Wizardry Party Phase II Writeup

Xintong (Robert) Wen & James Scott Reese

Our base algorithm reduces an instance of the wizardry party to become an instance of an ILP, another NP-complete problem. We have a python script `wizard_ordering.py` writes and executes an LP script.

Each wizard becomes a Integer LpVariable with possible values ranging from `1` to `n = num_wizards` because `num_wizards` would be the minimum number of integer values required to properly satisfy all constraints. We use `bigM` and `smallC` constasnts to force strict inequalities. Each constraint also makes use of a helper binary variable `z` that indicates on which end of a range a wizard lies. Using the constraint `Tony Bruce Thor` as an example, we would have the following lines in our script:

The following block would have been executed before iterating through the constraints:

```
smallC = 1
bigM = 5 + num_wizards
Tony = LpVariable("Tony", 1, num_wizards, cat="Integer")
Bruce = LpVariable("Bruce", 1, num_wizards, cat="Integer")
Thor = LpVariable("Thor", 1, num_wizards, cat="Integer")
```

Once we have reached this constraint through our iterations, the following block is executed:

```
z = LpVariable("z", cat="Binary")
prob += Thor + smallC <= Tony + bigM * z
prob += Thor + smallC <= Bruce + bigM * z
prob += Thor >= smallC + Tony - (1 - z) * bigM
prob += Thor >= smallC + Bruce - (1 - z) * bigM
```

This imposes the constraints that if `z = 0` then `Thor < Tony and Thor < Bruce` and if `z = 1` then `Thor > Tony and Thor > Bruce`

We Use the PuLP package which has documentation at `https://pypi.python.org/pypi/PuLP/1.6.8` and can be installed as simply as running `pip install pulp` from the command line. We also tried a number of LpSolvers that PuLP calls, but the one with the best performance was the Gurobi solver. Gurobi offers a free academic license here `https://user.gurobi.com/download/licenses/free-academic` and comes with an easy to use installer. We were able to use the Gurobi Optimizer to solve up to inputs of size 140. For input sizes above 140, the Optimizer would take far too long and far too many resources so we instead used a randomized approach called simulated annealing for greater input sizes.

The basic idea for simulated annealing can be found here: `katrinaeg.com/simulated-annealing.html`. Our implementation can be found in `simulated_annealing.py`, which is called by `wizard_ordering.py` as a module when dealing with

larger input sizes. This method is non-deterministic because it uses randomization, but it is much faster. We essentially used this approach to brute force larger input files. The idea is to output a random ordering and then continuously improve upon it by comparing an ordering to its neighbor, which is a swapping of one wizard's place to that of another. If this reduces the number of failed constraints (as the `cost_opt` function) then we proceed with the new ordering, otherwise we may or may not proceed with this ordering based on an `acceptance_probability` method that compares the previous and current number of constaints failed. The number of iterations performed is dependent on the paramters `alpha` and `i`. The greater these values were, the longer the algorithm would run for, but the more likely we would return an ordering that satisfies all constraints. We noticed that when we used a list, we never cared about anything except the index of a wizard, so we improved runtime by operating only on a map of a wizard to its rank. For example, `Tony : 0` in the map would mean that `Tony` is the left-most wizard in the ordering.

```
 1 import argparse
 2 import os
 3 import output_validator
 4 import simulated_annealing
 5
 6 staffMin = 160
 7
 8 def read_input(filename):
 9     with open(filename) as f:
10         num_wizards = int(f.readline())
11         num_constraints = int(f.readline())
12         constraints = []
13         wizards = set()
14         for _ in range(num_constraints):
15             c = f.readline().split()
16             constraints.append(c)
17             for w in c:
18                 wizards.add(w)
19
20     wizards = list(wizards)
21     return num_wizards, num_constraints, wizards,
   constraints
22
23 def solve(num_wizards, num_constraints, wizards,
   constraints, filename, outfile):
24
25     # used simulated annealing optimization algorithm to
   solve staff inputs
26     if num_wizards >= staffMin:
27         simulated_annealing.solve_opt(wizards, constraints
   , outfile)
28     else:
29         base = os.path.basename(filename)
30         filenameLP = os.path.splitext(base)[0] + "_LP.py"
31
32         # for smaller input sizes, reduce to ILP
33
34         with open(filenameLP, "w") as f:
35             # setting up LP script
36             f.write("import time\n")
37             f.write("from pulp import *\n\n")
38             f.write("prob = LpProblem(\"wiz\", LpMaximize)
   \n\n")
39
40             f.write("start = time.time()\n")
```

```
41              f.write("smallC = 1\n")
42              f.write("bigM = {0}\n".format(5 + num_wizards)
    )
43              f.write("wizzies = set()\n")
44              # wizard variables
45              for wiz in wizards:
46                  f.write("{0} = LpVariable(\"{0}\", 1, {1
    }, cat = \"Integer\")\n".format(wiz, num_wizards))
47                  f.write("wizzies.add(\"{0}\")\n".format(
    wiz))
48
49              # objective: max0 (no maximization or min,
    just want to see if a solution exists)
50              f.write("\nprob += 0\n\n")
51
52              # turn wiz constraints into LP constraints
53              for i in range(num_constraints):
54                  constraint = constraints[i]
55                  x_1 = constraint[2]
56                  x_2 = constraint[0]
57                  x_3 = constraint[1]
58
59                  # x1 must be outside the age bounds of x2
    , x3
60                  # e.g. constraint = x_2 x_3 x_1
61
62      #           # binary helper for constrained ordering
63                  z_1 = "z{0}_1".format(i)
64                  f.write("{0} = LpVariable(\"{0}\", cat=\"
    Binary\")\n".format(z_1))
65
66                  # z1 => x1 > x2 and x1 > x3
67                  # !z1 => x < x2 and x1 < x3
68                  f.write("prob += {0} + smallC <= {1} +
    bigM * {2}\n".format(x_1, x_2, z_1))
69                  f.write("prob += {0} + smallC <= {1} +
    bigM * {2}\n".format(x_1, x_3, z_1))
70                  f.write("prob += {0} >= smallC + {1} - (1
     - {2}) * bigM\n".format(x_1, x_2, z_1))
71                  f.write("prob += {0} >= smallC + {1} - (1
     - {2}) * bigM\n\n".format(x_1, x_3, z_1))
72
73              f.write("\n")
74              # f.write("GLPK().solve(prob)\n")
75              f.write("prob.solve(pulp.GUROBI_CMD())\n")
```

```
 76             f.write("ages = {}\n")
 77             f.write("for v in prob.variables():\n")
 78             f.write("\tif v.name in wizzies:\n")
 79             f.write("\t\tprint(v.name, \"=\", v.varValue)
    \n")
 80             f.write("\t\tages[v.name] = v.varValue\n")
 81             f.write("print(\"Time taken: {0}\".format((
    time.time() - start) * 1000))\n")
 82             f.write("sorted_ages = sorted(ages.items(),
    key = lambda x: x[1])\n")
 83             f.write("with open(\"{0}\", \"w\") as f:\n".
    format(outfile))
 84             f.write("\tfor i in range({0}):\n".format(
    num_wizards))
 85             f.write("\t\tf.write(\"{0} \".format(
    sorted_ages[i][0]))\n")
 86
 87         os.system("python {0}".format(filenameLP))
 88         os.remove(filenameLP)
 89
 90
 91 # python wizard_ordering.py ./inputs/inputs20/input20_0.
    in ./outputs/outputs20/output20_0.out
 92 if __name__ == "__main__":
 93     parser = argparse.ArgumentParser(description = "
    Constraint Solver.")
 94     parser.add_argument("input_file", type=str, help = "
    ___.in")
 95     parser.add_argument("output_file", type=str, help = "
    ___.out")
 96     args = parser.parse_args()
 97
 98     num_wizards, num_constraints, wizards, constraints =
    read_input(args.input_file)
 99     solve(num_wizards, num_constraints, wizards,
    constraints, args.input_file, args.output_file)
100     constraints_satisfied, num_constraints,
    constraints_failed = output_validator.processInput(args.
    input_file, args.output_file)
101     print("You satisfied {}/{} constraints. List of
    failed constraints: {}".format(constraints_satisfied,
    num_constraints, constraints_failed))
102
103
```

simulated_annealing.py

```
 1 import random
 2 import math
 3 import time
 4
 5 # Randomization Idea: http://katrinaeg.com/simulated-
   annealing.html
 6
 7 def swap_opt(wizards, mapping):
 8     wiz1 = random.choice(wizards)
 9     wiz2 = random.choice(wizards)
10     wiz1_prev_index = mapping[wiz1]
11     wiz2_prev_index = mapping[wiz2]
12     mapping[wiz2] = wiz1_prev_index
13     mapping[wiz1] = wiz2_prev_index
14     return wiz1, wiz2
15
16 def cost_opt(mapping, constraints):
17     failed = 0
18     for c in constraints:
19         wiz_a = mapping[c[0]]
20         wiz_b = mapping[c[1]]
21         wiz_mid = mapping[c[2]]
22         if (wiz_a < wiz_mid < wiz_b) or (wiz_b < wiz_mid <
    wiz_a):
23             failed += 1
24     return failed
25
26 def acceptance_probability(old_cost, new_cost, T):
27     if new_cost < old_cost:
28         return 1.0
29     else:
30         return math.exp((old_cost - new_cost) / T)
31
32 def anneal_opt(wizards, constraints):
33     random.shuffle(wizards)
34     tempSet = set(wizards)
35     # maps wiz order to reduce indexing runtime costs
36     mapping = {k: v for v, k in enumerate(tempSet)}
37     old_cost = cost_opt(mapping, constraints)
38     T = 1.0
39     T_min = 0.0001
40     alpha = 0.95 # increase to improve probability of
   satisfying all constraints
41     while T > T_min:
42         i = 0
```

```
43          while i < 8000: # increase to improve probability
   of satisfying all constraints
44              wiz1, wiz2 = swap_opt(wizards, mapping)
45              new_cost = cost_opt(mapping, constraints)
46              if new_cost == 0:
47                  print("Found!")
48                  return mapping
49              ap = acceptance_probability(old_cost, new_cost
   , T)
50              if ap > random.random():
51                  old_cost = new_cost
52              else:
53                  # revert back to pre-swap ordering
54                  temp = mapping[wiz1]
55                  mapping[wiz1] = mapping[wiz2]
56                  mapping[wiz2] = temp
57              i += 1
58          T = T * alpha
59      return mapping
60
61 def solve_opt(wizards, constraints, filename):
62      start = time.time()
63      mapping = anneal_opt(wizards, constraints)
64      solution = sorted(mapping.keys(), key=lambda x:
   mapping[x])
65      write_output(filename, solution)
66      print("Time taken: {0}".format(time.time() - start))
67
68 def write_output(filename, solution):
69      with open(filename, "w") as f:
70          for wizard in solution:
71              f.write("{0} ".format(wizard))
72
```

```
 1 import os
 2
 3 staffDirIn = "./inputs/inputs_staff/"
 4 staffDirOut = "./outputs/outputs_staff/"
 5 In20 = "./inputs/inputs20/"
 6 Out20 = "./outputs/outputs20/"
 7 In35 = "./inputs/inputs35/"
 8 Out35 = "./outputs/outputs35/"
 9 In50 = "./inputs/inputs50/"
10 Out50 = "./outputs/outputs50/"
11
12 # Executes wizard_ordering.py for a series of input files
13 # Input file locations must be relative to wizard_ordering
   .py as described above
14 # Comment out and edit ranges as necessary
15
16 # For Input 20s
17 for i in range(10):
18     inputFile = In20 + "input20_" + str(i) + ".in"
19     outputFile = Out20 + "input20_" + str(i) + ".out"
20     os.system("python wizard_ordering.py {0} {1}".format(
   inputFile, outputFile))
21
22 # For Input 35s
23 for i in range(10):
24     inputFile = In35 + "input35_" + str(i) + ".in"
25     outputFile = Out35 + "input35_" + str(i) + ".out"
26     os.system("python wizard_ordering.py {0} {1}".format(
   inputFile, outputFile))
27
28 # For Input 50s
29 for i in range(10):
30     inputFile = In50 + "input50_" + str(i) + ".in"
31     outputFile = Out50 + "input50_" + str(i) + ".out"
32     os.system("python wizard_ordering.py {0} {1}".format(
   inputFile, outputFile))
33
34 # For Staff Inputs
35 for i in range(3, 21):
36     j = i * 20
37     inputFile = staffDirIn + "staff_" + str(j) + ".in"
38     outputFile = staffDirOut + "staff_" + str(j) + ".out"
39     os.system("python wizard_ordering.py {0} {1}".format(
   inputFile, outputFile))
40
```

output_validator.py

```
# Released to students

import sys

def main(argv):
    if len(argv) != 2:
        print("Usage: python output_validator.py [path_to_input_file] [path_to_output_file]")
        return
    constraints_satisfied, num_constraints, constraints_failed = processInput(argv[0], argv[1])
    print("You satisfied {}/{} constraints. List of failed constraints: {}".format(constraints_satisfied, num_constraints, constraints_failed))

def processInput(input_file, output_file):
    fin = open(input_file, "r")
    fout = open(output_file, "r")

    num_wiz_in_input = int(fin.readline().split()[0])
    # input_wizard_set = set(fin.readline().split())
    num_constraints = int(fin.readline().split()[0])

    output_ordering = fout.readline().split()
    output_ordering_set = set(output_ordering)
    output_ordering_map = {k: v for v, k in enumerate(output_ordering)}


    if (len(output_ordering_set) != num_wiz_in_input):
        return "Input file has unique {} wizards, but output file has {}".format(num_wiz_in_input, len(output_ordering_set))

    if (len(output_ordering_set) != len(output_ordering)):
        return "The output ordering contains repeated wizards."

    # if (input_wizard_set != output_ordering_set):
        # return "The output ordering contains wizards that are different from the ones in the input ordering."

    # Counts how many constraints are satisfied.
    constraints_satisfied = 0
    constraints_failed = []
```

```
37      for i in range(num_constraints):
38          line_num = i + 4
39          constraint = fin.readline().split()
40
41          c = constraint # Creating an alias for easy
    reference
42          m = output_ordering_map # Creating an alias for
    easy reference
43
44          wiz_a = m[c[0]]
45          wiz_b = m[c[1]]
46          wiz_mid = m[c[2]]
47
48          if (wiz_a < wiz_mid < wiz_b) or (wiz_b < wiz_mid <
     wiz_a):
49              constraints_failed.append(c)
50          else:
51              constraints_satisfied += 1
52
53      return constraints_satisfied, num_constraints,
    constraints_failed
54
55  if __name__ == '__main__':
56      main(sys.argv[1:])
57
```