**TEMA 1 – PROBLEMA RUCSACULUI**

Student: Ichim Daniel Alexandru

Grupa: 30645

An universitar: 2016/2017

***CUPRINS***

[**1.** **Descrierea problemei** 3](#_Toc477816797)

[**1.1** **Cerinte** 3](#_Toc477816798)

[**1.2** **Problema Rucsacului** 4](#_Toc477816799)

[**2.** **Prezentare algoritmi** 5](#_Toc477816800)

[**2.1 Metoda de cautare exhaustiva (brute force)** 6](#_Toc477816801)

[**2.2 Metoda de cautare aleatoare** 7](#_Toc477816802)

[**3.** **Rezultate Experimentale** 8](#_Toc477816803)

[**4. Comparatii** 9](#_Toc477816804)

[**5. Bibliografie** 11](#_Toc477816805)

# **Descrierea problemei**

## **Cerinte**

Se consideră problema rucsacului pentru care se cere:

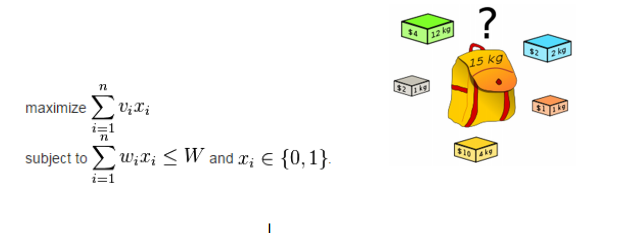
1. Să se implementeze o metoda de căutare aleatoare (random search) pentru problema rucsacului. Încercați să îmbunătățiți această metodă.
2. Să se implementeze o metodă de căutare greedy pentru problema rucsacului.
   1. Selectează cel mai bun obiec dacă adăugarea lui în rucsac nu conduce la depățirea limitei de greutate:
   2. Metrici posibile pentru cel mai bun ?
      1. Max valoare
      2. Min greutate
      3. Max valoarea/greutate
3. Să se implementeze una din cele trei variante ale metodei Hill-Climbing prezentate la curs.
   1. Random Hill-Climbing
   2. Steepset Ascent Hill-Climbing
   3. Next Ascent Hill-Climbing
4. Scrieți un raport care să conțină rezultatele obținute.

**Obs.**

* Metodele implementate la punctele 1, 2 si 3 vor fi testate pe cele 2 instante primite în cadrul laboratorului.
* Raportul trebuie să aibă structura prezentată în cadrul laboratorului

## **Problema Rucsacului**

* Există n obiecte
* Fiecare obiect are o valoare (v) și o greutate (w)
* Puneți în rucsac valoarea maximă fără a depăși greutatea maximă admisă W
* Xi = 1 înseamnă obiectul i este pus în rucsac
* Xi = 0 înseamnă obiectul i nu este pus în rucsac



# **Prezentare algoritmi**

Pentru dezvoltarea algoritmilor am folosit ca limbaj de programare Java. Structura problemei am reprezentat-o în modul următor.

* Clasa **BagItem -**  contine informatii referitoare la un obiect din rucsac, **ID-ul** acestuia,

**quantity –** reprezentand greutatea obiectului, **value –** reprezentand valoarea.

* Clasa **Bag -** contine informatii cu privire la o obiectele aflate in rucsac la un moment dat(**items)**, precum si greutatea maxima a acestuia(**maxWeight)**.

Pentru citirea datelor am folosit clasa **InfoReader** care citeste dintr-un fisier de input si seteaza urmatorii parametrii: **maxWeight** –greutatea maxima a rucsacului, **nrObjects –** numarul total de obiecte care pot fi adaugate in rucsac, **bagItemList –** lista efectiva cu obiectele respective.

Fiecare algoritm a fost implementat intr-o clasa sepaarata care implementeaza interfata **SearchStrategy**, metoda implementata este cea de **findBestBag** care returneaza o instanta a clasei Bag, reprezentand solutia furnizata de catre algoritm.

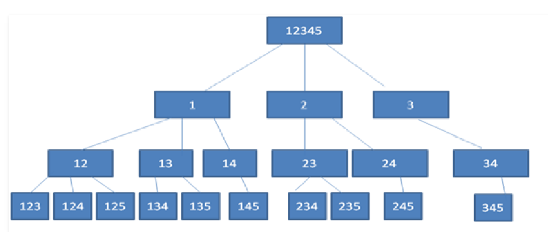
Pentru salvarea rezultatelor, am folosit clasa **ExcelUtils** in care am mai multe metode pentru printarea intr-un fisier Excel a solutiiilor furnizate de catre un algoritm, precum si a parametrilor de itnrare ai acestora.

## **Metoda de cautare exhaustiva (brute force)**

### **Prezentare algoritm**

Aceasta metoda este una cat se poate de directa, in mod uzual bazata pe parametrii de intrare ai problemei. Dacă avem un număr de n obiecte dintre care putem alege pentru a le adauga in rucsac, vom avea 2n combinatii posibile de obiecte. Un obiect poate fi ales sau poate sa nu fie ales. Acest mod poate fi reprezentat folosind un sir de biti continand 0 si 1. Daca al i-lea simbol din sirul de biti este 0, atunci al i-lea obiect nu este ales, iar daca este 1, obiectul este ales.

Pentru generarea tuturor solutiilor, am ales un algoritm de generare al combinarilor. Cream un array temporar data[], care stocheaza toate combiantiile posibile, una cate una. Ideea algoritmului este de a incepe de la primul index din data[], fixam aceste elemente unul cate unul si apelam in mod recurent pentru elementele ramase. De exemplu, pentru un array {1, 2, 3, 4, 5} pentru a afla combinariile si dimensiunea maxima a combinarilor 3. Fixam initial 1 la indexul 0 in data[], apoi apelam metoda in mod recurent pentru indecsii ramasi, apoi fixam 2 la indexul 0 si apelam recursive. In final, vom seta 3 si apelam recursive pentru indecsii ramasi.



Figură Recurenta pentru combinari de 5 luate cate 3

*exhaustiveSearch()*

*for i = 1, nrObjects*

*generateCombination(data[nrObjects], i)*

*generateCombination(arr, n, r)*

*int data[] = new int[r]*

*combinationUtil(arr, data, 0, n-1, r);*

*combinationUtil(arr, data, start, end, index, r)*

*if index == r*

*createBag()*

*calculateBestBagSoFar()*

*return;*

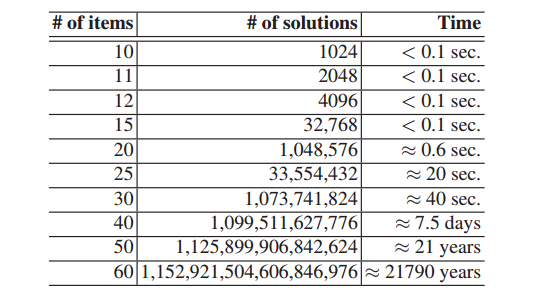
*for i = start; i <= end and end-i+1 >= r-index; i++*

*data[index] = arr.get(i)*

*combinationUtil(arr, data, i+1, end, index+1, r)*

### **Rezultate experimentale**

Metoda exhaustiva are o complexitate exponentiala, poate fi utilizata doar pentru instante mici ale problemei rucsacului, unde numarul de obiecte este mic. Mai jos avem un table cu privire la timpii de rulare ai aceste solutii.



## **Metoda de cautare aleatoare**

### **Prezentare algoritm**

Algoritmul de cautare folosind metoda aleatoare, genereaza o instanta a unui rucsac folosind obiecte alese in mod aleator (cate un obiect din fiecare), se alege un obiect in mod aleator, se verifica daca dupa adaugarea obiectului greutatea rucsacului nu depaseste limita admisa, daca limita este depasita, ultimul obiect introdus este scos din rucsac, si se returneaza solutia formata din obiectele introduse in rucsac pana in acest moment.

*Pseudocod*:

randomSearchBag()

indecsList = [0,1, .. n] /\* Lista cu indecsii tuturor obiectelor \*

while(true)

indecs = randomIndecsFrom(indecsList)

addItem(indecs, bag);

if checkOverfull(bag)

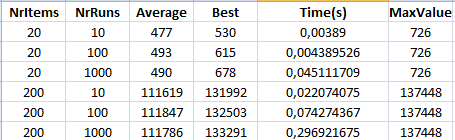
removeItem(indecs, bag);

break;

removeIndecs(indecs, indecsList) /\* Stergerea obiectului din lista initiala \*/

return bag

* + 1. **Rezultate experimentale**



Figură Comparare Rulari Random Search

## **Metoda de cautare Greedy**

Metodele de programare greedy, sunt folosite in general in problemele de optimizare. In mod tipic, aceste metode folosesc anumite euristici pentru a genera o secventa de suboptime care pot converge spre o valoare optima.

Strategii Greedy posibile pentru problema rucsacului:

* + Alegem obiectul care are valoarea cea mai mare din obiectele ramase.
  + Alegem obiectul care are greutatea cea mai mica din obiectele ramase.
  + Alegem obiectul care are cea mai mare ratie valoare/greutate din obiectele ramase.

### **Prezentare algoritm**

Pseudocod

GreedyAlgorithm(itemsList)

sortItemsByFilter(itemsList)

max = itemList.size

while(true)

addItem(itemsList[max], bag)

if (bagOverFull(bag))

removeItem(itemsList[max], bag)

break;

max—

return bag

### **Rezultate experimentale**

Dupa testarea tuturor strategiilor de alegere a obiectelor, cele mai bune rezultaet sunt obtinute folosind strategia de maximizare a ratiei valoare/greutate

## **Metoda de cautare Steepest Hill Climb**

### **Prezentare algoritm**

### **Rezultate experimentale**

## **Metoda de cautare Branch and Bound**

### **Prezentare algoritm**

### **Rezultate experimentale**

# **4. Comparatii**

# **5. Bibliografie**

<http://www.codeproject.com/>

<http://ro.wikipedia.org/wiki/Programare_orientat%C4%83_pe_obiecte>

<http://stackoverflow.com/>

<http://www.coned.utcluj.ro/~salomie/PT/Lab/Tema3_HW3.pdf>

<http://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html>