

Instituto Politécnico Nacional

ESC Superior de Cómputo

“EVOLUTIONARY COMPUTING” 1-KNAPSACK AND COIN CHANGE PROBLEMS

Jorge Luis Rosas Trigueros

Saldaña Aguilar Gabriela

30/08/16 - 06/09/16

THEORETICAL FRAMEWORK

Evolutionary algorithms have been shown to be successful for a wide range of optimization problems. While these randomized search heuristics work well for many optimization problems in practice, a satisfying and rigorous mathematical understanding of their performance is an important challenge in the area of genetic and evolutionary computing. It has been proven for various combinatorial optimization problems that they can be solved by evolutionary algorithms in reasonable time using a suitable representation together with mutation operators adjusted to the given problem. The representations used in these papers are different from the general encodings working with binary strings as considered earlier in theoretical works on the runtime behavior of evolutionary algorithms. The chosen representations reflect some properties of partial solutions of the problem at hand that allow to obtain solutions that can be extended to optimal ones for the considered problem.

Dynamic programming is a well-known algorithmic technique that helps to tackle a wide range of problems. A general framework for dynamic programming has been considered by e. g. Woeginger and Kogan. The technique allows the design of efficient algorithms, that solve the problem at hand to optimality, by extending partial solutions to optimal ones.

Framework for Evolutionary Algorithms An evolutionary algorithm consists of different generic modules, which have to be made precise by the user to best fit to the problem. Experimental practice, but also some theoretical work, demonstrate that the right choice of representation, variation operators, and selection method is crucial for the success of such algorithms. We assume that the problem to be solved is given by a multi-objective function g that has to be optimized. We consider simple evolutionary algorithms that consist of the following components. The algorithm (see Algorithm 2) starts with an initial population P_0 .

During the optimization the evolutionary algorithm uses a selection operator $\text{sel}(\cdot)$ and a mutation operator $\text{mut}(\cdot)$ to create new individuals. The d -dimensional fitness function together with a partial order \preceq_{par} on \mathbb{R}^d induce a partial order \preceq_{dom} on the phenotype space, which guides the search. After the termination of the EA, an output function $\text{out}(\cdot)$ is utilized to map the individuals in the last population to search points from the original search space.

THE KNAPSACK PROBLEM

Is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

THE COIN CHANGE PROBLEM

How many different ways can you make change for an amount, given a list of coins? In this problem, your code will need to efficiently compute the answer.

Task:

Given a value N , if we want to make change for N cents, and we have infinite supply of each of

$C = \{C_1, C_2, C_3, \dots, C_M\}$ valued coins, how many ways can we make the change? The order of coins doesn't matter.

EQUIPMENT AND MATERIAL.

SOFTWARE: The program is made in Python

BODY

We were told to write down the code for the DP problems: The knapsack problem and the Coin Change problem.

The first issue I found is how to implement the 2D array, because I didn't want to use a lot of code in that section, fortunately, I found an in-line Matrix construction, a nested for but in one line. The second problem I had was to define the reconstruction for that DP because I needed to be careful with the trace back, but at last my code worked. The final result are the values of the Table constructed and the assets taken to maximize or minimize the problem (see e. g. [1, 2]).

```
C:\WINDOWS\system32\cmd.exe
(c) 2016 Microsoft Corporation. Todos los derechos reservados.

C:\Users\GabrielaSaldaña>cd Documents

C:\Users\GabrielaSaldaña\Documents>cd Escuela

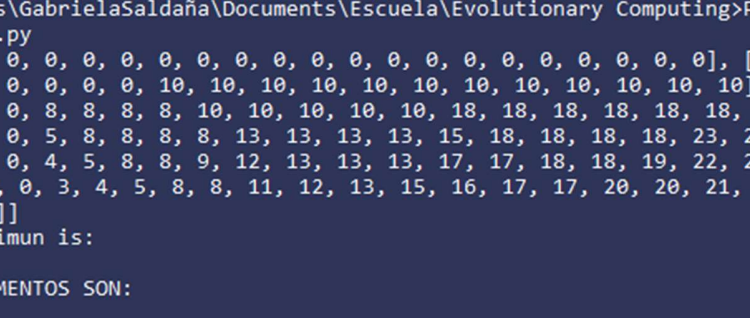
C:\Users\GabrielaSaldaña\Documents\Escuela>cd "Evolutionary Computing"

C:\Users\GabrielaSaldaña\Documents\Escuela\Evolutionary Computing>Python CoinProblem.py
The minimum is: 2 of 8
LOS ELEMENTOS SON:
0 de 1,
1 de 5,
1 de 6,
0 de 8,

C:\Users\GabrielaSaldaña\Documents\Escuela\Evolutionary Computing>Python CoinProblem.py
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11], [0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 2, 3], [0, 1, 2, 3, 4, 1, 1, 2, 3, 4,
2, 2], [0, 1, 2, 3, 4, 1, 1, 2, 1, 2, 2, 2]]
The minimum is: 2 of 8
LOS ELEMENTOS SON:
0 de 1,
1 de 5,
1 de 6,
0 de 8,

C:\Users\GabrielaSaldaña\Documents\Escuela\Evolutionary Computing>
```

F1: Coin Change problem



```
C:\WINDOWS\system32\cmd.exe

C:\Users\GabrielaSaldaña\Documents\Escuela\Evolutionary Computing>Python K
napSack.py
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0
, 0, 0, 0, 0, 0, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10], [0, 0
, 0, 0, 0, 8, 8, 8, 8, 10, 10, 10, 10, 10, 18, 18, 18, 18, 18, 18], [0
, 0, 0, 0, 5, 8, 8, 8, 8, 13, 13, 13, 13, 15, 18, 18, 18, 18, 23, 23, 23],
 [0, 0, 0, 4, 5, 8, 8, 9, 12, 13, 13, 13, 13, 17, 17, 18, 18, 19, 22, 23, 23,
23], [0, 0, 3, 4, 5, 8, 8, 11, 12, 13, 15, 16, 17, 17, 20, 20, 21, 22, 23,
25, 26]]
The maximun is:
26
LOS ELEMENTOS SON:
9,
5,
4,
2,

C:\Users\GabrielaSaldaña\Documents\Escuela\Evolutionary Computing>
```

F2: KnapSack problem

CODE

```
#COIN CHANGE PROBLEM
#Monto a juntar
C=11
#Denominacion de las monedas
W=[1,5,6,8]

#m requires len(w)+1 rows and c+1 columns
Matrix=[]
j, i = C+1, len(W)+1
Matrix = [[0 for x in range(j)] for y in range(i)]# la llenamos con ceros

for r in range(1,len(W)+1):#filas
    for d in range(1,C+1):#columnas
        #Si no cabe
        if r == 1:#si la denominacion es de 1
            Matrix[r][d]=d
        elif W[r-1] > d:#si la denominacion es mayor
            Matrix[r][d] = Matrix[r-1][d]#la denominacion anterior
        else:#Si cabe
            Matrix[r][d] = min(Matrix[r-1][d], Matrix[r][d-W[r-1]] +
1)#comparamos el de arriba contra
print Matrix
print 'The minimum is: ' + str(Matrix[len(W)][C]) + ' of ' + str(W[len(W)-1])
#Knowing the elements inserted
XR=[0 for x in range(len(W))]
i=0
a=len(W)#renglon
b=C#columna

while(b>= 0):
    if Matrix[a][b] == Matrix[a-1][b] :

        if(a <0 ):break

        else: a = a-1

    else:

        XR[a-1]=XR[a-1] + 1

        b= b - (W[a-1])

print 'LOS ELEMENTOS SON:'

for x in range(len(XR)):

    if XR[x] >= 0:

        print str(XR[x]) + ' de ' + str(W[x]) + ', '
```

```

#KNAPSACK PROBLEM
#Capacidad de la mochila
C=20
#Pesos de los objetos
W=[9,5,4,3,2]
#Beneficios de los Objetos
B=[10,8,5,4,3]
#m requires len(w)+1 rows and c+1 columns
Matrix=[]
j, i = C+1, len(W)+1 #11 columns, 5 rows
Matrix = [[0 for x in range(j)] for y in range(i)]# la llenamos con ceros

for r in range(1,len(W)+1):#filas
    for d in range(1,C+1):#columnas
        if W[r-1]>d: #Si no cabe
            Matrix[r][d]=Matrix[r-1][d]#a la hora de acceder
        else:
            Matrix[r][d]=max(Matrix[r-1][d], Matrix[r-1][d-W[r-1]] + B[r-1])

print Matrix

print 'The maximun is: '
print Matrix[len(W)][C]

#Knowing the elements inserted
XR=[0 for x in range(len(W))]

b=C #columna
#a= fila
for a in range(len(W),0,-1):
    if Matrix[a-1][b] == Matrix[a][b] : XR[a-1]=0
    else:    XR[a-1]=1
    b= C - (W[a-1]*XR[a-1])

print 'LOS ELEMENTOS SON:'

for x in range(len(XR)):
    if XR[x]== 1:
        print str(W[x]) + ','

```

CONCLUSION

I liked it the way we took the class in the lab like if it was free style, but with the help of the professor, also it is good to explain what will be the next topic to cover in the lab so the students can find it out and investigate a little bit before entering the class because it could be confusing if it is our first time coding evolutionary algorithms.

BIBLIOGRAPHY

NAME: <https://cs.adelaide.edu.au/~frank/papers/DynProgGECCO09.pdf>

AUTHOR: Benjamin Doerr, Anton Eremeev, Christian Horoba, Frank Neumann, Madeleine Theile

DATE: 07/12/2009

NAME: <https://www.hackerrank.com/challenges/coin-change>

AUTHOR: hackerrank nickname @alanl

NAME: https://en.wikipedia.org/wiki/Knapsack_problem

DATE: 17 August 2016