Competitive Milestone#2

Student: Weiyu Lin

Student Number: 932-479-491

Onid: Linweiy@oregonstate.edu

Introduction

I changed my testing object form MART (Mirror Adaptive Random Testing) to ART(Adaptive Random Testing) since MART is a bit difficult to testing. As we all known, Adaptive Radnom Testing is a famous black-box testing method. The random testing strategy is an extension of the random testing.The core code from T.Y. Chen's paper are following:

```
 Algorithm 1:
/*
selected set := { test data already selected };
candidate set := {};
total number of candidates := 10;
*/
function Select The Best Test Data(selected set, candidate set,
total number of candidates);
best distance := -1.0;
for i := 1 to total number of candidates do
candidate := randomly generate one test data from the program
input domain, the test data cannot be in
candidate set nor in selected set;
candidate set := candidate set + { candidate };
min candidate distance := Max Integer;
foreach j in selected set do
min candidate distance := Minimum(min candidate distance,
Euclidean Distance(j, candidate));
end foreach
if (best distance < min candidate distance) then
best data := candidate;
best distance := min candidate distance;
end if
end for
return best data;
end
```

It uses some special predefined values which can be simple boundary values or values that have high tendency of finding faults in the SUT[1]. For this assignment, I did a basic implementation of Adaptive Random Testing Algorithm using the TSTL APIs.

Implementation

My implementation file `tester2.py` and `tester1.py` can be found here.

For the program, there are five parsing parameters in my `tester1.py` , `tester2.py` , which is:

`BUDGET` : BUDGET is a parameter that can budge time in the program.

`SEED` : for random object and using for random number generation in the tester.py

`DEPTH` : DEPTH defines largest length of test in the algorithms.

`WIDTH` : WIDTH is for the deep search config of the algorithms.

`FAULT` : FAULT is applied for check the status (either 0 or 1) fault cheking in the SUTs. The working principle for FAULT is saving the failure found in the current directory.

`COVERAGE` : Coverage of testing of final coverage are reported by this parameter for using internalReport( ) function.

`RUNNING` : checking the brach of coverage(either 0 or 1) when after running by randomtester.py .
For implementation, you can tpying python BUDGET SEED DEPTH WIDTH FAULT , for example:

```
    python tester1.py 30 1 100 1 0 1 1
```

Algorithms

with the help of the algorithm in the T.Y. Chen's paper following:

1. At first, parsing parameter (DE)will be decleared.
2. randomEnable() will be refered with for loop to collect statement.
3. check the coverage of statement, find the coverage which lower than the tolerance.
4. Store the collected statement, if bugs were found, refer the failure in the TSTL API.
5. print out BUGs found, overall actions and total runtime.

Improvement for `tester2.py`

1. modify the threshold.
2. change time calculate method.

I select parameter as `30 1 100 1 1 1 1`

before the improvement I get following result:

```
TSTL BRANCH COUNT: 178
TSTL STATEMENT COUNT: 134
0 FAILED FOUND
178 New Branches
overall actions 29200
overall runtime 30.0029900074
linweiyutekiMacBook-Pro:CS569 WeiyuLin$
```

after the improvement I get following result:

```
TSTL BRANCH COUNT: 189
TSTL STATEMENT COUNT: 141
7 FAILED FOUND
189 New Branches
overall actions 31549
overall runtime 32.7211210728
```

Reference

[1] Richard Hamlet. Random testing. Encyclopedia of software Engineering, 1994.

word count [549]