ECE 322
Lab Report 2

Arun Woosaree
XXXXXXX

October 8, 2019

1 Introduction

The purpose of this lab was to serve as a practical introduction to some more black box testing techniques. In this lab, the testing methods introduced were the Extreme Point Combination (EPC), and the Weak n x 1 strategy methods. We tested two programs written in Java using both of these testing strategies. The first application, named Drone takes in three arguments, and outputs either 'Success!', 'Failure', or an error message based on whether the arguments are integers ≥ 0, and their sum is less than $k = 100$. The second program, named RemoteCar takes in two arguments which represent points on a Cartesian plane, and outputs either 'Ok', 'Out of range', or an error message based on whether the point is on a circle of radius 1 about the origin. The idea for EPC testing is to identify the input domain limits, and produce all possible combinations of inputs with each of the input variables taking on a minimum value, slightly under minimum, a maximum value, and slightly over maximum value. One additional test case is added somewhere within the valid subdomain to generate a total of $4^n + 1$ test cases, where n is the dimension of inputs, or put more simply, the number of input variables. With the weak n x 1 strategy, we attempt to find linear boundaries of the problem using domain analysis. n points are selected on each linear boundary, where n is the number of input variables, and one additional point is chosen outside of each boundary. An additional point within the boundary is chosen if the boundary is open, and if the boundary is instead closed an additional point outside the boundary is chosen. In the end, with the weak n x 1 strategy, $b \times (n + 1)$ test cases are generated, where b is the number of linear boundaries, and n is the dimensionality, or the number of inputs.

2 Task 1 — Drone

For task one in this lab, the Drone application was tested using both the EPC method and the weak n x 1 strategy. Drone is a command-line program written

in java, which takes three inputs, x_1, x_2, and x_3. If $x_1 + x_2 + x_3 \leq k$, where $k = 100$, the program should output 'Success', and if not, it should output 'Failure'. The three inputs must be integers ≥ 0.

2.1 Sudomain Plot

The subdomain for this problem is illustrated in the 3-Dimensional graph below:

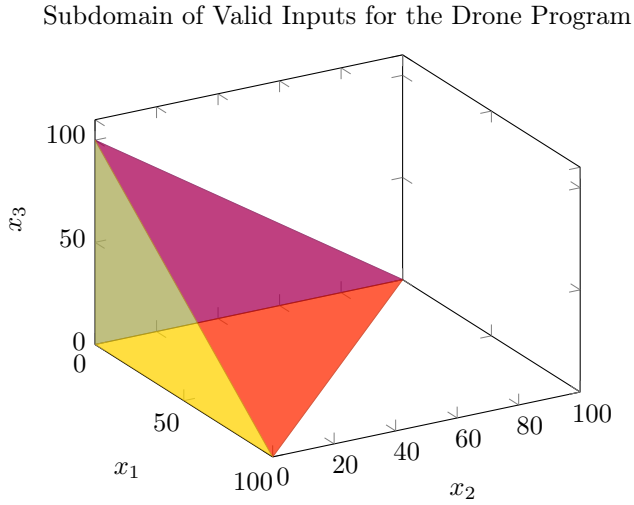

Figure 1: Valid inputs for the Drone program. $x_1 + x_2 + x_3 \leq 100 \mid x_1, x_2, x_3 \geq 0$

2.2 EPC Strategy

Using the EPC strategy, since there are 3 input variables, there are $4^3 + 1 = 65$ test cases. The values chosen were:

- max: 100
- min: 0
- slightly under min: -1
- slightly over max: 101

All the permutations of the 3 inputs taking on these values were generated and tested. An additional test case was created within the valid subdomain, which was $(x_1, x_2, x_3) = (10, 20, 30)$. The full table of test cases along with their expected inputs and outputs can be found in Appendix A of this report. The failed test cases are highlighted in red for convenience.

2.3 Weak n x 1 Strategy

Using the weak n x 1 strategy, there are 4 boundary surfaces to take into consideration. Since there are 3 inputs, we pick 3 points on each boundary surface, and one additional point outside of each boundary, for a total of $4 \times (3 + 1) + 1 = 17$ test cases. The additional test case chosen inside the boundary was $(x_1, x_2, x_3) = (30, 20, 10)$. The table of test cases along with their expected inputs and outputs can be found in Appendix B of this report. The failed test cases are highlighted in red for convenience.

2.4 Discussion

Using both the EPC and the weak n x 1 strategy, we see that for all of the failed test cases, the input variable x_2 is a negative. For negative inputs, the program is expected to give an error telling the user that negative inputs are not valid. Instead, the program outputs 'Success!' or 'Failure!' when $x_2 < 0$. Using the EPC strategy, we see 2 types of failures, where the program outputs both 'Success!' or 'Failure!'; however, with the weak n x 1 strategy we only saw the case where 'Success!' is outputted mistakenly. It should be noted however, that both testing strategies found an error which seems to stem from the same issue, (the application not checking if x_2 is negative, and the weak n x 1 strategy did so with less test cases (17 versus 65). As a result, for this problem, the weak n x 1 strategy seemed to be both more efficient and effective, since the same problem was caught using the EPC strategy but with far fewer test cases.

3 Part 2 - RemoteCar

For task two in this lab, the RemoteCar application was tested using both the EPC method and the weak n x 1 strategy. RemoteCar is a command-line program written in Java, which takes two inputs, x, and y, which represent a point on the Cartesian plane. If (x, y) falls on the circle with radius $r = 1$, centered about the origin, the program will output'Ok.', or 'Out of range!'. The input arguments must be real numbers, or the program should output an error letting the user know that the argument entered was not a number.

3.1 Subdomain Plot

Since circular boundaries are too complex to test using linear methods, we approximate the circular subdomain with m linear segments. In this lab, we carried out the testing with $m = 4$ linear boundaries to test a circular boundary of radius 1 about the origin. The accuracy of the approximation could have been increased by increasing the number of linear boundaries. For example, $m = 8$ which would make an octagon shape would be a more accurate representation of the circle, and as $m \to \infty$ we would get closer and closer to the actual boundary.

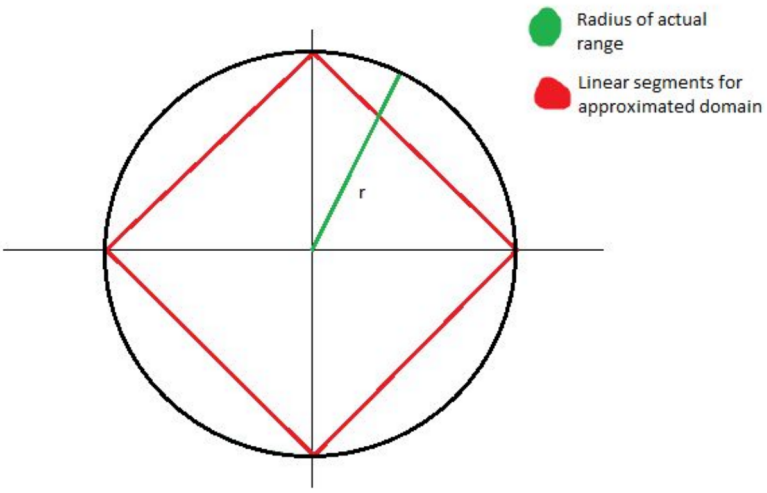

Figure 2: Approximate subdomain $m = 4$ shown in red. Circle is centered at the origin, $r = 1$. Plot taken from the lab instructions.

The equations of the approximated linear boundaries are as follows:

$$y = \begin{cases} 1 - x & 0 \leq x \leq 1 \\ 1 + x & -1 \leq x \leq 0 \\ -(1 + x) & -1 \leq x \leq 0 \\ x - 1 & 0 \leq x \leq 1 \end{cases}$$

3.2 EPC Strategy

Using the EPC strategy, since there are 2 input variables, there are $4^2 + 1 = 17$ test cases. The values chosen were:

- max: 1
- min: -1
- slightly under min: -1.1
- slightly over max: 1.1

All the permutations of the 2 inputs taking on these values were generated and tested. An additional test case was created within the valid subdomain, which was $(x, y) = (0, 0)$. The full table of test cases along with their expected inputs and outputs can be found in Appendix C of this report. Failed test cases would have been highlighted in red, however, there were no test faliures were found using this strategy.

3.3 Weak n x 1 Strategy

Using the weak n x 1 strategy, there are 4 boundary lines to take into consideration. Since there are 2 inputs, we pick 2 points on each boundary line, and one additional point outside of each boundary, for a total of $4 \times (2+1)+1 = 13$ test cases. The additional test case chosen inside the boundary was $(x, y) = (0.3, 0.7)$. The table of test cases along with their expected inputs and outputs can be found in Appendix D of this report. The failed test cases are highlighted in red for convenience.

3.4 Discussion

The domain approximation is somewhat effective. It seems to work best if test inputs chosen are far away from the boundaries, since there is a risk of a false negative test result if the chosen test input is too close to the boundary. This can happen when the test case chosen is outside of the approximated boundary, while it is still in the actual subdomain's boundary. In such a scenario, we might expect a test case to fail using the approximation, yet if the program is running correctly, the test case will pass. The configuration could have been adjusted to yield higher accuracy of the subdomain. For example, we could have increased the number of linear boundaries. Using the EPC strategy, the complexity of testing is unaltered. There are still 17 test cases, and they would all be the same. However, using the weak n x 1 strategy, the complexity would be increased. We would have to make $b \times (2+1)+1$ test cases, where b is the number of linear boundaries we choose to use for the approximation. In other words, we would need an additional 3 test cases for each boundary we add to increase the accuracy.

For this problem, it seems that the EPC testing strategy is more accurate. Using the weak n x 1 strategy, we ran into some cases where the test input was outside of our approximated boundary, yet it was inside the actual subdomain. This resulted in 4 test case failures, which were not actual errors with the program itself, but rather due to the innacurate linear approximation of the subdomain. As a result, no actual errors were identified in the application. In this case, with $m = 4$ boundaries, the number of test cases for the EPC and weak n x 1 strategies were similar. However, the weak n x 1 strategy could have a lot more test cases if we added more linear boundaries to make a better approximation of the problem's subdomain.

4 Conclusion

In this lab, we were introduced to two more black box testting strategies. The techniques learned were the Extreme Point Combination, or EPC strategy, and the weak n x 1 strategy. It would not be fair to say that one testing strategy is universally better than the other, since it would be better to match each problem at hand to the appropriate testing strategy. The weak n x 1 strategy seems to shine when the problem's subdomain is already linear in nature. This way, no

approximation needs to be made, and we get very few test cases $b \times (n+1)+1$, where b is the number of boundaries, and n is the number of input variables. The weak n x 1 strategy appears to not be very effective however, when the problem's subdomain is approximated as a collection of linear boundaries. This configuration appears to result in false negative test cases, and additionally, if more input boundaries are used to increase the accuracy of the input domain, it is still not perfect, and furthermore, $n+1$ additional test cases need to be added for each additional boundary, which can result in more test cases required to be carried out, which all still have the potential issue of resulting in a false negative result. On the other hand, the EPC strategy might seem tedious at first, generating $4^n + 1$ test cases, which might seem excessive. However, it seems to overall be more accurate than the weak n x 1 stratrgy. There were no false positive test cases, and furthermore, more failure scenarios were found, even though they are likely from the same root cause. (For the Drone program, we found using the EPC strategy 2 types of failures, where the program outputs both 'Success!' or 'Failure!' when x_2 was negative instead of an error message, however, with the weak n x 1 strategy we only saw one case where 'Success!' was outputted mistakenly. From a purely subjective standpoint, I personally prefer the EPC testing strategy over weak n x 1, since even though a lot of test cases are typically generated, this step can be automated using a script, and not lot of thought, since only four values (min, slightly under min, max, slightly over max) plus one additional valid test case needs to be thought of. Computers are fast, so we can simply generate most of the test cases without much thought, and then compare the results with ease using more automation. However, using the weak n x 1 strategy, one has to really think about each boundary, and choose n points on it, plus one point slightly outside each boundary, and additionally, one final test case. Additionally, weak n x 1 testing does not seem to work well for subdomains that are non-linear in nature, while the EPC strategy seems to overall work well and reliably.

Appendix A: Drone EPC Testing

Testid	description	x1	x2	x3	Expected	Actual
1	EPC permutation	100	100	100	Failure!	Failure!
2	EPC permutation	100	100	0	Failure!	Failure!
3	EPC permutation	100	100	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
4	EPC permutation	100	100	101	Failure!	Failure!
5	EPC permutation	100	0	100	Failure!	Failure!
6	EPC permutation	100	0	0	Success!	Success!
7	EPC permutation	100	0	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
8	EPC permutation	100	0	101	Failure!	Failure!
9	EPC permutation	100	-1	100	ERROR: Invald argument - negative value	Failure!
10	EPC permutation	100	-1	0	ERROR: Invald argument - negative value	Success!
11	EPC permutation	100	-1	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
12	EPC permutation	100	-1	101	ERROR: Invald argument - negative value	Failure!
13	EPC permutation	100	101	100	Failure!	Failure!
14	EPC permutation	100	101	0	Failure!	Failure!
15	EPC permutation	100	101	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
16	EPC permutation	100	101	101	Failure!	Failure!
17	EPC permutation	0	100	100	Failure!	Failure!
18	EPC permutation	0	100	0	Success!	Success!
19	EPC permutation	0	100	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
20	EPC permutation	0	100	101	Failure!	Failure!
21	EPC permutation	0	0	100	Success!	Success!
22	EPC permutation	0	0	0	Success!	Success!
23	EPC permutation	0	0	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
24	EPC permutation	0	0	101	Failure!	Failure!
25	EPC permutation	0	-1	100	ERROR: Invald argument - negative value	Success!
26	EPC permutation	0	-1	0	ERROR: Invald argument - negative value	Success!
27	EPC permutation	0	-1	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
28	EPC permutation	0	-1	101	ERROR: Invald argument - negative value	Success!
29	EPC permutation	0	101	100	Failure!	Failure!
30	EPC permutation	0	101	0	Failure!	Failure!
31	EPC permutation	0	101	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
32	EPC permutation	0	101	101	Failure!	Failure!
33	EPC permutation	-1	100	100	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
34	EPC permutation	-1	100	0	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
35	EPC permutation	-1	100	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
36	EPC permutation	-1	100	101	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
37	EPC permutation	-1	0	100	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
38	EPC permutation	-1	0	0	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
39	EPC permutation	-1	0	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
40	EPC permutation	-1	0	101	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
41	EPC permutation	-1	-1	100	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
42	EPC permutation	-1	-1	0	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
43	EPC permutation	-1	-1	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
44	EPC permutation	-1	-1	101	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
45	EPC permutation	-1	101	100	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
46	EPC permutation	-1	101	0	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value

Appendix A: Drone EPC Testing

47	EPC permutation	-1	101	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
48	EPC permutation	-1	101	101	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
49	EPC permutation	101	100	100	Failure!	Failure!
50	EPC permutation	101	100	0	Failure!	Failure!
51	EPC permutation	101	100	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
52	EPC permutation	101	100	101	Failure!	Failure!
53	EPC permutation	101	0	100	Failure!	Failure!
54	EPC permutation	101	0	0	Failure!	Failure!
55	EPC permutation	101	0	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
56	EPC permutation	101	0	101	Failure!	Failure!
57	EPC permutation	101	-1	100	ERROR: Invald argument - negative value	Failure!
58	EPC permutation	101	-1	0	ERROR: Invald argument - negative value	Success!
59	EPC permutation	101	-1	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
60	EPC permutation	101	-1	101	ERROR: Invald argument - negative value	Failure!
61	EPC permutation	101	101	100	Failure!	Failure!
62	EPC permutation	101	101	0	Failure!	Failure!
63	EPC permutation	101	101	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
64	EPC permutation	101	101	101	Failure!	Failure!
65	one valid test case	10	20	30	Success!	Success!

Appendix B: Drone Weak n x 1 Strategy

Testid	description	x1	x2	x3	Expected	Actual
1	x1+x2+x3=100	10	40	50	Success!	Success!
2	x1+x2+x3=100	35	45	20	Success!	Success!
3	x1+x2+x3=100	69	20	11	Success!	Success!
4	just outside of x1+x2+x3 = 100 boundary	33	34	34	Failure!	Failure!
5	x1=0, x2 + x3 < 100	0	20	30	Success!	Success!
6	x1=0, x2 + x3 < 100	0	50	20	Success!	Success!
7	x1=0, x2 + x3 < 100	0	21	36	Success!	Success!
8	just outside of x1=0 boundary	-1	25	62	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
9	x2=0, x2 + x3 < 100	32	0	16	Success!	Success!
10	x2=0, x2 + x3 < 100	48	0	25	Success!	Success!
11	x2=0, x2 + x3 < 100	42	0	14	Success!	Success!
12	just outside of x2=0 boundary	21	-1	42	ERROR: Invald argument - negative value	Success!
13	x3=0, x2 + x3 < 100	6	69	0	Success!	Success!
14	x3=0, x2 + x3 < 100	72	5	0	Success!	Success!
15	x3=0, x2 + x3 < 100	65	12	0	Success!	Success!
16	just outside of x3=0 boundary	45	7	-1	ERROR: Invald argument - negative value	ERROR: Invald argument - negative value
17	additional test case inside the boundary	10	20	30	Success!	Success!

Appendix C: RemoteCar EPC Testing

Testid	description	x	y	Expected	Actual
1	EPC permuation	-1.1	-1.1	Out of range!	Out of range!
2	EPC permuation	-1.1	-1	Out of range!	Out of range!
3	EPC permuation	-1.1	1	Out of range!	Out of range!
4	EPC permuation	-1.1	1.1	Out of range!	Out of range!
5	EPC permuation	-1	-1.1	Out of range!	Out of range!
6	EPC permuation	-1	-1	Out of range!	Out of range!
7	EPC permuation	-1	1	Out of range!	Out of range!
8	EPC permuation	-1	1.1	Out of range!	Out of range!
9	EPC permuation	1	-1.1	Out of range!	Out of range!
10	EPC permuation	1	-1	Out of range!	Out of range!
11	EPC permuation	1	1	Out of range!	Out of range!
12	EPC permuation	1	1.1	Out of range!	Out of range!
13	EPC permuation	1.1	-1.1	Out of range!	Out of range!
14	EPC permuation	1.1	-1	Out of range!	Out of range!
15	EPC permuation	1.1	1	Out of range!	Out of range!
16	EPC permuation	1.1	1.1	Out of range!	Out of range!
17	one valid test case	0	0	Ok.	Ok.

Appendix D: RemoteCar Weak n x 1 Strategy

Testid	description	x	y	Expected	Actual
1	y=1-x boundary	0.9	0.1	Ok.	Ok.
2	y=1-x boundary	0.1	0.9	Ok.	Ok.
3	just outside of y=1-x boundary	0.5	0.6	Out of range!	Ok.
4	y=1+x boundary	-0.1	0.9	Ok.	Ok.
5	y=1+x boundary	-0.9	0.1	Ok.	Ok.
6	just outside of y=1+x boundary	-0.5	0.6	Out of range!	Ok.
7	y=-(1+x) boundary	-0.9	-0.1	Ok.	Ok.
8	y=-(1+x) boundary	-0.1	-0.9	Ok.	Ok.
9	just outside of y=-(1+x) boundary	-0.5	-0.6	Out of range!	Ok.
10	y=x-1 boundary	0.1	-0.9	Ok.	Ok.
11	y=x-1 boundary	0.9	-0.1	Ok.	Ok.
12	just outside of y=x-1 boundary	0.5	-0.6	Out of range!	Ok.
13	additional test case inside the boundary	0.3	0.7	Ok.	Ok.