OPTIMISATION PROJECT

Reactive coverage of a large area using a constellation of observation satellites

G23
FERNÁNDEZ DE LA INFIESTA, Álvaro
LAHOZ GAITX, Pau
MUÑOZ MORALES, Víctor

10/11/2022

Introduction

Stage 1

Objectives:

- To become familiar with the input data of the problem
- Run the whole optimisation chain with the elements already provided
- Define metrics to evaluate a solution (calculate overall cost, calculate goal satisfaction scores)

Load Data

```
# Load data
obsRequests = data['ObservationRequests']
fixedPassCost = data['fixedPassCost']
minVisiDuration = data['minVisiDuration']
aois = data['AOIs']
accesses = data['Accesses']
referenceDate = data['referenceDate']
passCostPerTimeUnit = data['passCostPerTimeUnit']
goals = data['Goals']
satPasses = data['Passes']
```

Metrics to evaluate the solution

```
# Computation of the total cost
timeUses = [satPasses[id]['end'] - satPasses[id]['start'] for id in selectedPassIds]
totalCost = len(selectedPassIds)*data['fixedPassCost'] + sum(timeUses)*data['passCostPerTimeUnit']
```

```
# Computation of the quality of the covering: It is defined as the relation of the
# requests covered over the total amount of requests
quality = 0
for aoiId in range(nAois):
    aux = 0
    for goalId in range(nGoals):
        for step in range(goals[goalId]['nSteps']):
            obsReqId = aoiId + nAois*aux + step + (goals[goalId]['nSteps'] - 1)*aoiId
            requestStart = step*goals[goalId]['duStep']
            requestEnd = step*goals[goalId]['duStep'] + goals[goalId]['rctHorizon']
            for passId in selectedPassIds:
                if any(accessIdObsReq in satPasses[passId]['accessIds'] for accessIdObsReq in obsRequests[obsReqId]['accessIds']):
                    if (requestStart <= satPasses[id]['end'] - 30) and (requestEnd >= satPasses[id]['start'] + 30):
                        quality +=1
                        break
        aux += goals[goalId]['nSteps']

quality /= nObsRequests
```

Stage 2

- Objective:
  - Consider the problem with a selection of complete runs and test a MILP approach with DOCPLEX

Variables

```
# Variables
selectedPass = model.binary_var_list(range(nPasses), name='selectedPass')
```


min(fixedCost +
variableCost·(tEnd-tIni))
Objective function

Objective fcn. implementation

```
# Objective:
# Minimisation of the cost
model.minimize(model.sum(selectedPass)*data['fixedPassCost']
    + model.sum((satPasses[id]['end'] - satPasses[id]['start'])*selectedPass[id]
    for id in range(nPasses))*data['passCostPerTimeUnit'])
```

Definition of the constraints

Access constraints

1) **The site must be visible by the satellite (accessId)**
2) **The connection must last at least 30s to ensure proper data collection**

Pass constraints

1) **The satellite must view the site in order to make a connection**
2) **The connection must last at least 30s to ensure proper data collection**
3) **If a pass is not taken, its ‘booking’ time is 0.**

Requests constraints

1) **An observation request is only satisfied if at least one of its access requests is satisfied**
2) **All observation requests must be satisfied**

Implementation of the constraints

```python
# Ensuring that all the goals are covered bu at least one access of a pass
def covering(requestStart, requestEnd, passId, obsReqId):
    if any(accessIdObsReq in satPasses[passId]['accessIds'] for accessIdObsReq in obsRequests[obsReqId]['accessIds']):
        if requestStart<=(satPasses[passId]['end'] - 30) and requestEnd>=(satPasses[passId]['start'] + 30):
            return selectedPass[passId]
    return 0
```

Definition of the covering function

```python
# Checking for all the observation requests
for aoiId in range(nAois):
    aux = 0
    for goalId in range(nGoals):
        for step in range(goals[goalId]['nSteps']):
            obsReqId = aoiId + nAois*aux + step + (goals[goalId]['nSteps'] - 1)*aoiId
            requestStart = step*goals[goalId]['duStep']
            requestEnd = step*goals[goalId]['duStep'] + goals[goalId]['rctHorizon']
            coverings = [covering(requestStart, requestEnd, passId, obsReqId) for passId in range(nPasses)]
            sum_covering = model.sum(coverings)
            if not str(sum_covering) == "0":
                model.add_constraint(sum_covering >= 1)
        aux += goals[goalId]['nSteps']
```

At least one Access Request satisfied per Observation Request

Stage 3

Objective:

- Consider the problem with partial passage selection and test a MILP approach with DOCPLEX

Variables

```
# Variables
lbs = []
ube = []

for i in range(nPasses):
    lbs.append(satPasses[i]['start'])
    ube.append(satPasses[i]['end'])

selectedPass = model.binary_var_list(range(nPasses), name='selectedPass')
tStart = model.continuous_var_list(range(nPasses), lb=lbs, name='tStart')
tEnd = model.continuous_var_list(range(nPasses), ub=ube, name='tEnd')
```


min(fixedCost +
variableCost·(tEnd-tIni))
Objective function

Objective fcn. implementation

```
# Objective function: Minimisation of the cost
model.minimize(model.sum(selectedPass)*data['fixedPassCost']
               + (model.sum((tEnd[id] - tStart[id])for id in range(nPasses))
                  - 30*(nPasses - model.sum(selectedPass)))*data['passCostPerTimeUnit'])
```

Variables:

- **selectedPass:** Binary List
- **tStart:** Continuous List
- **tEnd:** Continuous List

Definition of the constraints

Access constraints

1) The site must be visible by the satellite (accessId)
2) The connection must last at least 30s to ensure proper data collection

Pass constraints

1) The satellite must view the site in order to make a connection
2) The connection must last at least 30s to ensure proper data collection
3) If a pass is not taken, its ‘booking’ time is 0.

Requests constraints

1) An observation request is only satisfied if at least one of its access requests is satisfied
2) All observation requests must be satisfied

Time constraints:

1) The connection start and end must be inside of the satellite pass duration
2) The connection must last at least 30s to ensure proper data collection

Implementation of the constraints 1

```
# tEnd - tStart >= 30
for passId in range(nPasses):
    model.add_constraint((tEnd[passId] - tStart[passId]) >= 30)
```

Connection at least 30s long

```
# Ensuring that all the goals are covered but at least one access of a pass
def accessing(requestStart, requestEnd, passId, obsReqId):
    if any(accessIdObsReq in satPasses[passId]['accessIds'] for accessIdObsReq in obsRequests[obsReqId]['accessIds']):
        if requestStart<=(satPasses[passId]['end'] - 30) and requestEnd>=(satPasses[passId]['start'] + 30):
            return selectedPass[passId]
    return 0
```

Definition of the accessing function

Implementation of the constraints 2

```
# Checking for all the observation requests
for aoiId in range(nAois):
    aux = 0
    for goalId in range(nGoals):
        for step in range(goals[goalId]['nSteps']):
            obsReqId = aoiId + nAois*aux + step + (goals[goalId]['nSteps'] - 1)*aoiId
            requestStart = step*goals[goalId]['duStep']
            requestEnd = step*goals[goalId]['duStep'] + goals[goalId]['rctHorizon']
            accessings = [accessing(requestStart, requestEnd, passId, obsReqId) for passId in range(nPasses)]
            sum_accessing = model.sum(accessings)
            if not str(sum_accessing) == "0":
                # 1) Ensuring that all the goals are accessed
                model.add_constraint(sum_accessing >= 1)
                for passId in range(nPasses):
                    if not str(accessings[passId]) == "0":
                        # 2) tStart covering constraint
                        model.add_constraint(requestStart <= tEnd[passId]-30)
                        # 3) tEnd covering constraint
                        model.add_constraint(requestEnd >= tStart[passId]+30)
        aux += goals[goalId]['nSteps']
```

At least one Access Request satisfied with a connection longer that 30s per Observation Request

Metrics to evaluate the solution

```
# Computation of the total cost
timeUses = [tEnd[id].solution_value - tStart[id].solution_value for id in selectedPassIds]
totalCost = len(selectedPassIds)*data['fixedPassCost'] + model.sum(timeUses)*data['passCostPerTimeUnit']
```

```
# Computation of the quality of the covering: It is defined as the relation of the
# requests covered over the total amount of requests
quality = 0
for aoiId in range(nAois):
    aux = 0
    for goalId in range(nGoals):
        for step in range(goals[goalId]['nSteps']):
            obsReqId = aoiId + nAois*aux + step + (goals[goalId]['nSteps'] - 1)*aoiId
            requestStart = step*goals[goalId]['duStep']
            requestEnd = step*goals[goalId]['duStep'] + goals[goalId]['rctHorizon']
            for passId in selectedPassIds:
                if any(accessIdObsReq in satPasses[passId]['accessIds'] for accessIdObsReq in obsRequests[obsReqId]['accessIds']):
                    if (requestStart<=tEnd[passId].solution_value-30) and (requestEnd>=tStart[passId].solution_value+30):
                        quality +=1
                        break
        aux += goals[goalId]['nSteps']

quality /= nObsRequests
```

Stage 4

- Objective:
  - Approach the multi-criteria problem and explore one or more possible approaches (aggregation of criteria, production of Pareto-optimal solutions, hierarchisation of criteria, etc.)

Multi-criteria optimisation

Multi-criteria optimisation

$$\min(\alpha \cdot C1 + (1-\alpha) \cdot (1-C2))$$

Objective function

Objective fcn. implementation

```
model.minimize(alpha*cost/maxcost+(1-alpha)*(1-quality))
```

C1 definition: cost function

```
# Cost calculation
cost = (model.sum(selectedPass)*data['fixedPassCost']
    + model.sum((satPasses[id]['end'] - satPasses[id]['start'])*selectedPass[id]
    for id in range(nPasses))*data['passCostPerTimeUnit'])

maxcost = 0
for id in range(nPasses):
    maxcost += fixedPassCost + (satPasses[id]['end'] - satPasses[id]['start'])*data['passCostPerTimeUnit']
```

C2 definition: quality function

```
# Quality calculation
quality = 0
for aoiId in range(nAois):
    aux = 0
    for goalId in range(nGoals):
        for step in range(goals[goalId]['nSteps']):
            obsReqId = aoiId + nAois*aux + step + (goals[goalId]['nSteps'] - 1)*aoiId
            requestStart = step*goals[goalId]['duStep']
            requestEnd = step*goals[goalId]['duStep'] + goals[goalId]['rctHorizon']
            for i in range(len(selectedPass)):
                if any(accessIdObsReq in satPasses[i]['accessIds'] for accessIdObsReq in obsRequests[obsReqId]['accessIds']):
                    quality += selectedPass[i]
                    break
        aux += goals[goalId]['nSteps']
quality /= nObsRequests
```

Definition of the constraints

Access constraints

1) The site must be visible by the satellite (accessId)
2) The connection must last at least 30s to ensure proper data collection

Pass constraints

1) The satellite must view the site in order to make a connection
2) The connection must last at least 30s to ensure proper data collection
3) If a pass is not taken, its ‘booking’ time is 0.

Requests constraints

1) An observation request is only satisfied if at least one of its access requests is satisfied
2) ~~All observation requests must be satisfied~~

Time constraints:

1) The connection start and end must be inside of the satellite pass duration
2) The connection must last at least 30s to ensure proper data collection

Analysis:
Effect of the variable α

α = 0

min(1-C2)

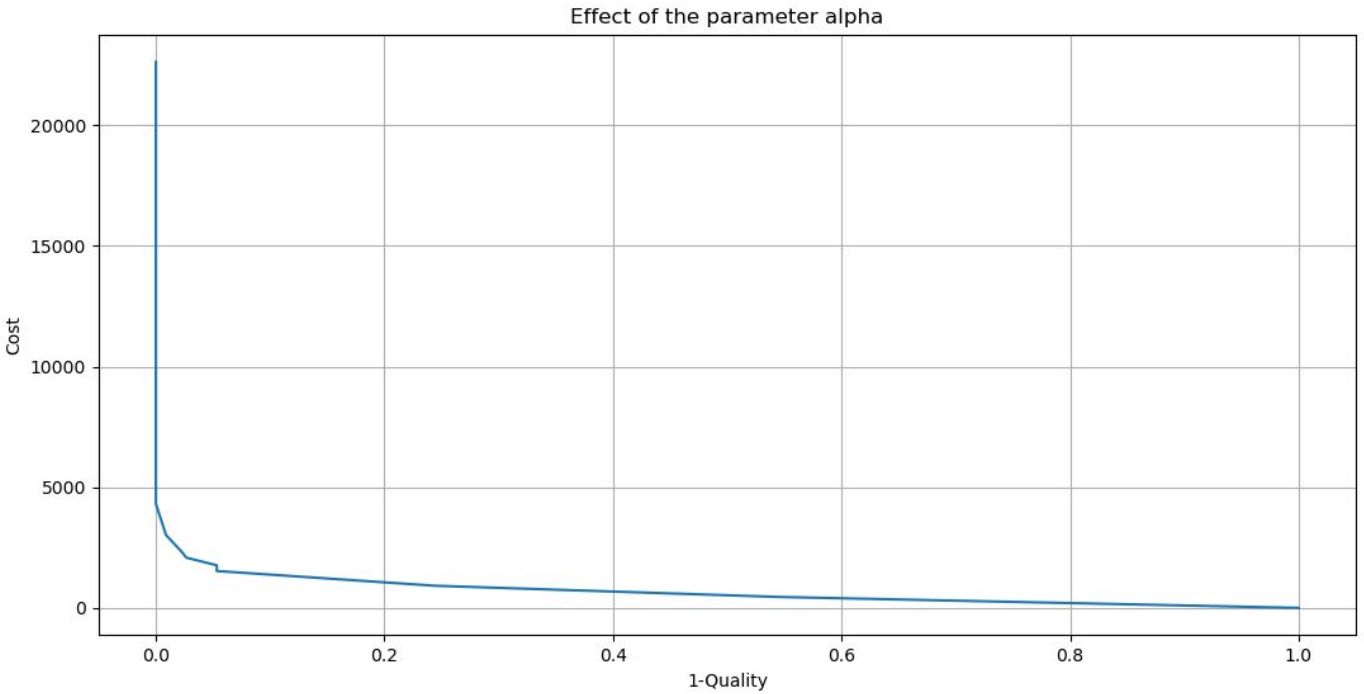
Maximum quality
(maximum cost)

α = 1

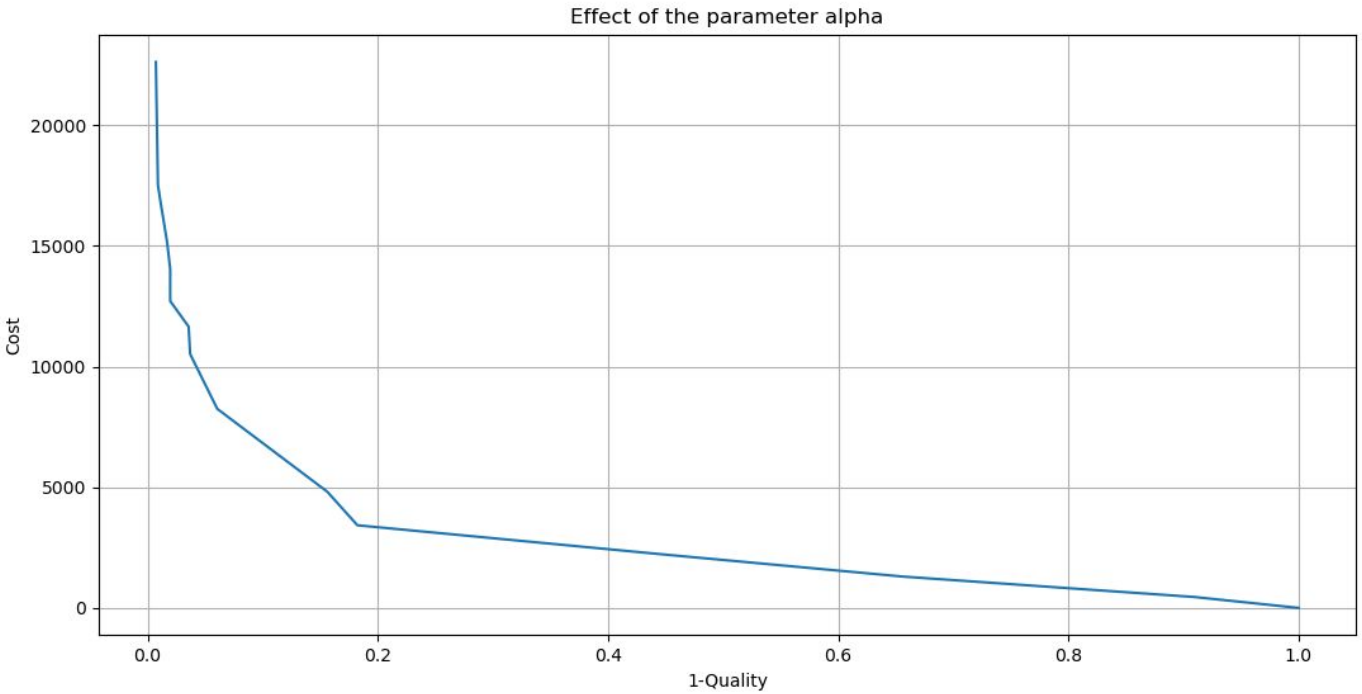
min(C1)

Minimum cost
(minimum quality)

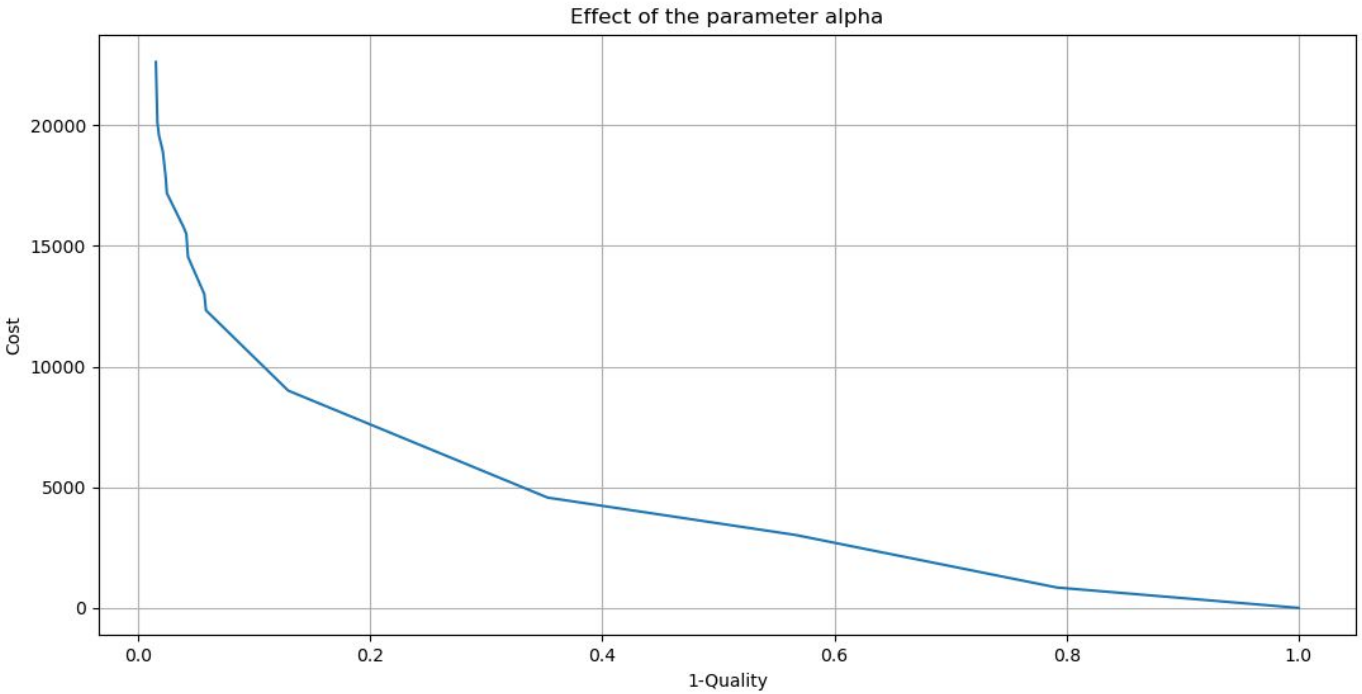
Data12


Effect of the parameter alpha
Cost
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0
1-Quality

Data4


Effect of the parameter alpha
Cost
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0
1-Quality

Data2


Effect of the parameter alpha
Cost
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0
1-Quality

Greedy algorithm

For every satellite pass:

nObsReq/Cost

Parameter to optimise

Loop to create the optimization variables:

```
for i in range(len(satPasses)):
    # Stock passes in new lists containing the data, start time and end time
    passesToCheck.append(satPasses[i])
    satStart.append(accesses[satPasses[i]['accessIds'][0]]['start'])
    satEnd.append(accesses[satPasses[i]['accessIds'][-1]]['end'])

    # Stock in a list of lists the obsReqs to whom the accessIds of the satellites refer to.
    # We optimise per obsReq, not accesReq
    obsReqList.append([])
    for aoiId in range(nAois):
        aux = 0
        for goalId in range(nGoals):
            for step in range(goals[goalId]['nSteps']):
                obsReqId = aoiId + nAois*aux + step + (goals[goalId]['nSteps'] - 1)*aoiId # For all obsReqs
                for accessIdObs in satPasses[i]['accessIds']:
                    if accessIdObs in obsRequests[obsReqId]['accessIds'] and obsReqId not in obsReqList[i]:
                        obsReqList[i].append(obsReqId)
```

Popping the covered obsReqs from the lists in the while loop

```
# Remove chosen satellite from the list of possible sats
passesToCheck.pop(maxRatioIndex)
satStart.pop(maxRatioIndex)
satEnd.pop(maxRatioIndex)

# Remove obsRequests from pool of requests to observe
for obsId in obsReqList[maxRatioIndex]: # for each obsId within the list of obsIds of the chosen satellite
    for sat in passesToCheck: # for each satellite in the satellites to watch
        if obsId in obsReqList[passesToCheck.index(sat)]: # if obsId is inside the "minilist" of obsReqs
            obsReqList[passesToCheck.index(sat)].pop(obsReqList[passesToCheck.index(sat)].index(obsId))

obsReqList.pop(maxRatioIndex)
```

Conclusions

- From the stage 1 it has been possible to define the metrics that make it possible to define the optimisation problem.
- From the stages 2 and 3 has been possible to see the improvement of the results when adding the new degree of liberty (selection of partial passes). This increased the computational complexity, thus, the computation time.
- The implementation of the multi-variable criteria in stage 4 made it possible to see the effect of prioritizing the maximisation of the quality or the minimisation of the cost.
- The Glutton algorithm implemented showed another alternative to face the problem, obtaining different optimal results.
- From this final two points it is possible to realise that for this type of optimisation problems there is not an algorithm that ensures the best solution for all the cases, but different alternatives can be checked in other to find a suitable solution.

THANKS!

Do you have any question?

OPTIMISATION PROJECT

Reactive coverage of a large area using a constellation of observation satellites