Jason Perrella & Vijay Manchiraju

Yoon

CSC345-02

10 March 2024

Project 2

For project two, we were tasked with developing a sudoku-board validity checker, with three methods of checking. For option one, one thread was to be used for checking rows, one thread for columns, and nine for each subgrid in the 9x9 grid, totaling 11 threads. Option two was similar, but instead of one thread for the rows and columns, it was nine for each, resulting in 27 total threads. Option three was a multiprocess implementation in which three child processes were created, one for checking rows, one for checking columns, and one for checking subgrids. After whichever option was chosen completes, the variable used to track the validity is used to determine what is outputted (valid or not), along with a variable used to hold the amount of time the execution took. Implementing the first two options was not that difficult, however, option three gave some trouble. At one point just before we finished, the program was able to successfully identify an invalid board with every option, but if the board was valid, only options one and two worked. We were using a pipe to allow the processes to communicate and update the variable *valid,* but every time the program got to the parent process and attempted to read from the pipe, nothing would occur because the problem, which was very simple, was that we were not closing the write end of the pipe in the parent process. After implementing that, the program was finished. To run each option 50 times, we wrote a bash script that would write the execution times to a .dat file and used the MATPLOTLIB python library to generate graphs to analyze options 1 and 2, options 2 and 3, and options 1 and 3. We used Pandas and SciPy to load the data into a dataframe and run the hypothesis tests(This was not  required by the project requirements but we wanted to make it easier to plot graphs, and run hypothesis tests as the math could easily be done by these libraries and is less error prone).

<u>Figure 1: Option 1 Runtimes</u>

main OSLab2 / option_1_runtimes.dat

jasonperrella Updated files

Code Blame 50 lines (50 loc) · 691 Bytes Code 55% faster with GitHub Copilot

```
1 .017245814
2 .016313378
3 .014957006
4 .022534150
5 .025849564
6 .015366739
7 .017366530
8 .016903476
9 .012517209
10 .010436217
11 .011960331
12 .013534346
13 .014146983
14 .010433182
15 .013714628
16 .012257634
17 .008679469
18 .017206198
19 .012395238
20 .011537643
21 .015143557
22 .018332618
23 .012074700
24 .024994175
25 .021443563
26 .016412027
27 .015480863
28 .012235947
29 .009236607
30 .017484162
31 .010112318
32 .012748711
33 .010897103
34 .016732125
35 .009695047
36 .012940634
37 .021518068
38 .015115652
39 .013821073
40 .016868457
41 .016258496
42 .013457183
43 .014342697
44 .012821489
45 .014270013
46 .014263895
47 .014272924
48 .014777892
49 .012979657
50 .013522153
```

Figure 2: Option 2 Runtimes

main OSLab2 / option_2_runtimes.dat

jasonperrella Updated files

Code Blame 50 lines (50 loc) · 691 Bytes Code 55% faster with GitHub Copilot

```
1 .018613800
2 .010944097
3 .015530889
4 .013781560
5 .014614781
6 .016515729
7 .024270837
8 .011485475
9 .011714872
10 .026377092
11 .011497977
12 .012644405
13 .016682050
14 .010267979
15 .010681346
16 .011874688
17 .015754813
18 .016269488
19 .017545942
20 .010747655
21 .009006268
22 .015333564
23 .020417211
24 .019326457
25 .021323390
26 .013426623
27 .014204130
28 .017806570
29 .013778507
30 .010569094
31 .010690756
32 .037393376
33 .018844388
34 .034684742
35 .016392969
36 .022339786
37 .019765931
38 .016011921
39 .017890056
40 .010961357
41 .012139820
42 .015300866
43 .017343163
44 .016819557
45 .010310621
46 .010777233
47 .050114654
48 .027364207
49 .025276477
50 .011889248
```

Figure 3: Option 3 Runtimes

main OSLab2 / option_3_runtimes.dat

jasonperrella Updated files

Code Blame 50 lines (50 loc) · 691 Bytes Code 55% faster with GitHub Copilot

```
1 .005412087
2 .003514335
3 .003740979
4 .004892834
5 .003367322
6 .003964923
7 .005615284
8 .006204955
9 .003592229
10 .006132170
11 .003422622
12 .005900323
13 .005399378
14 .003897485
15 .004063059
16 .003521236
17 .008066110
18 .003700216
19 .005949083
20 .003948251
21 .004892786
22 .003378555
23 .003704754
24 .006517813
25 .003299146
26 .004609521
27 .003564707
28 .004157505
29 .003390152
30 .005778085
31 .004495706
32 .003796193
33 .003837745
34 .003732854
35 .004826621
36 .003909105
37 .004611073
38 .004224475
39 .004790063
40 .004094846
41 .005278966
42 .003699753
43 .003595160
44 .006424072
45 .004008058
46 .003404253
47 .003762068
48 .006393179
49 .003378342
50 .005367807
```

Figure 4: Bash shell script. The graphing did not work. We needed to use our own local machines to graph everything.

main OSLab2 / sudoku.sh

jasonperrella adding files to repo

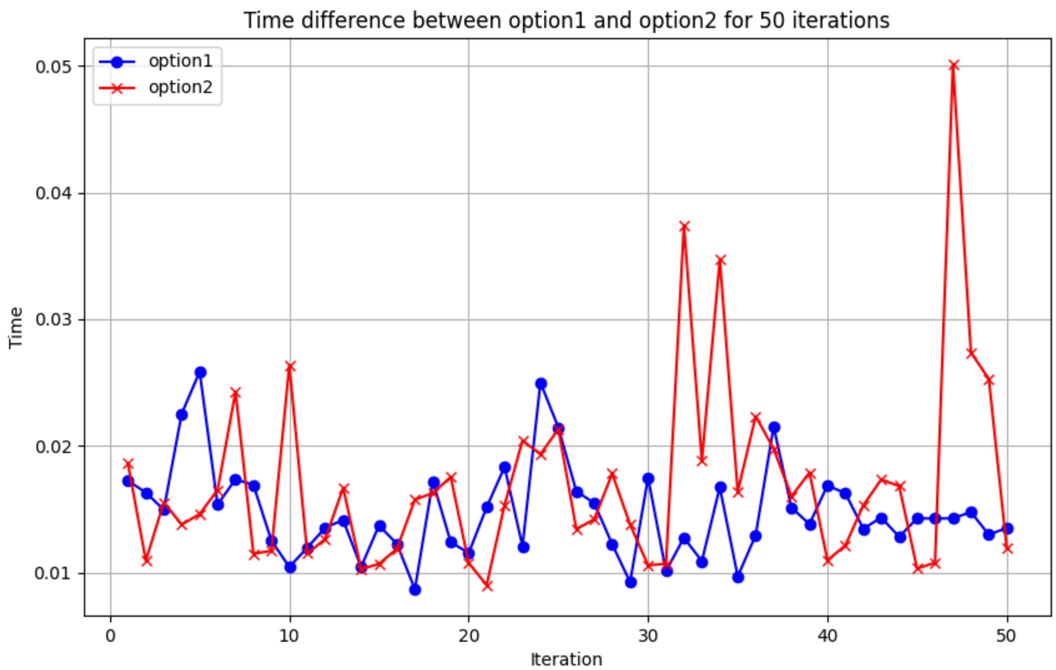
Code Blame 51 lines (40 loc) · 1.42 KB Code 55% faster with GitHub Copilot

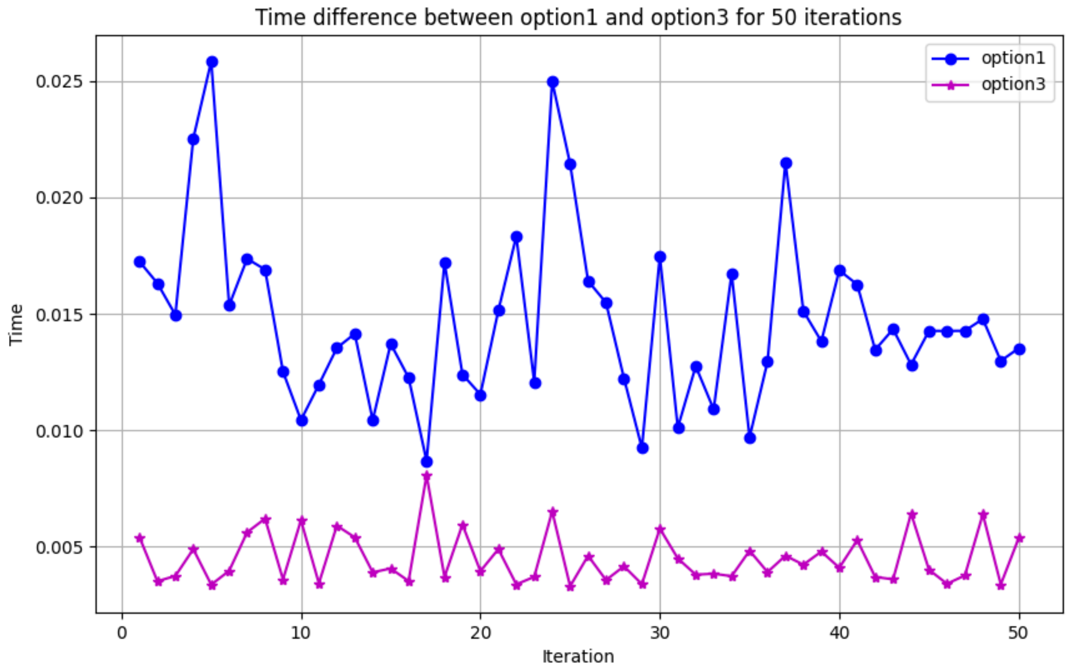
```
#!/bin/bash

for option in 1 2 3; do
    data_file="option_${option}_runtimes.dat"
    graph_file="option_${option}_graph.png"

    total_time=0
    iterations=50

    echo "Option $option Runtimes:"
    for i in $(seq 1 $iterations); do
        start_time=$(date +%s.%N)
        ./main $option
        end_time=$(date +%s.%N)
        duration=$(echo "$end_time - $start_time" | bc)
        total_time=$(echo "$total_time + $duration" | bc)

        # Print '*' for each second of runtime
        seconds=$(echo "$duration/1" | bc)
        printf "%3s | %s\n" "$i" "$(yes '*' | head -n $seconds | tr -d '\n')"
        echo "$i $duration" >> "$data_file"
    done

    avg_time=$(echo "$total_time/$iterations" | bc -l)
    echo -e "\nAverage execution time for option $option: $avg_time seconds\n"
done

# Plot graphs after the 3rd option using Python and matplotlib
if command -v python3 &> /dev/null; then
    python3 - <<EOF
import matplotlib.pyplot as plt

for option in [1, 2, 3]:
    data_file = f"option_{option}_runtimes.dat"
    graph_file = f"option_{option}_graph.png"

    with open(data_file, 'r') as f:
        data = [line.split() for line in f.readlines()]

    iterations, runtimes = zip(*data)
    plt.plot(iterations, runtimes, label=f'Option {option} Runtimes')

plt.title('Runtimes for Options 1, 2, and 3')
plt.xlabel('Iteration')
plt.ylabel('Runtime (seconds)')
plt.legend()
plt.savefig('combined_graph.png')
plt.show()
EOF
fi
```

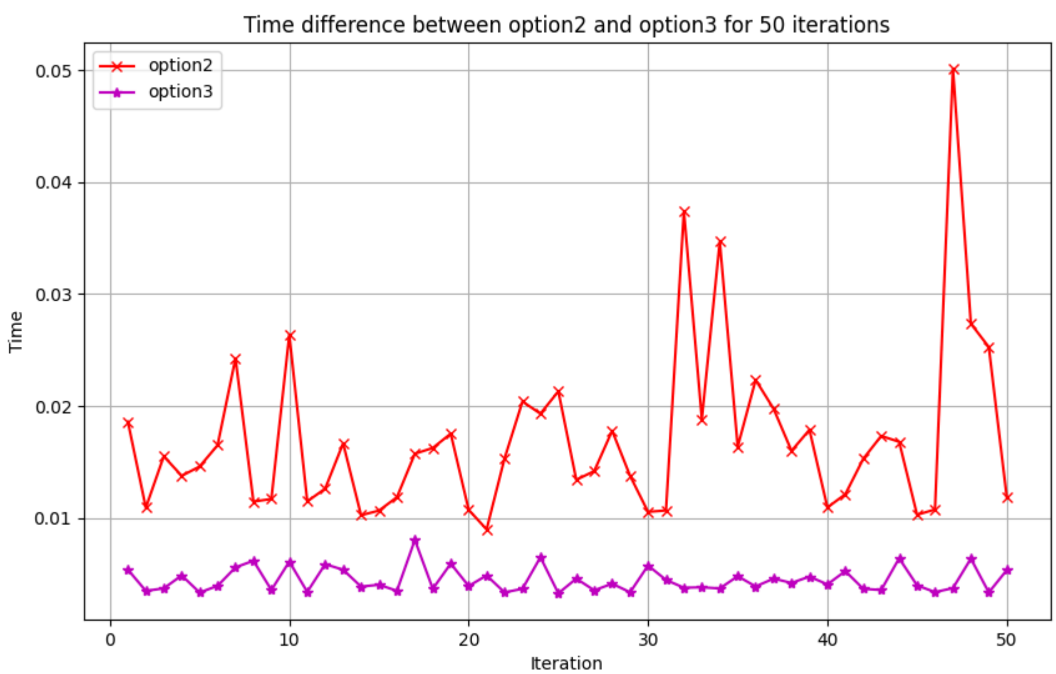
Figure 5: Hypothesis Testing Script

```python
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
# Replace 'your_file.dat' with the path to your .dat file
filename = '/mnt/c/Users/Padma/PycharmProjects/STA307/ranom/option_1_runtimes.dat'
filename2 = '/mnt/c/Users/Padma/PycharmProjects/STA307/ranom/option_2_runtimes.dat'
filename3 = '/mnt/c/Users/Padma/PycharmProjects/STA307/ranom/option_3_runtimes.dat'

# Read the .dat file into a DataFrame
# Adjust the separator if your file uses tabs or spaces ('sep='\t' or 'sep=' ')
df = pd.read_csv(filename, names=['Iteration', 'Time'], sep=' ')
df2 = pd.read_csv(filename2, names=['Iteration', 'Time'], sep=' ')
df3 = pd.read_csv(filename3, names=['Iteration', 'Time'], sep=' ')
print(df)
print(df2)
print(df3)
times_option1 = df['Time']
times_option2 = df2['Time']
times_option3 = df3['Time']
# Test for normality
_, p_value_normality_option1 = stats.shapiro(times_option1)
_, p_value_normality_option2 = stats.shapiro(times_option2)
_, p_value_normality_option3 = stats.shapiro(times_option3)
print(f"Normality test p-value for option 1: {p_value_normality_option1}")
print(f"Normality test p-value for option 2: {p_value_normality_option2}")
print(f"Normality test p-value for option 3: {p_value_normality_option3}")

# Decide which test to use based on normality
if p_value_normality_option1 > 0.05 and p_value_normality_option2 > 0.05 and p_value_normality_option3 > 0.05:
    # Use Welch's t-test by default as we do not assume equal variances
    t_stat, p_value = stats.ttest_ind(times_option1, times_option3, equal_var=False)
else:
    # Use the non-parametric Mann-Whitney U test
    print("Not all distributions are normal")
    u_stat, p_value = stats.mannwhitneyu(times_option1, times_option3)
    print(u_stat)
    print(p_value)

print(f"Method 1 average runtime: {times_option1.mean()}")
print(f"Method 2 average runtime: {times_option3.mean()}")
if p_value < 0.05:
    print("Reject the null hypothesis: there is a significant difference between the two methods.")
else:
    print("Fail to reject the null hypothesis: there is no significant difference between the two methods.")
```

Figures 6 - 8: Graphs




Time difference between option1 and option3 for 50 iterations
option1
option3
Time
Iteration
0.025
0.020
0.015
0.010
0.005
0
10
20
30
40
50


Time difference between option2 and option3 for 50 iterations
option2
option3
Time
Iteration
0.05
0.04
0.03
0.02
0.01
0
10
20
30
40
50

**Hypothesis Testing:**

The hypothesis tests we will be running are based on the assumptions from the Mann-Whitney U test:

1. All of the observations from both groups are independent of each other.
2. The responses are at least ordinal.
3. Under the null hypothesis, the distributions of both populations are identical.
4. Under the alternative hypothesis, the distributions of both populations are not identical.

**Hypothesis Testing:**

We generated a python script to test whether the population for method A and method B were normal. By utilizing the Shapiro-Wilk Test(SWT) for normality, we determined that neither of the population distributions were normal for an alpha of 0.05:

- p-value for option 1(SWT) = 0.0037256674841046333 < 0.05 we reject that the population distribution for option 1 is normal because the p-value is less than alpha.
- p-value for option 2(SWT) = 3.005404494160757e-07 < 0.05 we reject that the population distribution for option 2 is normal for the same reason above.
- p-value for option 3(SWT) = 7.0082867750898e-05 < 0.05 we reject that the population distribution for option 3 is normal for the same reason above.

Therefore, we will use the Mann-Whitney U-test(MWU) for all of our tests. This test does not assume that the population distributions for methods A and B are normal. From the

Mann-Whitney U-test assumptions, we can determine that the observations from methods A and B are independent of each other since we ran 50 experiments in succession(technically a simple random sample) for each option and the times we received from those runs had no effect on the other runs. So, the independence condition is satisfied. The responses(execution times) are at least ordinal since we can determine a clear ordering of the time values. These values were measured on a continuous scale and therefore, allows for a clear measurement of the magnitude of difference for each run and allows for the ordering of each run from fastest to slowest run time. Therefore, the ordinal condition is satisfied. We are assuming that there is no statistically significant difference between methods A and B in our null hypothesis($H_0$) meaning that the distributions for both populations are identical. We are assuming that there is a statistically significant difference between methods A and B in our alternative hypothesis($H_1$) meaning that the distributions for both populations are not identical. The notation A and B are used as a way to generalize these assumptions to all three methods since the conclusion of the SWT was the same for all three methods and these assumptions were already met for comparison of each method.

**RESULTS:**

**Option 1 and Option 2 comparison:**

$H_0$: There is no statistically significant difference between methods 1 and 2 for an alpha of 0.05.

$H_1$: There is a statistically significant difference between methods 1 and 2.

The assumptions to be able to run the MWU-test have already been satisfied. Therefore, we can perform this test. After executing the python script(Figure 5), the resulting U-statistic calculated was U = 1084.0. The sample mean for methods 1 and 2 were 0.014792190820000002 and

0.01710636834 average run time respectively. The resulting p-value was p = 0.25390022371319265. Since we chose the alpha to be 0.05, and p >= alpha, we fail to reject the null hypothesis which means that based on our calculations we found that methods 1 and 2 are statistically indistinguishable.

**Option 2 and Option 3 comparison:**

$H_0$: There is no statistically significant difference between methods 2 and 3 for an alpha of 0.05.

$H_1$: There is a statistically significant difference between methods 2 and 3.

The assumptions to be able to run the MWU-test have already been satisfied. Therefore, we can perform this test. After executing the python script(Figure 5), the resulting U-statistic calculated was U = 2500.0. The sample mean for methods 2 and 3 were 0.01710636834 and 0.00450456488 average run time respectively. The resulting p-value was p = 7.066071930388932e-18. Since we chose the alpha to be 0.05, and p < alpha, we reject the null hypothesis which means that based on our calculations, we found that methods 2 and 3 are statistically distinguishable. Therefore, we determine that method 3 is better than method 2 in terms of run-time.

**Option 1 and Option 3 comparison:**

$H_0$: There is no statistically significant difference between methods 1 and 3 for an alpha of 0.05.

$H_1$: There is a statistically significant difference between methods 1 and 3.

The assumptions to be able to run the MWU-test have already been satisfied. Therefore, we can perform this test. After executing the python script(Figure 5), the resulting U-statistic calculated was U = 2500.0. The sample mean for methods 1 and 3 were 0.014792190820000002 and 0.00450456488 average run time respectively. The resulting p-value was p =

7.066071930388932e-18. Since we chose the alpha to be 0.05, and p < alpha, we reject the null hypothesis which means that based on our calculations, we found that methods 1 and 3 are statistically distinguishable. Therefore, we determine that method 3 is better than method 1 in terms of run-time.