**Intelligent Systems**

**Exercise 8. Local Search**

# Exercise description

The objective of this exercise is to apply the concepts and methods for Solving Problems with Local Search.

**Team members**

Write the student id, name, and campus of each member in a different line.

1: Carlos Hinojosa A01137566, Campus Monterrey

2: Eider Diaz A00828174, Campus Monterrey

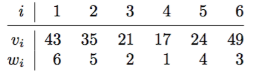
3: Miguel Cortes A01270966, Campus Monterrey

**Combinatorial Optimization Problem**

**Knapsack Problem**

The Knapsack problem is defined as: given n items of known weights w1, w2, …, wn and values v1, v2, …, vn, and a knapsack of capacity W, find the most valuable subset of items that fit into the knapsack.

**Problem**: You have to decide which items to take when traveling to Mexico City. For this, you will carry a Knapsack with a weight capacity (W) of 9. If you have possible items to take with values and weights defined in the following table:



Which items will you fit in the knapsack to maximize the value of your load?

**Hill Climbing**

1. Suppose you want to solve the knapsack problem by using the Hill Climbing algorithm. Use as objective function to MAXIMIZE, the sum of the value of the items fitted in the knapsack. The neighborhood is constructed by fitting in or removing from the knapsack an item. Manually run up 3 iterations of the algorithm from the following initial state. In each iteration, you must show all the neighbors of the current state, its evaluation, and the chosen movement.

Knapsack = {3, 4, 6}

Maximize function =

={−1, if ∑(wi)>MaxWeight

∑(vi) otherwise

Neighbor generation method:

-Swap and item

-Add an item

**Iteration 1**

Initial state = [3,4,6] - 87

Generated neighbors:

|  |  |
| --- | --- |
| Neighbor | Evaluation |
| [1,4,6] | -1 |
| [2,4,6] | 101 |
| [5,4,6] | 90 |
| [3,1,6] | -1 |
| [3,2,6] | -1 |
| [3,5,6] | 94 |
| [3,4,1] | 81 |
| [3,4,2] | 73 |
| [3,4,5] | 62 |
| [3,4,6,1] | -1 |
| [3,4,6,2] | -1 |
| [3,4,6,5] | -1 |

Selected neighbor: [2,4,6], value 101

**Iteration 2**

Initial state = [2,4,6] - 101

Generated neighbors:

|  |  |
| --- | --- |
| Neighbor | Evaluation |
| [1,4,6] | -1 |
| [3,4,6] | 87 |
| [5,4,6] | 90 |
| [2,1,6] | -1 |
| [2,3,6] | -1 |
| [2,5,6] | -1 |
| [2,4,1] | -1 |
| [2,4,3] | 73 |
| [2,4,5] | -1 |
| [2,4,6,1] | -1 |
| [2,4,6,3] | -1 |
| [2,4,6,5] | -1 |

No selected neighbor since there was no improvement on the evaluation function

Final result: [2,4,6] - 101

**Simulated Annealing**

1. Manually run up several iterations of the SIMULATED ANNEALING algorithm now to solve the knapsack problem from the initial configuration of the previous problem. Use the same neighborhood strategy, but in this case randomly select between fitting or removing, and the object to fit in or remove from the knapsack. To verify the eligibility for the successor, start at the temperature T=60 with a constant cooling rate of 10. For each iteration show the current state, its evaluation, the two random numbers used to select the modification of the state, the resulting neighbor state, its evaluation, the current temperature, the acceptance probability, and (if needed) the random number used to decide if the neighbor will become the new current solution.

**Genetic algorithms**

1. In this exercise, you will use a genetic algorithm to start solving the knapsack problem (2 epochs). For this, you must do the following:
   1. Create a random population of 4 individuals (knapsacks with items fitted in).
   2. Represent each individual as a binary chromosome of 6 bits (1 bit for each item).
   3. Compute the fitness of each individual as the value of the knapsack.
   4. Compute the reproduction probability and the cumulative probability of each individual.
   5. Randomly select 4 individuals in pairs for crossover using the Roulette wheel method.
   6. Apply the one-point crossover to both pairs to create their 4 offsprings.
   7. Compute the fitness of the new individuals.
   8. Mutate every bit of the individuals with a probability of 0.02.
   9. Repeat from d.