Tutorial - Pyomo

Janusz Granat

Outline

- Introduction
- Concrete models
- Abstract models
- Multicriteria and pyomo - introduction

Introduction

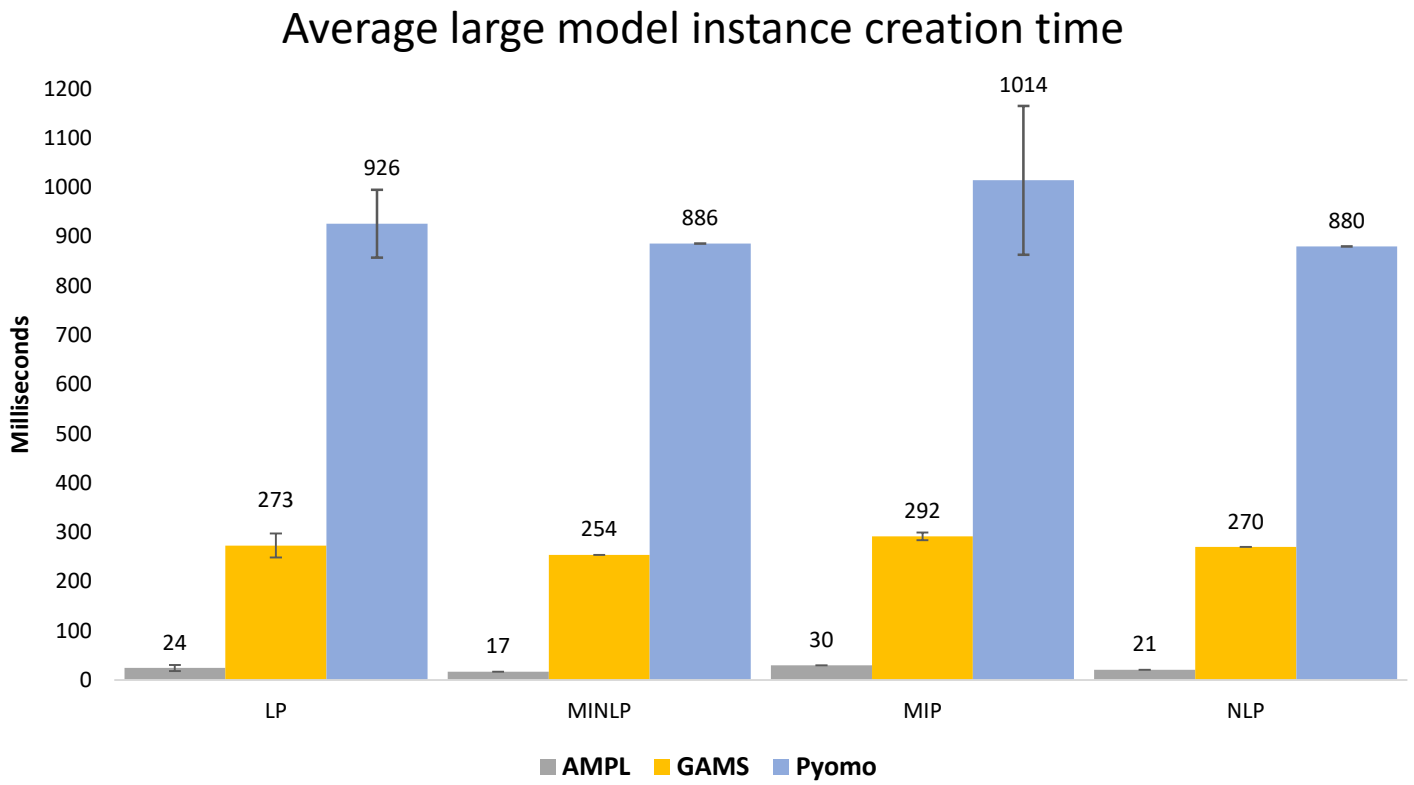
Modeling languages

- AMPL
- GAMS
- AIMMS
- ….

Why Pyomo?

- The Python Optimization Modeling Objects (Pyomo) software package supports the definition and solution of optimization applications using the Python scripting language.
- Pyomo is not intended to facilitate modeling better than existing, primarily commercial AML tools.
- It supports a different modeling approach in which the software is designed for flexibility, extensibility, portability, and maintainability.
- Direct integration with a high-level programming language is needed to allow modelers to leverage modern programming constructs, ensure cross-platform portability, and access the broad range of functionality found in standard software libraries.
- Pyomo incorporates the central ideas in modern AMLs, e.g., differentiation between abstract models and concrete problem instances.

Performance – Pyomo vs GAMS vs AMPL



Vaidas Jusevičius, Remigijus Paulavičius International EURO Mini Conference, 2019, Modelling and Simulation of Social-Behavioural Phenomena in Creative Societies

Performance – Pyomo vs GAMS vs AMPL

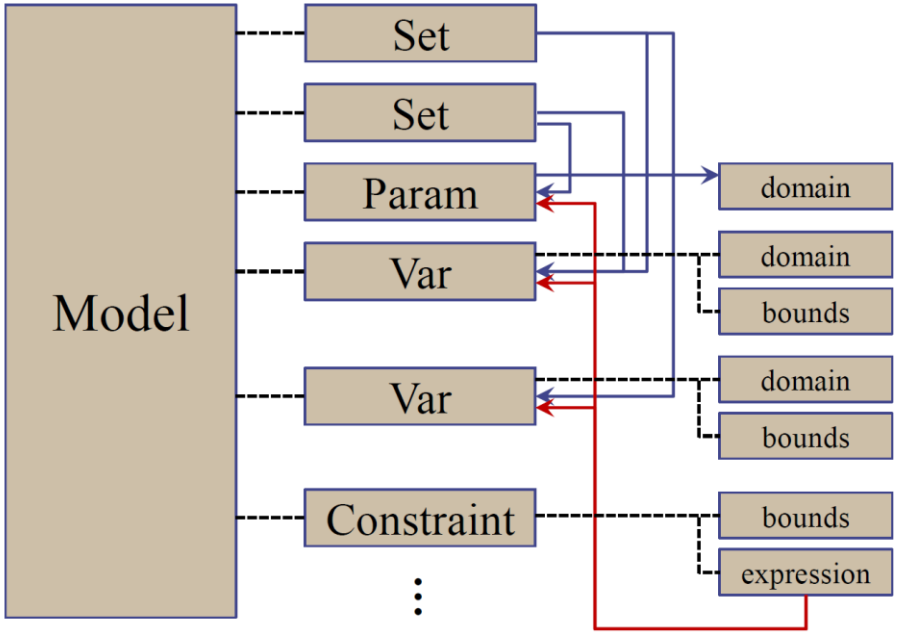
- Model building – no difference
- Model instance generation – GAMS and AMPL is better
- Solving the model instance – no difference

Python

- Python is a powerful high-level programming language that has a very clear, readable syntax and intuitive object orientation.
- Python Packages (e.g.):
  - SciPy (Scientific Python for mathematics and engineering)
  - Numpy (Numeric array package)
  - Matplotlib (2D plotting library)
  - Pandas (Data structures and analysis)

Pyomo basic ideas

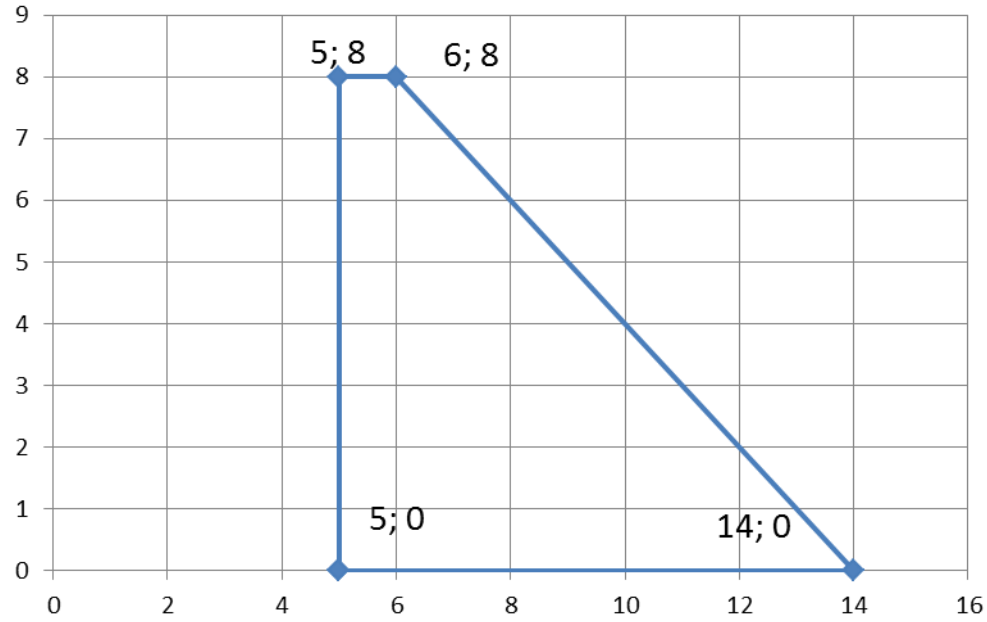
- Pyomo incorporates the central ideas in modern AMLs, e.g., differentiation between abstract models and concrete problem instances.
- The fundamental objects used to build models are *Components*



Concrete Model

The first model

- Decision variables: $p\ w$
- Objective: $p + 5w$
- Constraints:
  - $p + w \leq 14$
  - $p \geq 5$
  - $w \leq 8$
  - $p \geq 0, w \geq 0$



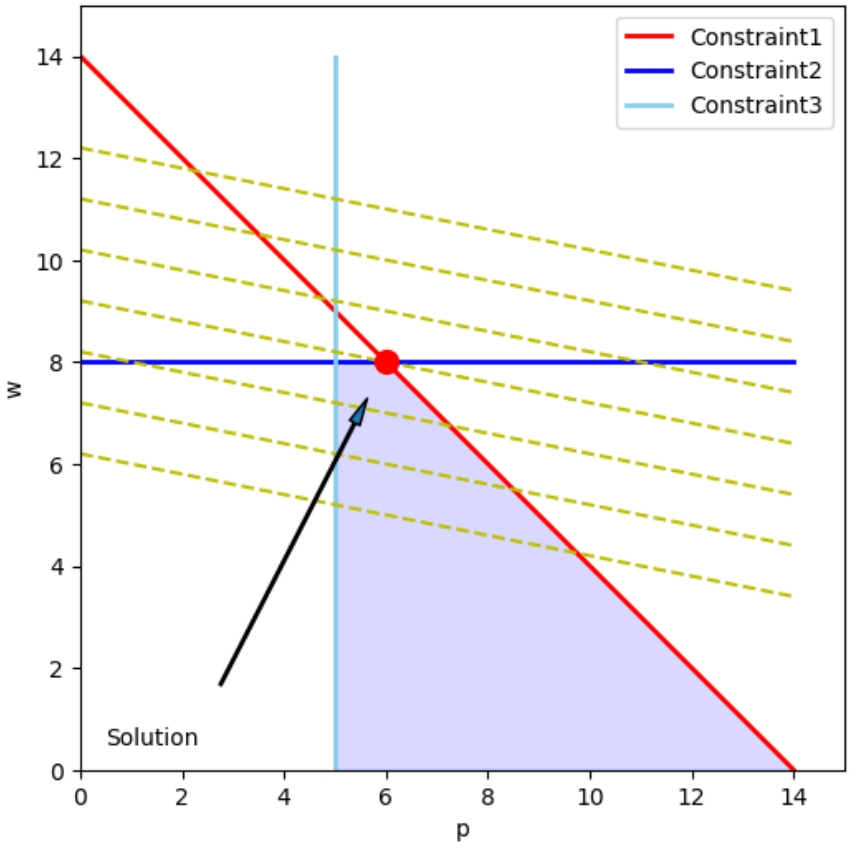
The first model - pyomo

```
from pyomo.environ import *
model = ConcreteModel()
# Variables
model.p = Var(within=NonNegativeReals)
model.w = Var(within=NonNegativeReals)
# Objective
model.obj =
Objective(expr=model.p+5*model.w,sense=m
aximize)
# Constraints
model.con1 = Constraint(expr=model.p +
model.w  <= 14)
model.con2 = Constraint(expr=model.p  >= 5)
model.con3 = Constraint(expr=model.w <= 8)
#call solver
opt = SolverFactory('glpk')
opt.solve(model)
```

- Decision variables: $p\ w$
- Objective: $p + 5w$
- Constraints:
  - $p + w \leq 14$
  - $p \geq 5$
  - $w \leq 8$
  - $p \geq 0, w \geq 0$

The first model - solution

```
from pyomo.environ import *
model = ConcreteModel()
# Variables
model.p = Var(within=NonNegativeReals)
model.w = Var(within=NonNegativeReals)
# Objective
model.obj =
Objective(expr=model.p+5*model.w,sense=m
aximize)
# Constraints
model.con1 = Constraint(expr=model.p +
model.w  <= 14)
model.con2 = Constraint(expr=model.p  >= 5)
model.con3 = Constraint(expr=model.w <= 8)
#call solver
opt = SolverFactory('glpk')
opt.solve(model)
```



Abstract Model

Production Models: Maximizing Profits

Parameters

P- a set of products

a_j - tons per hour of product j, for each $j \in P$

b - hours available at the mill

c_j - profit per ton of product j, for each j ∈P

u_j - maximum tons of product j, for each j ∈P

Variables

x_j – tons of product j to be made, for each j ∈P

Objective

$$\max_{x_j \in X} \sum_{j \in P} c_j \, x_j$$

Constraints

$\sum_{j \in P}(1/a_j)x_j \leq b$ (Time constraints)

$0 \leq x_j \leq u_j \; for \; each \; j \in P$ (Limit constraint)

Production Models - data

```
set P := bands coils;

param: a c u :=
bands 200 25 6000
coils 140 30 4000 ;

param b := 40;
```

Pyomo – abstract model

```
from pyomo.environ import *

model = AbstractModel()
# Sets
model.P = Set()
# Parameters
model.a = Param(model.P)
model.b = Param()
model.c = Param(model.P)
model.u = Param(model.P)
# Variables
model.X = Var(model.P)
# Objective
def objective_rule(model):
    return summation(model.c, model.X)
model.Total_Profit = Objective(rule=objective_rule, \
                               sense=maximize )

# Time Constraint
def time_rule(model):
    return summation(model.X, denom=model.a ) <= model.b
model.Time = Constraint(rule=time_rule)
# Limit Constraint
def limit_rule(model, j):
    return (0, model.X[j], model.u[j])
model.Limit = Constraint(model.P, rule=limit_rule)

data = DataPortal()
data.load(filename='prod.dat')
instance = model.create_instance(data)

instance.pprint()
```

Parameters

P- a set of products

a_j - tons per hour of product j, for each $j \in P$

b - hours available at the mill

c_j - profit per ton of product j, for each j ∈P

u_j - maximum tons of product j, for each j ∈P

Variables

x_j - tons of product j to be made, for each j ∈P

Objective

$\max_{x_j \in X} \sum_{j \in P} c_j \, x_j$

Constraints

$\sum_{j \in P}(1/a_j)x_j \leq b$ (Time constraints)

$0 \leq x_j \leq u_j \; for \; each \; j \in P$ (Limit constraint)

Pyomo – model instance

```
from pyomo.environ import *

model = AbstractModel()
# Sets
model.P = Set()
# Parameters
model.a = Param(model.P)
model.b = Param()
model.c = Param(model.P)
model.u = Param(model.P)
# Variables
model.X = Var(model.P)
# Objective
def objective_rule(model):
    return summation(model.c, model.X)
model.Total_Profit = Objective(rule=objective_rule, \
                               sense=maximize )

# Time Constraint
def time_rule(model):
    return summation(model.X, denom=model.a ) <= model.b
model.Time = Constraint(rule=time_rule)
# Limit Constraint
def limit_rule(model, j):
    return (0, model.X[j], model.u[j])
model.Limit = Constraint(model.P, rule=limit_rule)

data = DataPortal()
data.load(filename='prod.dat')
instance = model.create_instance(data)

instance.pprint()
```

Parameters

P- a set of products

a_j - tons per hour of product j, for each $j \in P$

b - hours available at the mill

c_j - profit per ton of product j, for each j ∈P

u_j - maximum tons of product j, for each j ∈P

Variables

x_j - tons of product j to be made, for each j ∈P

Objective

$\max_{x_j \in X} \sum_{j \in P} c_j \, x_j$

Constraints

$\sum_{j \in P}(1/a_j)x_j \leq b$ (Time constraints)

$0 \leq x_j \leq u_j \; for \; each \; j \in P$ (Limit constraint)

Pyomo – solution

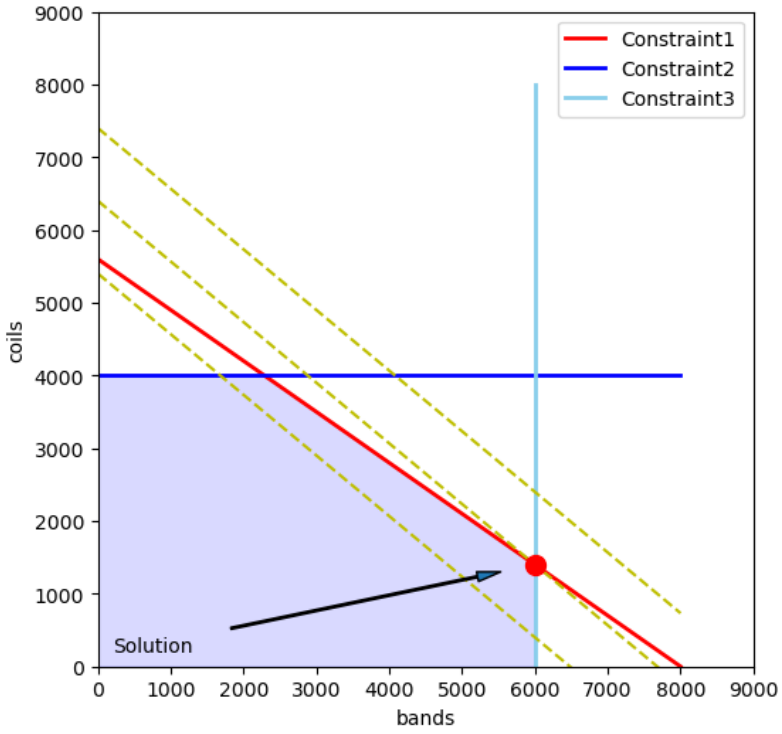
```
from pyomo.environ import *

model = AbstractModel()
# Sets
model.P = Set()
# Parameters
model.a = Param(model.P)
model.b = Param()
model.c = Param(model.P)
model.u = Param(model.P)
# Variables
model.X = Var(model.P)
# Objective
def objective_rule(model):
    return summation(model.c, model.X)
model.Total_Profit = Objective(rule=objective_rule, \
                               sense=maximize )

# Time Constraint
def time_rule(model):
    return summation(model.X, denom=model.a ) <= model.b
model.Time = Constraint(rule=time_rule)
# Limit Constraint
def limit_rule(model, j):
    return (0, model.X[j], model.u[j])
model.Limit = Constraint(model.P, rule=limit_rule)

data = DataPortal()
data.load(filename='prod.dat')
instance = model.create_instance(data)

instance.pprint()
```



Pyomo installation

1. Install anaconda (https://docs.anaconda.com/)
2. Create dedicated Pyomo environment
3. Install pyomo

   conda install -c conda-forge pyomo
4. Install solvers

   conda install -c conda-forge ipopt glpk

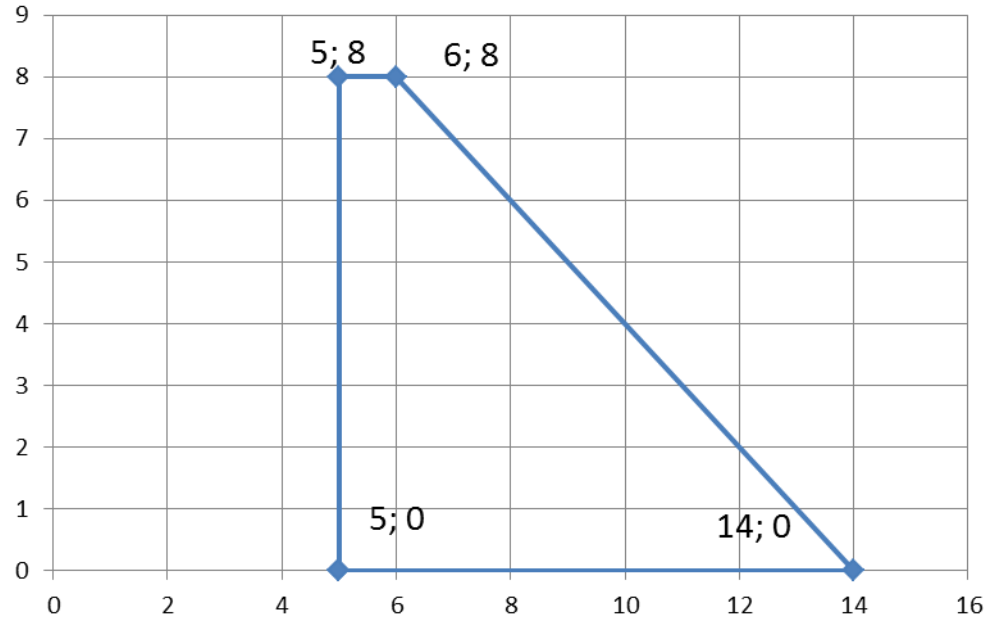
Documentation

- Bynum, M.L., Hackebeil, G.A., Hart, W.E., Laird, C.D., Nicholson, B., Siirola, J.D., Watson, J.-P., Woodruff, D.L.. Pyomo – Optimization Modeling in Python. Third Edition. Springer, 2021.
- Nicholson, Bethany L., Laird, Carl Damon, Siirola, John Daniel, Watson, Jean-Paul, and Hart, William E.. 2016. "Pyomo Tutorial.". United States. https://www.osti.gov/servlets/purl/1376827.

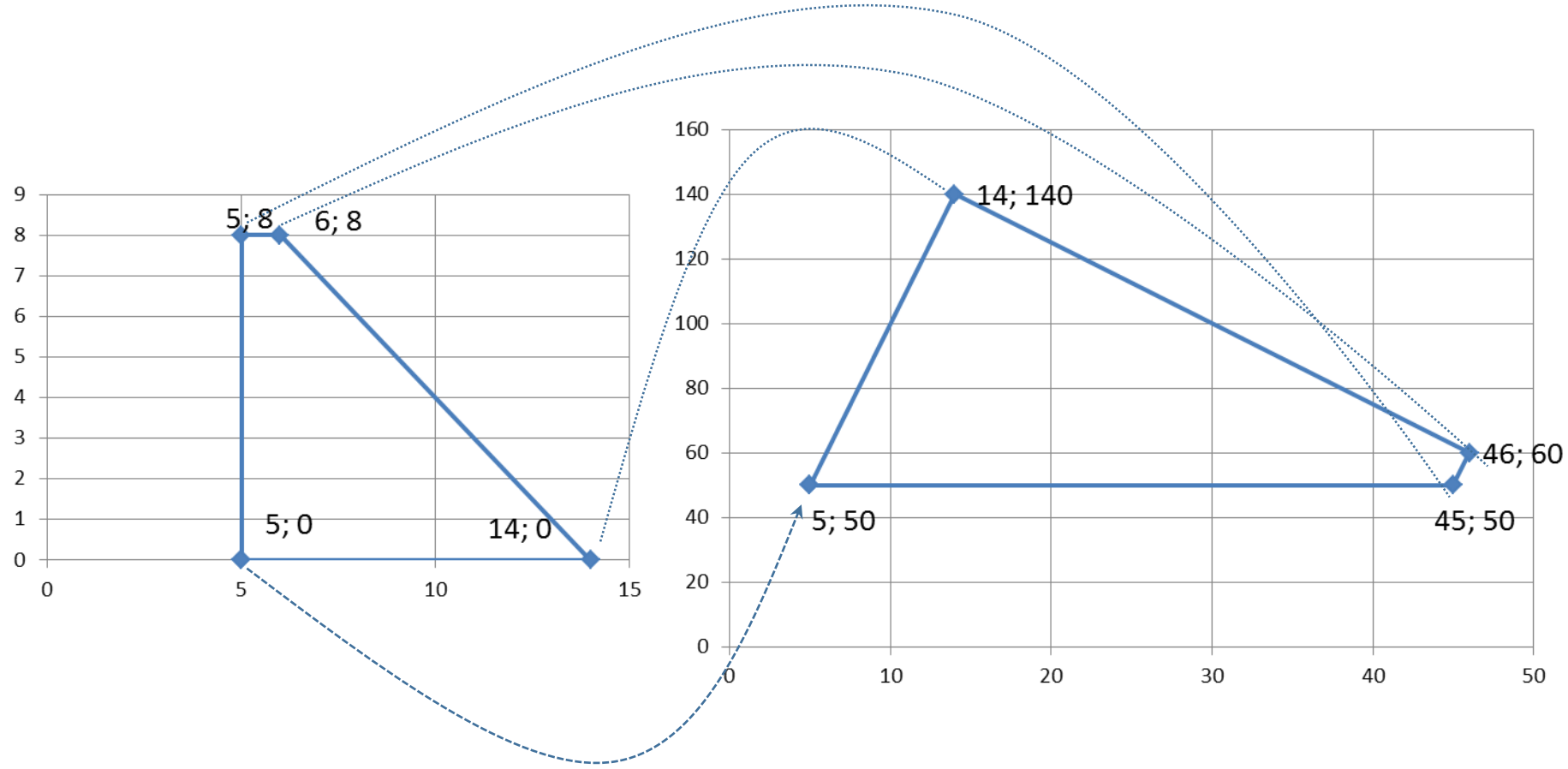
Pyomo and multicriteria

The multicriteria model

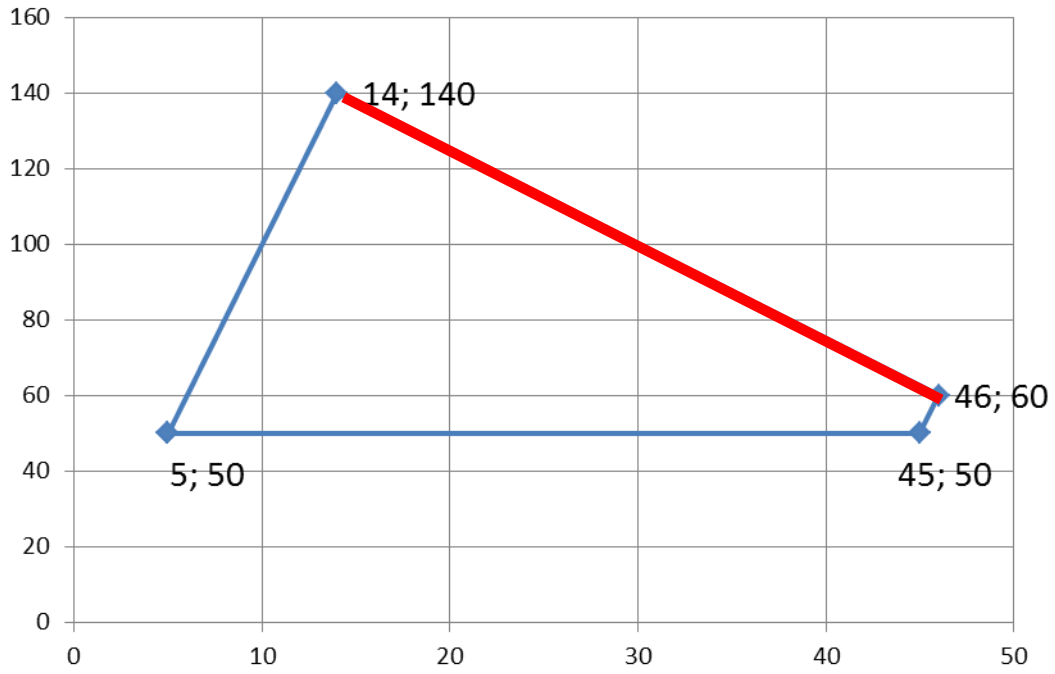
- Decision variables: p w
- Objectives:
  - $q_1 = 10p$
  - $q_2 = p + 5w$
- Constraints:
  - $p + w \leq 14$
  - $p \geq 5$
  - $w \leq 8$
  - $p \geq 0, w \geq 0$



Variable space -> critera space

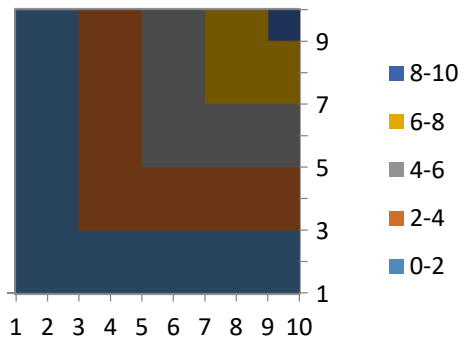


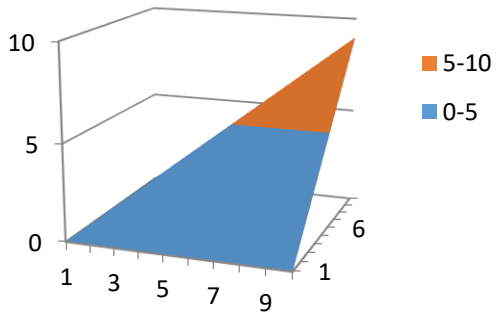
Rozwiązanie Pareto-optymalne



Reference point (the same scale)

$$S(q,\bar{q}) = \min_{1\le i\le m}(q_i - \bar{q}_i) + \frac{\varepsilon}{m}\sum_{i=1}^{m}(q_i - \bar{q}_i)$$





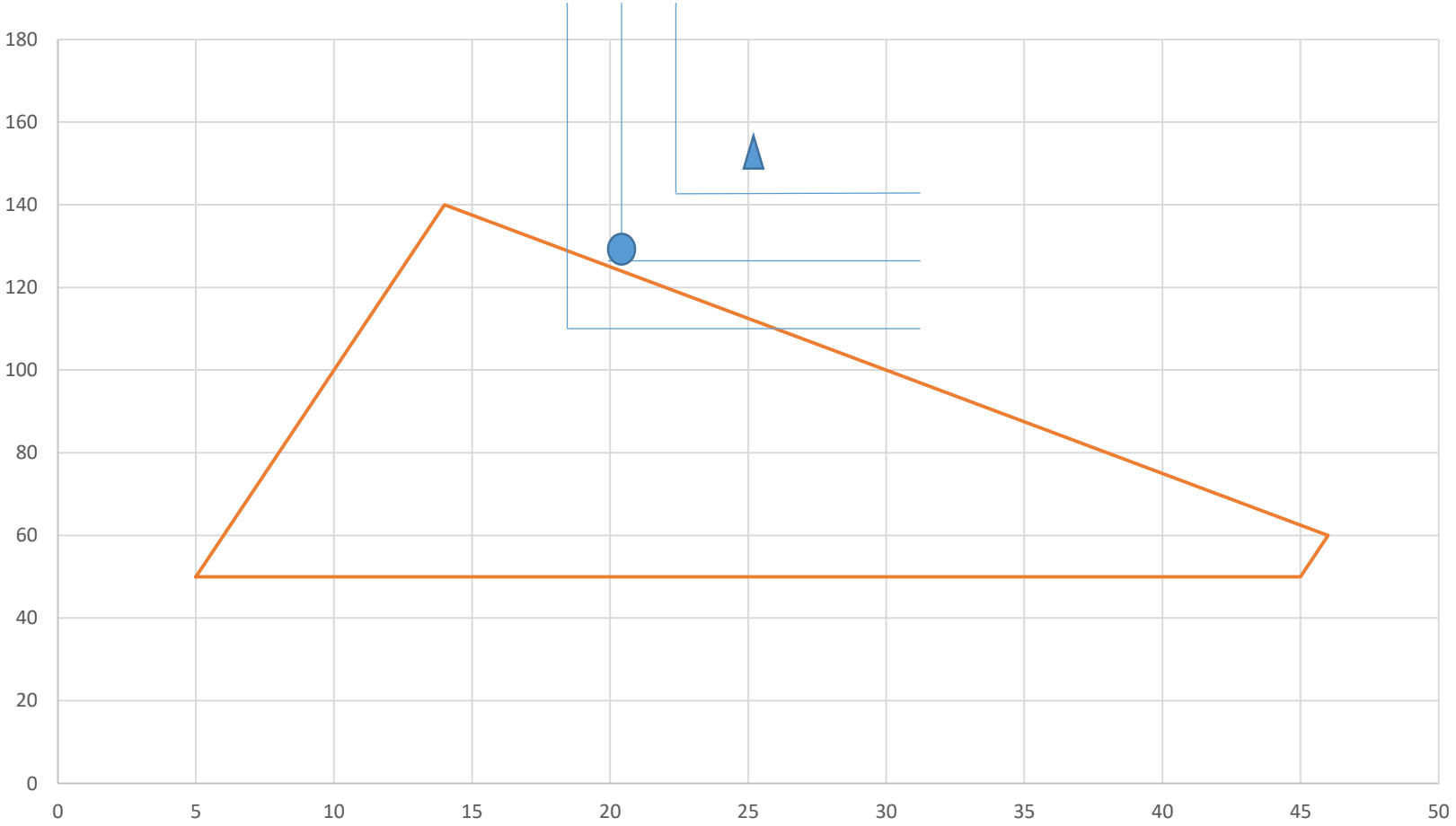
Reference point (the same scale)

$$S(q,\bar{q}) = \min_{1\le i\le m}(q_i - \bar{q}_i) + \frac{\varepsilon}{m}\sum_{i=1}^{m}(q_i - \bar{q}_i)$$

$$\max_{x\in X} z + \frac{\varepsilon}{m}\sum_{i=1}^{m}(q_i - \bar{q}_i)$$

$$q_i - \bar{q}_i \ge z$$
$$x \in X$$

Punkt odniesienia



Pyomo and multicriteria

```
from pyomo.environ import *
model = ConcreteModel()
model.rfp_d = Param(initialize=60)
model.rfp_z = Param(initialize=20)
model.p = Var(within=NonNegativeReals)
model.w = Var(within=NonNegativeReals)
model.z = Var(within=Reals)
model.obj = Objective(expr=model.z + 0.01/2 * ((10*model.p - model.rfp_d) +
(model.p+5*model.w-model.rfp_z)),sense=maximize)
model.con1 = Constraint(expr=10*model.p - model.z -model.rfp_d >= 0)
model.con2 = Constraint(expr=model.p + 5 * model.w - model.z -model.rfp_z >= 0)
model.con3 = Constraint(expr=model.p + model.w  <= 14)
model.con4 = Constraint(expr=model.p  >= 5)
model.con5 = Constraint(expr=model.w <= 8)
```

Pyomo and multicriteria

- Presented model is only for understanding reference point method
- Scaling must be introduced
- Aspiration and reservation point should be introduced