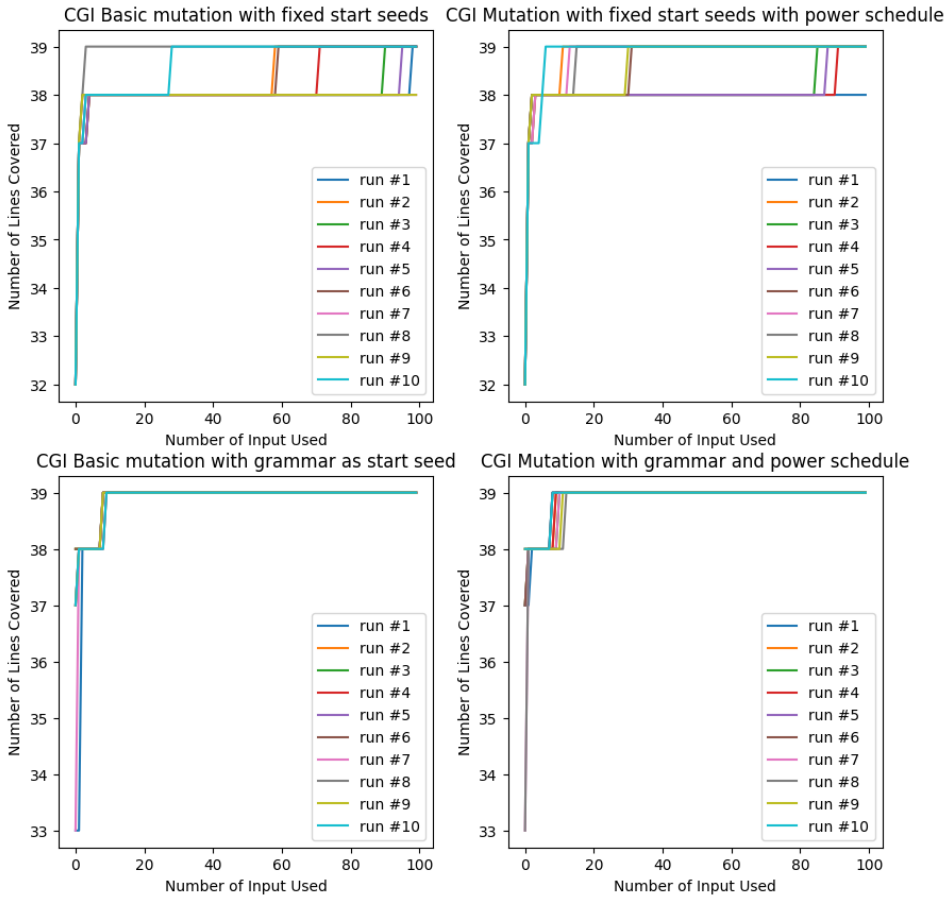
LOG 6305 – Techniques avancées de test du logiciel
Assignment 2

Question 1

The goal of the *cgi_decode.py* module is to transform escaped characters from their hexadecimal value or + to the corresponding character.
To test it, I defined a grammar with 4 sections in a row, with a section being either a +, an escaped character, or a random string.
For the Fuzzer, I added two mutators, one that inserts +, and one that inserts an escaped character.
For the power schedule, I defined the energy of a seed by two times the normalized coverage subtracted by the normalized execution time.
I normalized before doing the sum so the coverage and execution time can be on the same scale.
In the fuzzer, I decided to remove a seed if it created an exception, this was an arbitrary choice because a syntax invalid input that would be mutated will certainly throw the same error. And the mutation operators are enoughly rough, that a lot of invalid input are still created, even with the removal of invalid seeds.
For the start seeds for the non-grammar part, I used two seeds, one being a random string, and another with an escaped character inside it.

This was the base that was used on the others fuzzer as well.



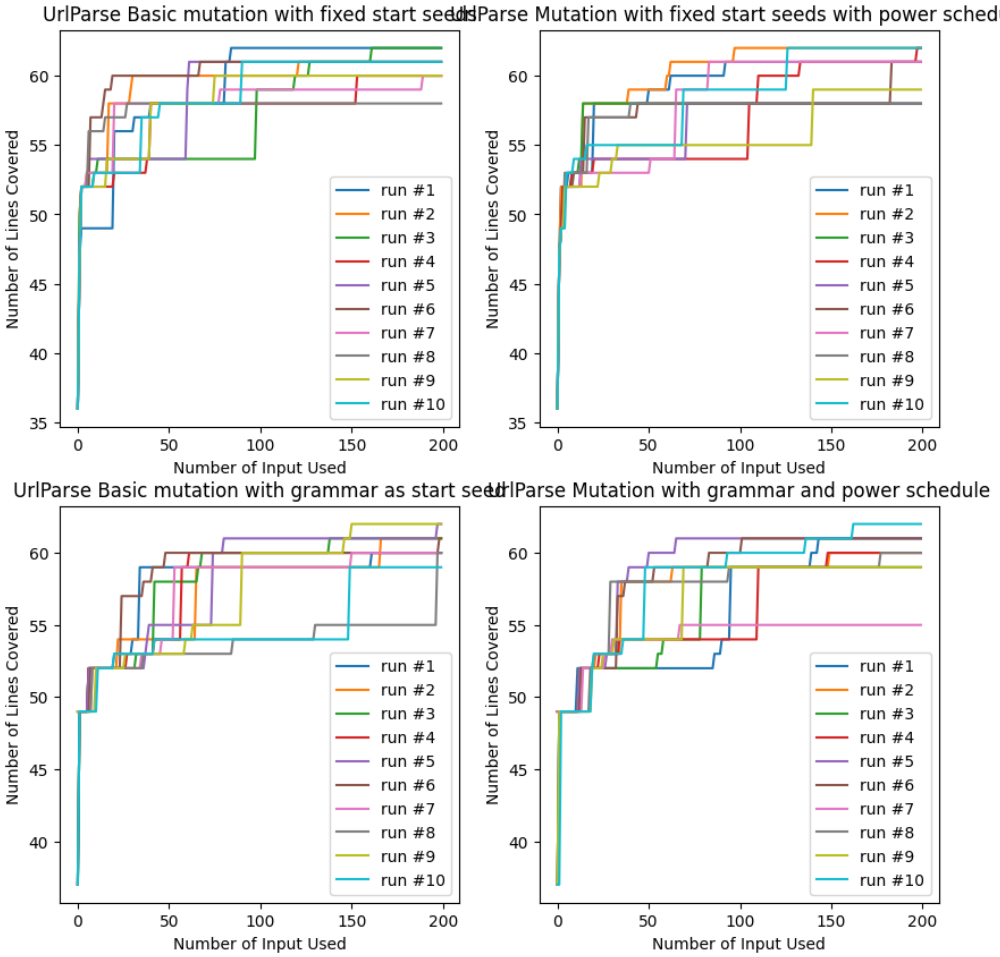
This was the result with a budget of 100, we see that all of the versions seem to converge to the same covered lines pretty quickly, but we see that the ones with grammar converged faster. Because we don't know the total number of lines, we can't know how much of the module is covered. I tried to add code in the executor to get the total number of lines in the module, but It got 200% coverage, and I wasn't able to find a logical number of total lines.

Question 2

For the urlparse module, I defined a grammar to represent urls, there is not much to add on it.

For the fuzzer, I couldn't think of new mutators, because I didn't see what I could possibly add more than deleting random character, inserting a random character and replacing a random character.

For the power schedule, I kept the version of the first question.



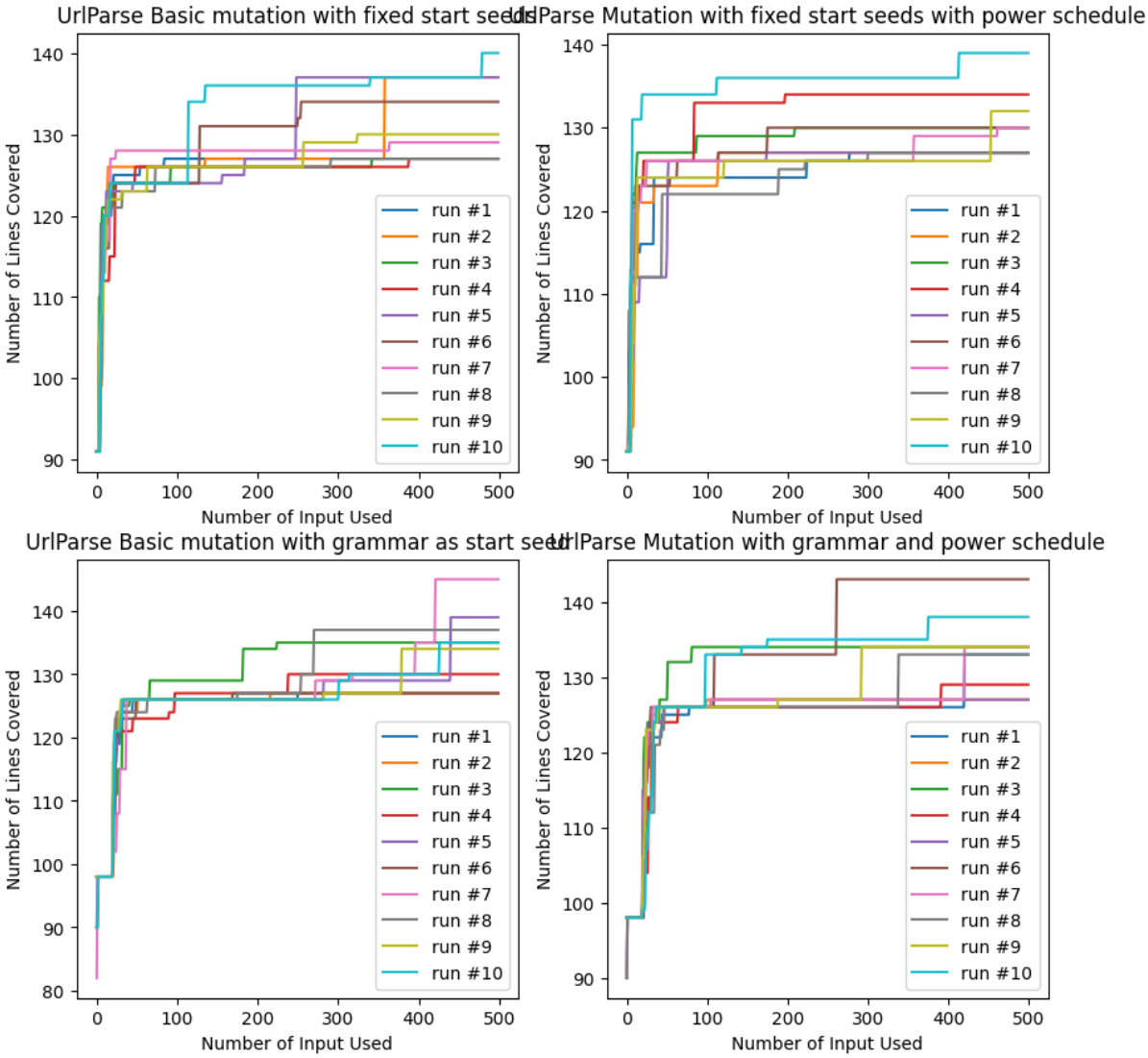
For the comparison of the different versions, almost all of them converge to the same value-ish. But we see that the grammar version performs worse in the beginning, that is strange because the grammar creates inputs that are more broad that the fixed seeds that I chose.

Question 3

For the html parsing, I tried to do a really exhaustive grammar, but I couldn't reasonably cover all of the HTML5 specification, but that would be what would yield the better results, analyzing the specification, and creating a grammar that could replicate the use of every feature. To be able to have <a> and other tags in HTML that uses <> that are misrepresented by the grammar engine, I added a feature that at the end of the grammar process, it changes [in < and] in >, so I was able to generate grammar with <> tags.

For the power schedule I didn't change a thing from the previous ones.

For the fuzzer part, I added 10 new mutators, based on my memories of HTML : *insert_div*, *insert_a*, *insert_ul*, *insert_ol*, *insert_image*, *insert_nav*, *insert_link*, *insert_table*, *insert_button*, *insert_script*.
That would insert in a random part of the HTML a div, or an <a> etc...



With this one we see that the exhaustive grammar paid off in the long run, because the grammar with and without power schedule achieved the best coverage.

Question 4

I chose for the cgi one the grammar with the power schedule, for the second one I chose the grammar one without the power schedule, and for the third one I chose the grammar with power schedule.
I chose them because they were the most performant between the four

models. For the url parser, the power schedule that I implemented using coverage and time of execution was not helpful to find seeds that were more efficient. But for the other two ones, it helped choosing better seeds.

So the results are :

Module	Fuzzer	Max Coverage
cgi_decode	Grammar and Power Schedule	39
urlparse	Grammar	62
HTMLParser	Grammar and Power Schedule	145