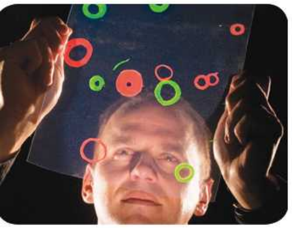
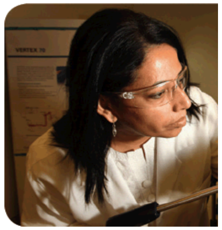
SAND2015-4321C





Mixed-Integer Programming of The Constellation Scheduling Problem

WG-8 Space Acquisition, Testing and Operations

June 25th, 2015

Sandia National Laboratories

Exceptional service in the national interest

Christopher G. Valicka, PhD*
M. Danny Rintoul, PhD
William E. Hart, PhD

*Correspondence: cgvalic@sandia.gov

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2015-*****

Overview

- **A remote sensing constellation scheduling problem**
- What is needed in a constellation scheduling tool?
- Modeling with Pyomo
  - Preliminary results
- Future work

Optimization Models for Managing Mobile Sensors

- Problem:
  - Manage a collection of mobile sensors that are scheduled to monitor physical locations in space and time
    - Examples: stationary video cameras, drones, **satellites**

- Challenge:
  - Sensors have highly flexible capabilities

- Assumption:
  - The performance of the mobile sensors will be evaluated w.r.t a fixed set of *activities*

How is an Activity Defined?

- **Start time**
  - **Time window:** list of potential start times
  - **Duration:** fixed and known before building schedule

- **Configuration**: operational configuration needed to observe a location
  - **Physical location**:
    - The location that needs to be *observed*; precise requirements depend on the sensor technology

- **Quality**: a minimum observation quality. Impacted by sensor location, time of day, etc.

- **Priority**: importance relative to other activities

- **Category:** hierarchical importance (required, essential, desired)

Activity Categories

- Activities are categorized into the following:
  - **Category 1:** Unique to a given sensor. Must be scheduled.
    - Example: activities scheduled for the safety and proper operation of a sensor
  - **Category 2:** Cannot be scheduled during Category 1 activities. In general, of high priority. In some cases, preempted by a Category 3 activity
    - Example: periodic sensor calibration activities
  - **Category 3:** Cannot be scheduled during Category 1 activities. The vast majority of activities to be scheduled. In general, lower priority than Category 2 activities.
    - Example: observation activities

Overview

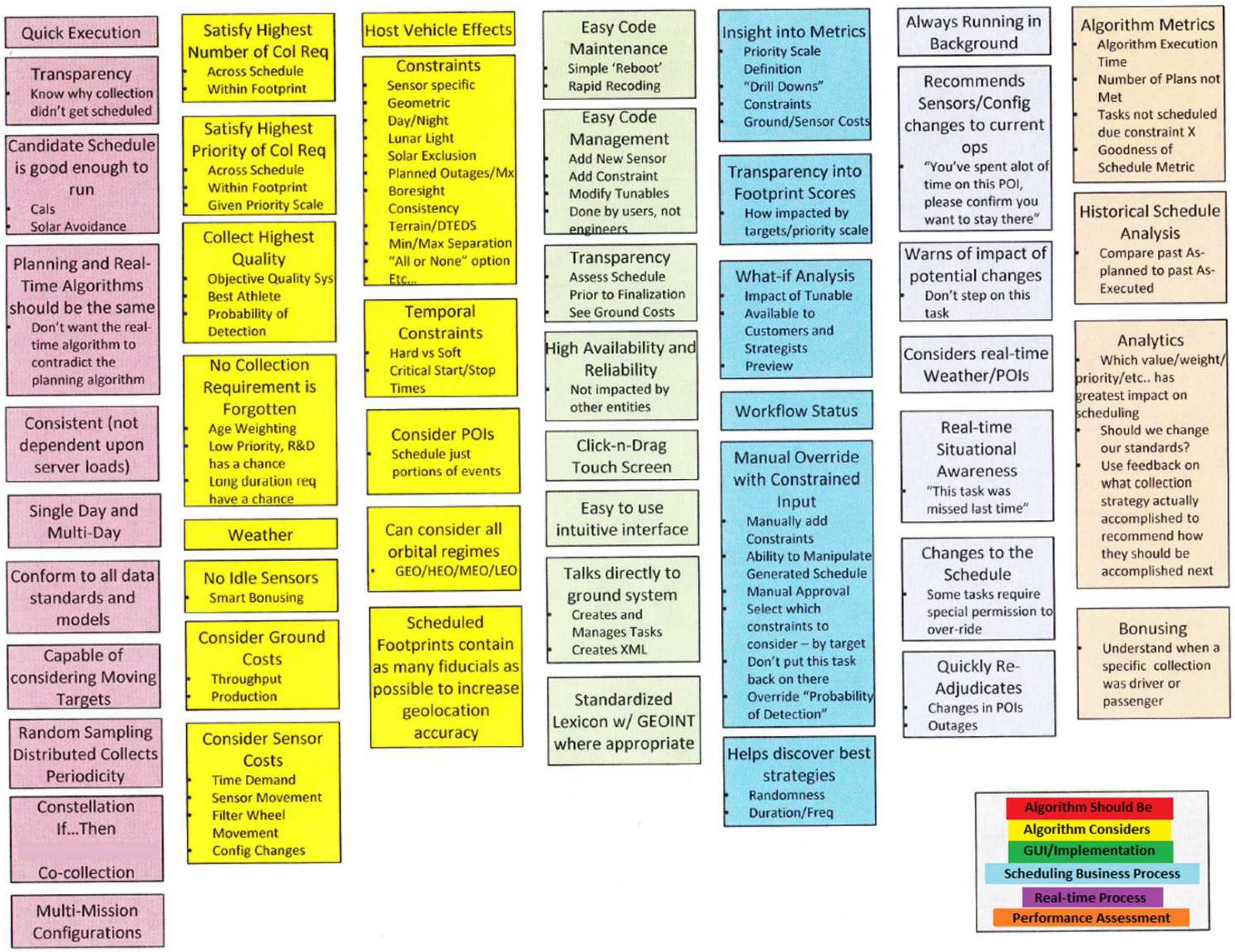
- A remote sensing constellation scheduling problem

- **What is needed in a constellation scheduling tool?**

- Modeling with Pyomo
  - Preliminary results

- Future work

A Constellation Scheduling Tool Should Be…



What to Optimize- What is the Strategy?

- Minimize the number of sensors needed to observe a given set of activities
- Minimize the amount of time it takes to observe a given set of activities
- Minimize the number of schedule gaps
- Maximize the average or total priority of scheduled activities
- Maximize the quality of scheduled activities
- …, etc.
- Hybrid strategies

Initial work has focused on scheduling the largest number of high priority, high quality activities

Related Work

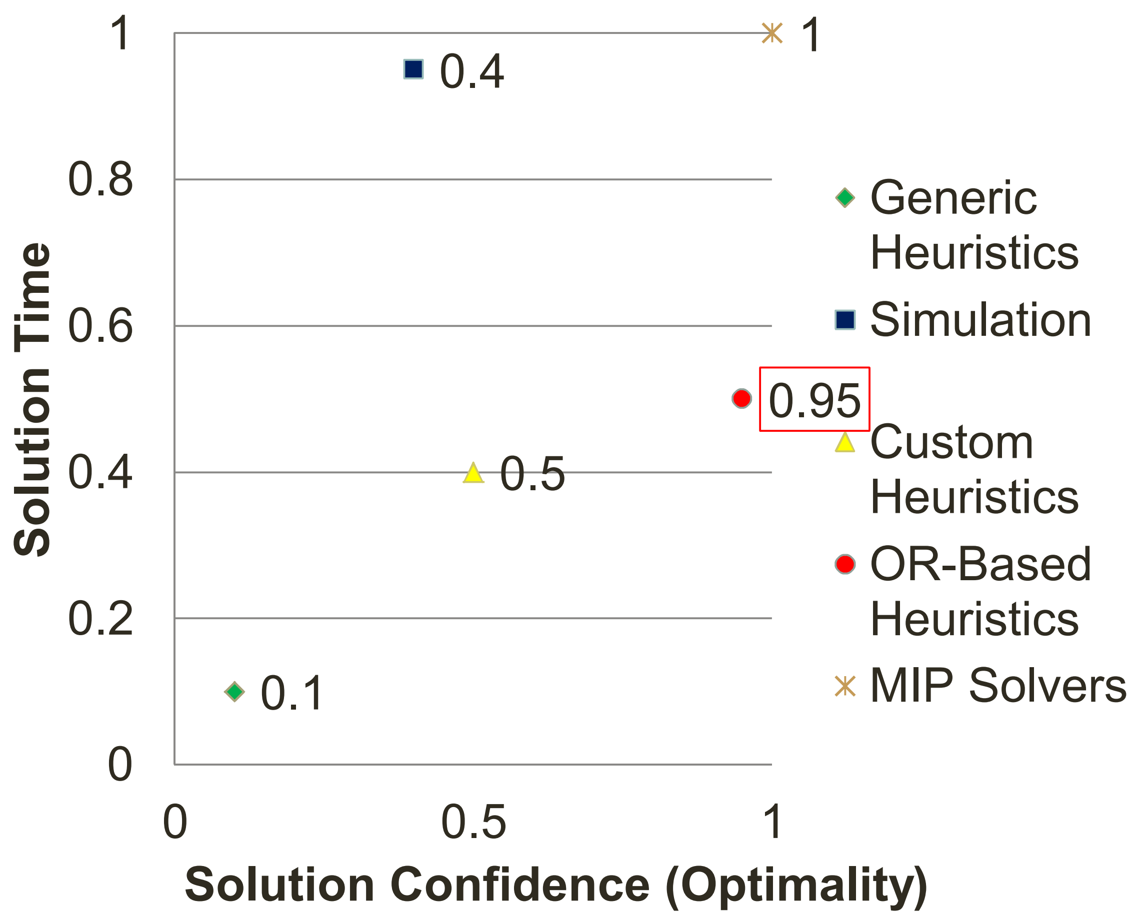
- Community practice uses rule-based techniques

- Academic research is divided into two camps
  - Heuristics (Globus et. al 2004)
    - Genetic algorithms (Lining et. Al 2009)
    - Simulated annealing (Peng et. al 2011)
    - Greedy local (Dungan et. al 2011)
    - Ant colony optimization (Wang et. al 2009)

  - Exact methods (less research)
    - Integer programming (Liao, 2007)
      - Simple models

Why do Optimization Modeling?

- The goal is to manage the tradeoff between solution time and optimality
- Many national security problems require near real-time answers
  - 95% solution is acceptable



Benefits of OR-Based Heuristics

- Scheduling problems can be notoriously hard to solve

- An OR-based heuristic:
  - Apply a MIP solver using an optimality tolerance (e.g. 5%)
  - Final solution guaranteed to be near optimal
  - Small optimality tolerances can significantly reduce time to solution

- MIPs facilitate exploration of alternate formulations
  - Quickly assess different formulations
    - Objective functions, constraint equations

- Sensitivity analysis
  - Determine active/limiting constraints
  - Rigorously determine the effects of changing objectives, constraints, and decision variables

Overview

- A remote sensing constellation scheduling problem
- What is needed in a constellation scheduling tool?
- **Modeling with Pyomo**
  - Preliminary results
- Future work

Pyomo Overview

Idea: a framework for Python used to formulate optimization models

- Provide a natural syntax to describe mathematical models
- Formulate large models with a concise syntax
- Separate modeling and data declarations
- Enable data import and export in commonly used formats

Highlights:

- Python provides a clean, intuitive syntax
- Python scripts provide a flexible context for exploring the structure of Pyomo models

```
# simple.py
import  pyomo.environ as pyomo

M = ConcreteModel()
M.x1 = Var()
M.x2 = Var(bounds=(-1,1))
M.x3 = Var(bounds=(1,2))
M.o  = Objective(expr=M.x1**2 +
            (M.x2*M.x3)**4 + \
                 M.x1*M.x3 + \
M.x2*sin(M.x1+M.x3) +
                       M.x2)
model = M
```

Pyomo Example: The Knapsack Problem

$$\max \quad \sum_{i=1}^{N} v_i x_i$$

$$s.t. \quad \sum_{i=1}^{N} w_i x_i \leq W_{\max}$$

$$x_i \in \{0,1\}$$

Item	Weight	Value
hammer	5	8
wrench	7	3
screwdriver	4	6
towel	3	11
Max weight:	14	

Given the set of items, each with a weight and a value, determine which to place in a knapsack so that the total weight is less than or equal to $W_{\max}$ and so that the total value is a large as possible.

Solution: The Knapsack Problem

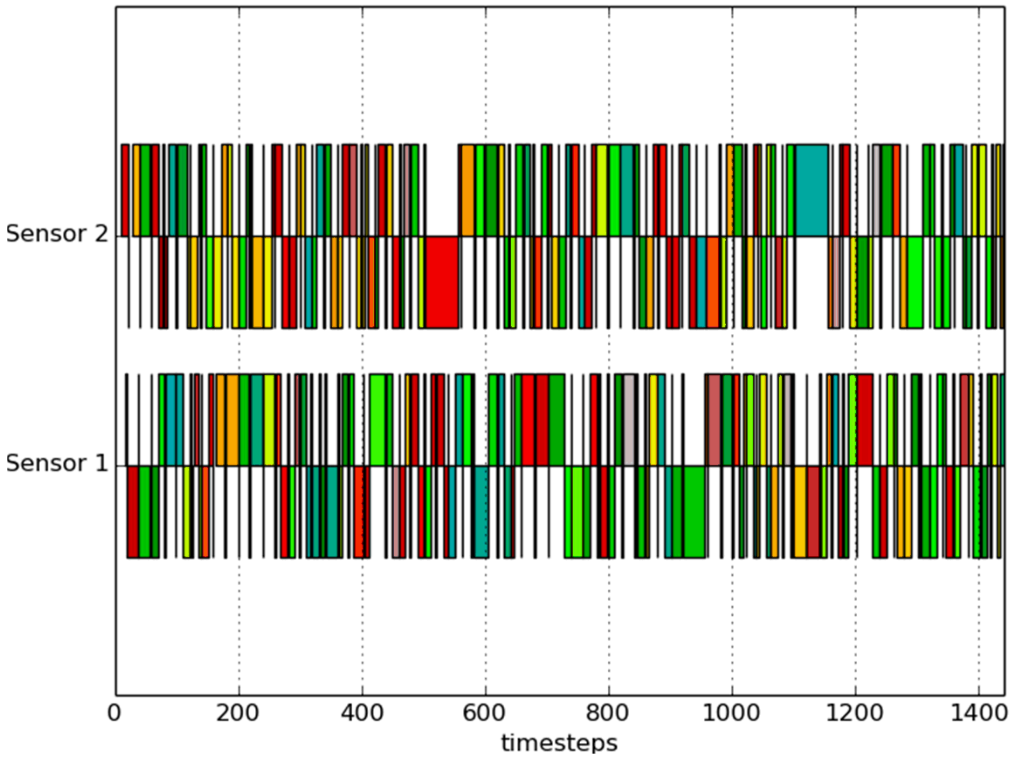
```
from pyomo.environ import *

v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14

model = ConcreteModel()
model.ITEMS = v.keys()
model.x = Var( model.ITEMS, within=Binary )

model.value = Objective(
   expr = sum( v[i]*model.x[i] for i in model.ITEMS ),
   sense = maximize )

model.weight = Constraint(
   expr = sum( w[i]*model.x[i] for i in model.ITEMS ) <= W_max )
```

How does a MIP solver find an optimal solution?

- **Linear-programming branch-and-bound algorithm**
  - Initial formulation is linear
    - *Relax* the decision variables' integrality constraint
  - Attempt to solve the linear program
    - Linear programs will either be: **infeasible**, have **a set of optimal solutions**, or have **exactly one optimal solution**
  - If solution is found, **branch** on a fractional variable (two sub-MIPs)
  - Continue and in doing so create a **search tree**
    - New MIPs from branched variables: **node**
    - Original MIP: **root node**
  - Integer solutions become incumbents; node becomes permanent **leaf**
    - Discard infeasible LP solutions and incumbent inferior solutions
  - Incumbents are valid upper/lower bounds, best incumbent is **best bound**, difference is optimality gap

Preliminary Results – Two MIP Formulations

- Solutions within 99.5% optimal quickly (~10s of seconds – minutes)
  - provably optimal can takes hours
- Produce schedules over an arbitrary timeframe
  - Re-plan or rebuild schedules after adhoc changes
- Create schedules for multiple sensors, simultaneously
- Guarantees Category 1 activities make the schedule



- Alternative is faster, facilitates transition costs, but more restrictive

Overview

- A remote sensing constellation scheduling problem
- What is needed in a constellation scheduling tool?
- Modeling with Pyomo
  - Preliminary results
- **Future work**

Future Work- Quality scale: $q_{i,k,t}$

- Build a composite quality score, tentatively normalized between 0 and 1 based on:
  - Geometric access
  - Coverage
  - Probability of detection (PD)
  - Closely Spaced Objects (CSO)

- Quality score will also incorporate weather

- Certain portions typically built in advance with others updated near scheduling time

- Collaborating with sensor performance experts

Future Work: Stochastic Programming Scenarios

- We are currently considering generating scenarios based on:
  - Uncertain activity timewindows
  - New high priority activities (go/no-go activities)
  - Uncertainty in quality scores (weather)
- Key challenges:
  - What data to model uncertainties do we have or can we obtain?
    - Lots of realization data, missing forecast data
  - How frequently is uncertain information updated?

Future Work- Related Efforts

- Currently drawing from remote sensor scheduling examples
  - Example sensor activities (durations, configurations, scheduling constraints)
- Formulation aims to be sensor agnostic
  - Variety of Mobility constraints (orbits)
  - Different sensor and mission types
- Also working on computational geometry algorithms to group and optimally position activity locations
  - Multiple sensors observing a single activity
  - Sub-activity partitioning problems
- Additional activity constraints and solver tuning

Backup Slides

Pyomo is Open Source

- Transparent and reliable (developed at Sandia w/ external collaborators)

- Fosters community involvement
  - Extend the modeling language
  - Develop new solvers / algorithms
  - Interface with additional external utilities

- Flexible licensing
  - Pyomo released under 3-clause BSD license
  - No restrictions on deployment or commercial use

- Interfaces with open-source and commercial solvers
  - IPOPT, GLPK, CBC, PICO, GUROBI, CPLEX

Going forward, GUROBI will replace CPLEX as the preferred solver

For More Information

See the new Pyomo homepage

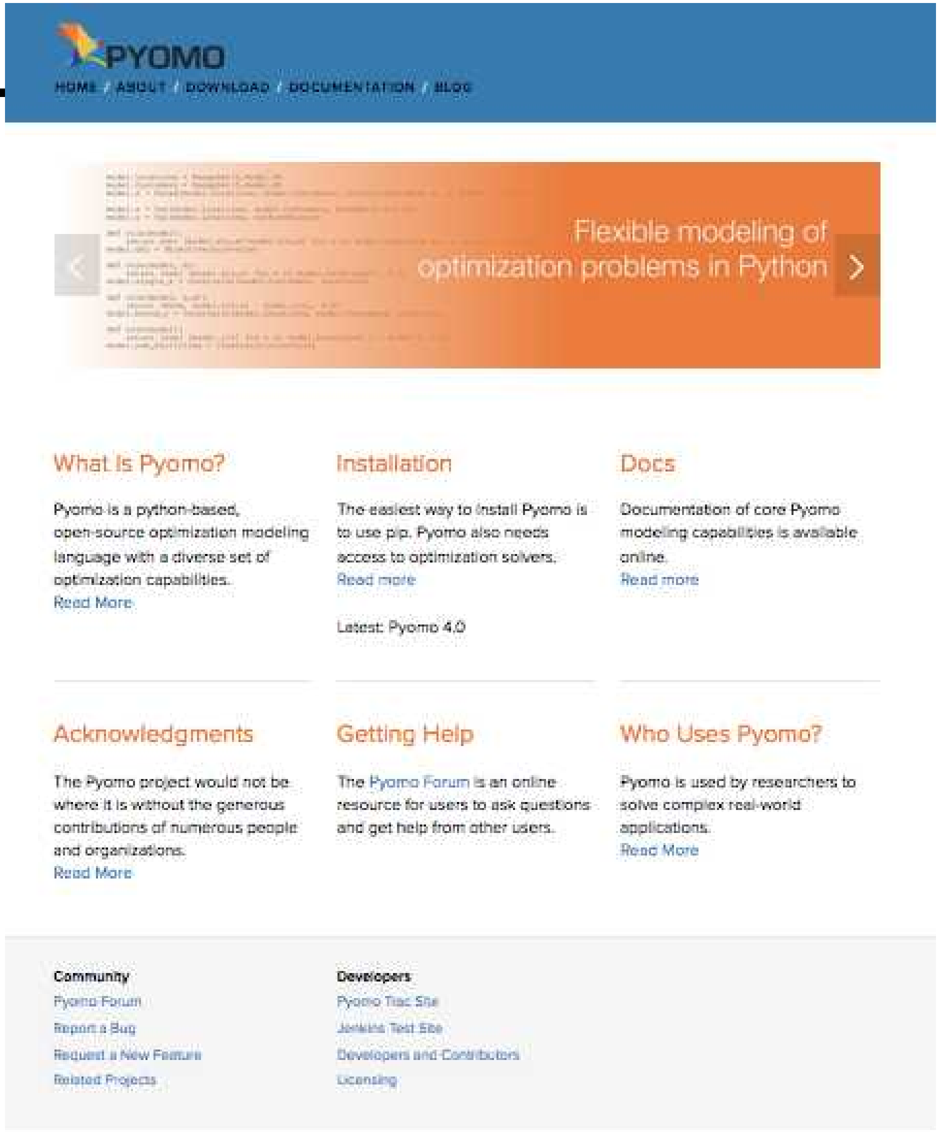
- www.pyomo.org

The Pyomo homepage provides a portal for:

- Online documentation
- Installation instructions
- Help information
- Developer links

Coming soon:

- A gallery of simple examples



Constellation Scheduling Mixed Integer Program

$$
\begin{aligned}
\max \quad & \sum \frac{\alpha(\delta_{i,k,t} p_k d_k q_{k,t})}{\sum_{k \in K} p_k d_k q_k^*}, && i \in I,\ k \in K,\ t \in T \\
& w_k = \sum_{i \in I,\ t \in T} \delta_{i,k,t}, && \forall k \in K \\
& w_k = 1, && \forall k \in K_1 \\
s.t. \quad & w_k \le 1, && \forall k \in K \setminus K_1 \\
& q_k^{'} \delta_{i,k,t} \le \sum_{i \in I,\ k \in K,\ t \in T} q_{i,k,t} \delta_{i,k,t}, && \forall k \in K_1 \\
& \sum_{k \in K} \delta_{i,k,t} \le 1 - \sum_{k \in K, t \in C(k,t)} \delta_{i,k,t}, && \forall i \in I,\ t \in T \\
& \delta_{i,k,t} \in \{0,1\} && i \in I,\ k \in K,\ t \in T
\end{aligned}
$$

- Where:
  - $\delta_{i,k,t}$: whether activity *k* starts at time *t* on sensor *i*
  - $q_{i,k,t}$: quality associated with starting activity *k* at time *t* on sensor *i*
  - d_k, p_k : duration and priority of activity *k*
  - $C(k,t)$: set of feasible start times for activity *k* before time *t*
  - α : scaling constant (e.g. 100)

Constellation Scheduling Knapsack Formulation

$$\max \quad \sum \frac{\alpha(\delta_{i,a,k} p_k d_k q_{i,a,k})}{\sum_{k \in K} p_k d_k q_k^*}, \qquad i \in I,\ a \in A,\ k \in K$$

$$
\begin{aligned}
s.t. \quad & w_k = \sum_{i \in I,\, t \in T} \delta_{i,a,k}, && \forall a \in A \\
& w_k = 1, && \forall k \in K_1 \\
& w_k \le 1, && \forall k \in K \setminus K_1 \\
& W_{\min} \le U_{i,k} \le W_{\max} && \forall i \in I, \forall k \in K \\
& \sum_{k \in K} U_{i,k} \le T_{horizon} && \forall i \in I \\
& \sum_{a \in A, k \in K} \delta_{i,a,k} + |K| \le T_{horizon} && \forall i \in I \\
& \sum_{a \in A} \delta_{i,a,k} d_k \le U_{i,k} && \forall i \in I \\
& \delta_{i,a,k} \in \{0,1\} && i \in I,\ a \in A,\ k \in K
\end{aligned}
$$

- Where:
  - $\delta_{i,a,k}$: whether activity *a* starts in knapsack *k* on sensor *i*
  - $U_{i,k}, W_{\max}, W_{\min}$: duration of knapsack *k* of sensor *I*; max/min knapsack duration
  - $q_{i,a,k}$: quality associated with starting activity *a* in knapsack *k* of sensor *i*
  - d_k, p_k : duration and priority of activity *k*
  - α : scaling constant (e.g. 100)

Why Model within Python?

Full-Featured Library

- Language features includes functions, classes, looping, namespaces, etc
- Introspection facilitates the development of generic algorithms
- Python’s clean syntax facilitates **rapid prototyping**

Open Source License

- No licensing issues w.r.t. the language itself

Extensibility and Robustness

- Highly stable and well-supported

Support and Documentation

- Extensive online documentation and several excellent books
- Long-term support for the language is not a factor

Standard Library

- Includes a large number of useful modules

Portability

- Widely available on many platforms