Python for GRASP

Data based Logistics

Rafael Martí

Statistics and Operations Research Department

University of Valencia

Getting started with Python

- Download from python.org/downloads/.
  - Select the option “Add Py to PATH” when installing it.

- IDE (Integrated Development Environment)
  - We may use Pycharm
  - jetbrains.com/pycharm/download
  - Install the Community version (it’s free).

Running a project

- Copy our files in a folder
- Open Pycharm
- Create a new project
  - Don’t create new environment
  - Previously configured
  - Create from existing sources
  - Do not touch the folder venv (virtual environment)

- The code is organized in modules:
  - Structure
  - Instances
  - Algorithms

Data files

100 10 ← n m

0 1 8.01

0 2 8.77

0 3 1.23

0 4 4.37 ← i j $d(i,j)$

0 5 4.84

0 6 5.71

0 7 1.60

0 8 4.83

0 9 0.70

0 10 7.86

0 11 1.74

0 12 1.32

…

Structure instance

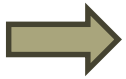
Instance data is stored in a **dictionary**: an ‘associative array’ indexed by keys.

Pairwise distances between elements are stored in a **matrix** within key ‘d’ in the dictionary.

```
def readInstance(path):
    instance = {}
    with open(path, "r") as f:
        # First line in file has two numbers: n p
        n, p = f.readline().split()
        n = int(n)
        p = int(p)
        instance['n'] = n
        instance['p'] = p
        instance['d'] = []
        for i in range(n):
            instance['d'].append([0] * n)
        for i in range(n):
            for j in range(i+1, n):
                u, v, d = f.readline().split()
                u = int(u)
                v = int(v)
                d = round(float(d), 2)
                instance['d'][u][v] = d
                instance['d'][v][u] = d
    return instance
```

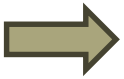
Structure solution

Create
a dictionary



```
def createEmptySolution(instance):
    sol = {}
    sol['instance'] = instance
    sol['sol'] = set()
    sol['of'] = 0
    return sol
```

Access to the instance
through the solution



```
def evaluate(sol):
    of = 0
    for s1 in sol['sol']:
        for s2 in sol['sol']:
            if s1 < s2:
                of += sol['instance']['d'][s1][s2]
    return of
```

GRASP construction

Select first element at random.

Matrix cl has two columns: $d(i)$ i

Matrix rcl has some of the rows of cl

```
def construct(inst, alpha):
    sol = solution.createEmptySolution(inst)
    n = inst['n']
    u = random.randint(0, n-1)
    solution.addToSolution(sol, u)
    cl = createCandidateList(sol, u)
    alpha = alpha if alpha >= 0 else random.random()
    while not solution.isFeasible(sol):
        gmin, gmax = evalGMinGMax(cl)
        threshold = gmax - alpha * (gmax - gmin)
        rcl = [ ]
        for i in range(len(cl)):
            if cl[i][0] >= threshold:
                rcl.append(cl[i])
        selIdx = random.randint(0, len(rcl)-1)
        cSel = rcl[selIdx]
        solution.addToSolution(sol, cSel[1], cSel[0])
        cl.remove(cSel)
        updateCandidateList(sol, cl, cSel[1])
    return sol
```

Local Search

- from structure import solution

- def improve(sol):
- improve = True
- while improve:
- improve = **tryImprove**(sol)

- def tryImprove(sol):
- sel, ofVarSel, unsel, ofVarUnsel = **selectInterchange**(sol)
- if ofVarSel < ofVarUnsel:
- solution.removeFromSolution(sol, sel, ofVarSel)
- solution.addToSolution(sol, unsel, ofVarUnsel)
- return True
- return False

sel is the element in solution with **minimum** sum of distances (bestSel) to the rest.

We look for the element v, not in sol, with **maximum** sum of distances (bestUnsel) to the elements in sol.
We exclude sel from this computation.

```
def selectInterchange(sol):
    n = sol['instance']['n']
    sel = -1
    bestSel = 0x3f3f3f
    unsel = -1
    bestUnsel = 0

    for v in sol['sol']:
        d = solution.distanceToSol(sol, v)
        if d < bestSel:
            bestSel = d
            sel = v

    for v in range(n):
        d = solution.distanceToSol(sol, v, without=sel)
        if not solution.contains(sol, v):
            if d > bestUnsel:
                bestUnsel = d
                unsel = v
    return sel, round(bestSel,2), unsel, round(bestUnsel,2)
```

Try the code!

- You may find in the zip folder with the code several implementations for the construction and the local search procedures:

- lsbestimp.py
- lsfirstimp.py
- lsfirstimp_sorte.py

- cgrasp.py
- cgrasp_eff.py
- greedy.py

- Run the different combinations, and compare the results.