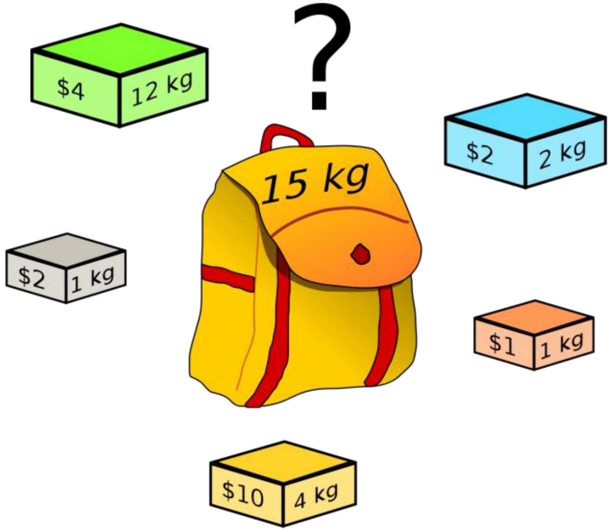
Artificial Intelligence

The knapsack problem
and genetic algorithms

POLITÉCNICO DE LEIRIA



The knapsack problem

- Let us consider that we have N items
- Each item i is characterized by its weight w_i and its value v_i
- There is also a sack where the items can be put in
- The weight of the items put in the sack cannot exceed some value W
- **The goal is to choose a subset of the N items to be put in the sack so that the total value of the items is maximized while the total weight does not exceeds W**

Individuals representation

- Each individual in the genetic algorithm represents a possible solution to the problem
- So, each individual prescribes the subset of the items that should be put in the sack
- Each individual will be represented has a binary vector of *N* genes
- Each gene corresponds to an item
- A gene with value 1 means that the corresponding item is to be put in the sack; A gene with value 0 means that the corresponding item is not to be put in the sack

Fitness function, version 1

- For some individual ind:
  - Let $w(ind)$ be the sum of the weight of the items to be put in the sack
  - Let $v(ind)$ be the sum of the value of the items to be put in the sack
- $fitness(ind) = v(ind)$ if $w(ind) \leq W$;

  otherwise, $fitness(ind) = 0$

Fitness function, version 2

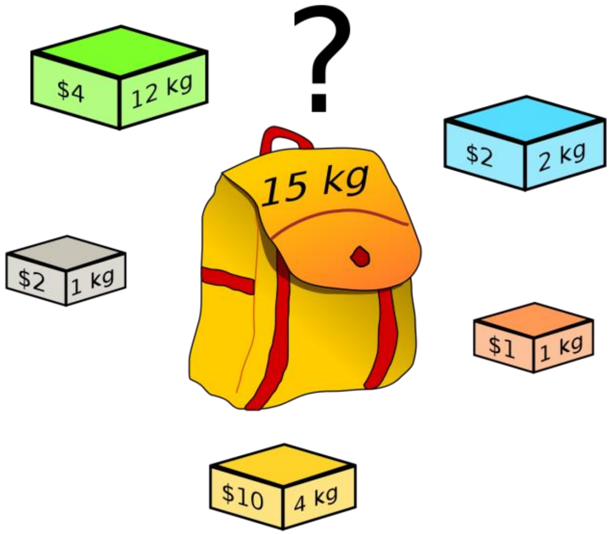
- For some individual ind:
  - Let $w(ind)$be the sum of the weight of the items to be put in the sack
  - Let $v(ind)$be the sum of the value of the items to be put in the sack
- $fitness(ind) = v(ind)$ if $w(ind) \leq W$

  otherwise, $fitness(ind) = v(ind) - penalty$,

  where $penalty = MAXVP \times (w(ind) - W)$ and $MAXVP$ is the maximum value for $\frac{v_{item}}{w_{item}}$ over all items

Example

- Let us consider that we have a sack with maximum capacity of 15kg and the following 5 items:
  - Item 1: (w = 12, v = 04)
  - Item 2: (w = 02, v = 02)
  - Item 3: (w = 01, v = 01)
  - Item 4: (w = 04, v = 10)
  - Item 5: (w = 01, v = 02)



Individual example 1

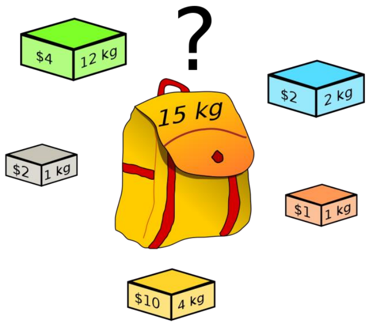
Genotype

1	0	1	0	1

Items 1, 3 and 5 will be put in the sack

$w(ind) = 12 + 1 + 1 = 14$

$v(ind) = 4 + 1 + 2 = 7$



Fitness, version 1:

Since $w(ind) < W$ (14 < 15), $fitness(ind) = 7$

Fitness, version 2:

Since $w(ind) < W$ (14 < 15), $fitness(ind) = 7$

Individual example 2

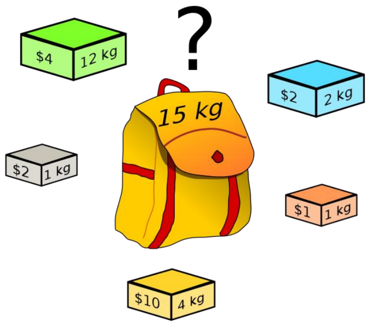
Genotype

1	0	0	1	0

Items 1 and 4 will be put in the sack

$w(ind) = 12 + 4 = 16$

$v(ind) = 4 + 10 = 14$



Fitness, version 1:

Since $w(ind) > W$ (16 > 15), $fitness(ind) = 0$

Fitness, version 2:

Since $w(ind) > W$ (16 < 15),

$$fitness(ind) = 14 - 2.5 \times (16 - 15) = 11.5$$

$MAXVP$ computation:

Item 1: 3/12 = 0.33 ($\frac{v_{item}}{w_{item}}$)

Item 2: 2/2 = 1

Item 3: 1/1 = 1

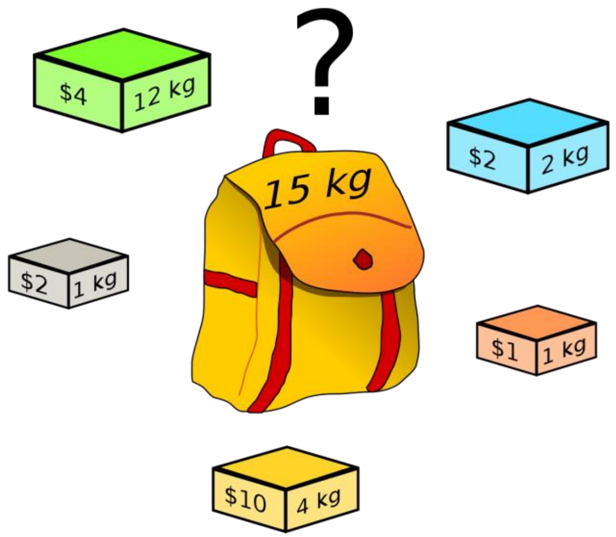
Item 4: 10/4 = 2.5 ← maximum

Item 5: 2/1 = 1

Knapsack

Example of the first iteration of the genetic algorithm

- Population size: 6 individuals
- Fitness function, version 2
- Selection method: tournament with size 2
- Crossover operator: 1 cut, occurring with probability of 0.7
- Mutation operator probability: 0.1



create_initial_population(P(*t*))

```
t = 0
create_initial_population(P(t))
evaluate(P(t))
while stop condition not met
    P'(t) = select(P(t))
    P''(t) = apply_genetic_operators(P'(t))
    P(t + 1) = create_next_population(P(t), P''(t))
    evaluate(P(t + 1))
    t = t + 1
return best individual found
```

create_initial_population(P(t))

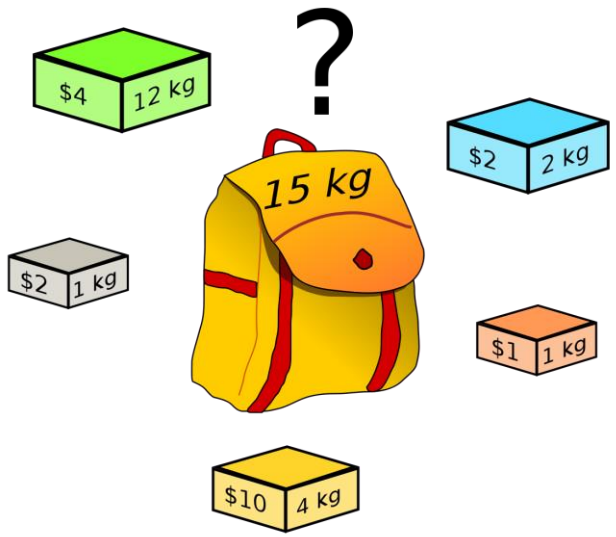
- Let us consider that the population is randomly inicialized with the following individuals
  - 10101
  - 10010
  - 11000
  - 10110
  - 00100
  - 00101

evaluate(P(t))

```
t = 0
create_initial_population(P(t))
evaluate(P(t))
while stop condition not met
    P'(t) = select(P(t))
    P''(t) = apply_genetic_operators(P'(t))
    P(t + 1) = create_next_population(P(t), P''(t))
    evaluate(P(t + 1))
    t = t + 1
return best individual found
```

evaluate(P(t))

- Using fitness function, version 2, we compute each individual's fitness:
  - $10101 \rightarrow fitness = 7$
  - $10010 \rightarrow fitness = 14 - 2.5 \times (16 - 15) = 11.5$
  - $11000 \rightarrow fitness = 6$
  - $10110 \rightarrow fitness = 15 - 2.5 \times (17 - 15) = 10$
  - $00100 \rightarrow fitness = 1$
  - $00101 \rightarrow fitness = 3$



P'(t) = select(P(t))

```
t = 0
create_initial_population(P(t))
evaluate(P(t))
while stop condition not met
    P'(t) = select(P(t))
    P''(t) = apply_genetic_operators(P'(t))
    P(t + 1) = create_next_population(P(t), P''(t)))
    evaluate(P(t + 1))
    t = t + 1
return best individual found
```

P’(t) = select(P(t))

- We now use the tournament method to build P’(T)

- We repeat the following procedure 6 times:

  - Randomly choose two individuals from P(t)

  - Choose the best among the two selected individuals and put it in P’(t)

P’(t) = select(P(t))

- Application of the tournament method: **P’(t)**
  - 10101 (7) and 00100 (1) chosen → $10101 \rightarrow fitness = 7$
  - 10010 (11,5) and 10110 (10) chosen → $10010 \rightarrow fitness = 11.5$
  - 00101 (3) and 10010 (11,5) chosen → $10010 \rightarrow fitness = 11.5$
  - 10110 (10) and10101 (7) chosen → $10110 \rightarrow fitness = 10$
  - 00101 (3) and 00100 (1) chosen → $00101 \rightarrow fitness = 3$
  - 10110 (10) and 00101 (3) chosen → $10110 \rightarrow fitness = 10$

P''(t) = apply_genetic_operators(P'(t))

```
t = 0
create_initial_population(P(t))
evaluate(P(t))
while stop condition not met
    P'(t) = select(P(t))
    P''(t) = apply_genetic_operators(P'(t))
    P(t + 1) = create_next_population(P(t), P''(t))
    evaluate(P(t + 1))
    t = t + 1
return best individual found
```

P’’(*t*) = apply_genetic_operators(P’(*t*)) - **crossover**

P’(t)

10101

10010

10010

10110

00101

10110

1. First, we generate a value in the interval [0, 1]
2. Let’s consider that 0.4 is generated
3. Since 0.4 < 0.7, the two individuals should be crossedover (0.7 is the crossover probability)
4. Now, we must randomly select the crossover point
5. Let’s consider that the generated crossover point is between the 3rd and the 4th genes
6. The resulting individuals are 10110 and 10001

P’’(t)

10110

10001

P''(t) = apply_genetic_operators(P'(t)) - **crossover**

P'(t)

10101

10010

10010

10110

00101

10110

1. First, we generate a value in the interval [0, 1]
2. Let's consider that 0.8 is generated
3. Since 0.8 > 0.7, the two individuals should not be crossedover
4. This means that the two individuals are copied to P''(t) with no modifications

P''(t)

10110

10001

10010

10110

P''(t) = apply_genetic_operators(P'(t)) - **crossover**

P'(t)		**P''(t)**
10101		10110
10010		10001
10010		10010
10110		10110
00101		00110
10110		10101

1. First, we generate a value in the interval [0, 1]
2. Let's consider that 0.2 is generated
3. Since $0.2 < 0.7$, the two individuals should be crossedover
4. Now, we must randomly select the crossover point
5. Let's consider that the generated crossover point is between the 2nd and the 3rd genes
6. The resulting individuals are 10101 and 00101

P''(*t*) = apply_genetic_operators(P'(*t*)) - **mutation**

P''(t) before mutation		**P''(t)** after mutation
10110		10110
10001		10011
10010		10010
10110		10110
00110		00110
10101		00111

1. For each gene of all individuals, we generate a value in the interval [0, 1]
2. If the value is smaller than 0.1 (the mutation probability), we mutate the gene: if it is 0, it is flipped to 1 and vice-versa
3. Let's consider that, after doing this for all genes of all individuals, the mutated genes are the red ones on the population on the left
4. This results on the population on the right

P(t + 1) = create_next_population(P(t), P''(t))

```
t = 0
create_initial_population(P(t))
evaluate(P(t))
while stop condition not met
    P'(t) = select(P(t))
    P''(t) = apply_genetic_operators(P'(t))
    P(t + 1) = create_next_population(P(t), P''(t))
    evaluate(P(t + 1))
    t = t + 1
return best individual found
```

P(t + 1) = create_next_population(P(t), P’’(t)))

P’’(t)

10110

10011

10010

10110

00110

00111

- We just do P(t + 1) = P’’(t)

- This is the so called **generational strategy**, where the parents population is completely replaced by the descendants one

P(t + 1)

10110

10011

10010

10110

00110

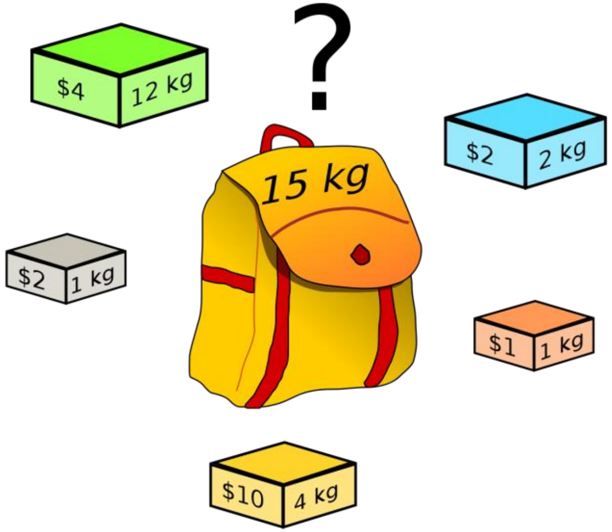
00111

evaluate(P(t + 1))

```
t = 0
create_initial_population(P(t))
evaluate(P(t))
while stop condition not met
    P'(t) = select(P(t))
    P''(t) = apply_genetic_operators(P'(t))
    P(t + 1) = create_next_population(P(t), P''(t)))
    evaluate(P(t + 1))
    t = t + 1
return best individual found
```

evaluate(P(t))

- Using fitness function, version 2, we compute each individual's fitness:
  - $10110 \rightarrow fitness = 15 - 2.5 \times (17 - 15) = 10$
  - $10011 \rightarrow fitness = 16 - 2.5 \times (17 - 15) = 11$
  - $10010 \rightarrow fitness = 14 - 2.5 \times (16 - 15) = 11.5$
  - $10110 \rightarrow fitness = 15 - 2.5 \times (17 - 15) = 10$
  - $00110 \rightarrow fitness = 11$
  - $00111 \rightarrow fitness = 13$



while stop condition not met

```
t = 0
create_initial_population(P(t))
evaluate(P(t))
while stop condition not met
    P'(t) = select(P(t))
    P''(t) = apply_genetic_operators(P'(t))
    P(t + 1) = create_next_population(P(t), P''(t)))
    evaluate(P(t + 1))
    t = t + 1
return best individual found
```

- The algorithm would now proceed to the next iteration
- In our case, it would stop after a predefined number of iterations (generations)
- After stoping, the algorithm returns the best individual found during the entire process (not necessarily the best individual in the last generation)
- This means that we must do some bookeeping in order to save the best individual found so far (not shown in the algorithm)