Solving using Pyomo

In [1]:
```
import random
import pyomo.environ as pyo
from pyomo.environ import *
```

In [2]:
```
random.seed(1)
n=10000 #number of packages
b=200 #number of bins
c=50

#Generate random locations
vj=random.choices(range(10, 100),k=n)
wj=random.choices(range(5, 20),k=n)
```

In [7]:
```
model=pyo.ConcreteModel()

model.i=RangeSet(0,b-1)
model.j=RangeSet(0,n-1)
model.p=Param(model.j,initialize=vj)
model.w=Param(model.j,initialize=wj)
model.x=Var(model.j,model.i,within=Binary)

def objective(model):
    result=0
    for j in model.j:
        for i in model.i:
            result+=model.p[j]*model.x[j,i]
    return result
model.cost=Objective(rule=objective,sense=maximize)

def constraint1(model,i):
    total=0
    for j in model.j:
        total+=model.w[j]*model.x[j,i]
    return total<=c
model.cons1=Constraint(model.i,rule=constraint1)

def constraint2(model,j):
    total=0
    for i in model.i:
        total+=model.x[j,i]
    return total<=1
model.cons2=Constraint(model.j,rule=constraint2)

instance=model.create_instance()
opt=pyo.SolverFactory('gurobi')
opt.solve(instance,options={'TimeLimit': 10000},tee=True)
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2023-01-29
Read LP format model from file C:\Users\dell\AppData\Local\Temp\tmphbe4gbq_.pyomo.lp
Reading time = 6.50 seconds
x2000001: 10201 rows, 2000001 columns, 4000001 nonzeros
Set parameter TimeLimit to value 10000
Gurobi Optimizer version 9.5.0 build v9.5.0rc5 (win64)
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 10201 rows, 2000001 columns and 4000001 nonzeros
Model fingerprint: 0x20eebf97
```

```
Variable types: 1 continuous, 2000000 integer (2000000 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+01]
  Objective range  [1e+01, 1e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 5e+01]
Found heuristic solution: objective 44250.000000
Presolve removed 1 rows and 1 columns (presolve time = 5s) ...
Presolve removed 1 rows and 1 columns
Presolve time: 6.43s
Presolved: 10200 rows, 2000000 columns, 4000000 nonzeros
Variable types: 0 continuous, 2000000 integer (2000000 binary)

Deterministic concurrent LP optimizer: primal and dual simplex
Showing first log only...


Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    5.3257000e+04   0.000000e+00   1.362713e+07     22s
     258    1.1792655e+05   0.000000e+00   0.000000e+00     22s
     258    1.1792655e+05   0.000000e+00   0.000000e+00     22s

Use crossover to convert LP symmetric solution to basic solution...

Root crossover log...

       0 DPushes remaining with DInf 0.0000000e+00                23s

  294328 PPushes remaining with PInf 0.0000000e+00                23s
       0 PPushes remaining with PInf 0.0000000e+00                24s

  Push phase complete: Pinf 0.0000000e+00, Dinf 1.9895197e-13     24s


Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
  296047    1.1792655e+05   0.000000e+00   0.000000e+00     24s
Concurrent spin time: 0.01s

Solved with dual simplex

Root relaxation: objective 1.179265e+05, 295790 iterations, 5.52 seconds (2.50 work un
its)
Total elapsed time = 34.24s
Total elapsed time = 36.83s

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 117926.545    0    9 44250.0000 117926.545   167%     -   49s
H    0     0                     117810.00000 117926.545  0.10%     -   60s
H    0     0                     117926.00000 117926.545  0.00%     -   66s
     0     0 117926.545    0    9 117926.000 117926.545  0.00%     -   66s

Explored 1 nodes (375473 simplex iterations) in 66.76 seconds (30.68 work units)
Thread count was 8 (of 8 available processors)

Solution count 3: 117926 117810 44250

Optimal solution found (tolerance 1.00e-04)
Best objective 1.179260000000e+05, best bound 1.179260000000e+05, gap 0.0000%
```

Out[7]:
{'Problem': [{'Name': 'x2000001', 'Lower bound': 117926.0, 'Upper bound': 117926.0, 'Number of objectives': 1, 'Number of constraints': 10201, 'Number of variables': 2000001, 'Number of binary variables': 2000000, 'Number of integer variables': 2000000, 'Number of continuous variables': 1, 'Number of nonzeros': 4000001, 'Sense': 'maximize'}], 'Solver': [{'Status': 'ok', 'Return code': '0', 'Message': 'Model was solved to optimality (subject to tolerances), and an optimal solution is available.', 'Termination condition': 'optimal', 'Termination message': 'Model was solved to optimality (subject to tolerances), and an optimal solution is available.', 'Wall time': '66.78540992736816', 'Error rc': 0, 'Time': 81.06724262237549}], 'Solution': [OrderedDict([('number of solutions', 0), ('number of solutions displayed', 0)])]}

Random Sampling

In [3]:
```
import random
import numpy as np
import copy
import time
import matplotlib.pyplot as plt
import math
```

In [4]:
```
#convert the binary result to a table.
import pandas as pd

def final_table(x):
        container=list(range(1,b+1))
        container.append('Total')
        package=[]
        for i in range(b):
            package.append([])
            for j in range(n):
                if x[j,i]==1:
                    package[i].append(j+1)
        package.append('')

        profits=[]
        weights=[]
        for i in range(b):
            profit=0
            weight=0
            for j in package[i]:
                profit+=vj[j-1]
                weight+=wj[j-1]
            profits.append(profit)
            weights.append(weight)
        profits.append(sum(profits))
        weights.append('')
        data={'Container':container,'Packages':package,'Profit':profits,'Weight':weigh
        return pd.DataFrame(data)
```

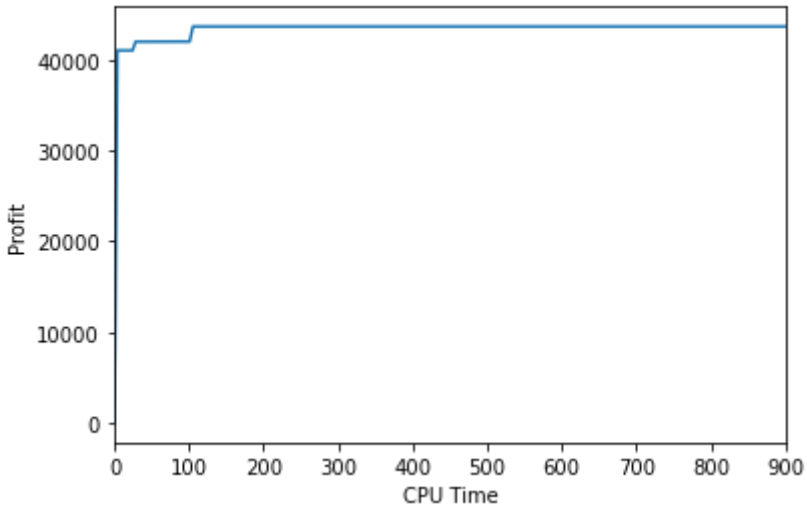
In [47]:
```
result=np.zeros((n,b))
iteration=0
best_value=0
start=time.time()
end=time.time()
time_list=[0]
value_list=[0]

while end-start<60*15:
    #randomly generate a result
    for i in range(n):
        #whether to put the item in a knapsack
```

```
            a=random.randint(0,3)
            if a==1:
                #decide to put in which knapsack
                j=random.randint(0,b-1)
                result[i,j]=1

        #check whether the result is valid
        valid=True
        overweight={}
        for i in range(b):
            weight=0
            for j in range(n):
                weight+=wj[j]*result[j,i]
            if weight>c:
                valid=False
                overweight[i]=weight

        #randomly remove items until it meets the constraint
        if valid==False:
            for i,w in overweight.items():
                in_bag=[]
                for j in range(n):
                    if result[j,i]==1:
                        in_bag.append(j)
                while w>c:
                    a=random.randint(0,len(in_bag)-1)
                    result[in_bag[a],i]=0
                    w-=wj[in_bag[a]]
                    in_bag.pop(a)

        #calculate the new value
        cur_value=0
        for i in range(n):
            for j in range(b):
                cur_value+=vj[i]*result[i,j]
        if cur_value>best_value:
            best_value=cur_value
        end=time.time()
        time_list.append(end-start)
        value_list.append(best_value)
    print(final_table(result))
    plt.xlabel('CPU Time')
    plt.ylabel('Profit')
    plt.xlim((0,60*15))
    plt.plot(time_list,value_list)
```

	Container	Packages	Profit	Weight
0	1	[1351, 1636, 4604, 7262]	216.0	48
1	2	[98, 409, 2043, 8172]	142.0	44
2	3	[91, 3106, 3759, 3941]	162.0	46
3	4	[2372, 3346, 7994]	240.0	39
4	5	[1099, 3968, 6789, 7359, 9998]	218.0	39
..	...	...	...	...
196	197	[569, 1419, 3920, 8654, 9090]	290.0	49
197	198	[646, 2380, 3549, 4888, 9402]	354.0	42
198	199	[436, 1938, 5951, 9901]	242.0	42
199	200	[2250, 3787, 8260, 9564]	183.0	46
200	Total		43685.0	

[201 rows x 4 columns]

Out[47]: [<matplotlib.lines.Line2D at 0x26d3f3e6eb0>]



Local Search

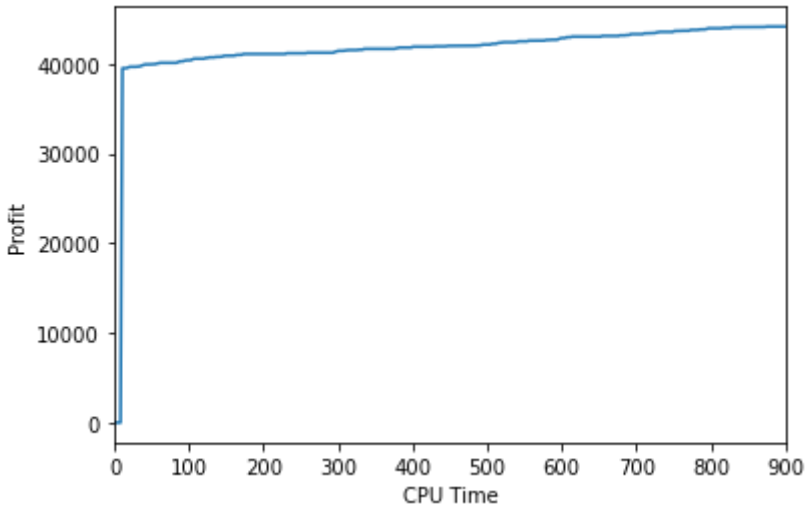
In [38]:

```
result=np.zeros((n,b))
iteration=0
best_value=0
start=time.time()
time_list=[0]
value_list=[0]

#generate a random solution
for i in range(n):
    #whether to put the item in a knapsack
    a=random.randint(0,5)
    if a==1:
        #decide to put in which knapsack
        j=random.randint(0,b-1)
        result[i,j]=1

#check whether the result is valid
valid=True
overweight={}
for i in range(b):
    weight=0
    for j in range(n):
        weight+=wj[j]*result[j,i]
    if weight>c:
        valid=False
        overweight[i]=weight

#randomly remove items until it meets the constraint
if valid==False:
    for i,w in overweight.items():
        in_bag=[]
        for j in range(n):
            if result[j,i]==1:
                in_bag.append(j)
        while w>c:
            a=random.randint(0,len(in_bag)-1)
            result[in_bag[a],i]=0
            w-=wj[in_bag[a]]
            in_bag.pop(a)

end=time.time()
while end-start<60*15:
    pre_result=copy.deepcopy(result)
```

```
    #choose which item to change
    i=random.randint(0,n-1)

    #check whether the item is already put in a bag, if so, remove it.
    for j in range(b):
        if result[i,j]==1:
            result[i,j]=0

    #choose a random bag to put it in
    j=random.randint(0,b-1)
    result[i,j]=1

    #check whether the result is valid
    valid=True
    for i in range(b):
        weight=0
        for j in range(n):
            weight+=wj[j]*result[j,i]
        if weight>c:
            valid=False
            result=pre_result

    #calculate the new value
    if valid==True:
        cur_value=0
        for i in range(n):
            for j in range(b):
                cur_value+=vj[i]*result[i,j]
        if cur_value>best_value:
            best_value=cur_value
        else:
            result=pre_result
    end=time.time()
    time_list.append(end-start)
    value_list.append(best_value)

print(final_table(result))
plt.xlabel('CPU Time')
plt.ylabel('Profit')
plt.xlim((0,60*15))
plt.plot(time_list,value_list)
```

```
    Container                        Packages   Profit Weight
0           1  [3030, 3195, 5309, 9320, 9413]    290.0     49
1           2  [4455, 6405, 7495, 8945, 9264]    235.0     47
2           3  [2964, 3961, 4074, 6736, 7096]    166.0     50
3           4  [3990, 4313, 5141, 5876, 5904]    250.0     48
4           5        [4069, 5970, 6347, 9368]    214.0     49
..        ...                             ...      ...    ...
196       197             [4859, 4951, 7537]    225.0     40
197       198        [3339, 4961, 7186, 7471]    248.0     43
198       199  [1038, 2391, 4566, 5263, 8935]    237.0     43
199       200             [1180, 7617, 9249]    203.0     49
200     Total                                  44142.0

[201 rows x 4 columns]
```

Out[38]: [<matplotlib.lines.Line2D at 0x2369a4ef1c0>]



Simulated Annealing

In [47]:

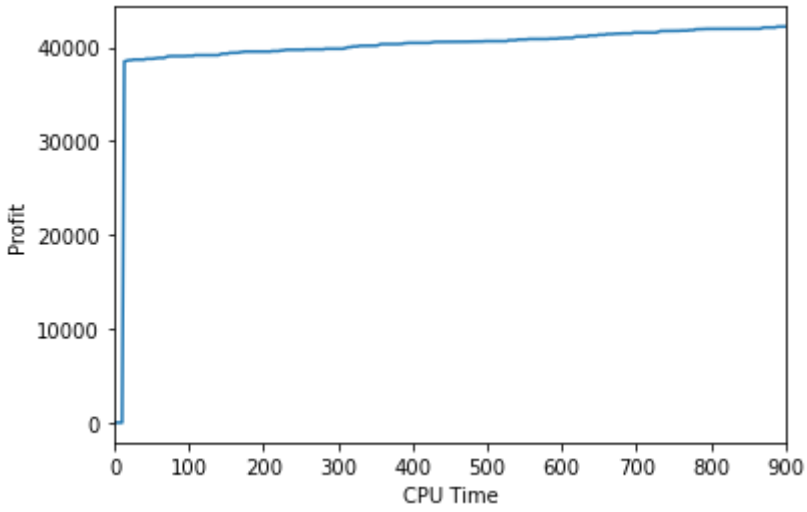
```
result=np.zeros((n,b))
iteration=0
best_value=0
start=time.time()
time_list=[0]
value_list=[0]
T=1000

#generate a random solution
for i in range(n):
    #whether to put the item in a knapsack
    a=random.randint(0,5)
    if a==1:
        #decide to put in which knapsack
        j=random.randint(0,b-1)
        result[i,j]=1

#check whether the result is valid
valid=True
overweight={}
for i in range(b):
    weight=0
    for j in range(n):
        weight+=wj[j]*result[j,i]
    if weight>c:
        valid=False
        overweight[i]=weight

#randomly remove items until it meets the constraint
if valid==False:
    for i,w in overweight.items():
        in_bag=[]
        for j in range(n):
            if result[j,i]==1:
                in_bag.append(j)
        while w>c:
            a=random.randint(0,len(in_bag)-1)
            result[in_bag[a],i]=0
            w-=wj[in_bag[a]]
            in_bag.pop(a)
for i in range(n):
    for j in range(b):
        best_value+=vj[i]*result[i,j]
```

```
end=time.time()
while end-start<60*15:
    pre_result=copy.deepcopy(result)

    #choose which item to change
    i=random.randint(0,n-1)

    #check whether the item is already put in a bag, if so, remove it.
    for j in range(b):
        if result[i,j]==1:
            result[i,j]=0

    #choose a random bag to put it in
    j=random.randint(0,b-1)
    result[i,j]=1

    #check whether the result is valid
    valid=True
    for i in range(b):
        weight=0
        for j in range(n):
            weight+=wj[j]*result[j,i]
        if weight>c:
            valid=False
            result=pre_result

    #calculate the new value
    if valid==True:
        cur_value=0
        for i in range(n):
            for j in range(b):
                cur_value+=vj[i]*result[i,j]
        p=random.random()
        if cur_value>best_value or p>math.exp((cur_value-best_value)/T):
            best_value=cur_value
        else:
            result=pre_result
    end=time.time()
    time_list.append(end-start)
    value_list.append(best_value)
    T=0.9*T

print(final_table(result))
plt.xlabel('CPU Time')
plt.ylabel('Profit')
plt.xlim((0,60*15))
plt.plot(time_list,value_list)
```

	Container	Packages	Profit	Weight
0	1	[1074, 2568, 3055, 7841, 8175]	293.0	49
1	2	[783, 2024, 5897, 7934, 8312, 9399]	297.0	48
2	3	[48, 1447, 1807, 1923, 3860]	242.0	48
3	4	[1038, 3389, 8226, 9909]	205.0	50
4	5	[4931, 7875, 8467]	186.0	48
..	...	...	...	...
196	197	[1336, 2037, 7381]	191.0	50
197	198	[1574, 7426, 8283]	178.0	47
198	199	[3897, 5537, 7755, 9434]	230.0	50
199	200	[2599, 3732, 3836, 7546]	206.0	49
200	Total		42218.0	

[201 rows x 4 columns]

Out[47]: [<matplotlib.lines.Line2D at 0x2369b5ca640>]



Greedy Algorithm

In [9]:

```
start=time.time()
#generate an initial solution using greedy algorithm
z=[] #value/weight ratio
for i in range(len(wj)):
    z.append(wj[i]/vj[i])
data={'value':vj,'weight':wj,'ratio':z}
table=pd.DataFrame(data).sort_values('ratio')
order=list(table.index)

result=np.zeros((n,b))
bag=0
total_w=0
for i in order:
    result[i,bag]=1
    total_w+=wj[i]
    if total_w>c:
        if bag<b-1:
            result[i,bag]=0
            result[i,bag+1]=1
            bag+=1
            total_w=wj[i]
        else:
            result[i,bag]=0
            bag+=1
    if bag==b:
        break

#using simulated annealing to solve the problem
iteration=0
best_value=0
for i in range(n):
    for j in range(b):
        best_value+=vj[i]*result[i,j]
time_list=[0]
value_list=[best_value]
T=1000
end=time.time()

while end-start<60*15:
    pre_result=copy.deepcopy(result)

    #choose which item to change
    i=random.randint(0,n-1)
```

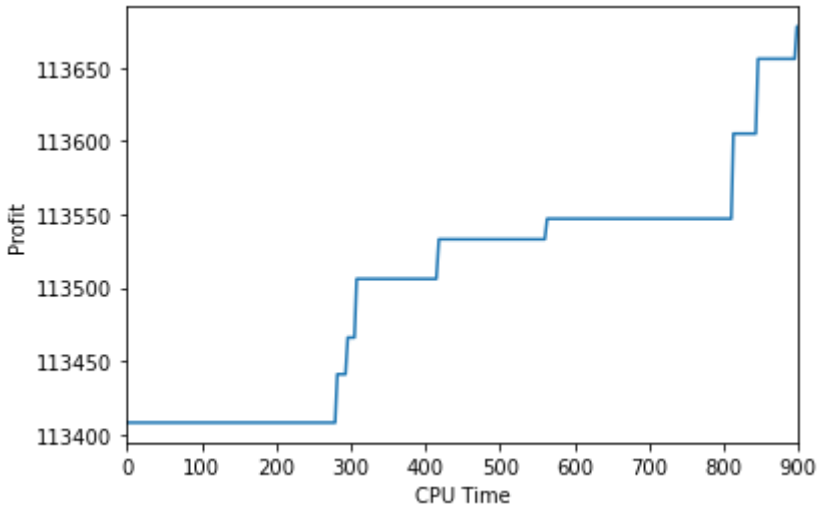
```
    #check whether the item is already put in a bag, if so, remove it.
    for j in range(b):
        if result[i,j]==1:
            result[i,j]=0

    #choose a random bag to put it in
    j=random.randint(0,b-1)
    result[i,j]=1

    #check whether the result is valid
    valid=True
    for i in range(b):
        weight=0
        for j in range(n):
            weight+=wj[j]*result[j,i]
        if weight>c:
            valid=False
            result=pre_result

    #calculate the new value
    if valid==True:
        cur_value=0
        for i in range(n):
            for j in range(b):
                cur_value+=vj[i]*result[i,j]
        p=random.random()
        if cur_value>best_value or p>math.exp((cur_value-best_value)/T):
            best_value=cur_value
        else:
            result=pre_result
    end=time.time()
    time_list.append(end-start)
    value_list.append(best_value)
    T=0.9*T

print(final_table(result))
plt.xlabel('CPU Time')
plt.ylabel('Profit')
plt.xlim((0,60*15))
plt.plot(time_list,value_list)
```

```
    Container                                           Packages  Profit  \
0           1  [265, 1979, 2112, 3507, 4141, 6304, 6461, 6472...     980
1           2  [340, 890, 2775, 4070, 4161, 4623, 7052, 8116,...     973
2           3  [572, 1403, 2892, 5190, 6129, 6143, 6424, 7134...     955
3           4  [2120, 2360, 3037, 4294, 4392, 4999, 5078, 587...     943
4           5  [176, 1148, 3013, 3063, 4654, 5994, 7110, 7712...     934
..        ...                                                ...     ...
196       197                      [661, 4064, 5960, 7321, 9517]     441
197       198          [2057, 2439, 5722, 6506, 8036, 8264, 9770]     436
198       199                      [315, 1619, 4178, 7478, 9097]     450
199       200                  [396, 604, 3926, 6344, 6429, 8035]     423
200     Total                                                     113678

    Weight
0       50
1       50
2       50
3       50
4       50
..     ...
196     49
197     50
198     50
```

```
199     47
200
```

[201 rows x 4 columns]

Out[9]: [<matplotlib.lines.Line2D at 0x23582b28640>]



Greedy Algorithm with Simulated Annealing

In [13]:

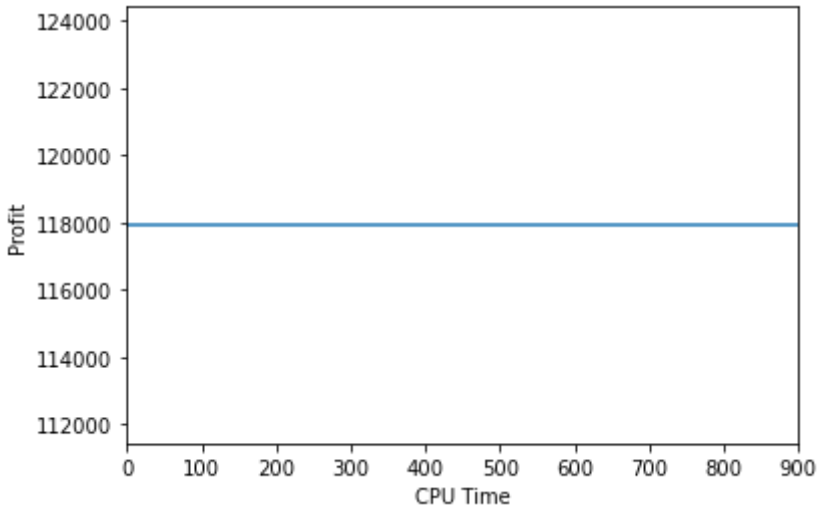
```
start=time.time()

#use optimisation to obtain a better initial solution
arrangement=[]
best_val=0
v=copy.deepcopy(vj)
w=copy.deepcopy(wj)
for a in range(b):
    table=np.zeros([len(v)+1,c+1])
    plan=[]
    for i in range(len(v)+1):
        plan.append([])
    for i in range(len(v)+1):
        for j in range(c+1):
            plan[i].append([])

    for i in range(1,len(v)+1):
        for j in range(1,c+1):
            if w[i-1]>j:
                table[i,j]=table[i-1][j]
                plan[i][j]=plan[i-1][j]
            else:
                table[i,j]=max(table[i-1][j],table[i-1][j-w[i-1]]+v[i-1])
                if table[i,j]==table[i-1][j]:
                    plan[i][j]=plan[i-1][j]
                else:
                    plan[i][j]=plan[i-1][j-w[i-1]]+[i]
    arrangement.append(plan[-1][-1])
    best_val+=table[-1][-1]
    x=plan[-1][-1]
    for i in x:
        v[i-1]=0
        w[i-1]=10000000

result=np.zeros((n,b))
for i in range(len(arrangement)):
    for j in arrangement[i]:
        result[j-1][i]=1
```

```
iteration=0
best_value=best_val
time_list=[0]
value_list=[best_val]
T=1000
end=time.time()

while end-start<60*15:
    pre_result=copy.deepcopy(result)

    #choose which item to change
    i=random.randint(0,n-1)

    #check whether the item is already put in a bag, if so, remove it.
    for j in range(b):
        if result[i,j]==1:
            result[i,j]=0

    #choose a random bag to put it in
    j=random.randint(0,b-1)
    result[i,j]=1

    #check whether the result is valid
    valid=True
    for i in range(b):
        weight=0
        for j in range(n):
            weight+=wj[j]*result[j,i]
        if weight>c:
            valid=False
            result=pre_result

    #calculate the new value
    if valid==True:
        cur_value=0
        for i in range(n):
            for j in range(b):
                cur_value+=vj[i]*result[i,j]
        p=random.random()
        if cur_value>best_value or p>math.exp((cur_value-best_value)/T):
            best_value=cur_value
        else:
            result=pre_result
    end=time.time()
    time_list.append(end-start)
    value_list.append(best_value)
    T=0.9*T

print(final_table(result))
plt.xlabel('CPU Time')
plt.ylabel('Profit')
plt.xlim((0,60*15))
plt.plot(time_list,value_list)
```

	Container	Packages	Profit	\
0	1	[265, 340, 1979, 2112, 3507, 4141, 6304, 6461,...	980	
1	2	[890, 2775, 4070, 4161, 4623, 7052, 8116, 9080...	973	
2	3	[572, 1403, 2892, 3037, 4999, 5190, 6129, 6143...	955	
3	4	[176, 1148, 2120, 2360, 3063, 4294, 4392, 6159...	943	
4	5	[2011, 3013, 4654, 5078, 5871, 5994, 6722, 690...	934	
..	...	...	...	
196	197	[5201, 5463, 5701, 6049, 6629, 6645]	444	
197	198	[667, 1521, 3241, 7116, 7441, 7822]	443	

```
198       199          [1948, 2558, 4518, 8087, 8300, 8569, 8964]     442
199       200                      [443, 1410, 2780, 2923, 9828]     441
200     Total                                                      117926

    Weight
0       50
1       50
2       50
3       50
4       50
..     ...
196     50
197     50
198     50
199     50
200

[201 rows x 4 columns]
```

Out[13]: [<matplotlib.lines.Line2D at 0x23582b40820>]



In [ ]: