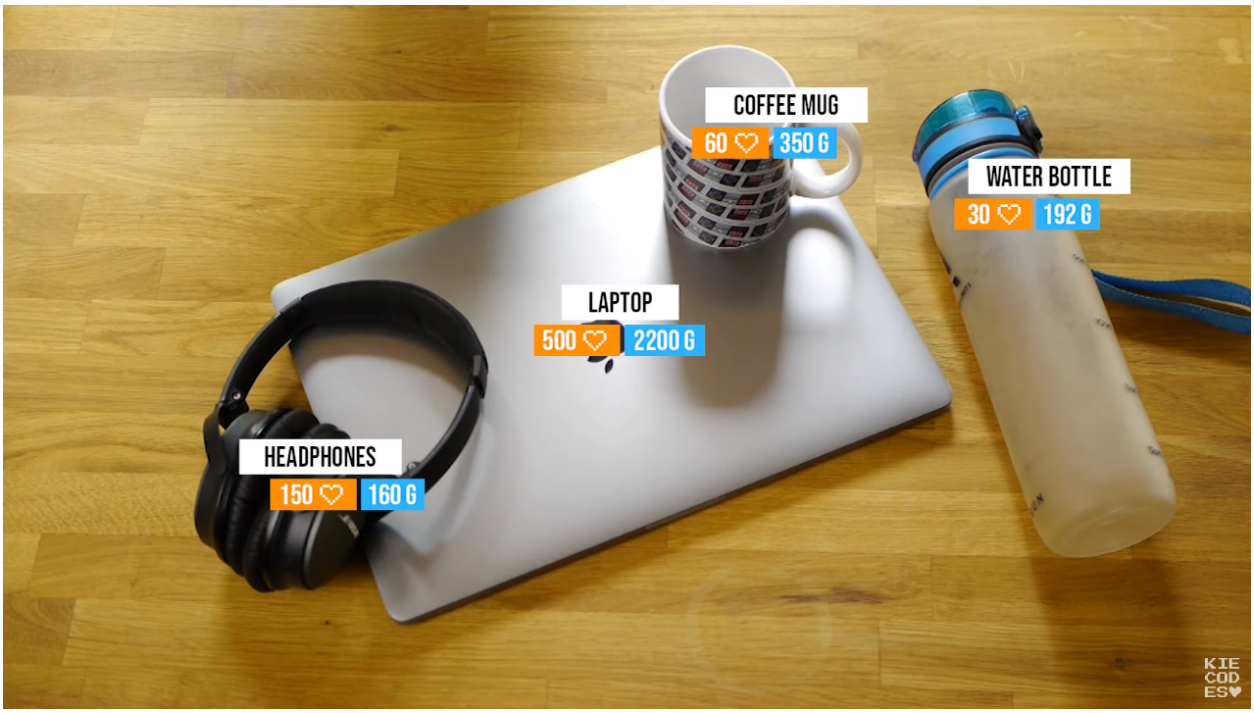
Genetic Algorithm

You can simulate evolution in your computer and by doing that it is possible to solve hard problems.

Let’s say you want to go on a trip and you can bring only 3kg, and you want to make the best of it. We have to find then the best combination:

The solution is:

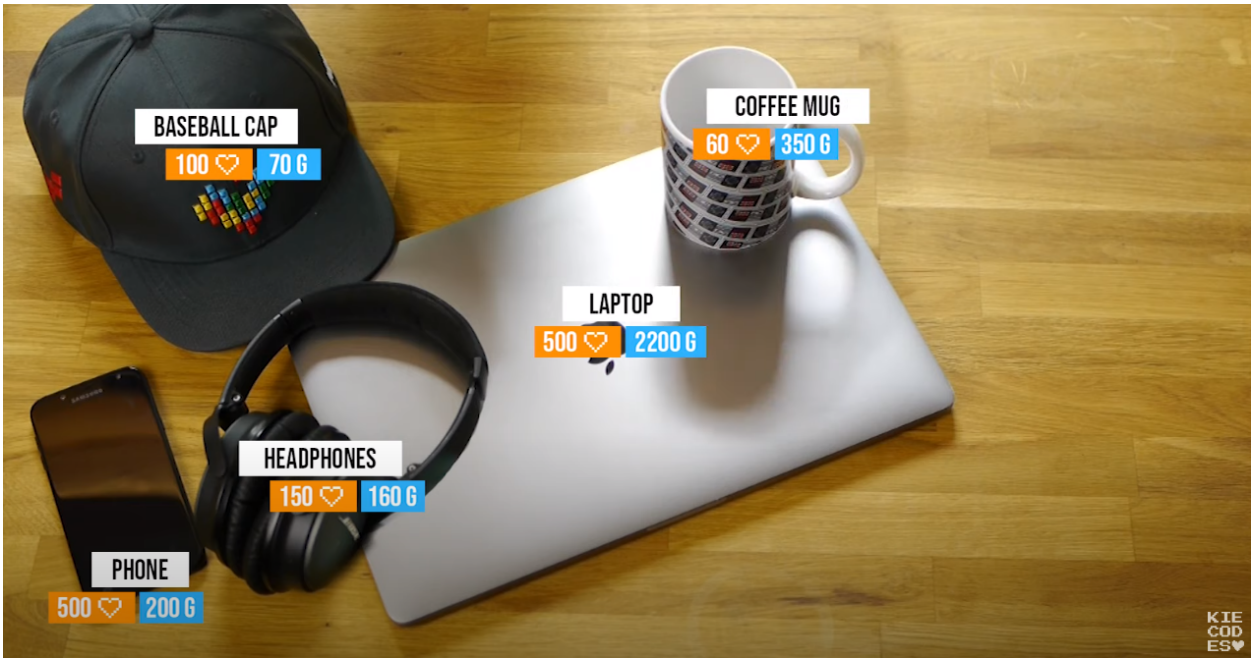


Our brain is really good at finding solution with small items. And computer are good as well, with the naive implementation we have

Items	Combination	Seconds
5	32	3 microseconds

Let’s complicate a bit the things:

To optimize the space, the better solution is:



With 20 items start to take 1 seconds and more than 1 million combinations. More items we have, more times we need.

We need to find a more intelligent and better solution. The time raises with the number of items: this is the **knapsack problem.**

Generic Algorithm

Can be used to generate solutions for problems that have no way to calculate the solution. This is what we need. Are part of the **Evolutionary Coding** that is a natural selection to approximate solutions for a given problem.

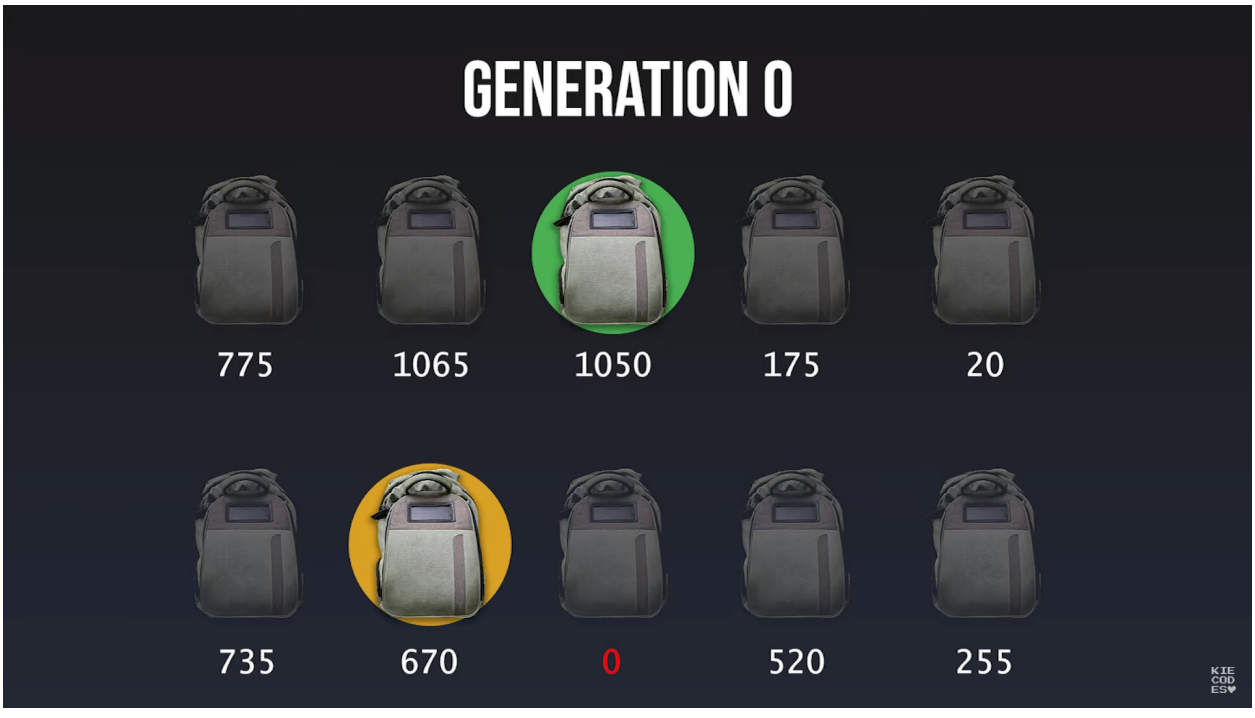
How they work?

Generic algorithm uses a population of possible solutions, in our case they would be the items we have to bring. Each item will have an encode (`0` or `1`, where `0` means outside the backpack, `1` means inside the backpack).

The set of all solutions, that contains the possibilities, is called **generation. The Generation 0** is random and is the initial solution.

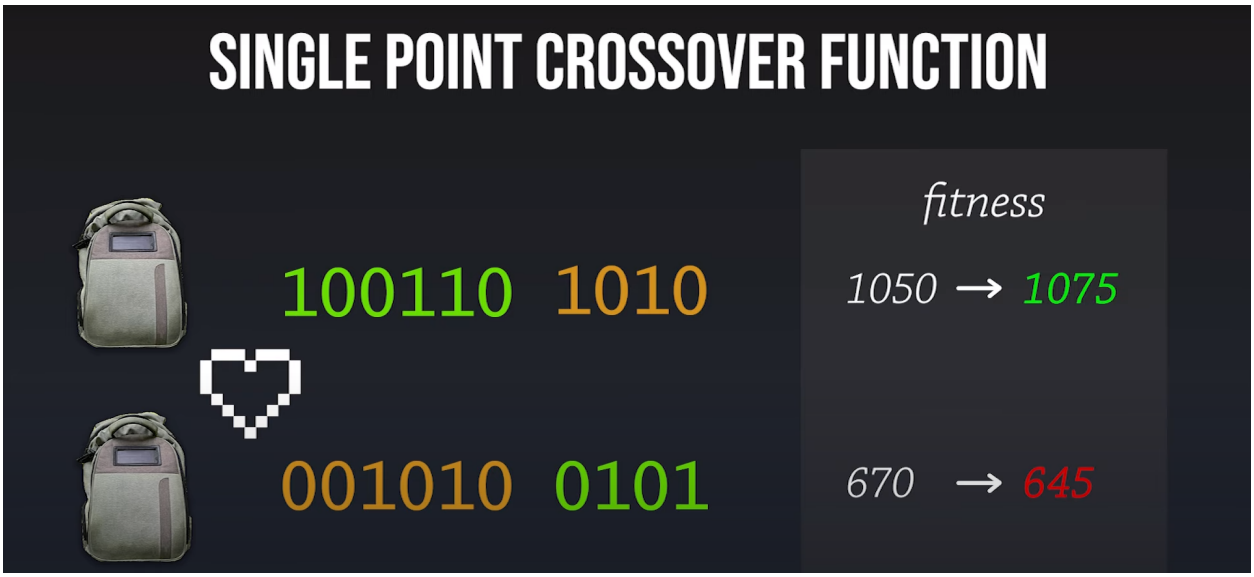
We have to start now the **Natural Selection** part. We use a `fitness` function that tells us how good the solution is. In our case it is the weight, and if the weight is exceeded it returns `0`.

A solution with the highest fitness score is more probably selected to reproduction instead of a lower one.



At this point we take the highest and the lower and we call the **parents**.

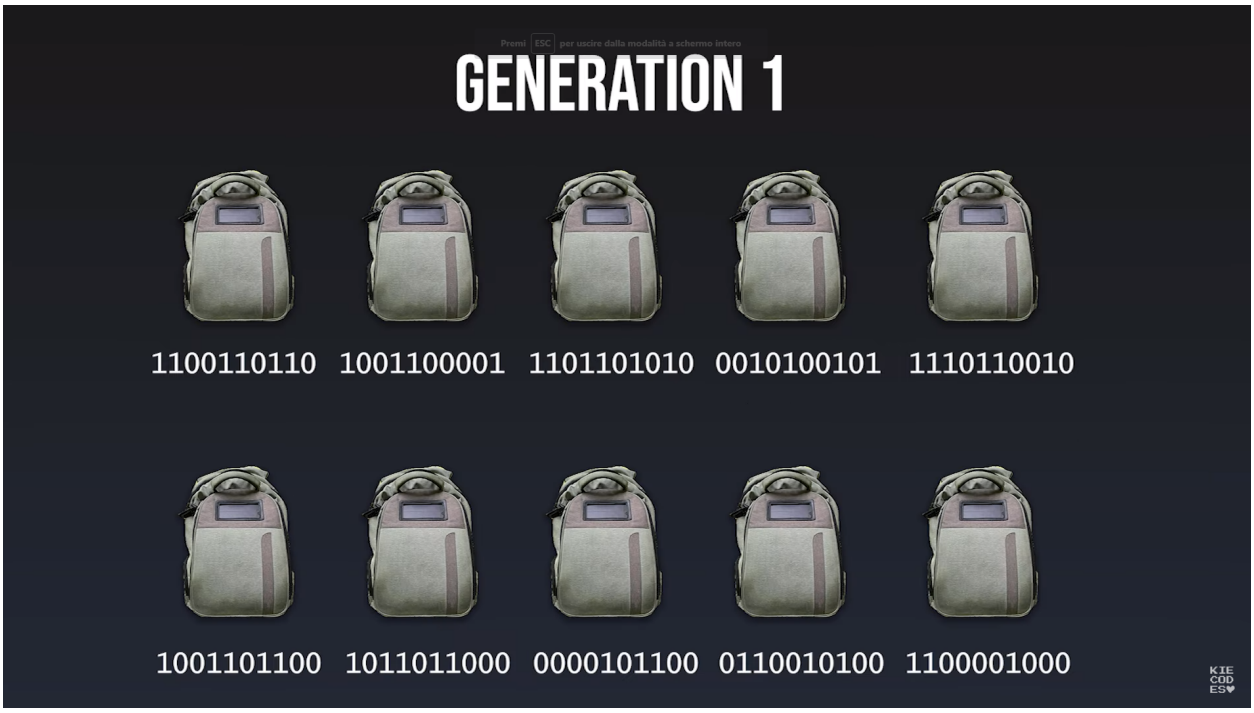
We select the **parents** and cut their genes randomly an switch them, at this point we create two new solutions. This is called **Single Point Crossover Function**



We repeat the process as long as we have as long as we don’t have enough solutions in the new generation. By crossing two solution, we may get a better one.

But there is one thing: there is no guarantee that the crossover function DOES NOT destroy our best solution. So what we can do is keep the two solutions we chose and copy it to the new generation.

The next step is called **Mutation**, helps to discover new solution and we simply change one bit of each solutions we have on the new generation.



Summary

What we need is

- genetic representation of a solution
- a function to generate new solutions
- fitness function
- selection function
- crossover function
- mutation function

Is it worth it?

Yes, but the limitation is the runtime.