Interpretation of ROAD's output

The RoadTest.java creates experiment output for three scenarios using input graph Metropolis. The output of the main method is generated by three scenarios performed by methods

1. *deliverAllPathsCaseStudy*
2. *emergencyAllPathsCaseStudy*
3. *deliverShortestPathParametric*

Directories for each scenario are created in the Eclipse project's directory. The directories are named in static String declarations at the top of the RoadTest class. The location of Prism's binary is also declared as a static string.

The directories contain

- DTMC model files expressed in Prism's high-level modelling language
- PCTL property files
- A script file to automate verification of model and properties in Prism

---

The main method in the class RoadTest.java creates a city graph

```
State city = Metropolis.cityModel();
```

of the Metropolis case study, shown in Figure 1. The graph comprises 6 vertices and 18 edges. Each vertex represents a location on the map and edges represent two-way streets.

The Roads framework is initialised: `Roads roads = new Roads(city);`
This code initialises the key components (Planner, Model Synthesiser and Probabilistic Model Checker) for use with on Metropolis graph input

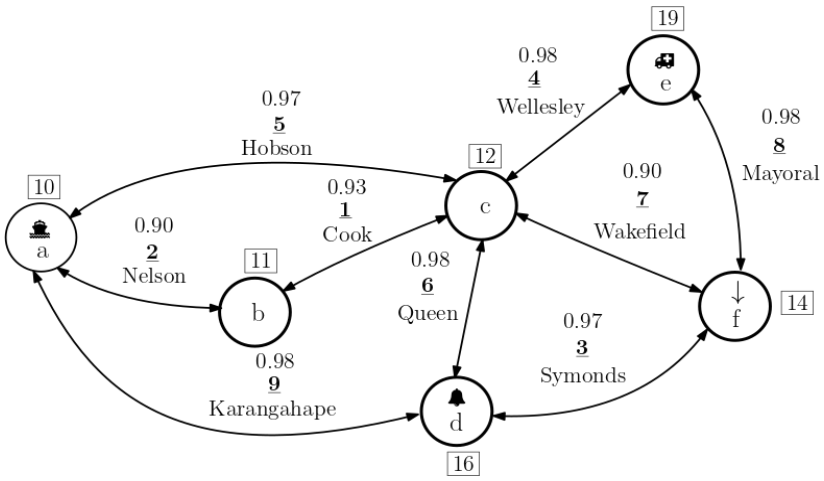

Figure 1Metropolis Topological Map labelled with distance and probabilities

---

Scenario 1: deliverAllPathsCaseStudy

In this scenario, a Deliver task transports 10 resources from Metropolis' supply vertex a to the destination vertex f. Upon arrival, an auditing task is performed.

We use Roads to model check each possible path between a and f against the PCTL properties:

P P=?[F(tasksuccess=1)]
R R=?[C]

to verify the probability a Deliver task is successful and the cost of attempting the task relative to the rewards structure associating distance to transitions in the path.

All paths between a -> f are computed by the Planner, stored in an ArrayList of Path objects:

[[(a : b), (b : c), (c : f)],
 [(a : b), (b : c), (c : d), (d : f)],
 [(a : b), (b : c), (c : e), (e : f)],
 [(a : c), (c : f)],
 [(a : c), (c : d), (d : f)],
 [(a : c), (c : e), (e : f)],
 [(a : d), (d : f)],
 [(a : d), (d : c), (c : f)],
 [(a : d), (d : c), (c : e), (e : f)]]

For each path a model and success and cost properties are synthesised and verified

```
RoadModel model = synth.synthesise(deliver, path);
RoadResult pmcResult = prob.verify(model, succ,cost);
```

The console output is below. Each line comprises: path, probabilistic model checking result for **P** and **R**, number of states, and number of transitions in the model.
The total amount of time required to complete model checking is 554ms.

```
Deliver (All Paths) Scenario
a -> b -> c -> f  0.70162  8.6714  20  28
a -> b -> c -> d -> f  0.74106  10.279  21  30
a -> b -> c -> e -> f  0.74870  12.682  21  30
a -> c -> f  0.81310  11.672  19  26
a -> c -> d -> f  0.85882  13.535  20  28
a -> c -> e -> f  0.86767  16.320  20  28
a -> d -> f  0.88538  11.821  19  26
a -> d -> c -> f  0.80506  21.387  20  28
a -> d -> c -> e -> f  0.85909  25.989  21  30
554ms
```

In the directory projectpath+*deliverScenarioPath* ROADS generates nine Prism model files corresponding to each path and a file storing properties **P** and **R**. A script file is generated to automatically verify each model against **P** and **R**.

Scenario 2: emergencyAllPathsScenario

In this scenario, an Emergency task travels from vertex e to attend an alarm notification at vertex d.

To verify the probability an Emergency task is successful and the cost of attempting the task relative to the rewards structure associating distance to transitions in the path.

We use Roads to model check each possible path between e and d against the PCTL properties **P** and **R**.

All paths between e -> d are computed by the Planner, stored in an ArrayList of Path objects:
[[(e : c), (c : d)],
[(e : c), (c : f), (f : d)],
[(e : c), (c : b), (b : a), (a : d)],
[(e : c), (c : a), (a : d)],
[(e : f), (f : d)],
[(e : f), (f : c), (c : d)],
[(e : f), (f : c), (c : b), (b : a), (a : d)],
[(e : f), (f : c), (c : a), (a : d)]]
[(e : c), (c : d)]
[(e : c), (c : f), (f : d)]
[(e : c), (c : b), (b : a), (a : d)]
[(e : c), (c : a), (a : d)]]

For each path a model and success and cost properties are synthesised and verified

```
RoadModel model = synth.synthesise(emergency, path);
RoadResult pmcResult = prob.verify(model, succ,cost);
```

The console output is below. Each line comprises: path, probabilistic model checking result for **P** and **R**, number of states, and number of transitions in the model.
The total amount of time required to complete model checking is 285ms.

```
Emergency All Paths Scenario
e -> c -> d  0.93178  9.7812  12  16
e -> c -> f -> d  0.83004  13.371  13  18
e -> c -> b -> a -> d  0.77990  14.043  14  20
e -> c -> a -> d  0.90383  17.281  13  18
e -> f -> d  0.92227  10.831  12  16
e -> f -> c -> d  0.83860  19.950  13  18
e -> f -> c -> b -> a -> d  0.70191  23.786  15  22
e -> f -> c -> a -> d  0.81344  26.700  14  20
```

285ms

In the directory projectpath+*emergencyScenario* ROADS generates eight Prism model files corresponding to each path and a file storing properties **P** and **R**. A script file is generated to automatically verify each model against **P** and **R**.

Scenario 3: deliverShortestPathParametricScenario

In this scenario, a deliver task must transport 10 resource units from the supply at vertex a to the destination at vertex f. In addition, the parameter list [pSoftware,pTraining, pRetry] is input to ROADS.

We use Roads to model check the shortest path between a and f against the PCTL properties:

P P=?[F(tasksuccess=1)]
R R=?[C]

to verify the probability a Deliver task is successful and the cost of attempting the task relative to the rewards structure associating distance to transitions in the path.

The shortest path computed by the Planner is: [(a : b), (b : c), (c : f)]

Roads synthesises a parametric model with parameters: pSoftware,pTraining, pRetry and verified using a parametric model checker (Prism)

```
RoadModel model = synth.synthesise(deliveraudit, route);
RoadResult pmcResult = prob.verify(model, succ,cost);
```

For this scenario, no console output is produced. The script for the task performs parametric model checking with the prism command:

prism parametricDeliver0.pm properties.pctl -param pSoftware=0.0:1.0, pTraining=0.0:1.0, pRetry=0.0:1.0 >> scenariooutput.txt

which invokes the Prism command line on the synthesised parametric DTMC in the Prism file parametricDeliver0.pm. Parametric model checking yields the expression for property **P**

Time for model construction: 0.077 seconds.
Time for model checking: 0.065 seconds.
Result (probability): ([0.0,1.0],[0.0,1.0],[0.0,1.0]): { 73830933 pTraining * pSoftware | 100000000 pRetry * pSoftware - 100000000 pRetry + 100000000 }

which can be used to compare the success of the deliver on the shortest path across different values for parameters for audit training, software reliability and likelihood of retry.