Mohamed Boudiaf university of M'sila
Faculty of MI
Department of informatics
Master 1 IDO
Combinatorial optimization 2 2025
Practice work: solving KSP using exact methods and local search

Create a new folder called PW CO2 2025, then, in NetBeans IDE, create a new project as java application called PWOC2. Rename the main class as KSP.
In order to prepare local search implementation and efficiency measure, we use the following program that allow solving 0-1 KSP by dynamic programming method.

```
def knapsack(n,weights, values, capacity):
      # CONSTRUCTION DE LA TABLE
    tab = [[0] * (capacity + 1) for _ in range(n + 1)]
    for i in range(1, n + 1):
       for w in range(1, capacity + 1):
          if weights[i - 1] > w:
             tab[i][w] = tab[i - 1][w]
          else:
             tab[i][w] = max(values[i - 1] + tab[i - 1][w - weights[i - 1]], tab[i - 1][w])
    # CALCUL DE LA SOLUTION (LES OBJETS PRIS)
    solution = []
    w = capacity
    for i in range(n, 0, -1):
      if tab[i][w] != tab[i - 1][w]:
          solution.append(i)
          w -= weights[i - 1]
    return tab[n][capacity], solution
# APPEL (MAIN)
n=5
weights = [3,1, 3, 4, 5]
values = [8,1, 4, 5, 7]
capacity = 11
max_value, solution = knapsack(n,weights, values, capacity)
print("Maximum value = ", max_value)
print("solution =", solution)
```

Part I

1. **Recall the KSP statement and mathematical formulation.**
2. **Write the recursive formula of DP for solving KSP.**
3. **Type then run the above code.**
4. **Run this code using your own input with different instances.**
5. **Add code to show processing time of the program.**
6. **Modify this code to enter random values and weights.**
   Modify this code to obtain the same instance for each run.
   Distinguish the cases of capacity ratio in {0.2,0.4,0.6,0.8} of the total weight and n in {10,20,50,100,500,1000,10000}. Print data and result in corresponding tables.
7. **Add necessary code to make your results in text file called KSP_DP_RESULTS.TXT.**

Part II

In this part, we propose to implement five metaheuristics that solve the simple KSP:

- Random Search (RS)
- First Deep (FD)
- Best Deep (BD)
- Simulated Annealing (SA)
- Tabu Search (TS)

After adjusting parameters of these approaches, we will measure the approximation of each one.

1. Implementing RS and adjusting the number of iterations:
   By using the one-bit flip neighborhood, write then run the code of RS.
   Run this code for 10*n, 20*n, 50*n iterations 10 times for each case and find the CPU time, the optimum and the approximation rate of the mean of each case.
   Construct the corresponding curve.
   Deduce the ideal number of iterations and the approximation factor.
2. Finding the best neighborhood:
   Using the same demarche above, compare between the one-bit flip and two different bits-flip neighborhoods.
3. Implement FD and BD then compare between them.
4. Implement RS and TS then compare between them.