University of Babylon, College of science for women
Dept. of Computer science

Evolutionary Computing

Forth year

Dr. Salah Al-Obaidi

Lecture #9

Fall 2024

Contents

7. Example Applications

7.1 The Knapsack Problem

The 0–1 knapsack problem, a generalisation of many industrial problems, can be briefly described as follows. We are given a set of n items, each of which has attached to it some value v_i, and some cost c_i. The task is to select a subset of those items that maximises the sum of the values, while keeping the summed cost within some capacity C_{max}. Thus, for example, when packing a backpack for a round-the-world trip, we must balance likely utility of the items against the fact that we have a limited volume (the items chosen must fit in one bag), and weight (airlines impose fees for luggage over a given weight).

It is a natural idea to represent candidate solutions for this problem as binary strings of length n, where a 1 in a given position indicates that an item is included and a 0 that it is omitted. The corresponding genotype space G is the set of all such strings with size 2^n, which increases exponentially with the number of items considered. Using this G, we fix the representation in the sense of data structure, and next we need to define the mapping from genotypes to phenotypes.

The first representation (in the sense of a mapping) that we consider

takes the phenotype space P and the genotype space to be identical. The quality of a given solution p, represented by a binary genotype g, is thus determined by summing the values of the included items, i.e., $q(p) = \sum_{i=1}^{n} v_i \cdot g_i$. However, this simple representation leads us to some immediate problems. By using a one-to-one mapping between the genotype space G and the phenotype space P, individual genotypes may correspond to invalid solutions that have an associated cost greater than the capacity, i.e., $\sum_{i=1}^{n} c_i \cdot g_i > C_{max}$.

The second representation that we outline here solves this problem by employing a decoder function, that breaks the one-to-one correspondence between the genotype space G and the solution space P. In essence, our genotype representation remains the same, but when creating a solution we read from left to right along the binary string, and keep a running tally of the cost of included items. When we encounter a value 1, we first check to see whether including the item would break our capacity constraint. In other words, rather than interpreting a value 1 as meaning include this item, we interpret it as meaning include this item IF it does not take us over the cost constraint. The effect of this scheme is to make the mapping from genotype to phenotype space many-to-one, since once the capacity has been reached, the values of all bits to the right of the current position are irrelevant, as no more items will be added to the solution. Furthermore, this mapping ensures that all binary strings represent valid solutions with a unique fitness (to be maximised).

Having decided on a fixed-length binary representation, we can now choose variation operators, because the bit-string representation is ‘standard’

there. A suitable (but not necessarily optimal) recombination operator is the so-called one-point crossover, where we align two parents and pick a random point along their length. The two offspring are created by exchanging the tails of the parents at that point. We will apply this with 70% probability, i.e., for each pair of parents there is a 70% chance that we will create two offspring by crossover and 30% that the children will be just copies of the parents. A suitable mutation operator is so-called bit-flipping: in each position we invert the value with a small probability $p_m \in [0, 1)$.

In this case we will create the same number of offspring as we have members in our initial population. As noted above, we create two offspring from each two parents, so we will select that many parents and pair them randomly. We will use a tournament for selecting the parents, where each time we pick two members of the population at random (with replacement), and the one with the highest value $q(p)$ wins the tournament and becomes a parent. We will institute a generational scheme for survivor selection, i.e., all of the population in each iteration are discarded and replaced by their offspring.

Finally, we should consider initialisation (which we will do by random choice of 0 and 1 in each position of our initial population), and termination. In this case, we do not know the maximum value that we can achieve, so we will run our algorithm until no improvement in the fitness of the best member of the population has been observed for 25 generations.

Our evolutionary algorithm to tackle this problem can be specified as below in Table 7.1.

Table 7.1: Description of the EA for the knapsack problem.

Representation	**Binary strings of length n**
Recombination	**One-point crossover**
Recombination probability	70%
Mutation	**Each value inverted with independent probability p_m**
Mutation probability p_m	$1/n$
Parent selection	**Best out of random 2**
Survival selection	**Generational**
Population size	500
Number of offspring	500
Initialisation	**Random**
Termination condition	**No improvement in last 25 generations**