Joshua Woodruff – Optimized Financial Systems

OPTIMIZED FINANCIAL SYSTEMS

A Quick Introduction to IBM CPLEX CPO

CPO is IBM's Constraint Programming Optimizer

- In this section we will model and solve a small CPO problem using python and the docplex API.

Problem Description

- We want to schedule a team of consultants to implement a set of projects.
- Each project has an earliest start and a latest end and a minimum and maximum duration.
- Each project has a requirements for skills and a duration of that skill on the project.
- Each consultant has one or more skills they can perform.

Imports


```
from docplex.cp.model import CpoModel

import pandas

import pdb
```

Load the data

- We will be using the pandas package to load the data and hold the variables.

```
projects = pandas.read_csv('projects.csv')

skillRequirements = pandas.read_csv('skills.csv')

consultants = pandas.read_csv('consultants.csv')


consultantSkillProjects = consultants.merge(skillRequirements,
     how='inner', left_on='skill', right_on='skill')
```

Create the variables

- The base class in scheduling for CPO is the interval.
- It has a start and finish, can be optional, and has a size and duration.

```
starts = [(row.earliestStart,row.latestFinish - row.minDuration) for idx, row in projects.iterrows()]

ends = [(row.earliestStart + row.minDuration,row.latestFinish) for idx, row in projects.iterrows()]

lengths = [(row.minDuration,row.maxDuration) for idx, row in projects.iterrows()]

names = ['project_%s' % row.project for idx, row in projects.iterrows()]

projects['projIntv'] =
[m.interval_var(start=starts[idx],end=ends[idx],length=lengths[idx],name=names[idx]
) for idx, row in projects.iterrows()]
```

This is enough to run

- Always make sure that a CPO run has a limit of some kind. Most commonly they are time, fails, or branches.

```
obj = m.max([m.end_of(row.projIntv) for idx, row in
projects.iterrows()])

m.add(m.minimize(obj))

sol = m.solve(TimeLimit=20)
```

Add the skills to the projects

- The **span** keyword enforces the beginning and end of an interval with other intervals

```
lengths = [row.weeks for idx, row in skillRequirements.iterrows()]

names = ['project_%s_%s' % (row.project,row.skill) for idx, row in
skillRequirements.iterrows()]

skillRequirements['projSkillIntv'] = [m.interval_var(length=lengths[idx],
      name=names[idx]) for idx, row in skillRequirements.iterrows()]

for idx, row in projects.iterrows():

      m.add(m.span(row.projIntv,[row2.projSkillIntv for idx2, row2 in
            skillRequirements[skillRequirements.project ==
            row.project].iterrows()]))
```

Assign the consultant to the project

- The **alternative** keyword chooses one (or n) intervals in an array to have the same solution as a specified interval

```
lengths = [row.weeks for idx, row in consultantSkillProjects.iterrows()]

names = ['%s_%s_%s' % (row.consultant,row.project,row.skill) for idx, row in
consultantSkillProjects.iterrows()]

consultantSkillProjects['consProjSkillIntv'] = [m.interval_var(optional=True,
      length=lengths[idx], name=names[idx]) for idx, row in
      consultantSkillProjects.iterrows()]

for idx, row in skillRequirements.iterrows():

      m.add(m.alternative(row.projSkillIntv,[row2.consProjSkillIntv for idx2, row2
            in consultantSkillProjects[(consultantSkillProjects.project ==
            row.project)  & (consultantSkillProjects.skill ==
            row.skill)].iterrows()]))
```

Keep the consultant from doing more than one thing at a time

- The **no_overlap** keeps the intervals in an array from overlapping

```
for consultant in consultantSkillProjects.consultant.unique():

        m.add(m.no_overlap([row.consProjSkillIntv for idx, row in
              consultantSkillProjects[consultantSkillProjects.consultant ==
              consultant].iterrows()]))
```

Display it all

- The solution indexed by the variable gives the start, end, and size.
- When an interval is optional, check for length.

```
projects['start'],projects['end'],projects['duration'] =
list(zip(*[sol[row.projIntv] for idx, row in projects.iterrows()]))

skillRequirements['start'],skillRequirements['end'],skillRequirements['duration'] =
list(zip(*[sol[row.projSkillIntv] for idx, row in skillRequirements.iterrows()]))

for idx, row in consultantSkillProjects.iterrows():
        solVal = sol[row.consProjSkillIntv]
        if len(solVal) == 3:
                consultantSkillProjects.loc[idx,'start'] = solVal[0]
                consultantSkillProjects.loc[idx,'end'] = solVal[1]
                consultantSkillProjects.loc[idx,'duration'] = solVal[2]
```