Project 1 – Nature-Inspired Algorithms

Willi Gierke, Lawrence Benson, Nico Ring

26th May 2017

1 Algorithm Runtimes

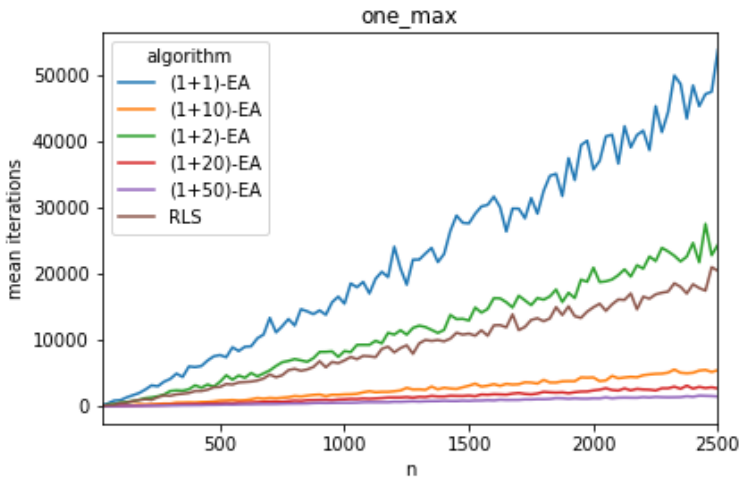

Figure 1: Mean iterations for the different algorithms with *OneMax*.

In Figure 1 we can see the approximate runtime for the different algorithms, i.e. RLS, 1+1-EA, etc., on *OneMax*. In the next section we can see why it is hard to compare runtimes with all value functions, i.e., *OneMax*, *Jump*, etc., because the runtimes highly depend on the value function. *OneMax* and *BinVal* are the best value functions, so we use *OneMax* to show the runtime of all algorithms in the sense of 'this is as good as the algorithm can get'. We can see that the $1 + \lambda$-EAs with $\lambda \geq 10$ have a linear runtime in $\mathcal{O}(n)$, as $n = 2500 \rightarrow |iterations| \approx 2500$. 1+2-EA and RLS show a $\mathcal{O}(n * log(n))$ runtime, with $n = 2500 \rightarrow |iterations| \approx 20000 \approx 2500 * ln(2500)$. 1+1-EA seems to also run in $\mathcal{O}(n * log(n))$, possibly with a constant factor of 2 compared to RLS. As mentioned above, these are the 'best' runtimes the algorithms can achieve. The following runtimes are calculated analogously.

Figure 2 shows us the runtimes of each algorithm with each value function except *Jump*, as this performs very badly and makes the plots useless. We use the **jump** plot from Figure 3 to approximate the runtimes with *Jump*.

Table 1: Algorithm runtimes

Function	RLS	(1+1)-EA	(1+2)-EA	(1+10)-EA	(1+20)-EA	(1+50)-EA
OneMax	$\mathcal{O}(n * log(n))$	$\mathcal{O}(n * log(n))$	$\mathcal{O}(n * log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
BinVal	$\mathcal{O}(n * log(n))$	$\mathcal{O}(n * log(n))$	$\mathcal{O}(n * log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
LeadingOnes	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n * log(n))$	$\mathcal{O}(n * log(n))$	$\mathcal{O}(n * log(n))$
Jump	$\mathcal{O}(2^n)$	$\mathcal{O}(2^n)$	$\mathcal{O}(2^n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
RoyalRoads	$\mathcal{O}(n * log(n))$	$\mathcal{O}(n * log(n))$	$\mathcal{O}(n * log(n))$	$\mathcal{O}(n * log(n))$	$\mathcal{O}(n * log(n))$	$\mathcal{O}(n * log(n))$

As we can tell from Figure 2, 1+1-EA needs roughly twice as many iterations as 1+2-EA to find the optimal solution. This intuitively makes sense as 1+1-EA needs two times as many iterations to generate the same amount of offsprings as 1+2-EA. This patterns also holds for the remaining $1 + \lambda$-EAs since 1+10 EA needs twice as many iterations as 1+20 EA which in turn needs 2.5 times as many as 1+50 EA to generate the offspring with the optimal fitness. The probability that the offspring generated in one iteration by 1+1-EA is

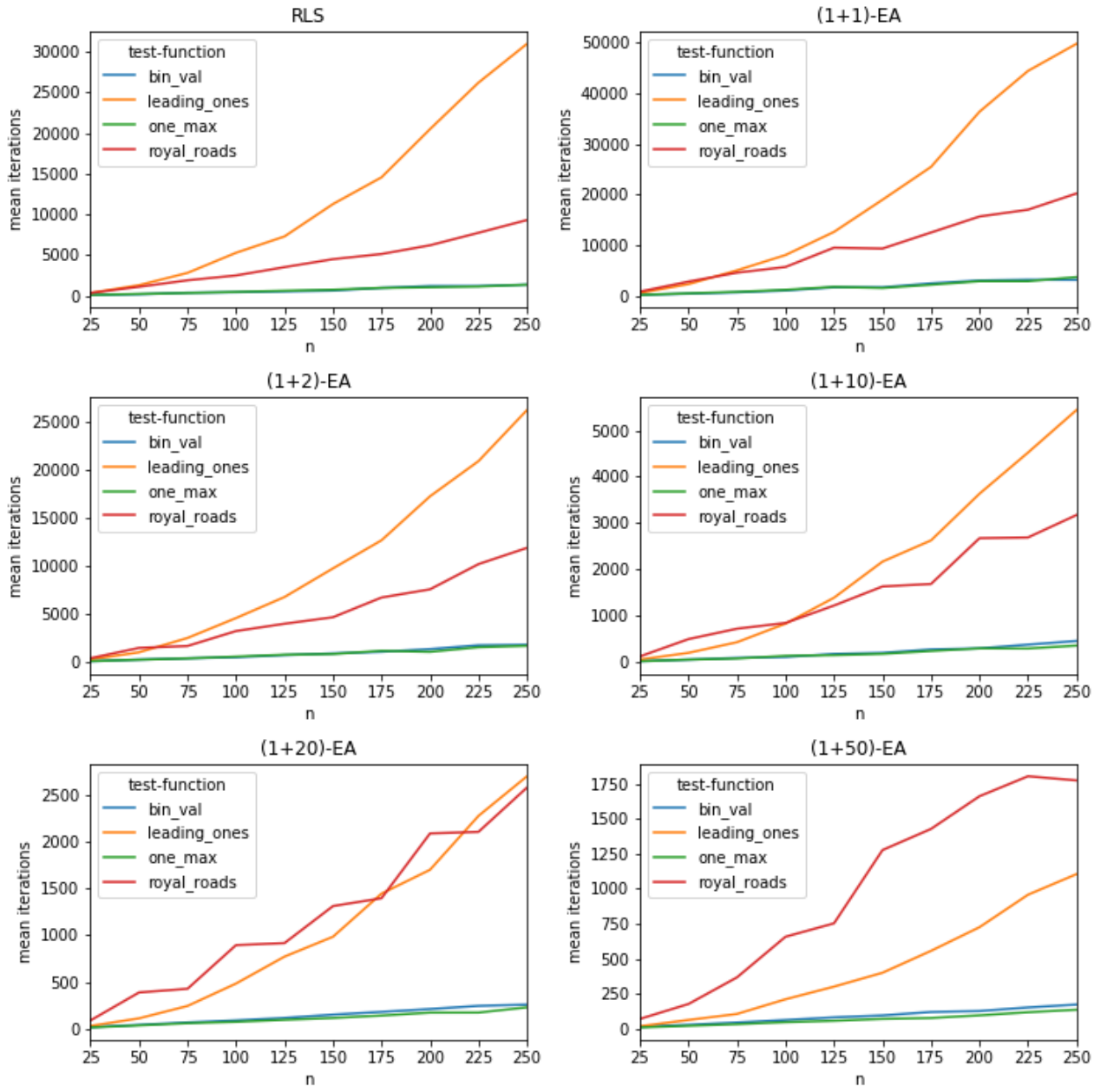

Figure 2: Mean iterations for the different algorithms with value functions.

the same as the parent (so no bits were flipped) is roughly $B(n, \frac{1}{n}) = (1 - \frac{1}{n})^{n-1} \geq \frac{1}{e} \approx 0.37$. Thus, 1+1-EA takes over 37 % longer to find the optimal solution than RLS since RLS always flips one bit while in 37 % of the iterations 1+1-EA does not generate a different offspring.

2 Value Functions

In Figure 3 we can see that each algorithm, i.e., RLS, 1+1-EA, etc., performs equally relative to the other functions. The 1+λ-EA functions with a high λ always perform best, then RLS and 1+2-EA. The worst performance is always by 1+1-EA. We can also see that *OneMax* and *BinVal* perfom best with a maximum of 3000 iterations. *LeadingOnes* and *RoyalRoads* are in the same order of magnitude with 20000 and 50000 iterations. *Jump* is by far the worst value funcion with up to 3000000 iterations. Explainations for this are found below.

Figure 4 shows us how the number of 1s relates to the number of iterations for a certain value function. For *OneMax* and *BinVal* the number of 1s increases rapidly and then takes roughly three quarters of the time to reach the optimum with the last few bits flipped. These algortithms can never get worse after a certain value is reached, e.g. after 47 1s, it can never go back to 46 because the value would be strictly less.

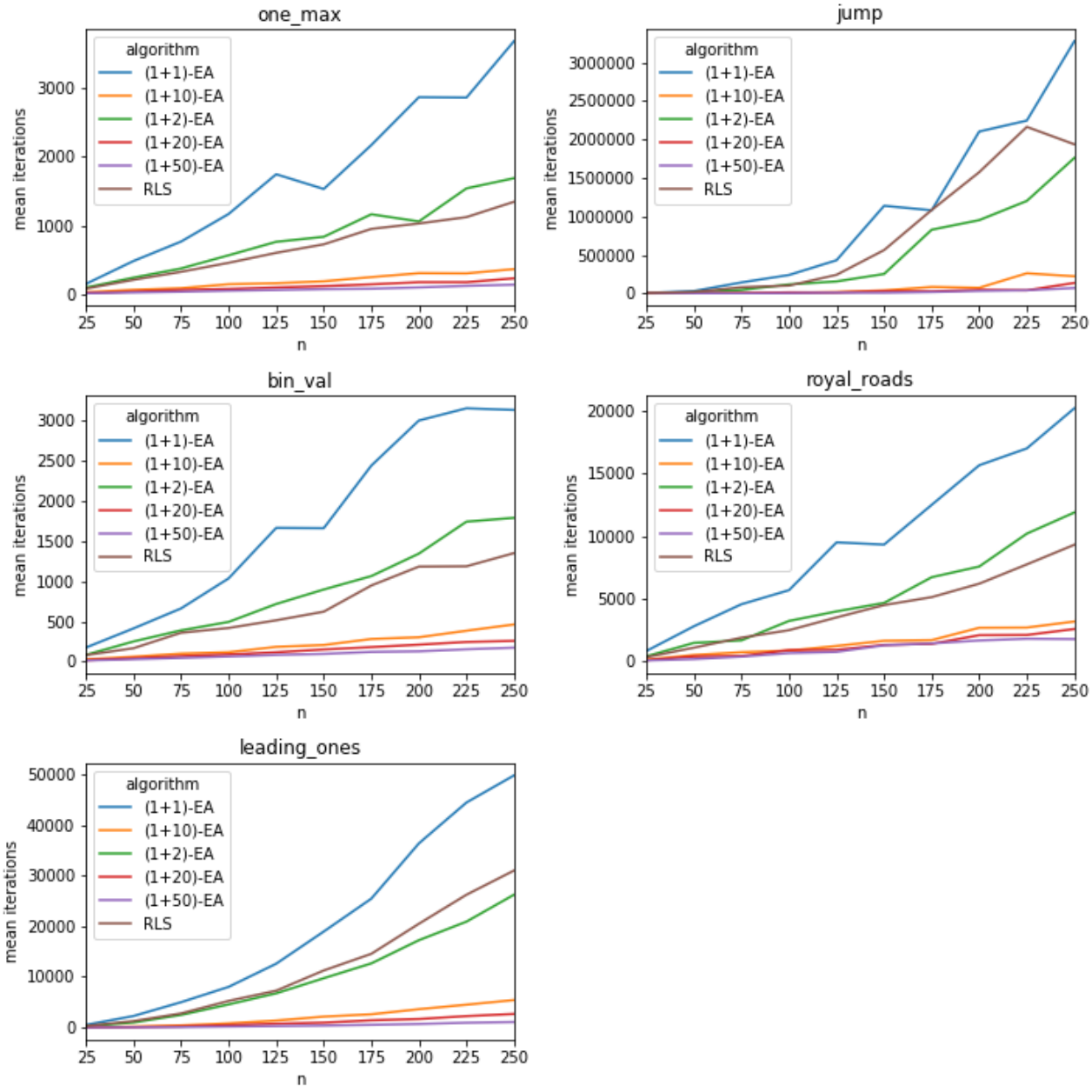

Figure 3: Mean iterations for the different test functions of each algorithm.

Jump behaves like *OneMax* until $n-k$ so after about 100 iterations it should reach $n-k$ 1s. However, it takes an enormous amount of time, i.e. 120000 iterations, to flip the last 3 bits correctly. This is due to the fact that after $n-k$ bits are set to 1, each additional 1 does not increase the value and thus allows the algorithm to flip a 1 back to 0 without decreasing the value, e.g., 48 1s has a value of 47 and a then flipped 1 with 47 1s also has a value of 47. This back and forth takes a long time. This behaviour increases with a higher k.
RoyalRoads takes 500-1000 iterations to reach a close to optimal number of 1s and then also roughly three quarters of the time to reach the optimum. Only that the number of iterations is order 10 larger than *OneMax*, i.e., 3000 instead of 300. It is 'harder' to get a valid road, but once that road is valid it can never be taken be removed again because that would decrease the value. It can only be lost if a new road is created, thus creating an equivalent population.
LeadingOnes is quite scattered because every 1 after the first 0 is irrelevant to the value of the population. However, it cannot get worse after a certain value is reached. It is the only function where there is no direct correlation between number of 1s and the value which makes it harder for the algorithm to optimise.

Figure 5 shows us what the value can be for any given number of 1s in a string of length 10. *OneMax* is linear so that each 1 corresponds to +1 in the value. *Jump* behaves similar until $n-3$, which then turns into a plateau, ending in a large jump of 3 bits. This shows the problem of trying to find the optimum. *BinVal*

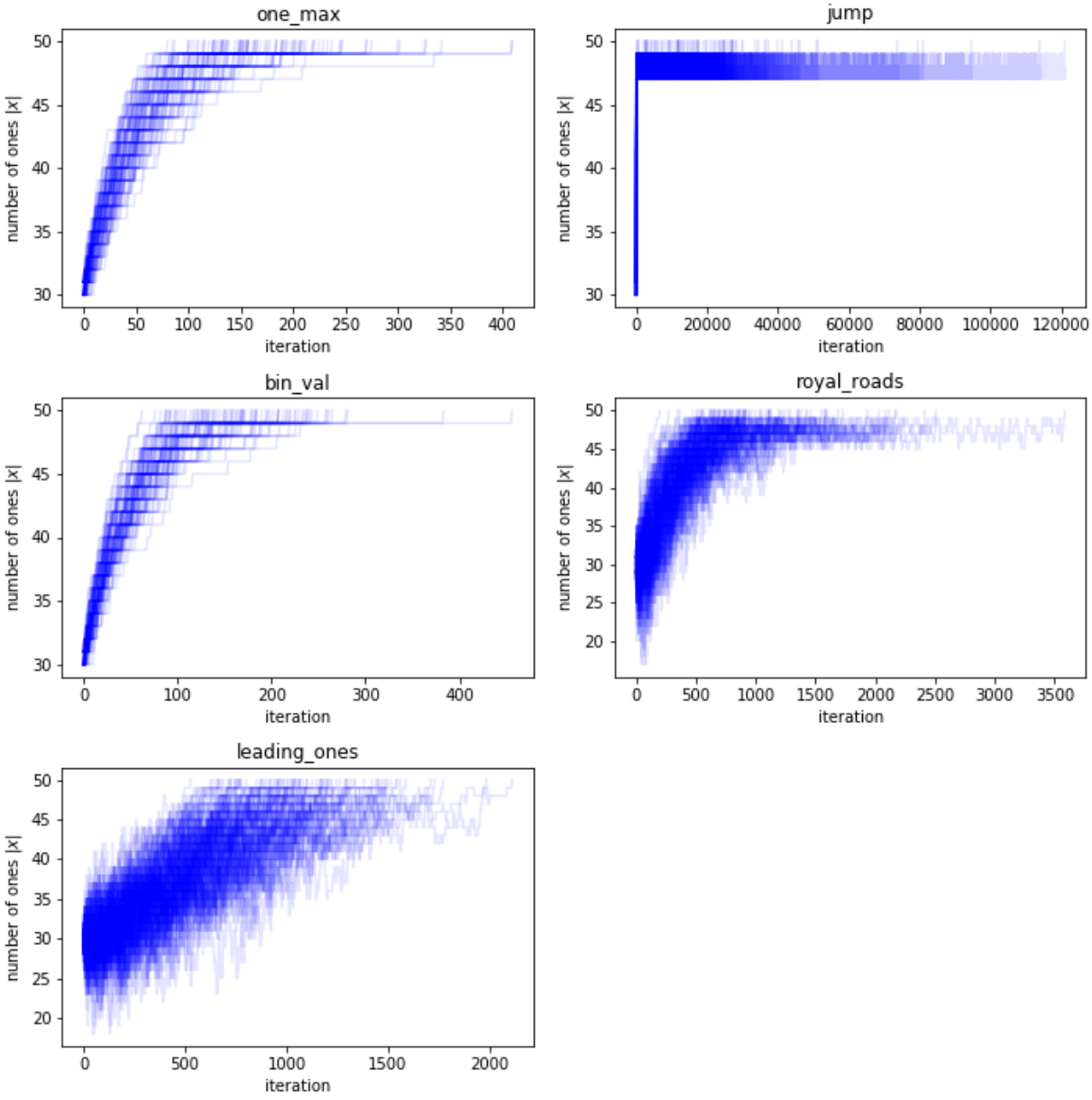

Figure 4: Number of ones in x after each iteration for *RLS* for each test function (100 runs each and $n = 50$)

is unique in that each instance has a unique value in the interval $[0, 2^n]$. However, this also means that each additional 1 always brings an increase in the value. *RoyalRoads* shows us that for 0 to 4 1s in the string, the value can only be 0 and from 5 on we can get value 1 but don't have to. From value 9 on, according to the pigeonhole principle, the value must be at least 1. *LeadingOnes* shows how this can be a good metric but also a bad one. In the best case, the number of 1s directly relates to the value as in *OneMax*, but in the worst case if the first value is a 0, even 9 1s still return a value 0.

3 Strictly greater than

In Figure 6 we show the runtime of each algorithm with the difference that we only select a mutation if it's value is strictly greater than the current population. For *OneMax*, *BinVal* and *LeadingOnes* this makes little difference in the number of iterations, because the functions are not dependant on 'jumps', i.e., with each change, the value can increase. *Jump* and *RoyalRoads* both rely on these 'jumps'. Intuitively this means that for *OneMax* (and the others) we can always get a better value with one bit flipped. *Jump* (analogous *RoyalRoads*), in our case, requires 3 bits flipped without any value increase. This makes it impossible for e.g. RLS to terminate, as it cannot flip 3 bits in one iteration. The other algorithms could theoretically achieve this but it would require

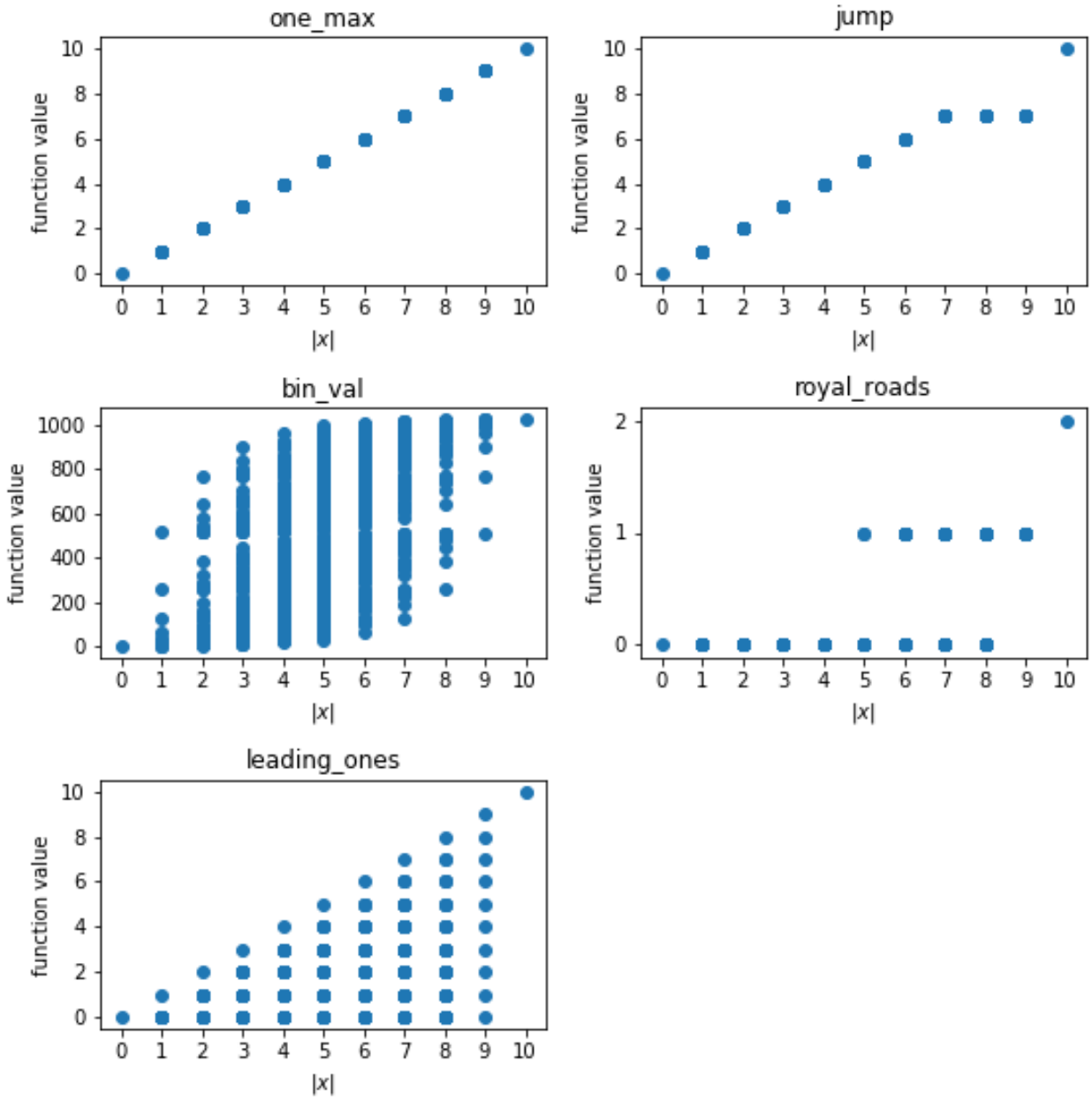

Figure 5: Possible values of the test functions for all possible instances of size 10.

a mutation to flip exactly the missing 3 bits and no other ones. This renders the evaluation unfeasable, which is why we only evaluated the non-'jump' value functions.

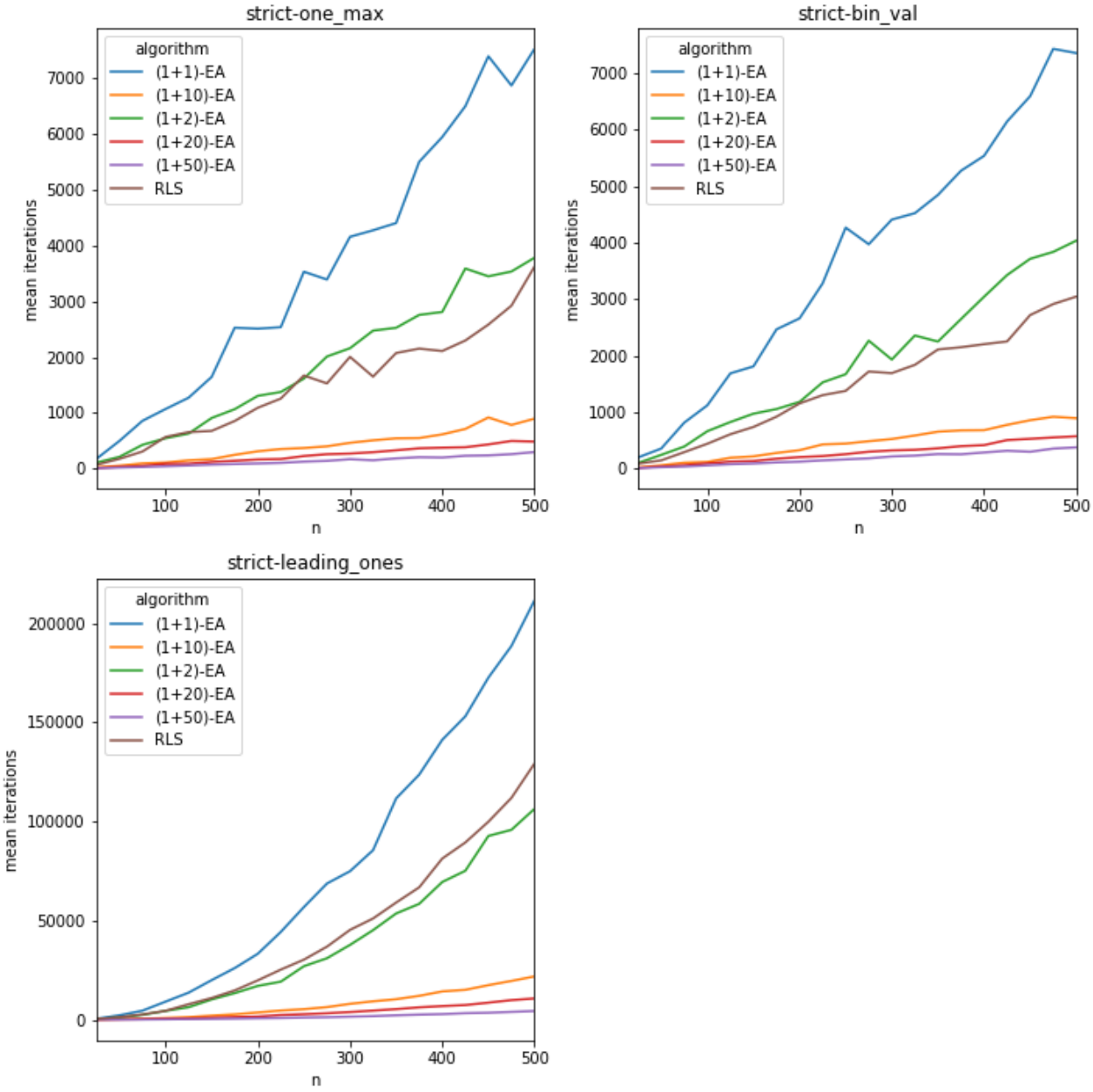

Figure 6: Mean iterations for the different test functions of each algorithm with strict greater than.