FIT3036 - Computer Science Project - Test Report

CCS: Cryptic Clue Solver
Test Plan and Report

Author:
Jaroslaw Glowacki

Supervisor:
Dr. Marc Cheong

Contents

1 Introduction **1**

2 Test Report **1**

2.1 Performance Test 1

2.2 Logging 2

2.3 Integration Tests 2

2.4 Unit Tests 6

2.5 Timing Tests 7

2.6 User Acceptance Testing 9

3 Conclusion **9**

4 References **9**

1 Introduction

This report covers the test schemes employed during project development, closely following those outlined in the original project specification.

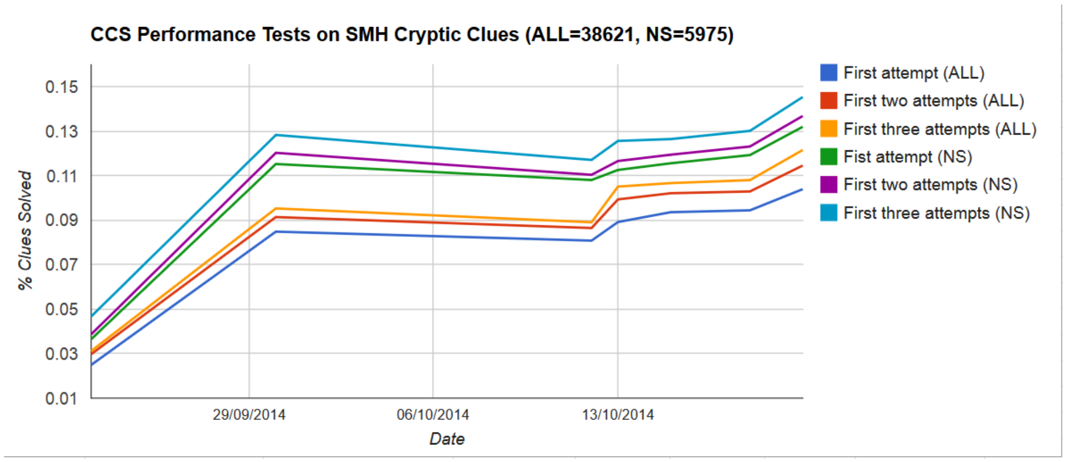
2 Test Report

CCS includes a sizeable test suite, which serves not only as a bug detection tool, but also to measure other useful information, such as the solver's performance at a given stage of development, and timing tests, to identify where bottlenecks are occuring. Each of these tests can be run from the console, by simply executing their respective scripts (they are all of the form 'ccs_<testtype>_tests<optionalsuffix>.py', located in the top level directory.)

2.1 Performance Test

The purpose of this test is to statistically measure CCS's solving capability. It subjects the tool to a huge pool of cryptic clues (with known solutions) from the Sydney Morning Herald newspaper [1], and returns some statistics based on how many clues were successfully solved within the top three attempts. Figure 1 shows the solver's performance over the course of the project till present.

Figure 1: Performance of CCS across project timeline



By running this test periodically during development, the results provided vital feedback, rating the usefulness of changes made to the design - as is evident in the chart, these changes didn't always end up being for the better.

2.2 Logging

Three dedicated loggers record information about each run, which can then be used for debugging purposes later:

- Last Run Log - 'CCS_lastRun.log', written to the current working directory; contains detailed information about the most recent run.
- Cumulative Log - 'CCS.log', in the same directory, containing less specific logs, but appends to the file rather than overwriting at each run.
- Console log - when running CCS from the console, prints certain logging information directly to it.

2.3 Integration Tests

These tests run basic clues that are expected to be solvable for different wordplay types and input formats through the full program, comparing the results output against expected solutions. They check not only the final solution, but also whether CCS correctly identifies the wordplay type. A simplified form of the tests that get run is included below. (Note that in the actual program each section would be getting run separately!):

```
s=cp.parseClue('Book in Habib Lew\'s handbag.')[0]
assertEqual('bible', s.solution, 'Wrong solution found at first position!')
assertEqual('run', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue('Punch a Rio Tinto official; find transport. (7)')[0]
assertEqual('chariot', s.solution, 'Wrong solution found at first position!')
assertEqual('run', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue('A pint makes colour.', 5)[0]
assertEqual('paint', s.solution, 'Wrong solution found at first position!')
assertEqual('anagram', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue('Frida, ILY — said Tom, holding a newspaper.', typ='run')[0]
assertEqual('daily', s.solution, 'Wrong solution found at first position!')
assertEqual('run', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue(Clue('Guide graphite'))[0]
assertEqual('lead', s.solution, 'Wrong solution found at first position!')
assertEqual('double definition', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue(Clue('House of God in Sencha Pellegrini bar. (6)'))[0]
assertEqual('chapel', s.solution, 'Wrong solution found at first position!')
assertEqual('run', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue(Clue('In job, sole technician, dated.', 8))[0]
assertEqual('obsolete', s.solution, 'Wrong solution found at first position!')
assertEqual('run', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue(Clue('Zoroastrian pairs dancing.', typ='anagram'))[0]
assertEqual('parsi', s.solution, 'Wrong solution found at first position!')
assertEqual('anagram', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue('Exuviate garage.')[0]
assertEqual('shed', s.solution, 'Wrong solution found at first position!')
assertEqual('double definition', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue('First man at carpet.', 3)[0]
assertEqual('mat', s.solution, 'Wrong solution found at first position!')
assertEqual('charade', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue('Initially babies are naked.', 4)[0]
assertEqual('bare', s.solution, 'Wrong solution found at first position!')
assertEqual('charade', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue('Prosciutto teases beds.', 8)[0]
assertEqual('hammocks', s.solution, 'Wrong solution found at first position!')
assertEqual('charade', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue('Rip lunch, last supper!', 4)[0]
assertEqual('tear', s.solution, 'Wrong solution found at first position!')
assertEqual('charade', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue('First male orphan on Io. (4)')[0]
assertEqual('moon', s.solution, 'Wrong solution found at first position!')
assertEqual('charade', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue('Purchased Game of Thrones, initially. (3)')[0]
assertEqual('got', s.solution, 'Wrong solution found at first position!')
assertEqual('initial', s.typ, 'Wrong wordplay type applied!')

s=cp.parseClue('Finally purchased Game Arena of hard of hearing. (4)')
assertEqual('deaf', s.solution, 'Wrong solution found at first position!')
assertEqual('final', s.typ, 'Wrong wordplay type applied!')
```

The output is generated by the standard Python unittest library [2], recognisable and auto-formatted by most Python-IDEs. Additionally, when printed rawly to the console, it is interjected by logging information, providing a more complete and informative result.

```
jarekwg@ubuntu:~/Desktop/CrypticClueSolver$ python3 ccs_integration_tests.py
2014-10-27 10:54:37,652 - INFO - clue_parser -
Parsing clue: Exuviate garage.
2014-10-27 10:54:41,799 - INFO - clue_parser -
20 solution(s) found. Best solution is 'shed' (certainty: 0.888889)
.2014-10-27 10:54:41,800 - INFO - clue_parser -
 Parsing clue: First man at carpet.
2014-10-27 10:54:46,549 - INFO - clue_parser -
70 solution(s) found. Best solution is 'mat' (certainty: 0.875000)
.2014-10-27 10:54:46,554 - INFO - clue_parser -
Parsing clue: Initially babies are naked.
2014-10-27 10:55:01,290 - INFO - clue_parser -
179 solution(s) found. Best solution is 'bare' (certainty: 1.000000)
.2014-10-27 10:55:01,293 - INFO - clue_parser -
Parsing clue: Prosciutto teases beds.
2014-10-27 10:55:04,048 - INFO - clue_parser -
61 solution(s) found. Best solution is 'hammocks' (certainty: 0.802005)
.2014-10-27 10:55:04,048 - INFO - clue_parser -
Parsing clue: Rip lunch, last supper!
2014-10-27 10:55:09,040 - INFO - clue_parser -
151 solution(s) found. Best solution is 'tear' (certainty: 0.875000)
.2014-10-27 10:55:09,041 - INFO - clue_parser -
Parsing clue: First male orphan on Io. (4)
2014-10-27 10:55:10,929 - INFO - clue_parser -
1 solution(s) found. Best solution is 'moon' (certainty: 0.823529)
.2014-10-27 10:55:10,929 - INFO - clue_parser -
Parsing clue: Purchased Game of Thrones, initially. (3)
2014-10-27 10:55:12,267 - INFO - clue_parser -
10 solution(s) found. Best solution is 'got' (certainty: 0.800000)
.2014-10-27 10:55:12,268 - INFO - clue_parser -
Parsing clue: Purchased Game of Thrones, initially. (3)
2014-10-27 10:55:13,231 - INFO - clue_parser -
10 solution(s) found. Best solution is 'got' (certainty: 0.800000)
.2014-10-27 10:55:13,232 - INFO - clue_parser -
```

```
Parsing clue: Guide graphite (None) <typ=None, letters=None>
2014—10—27 10:55:15,446 — INFO — clue_parser —
79 solution(s) found. Best solution is 'lead' (certainty: 0.705882)
.2014—10—27 10:55:15,447 — INFO — clue_parser —
Parsing clue: House of God in Sencha Pellegrini bar. (6) <typ=None, letters=None>
2014—10—27 10:55:18,018 — INFO — clue_parser —
6 solution(s) found. Best solution is 'chapel' (certainty: 0.941176)
.2014—10—27 10:55:18,019 — INFO — clue_parser —
Parsing clue: In job, sole technician, dated. (8) <typ=None, letters=None>
2014—10—27 10:55:24,172 — INFO — clue_parser —
1 solution(s) found. Best solution is 'obsolete' (certainty: 0.400000)
.2014—10—27 10:55:24,176 — INFO — clue_parser —
Parsing clue: Zoroastrian pairs dancing. (None) <typ=anagram, letters=None>
2014—10—27 10:55:24,304 — INFO — clue_parser —
3 solution(s) found. Best solution is 'parsi' (certainty: 0.631579)
.2014—10—27 10:55:24,305 — INFO — clue_parser —
Parsing clue: Book in Habib Lews handbag.
2014—10—27 10:55:41,769 — INFO — clue_parser —
12 solution(s) found. Best solution is 'bible' (certainty: 1.000000)
.2014—10—27 10:55:41,781 — INFO — clue_parser —
Parsing clue: Punch a Rio Tinto official; find transport. (7)
2014—10—27 10:55:47,554 — INFO — clue_parser —
1 solution(s) found. Best solution is 'chariot' (certainty: 0.800000)
.2014—10—27 10:55:47,560 — INFO — clue_parser —
Parsing clue: A pint makes colour.
2014—10—27 10:56:07,448 — INFO — clue_parser —
112 solution(s) found. Best solution is 'paint' (certainty: 0.933333)
.2014—10—27 10:56:07,453 — INFO — clue_parser —
Parsing clue: Frida, ILY — said Tom, holding a newspaper.
2014—10—27 10:56:07,628 — INFO — clue_parser —
7 solution(s) found. Best solution is 'daily' (certainty: 0.876190)
.
______________________________________________________________________

Ran 16 tests in 64.631s

OK
```

2.4 Unit Tests

Only three modules have unit tests written for them, as the remaining ones are either too ingrained with the remainder of CCS to be considered independently, or have been deemed too simple to require their own unit tests. The modules that get tested are `clue.py`, `wordnet.py` and `wordplay.py`. Below are included the test that get run for the Clue module. A similar approach is taken for the other two modules (refer to the source code for more information).

```
def test_validClueMinimal(self):
        c = Clue('This is a clue.')
        self.assertEqual(c.clue, 'This is a clue.')
        self.assertEqual(c.length, None)
        self.assertEqual(c.typ, None)
        self.assertEqual(c.known_letters, None)
        self.assertEqual(c.tokens, ['this', 'is', 'a', 'clue'])

def test_validClueWithLength(self):
        c = Clue('This is a clue. (7)')
        self.assertEqual(c.clue, 'This is a clue.')
        self.assertEqual(c.length, 7)
        self.assertEqual(c.typ, None)
        self.assertEqual(c.known_letters, None)
        self.assertEqual(c.tokens, ['this', 'is', 'a', 'clue'])

def test_validClueWithLengthAndType(self):
        c = Clue('This is a clue. (7)', typ='initial')
        self.assertEqual(c.clue, 'This is a clue.')
        self.assertEqual(c.length, 7)
        self.assertEqual(c.typ, 'initial')
        self.assertEqual(c.known_letters, None)
        self.assertEqual(c.tokens, ['this', 'is', 'a', 'clue'])

def test_validClueWithLengthAndKnownLetters(self):
        c = Clue('This is a clue. (7)', known_letters='????p??')
        self.assertEqual(c.clue, 'This is a clue.')
        self.assertEqual(c.length, 7)
        self.assertEqual(c.typ, None)
        self.assertEqual(c.known_letters, '????p??')
        self.assertEqual(c.tokens, ['this', 'is', 'a', 'clue'])

def test_validClueWithEverything(self):
        c = Clue('This is a clue. (7)', length=7, typ='anagram', known_letters='D???p??')
        self.assertEqual(c.clue, 'This is a clue.')
        self.assertEqual(c.length, 7)
        self.assertEqual(c.typ, 'anagram')
        self.assertEqual(c.known_letters, 'd???p??')
        self.assertEqual(c.tokens, ['this', 'is', 'a', 'clue'])

def test_invalidClueLengthMismatch(self):
        self.assertRaises(SolutionLengthMismatchException, Clue, 'Random clue. (4)', 5)
        self.assertRaises(SolutionLengthMismatchException, Clue, 'Random clue. (4)',
                                                                 known_letters='?f?e???')

def test_invalidClueMultiWordLength(self):
        self.assertRaises(UnsupportedClueException, Clue, 'Random clue. (4-4)')
        self.assertRaises(UnsupportedClueException, Clue, 'Random clue. (3,5)')
```

The output is of the same form as for the integration tests.

2.5 Timing Tests

Timing tests have been written for individual CCS functions deemed potentially problematic with regards to their execution speed. Timing a full integrated run is a somewhat pointless exercise due to it being closely dependent on the words chosen within each provided clue. That being said, the unit and integration tests do also print how long their respective tests took to run, which can be used to gain a vague idea this speed. The timing test code and output are below:

```
import timeit
import numpy as np
###################################################
function = 'wordnet.exists'
setup = """
import wordnet
def test():
        wordnet.exists('troubadour')
"""
reps = 1000000
print("Average time to execute function '%s' is %.3fus" %
(function, np.mean(timeit.Timer('test()', setup=setup).repeat(10, reps))/reps*1000000))
###################################################
function = 'wordnet.calcSimilarity'
setup = """
```

```
import wordnet
def test():
        wordnet.calcSimilarity('troubadour', 'valkyrie')
"""
reps = 10000
print("Average time to execute function '%s' is %.3fms" %
(function, np.mean(timeit.Timer('test()', setup=setup).repeat(reps, 1))*1000))
###################################################
function = 'wordnet.getSynonyms'
setup = """
import wordnet
def test():
        wordnet.getSynonyms('parsnip')
"""
reps = 1000000
print("Average time to execute function '%s' is %.3fus" %
(function, np.mean(timeit.Timer('test()', setup=setup).repeat(reps, 1))*1000000))
###################################################
function = 'wordnet.getAbbreviations'
setup = """
import wordnet
def test():
        wordnet.getAbbreviations('tungsten')
"""
reps = 1000000
print("Average time to execute function '%s' is %.3fus" %
(function, np.mean(timeit.Timer('test()', setup=setup).repeat(reps, 1))*1000000))
###################################################
function = 'wordnet.getWordsWithPattern'
setup = """
import wordnet
def test():
        wordnet.getWordsWithPattern('\A[a-z][a-z][a-z]i[a-z]a\Z')
"""
reps = 10
print("Average time to execute function '%s' is %.3fms" %
(function, np.mean(timeit.Timer('test()', setup=setup).repeat(reps, 1))*1000))
###################################################
function = 'wordplay.getAnagrams'
setup = """
```

```
from clue_parser import ClueParser
cp = ClueParser()
ana = cp.wordplays['anagram']
def test():
        ana.getPlay('pots')
"""
reps = 100000
print("Average time to execute function '%s' is %.3fus" %
(function, np.mean(timeit.Timer('test()', setup=setup).repeat(10, reps))/reps*1000000))
###################################################

Output:
Average time to execute function 'wordnet.exists' is 0.372us\\
Average time to execute function 'wordnet.calcSimilarity' is 0.276ms
Average time to execute function 'wordnet.getSynonyms' is 1.230us
Average time to execute function 'wordnet.getAbbreviations' is 0.611us
Average time to execute function 'wordnet.getWordsWithPattern' is 437.618ms
Average time to execute function 'wordplay.getAnagrams' is 1.799us
```

2.6 User Acceptance Testing

This testing is used to gauge the robustness and ease-of-use of the GUI. The tests are largely unstructured and consist of asking someone unfamiliar with the tool to perform some clue-parsing request of their choice. Over the course of the project, this test helped pick up several bugs in both the connections between interface and CCS parameters, along with some bugs in CCS's solving algorithms themselves.

3 Conclusion

Concluding, CCS employs a comprehensive test suite, which serves as a means of debugging problems, detecting new bugs introduced after changes made to the code, and gauging performance of the tool with respect to both speed and solving success rates. It is an invaluable resource during development, and will allow for a means of 'sanity checking' if changes/additions are made in the future.

4 References

[1] The DA Trippers. *The Motherlode: Crosswords from Way Back*. 2010. URL: `http://datrippers.com/2010/01/22/the-motherlode-crosswords-from-way-back/` (visited on 08/28/2014).

[2] Python Software Foundation. *Unit testing framework*. URL: `https://docs.python.org/3/library/unittest.html` (visited on 10/15/2014).