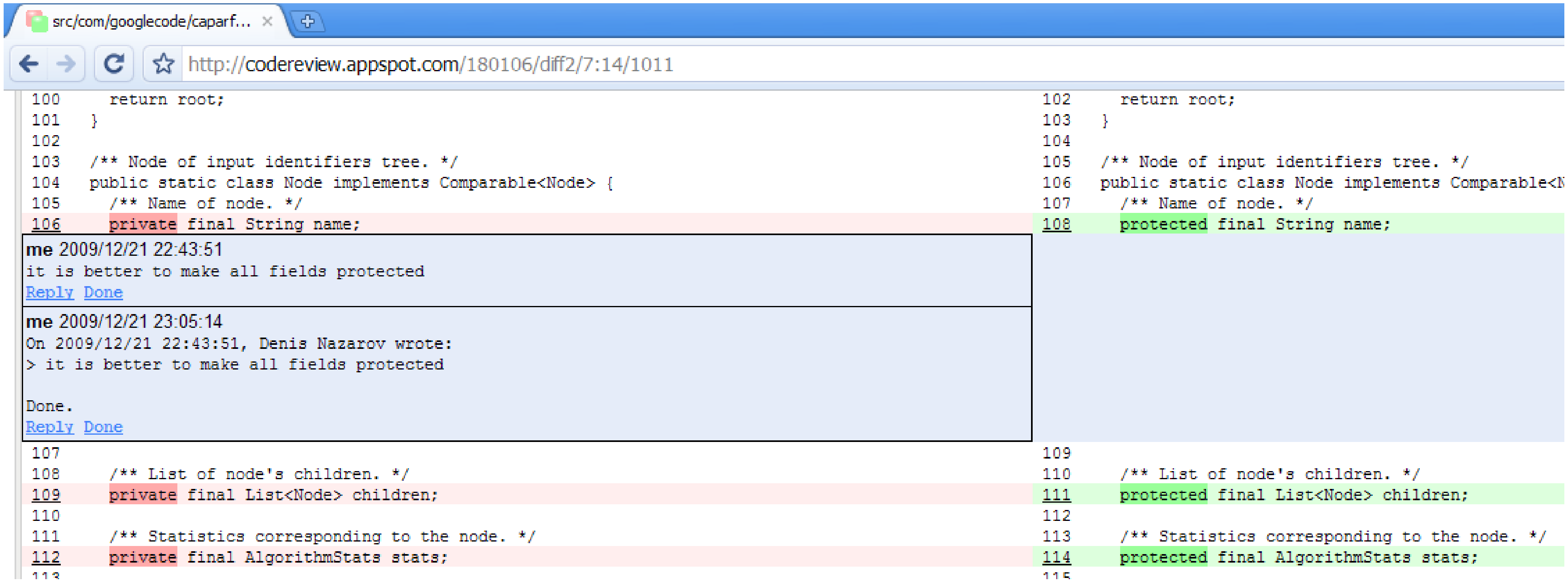
Roles in CAPARF developement:

contributor — create various changes and send them for code-review

developer — create various changes, do code-reviews, commit changes to the trunk

Code-review in CAPARF

All changes in CAPARF are required to pass code-review before commit using Rietveld Code Review Tool (for example, http://codereview.appspot.com/180106/show).



src/com/googlecode/caparf...

http://codereview.appspot.com/180106/diff2/7:14/1011

```
    return root;                                                  return root;
  }                                                             }

  /** Node of input identifiers tree. */                        /** Node of input identifiers tree. */
  public static class Node implements Comparable<Node> {        public static class Node implements Comparable<N
    /** Name of node. */                                          /** Name of node. */
    private final String name;                                    protected final String name;
```

me 2009/12/21 22:43:51

```
it is better to make all fields protected
```

Reply Done

me 2009/12/21 23:05:14

```
On 2009/12/21 22:43:51, Denis Nazarov wrote:
> it is better to make all fields protected

Done.
```

Reply Done

```

    /** List of node's children. */                               /** List of node's children. */
    private final List<Node> children;                            protected final List<Node> children;

    /** Statistics corresponding to the node. */                  /** Statistics corresponding to the node. */
    private final AlgorithmStats stats;                           protected final AlgorithmStats stats;
```