Operations Research
The Scalable AI for Complex Decision Problems

Andrea Taverna, PhD CAP

Alumnus @ OptLab (UniMi)

Currently Operations Researcher in private sector

andrea.taverna@outlook.com

in https://www.linkedin.com/in/andrea-taverna-data-analytics

http://optlab.di.unimi.it

Disclaimer

Opinions stated here are my own and do not necessarily reflect the official policy or position of my past, present or future employers.

Example code

Code used to run the examples and generate the charts can be found on git at lagrangian-example.

OptLab: a local reference for Operations Research!

- Research Laboratory at the Computer Science Dept. of UniMi
- Lots of **success stories on business applications** of Operations Research
- **Just ask!**

http://optlab.di.unimi.it

Contact:
Prof. Alberto Ceselli
alberto.ceselli@unimi.it

Topics

1. **Operations Research,** what is it?

2. **Mathematical Programming,** one versatile tool for solving Complex Decision Problems

3. **Decomposition:** one versatile tool for massive scaling

Operations Research

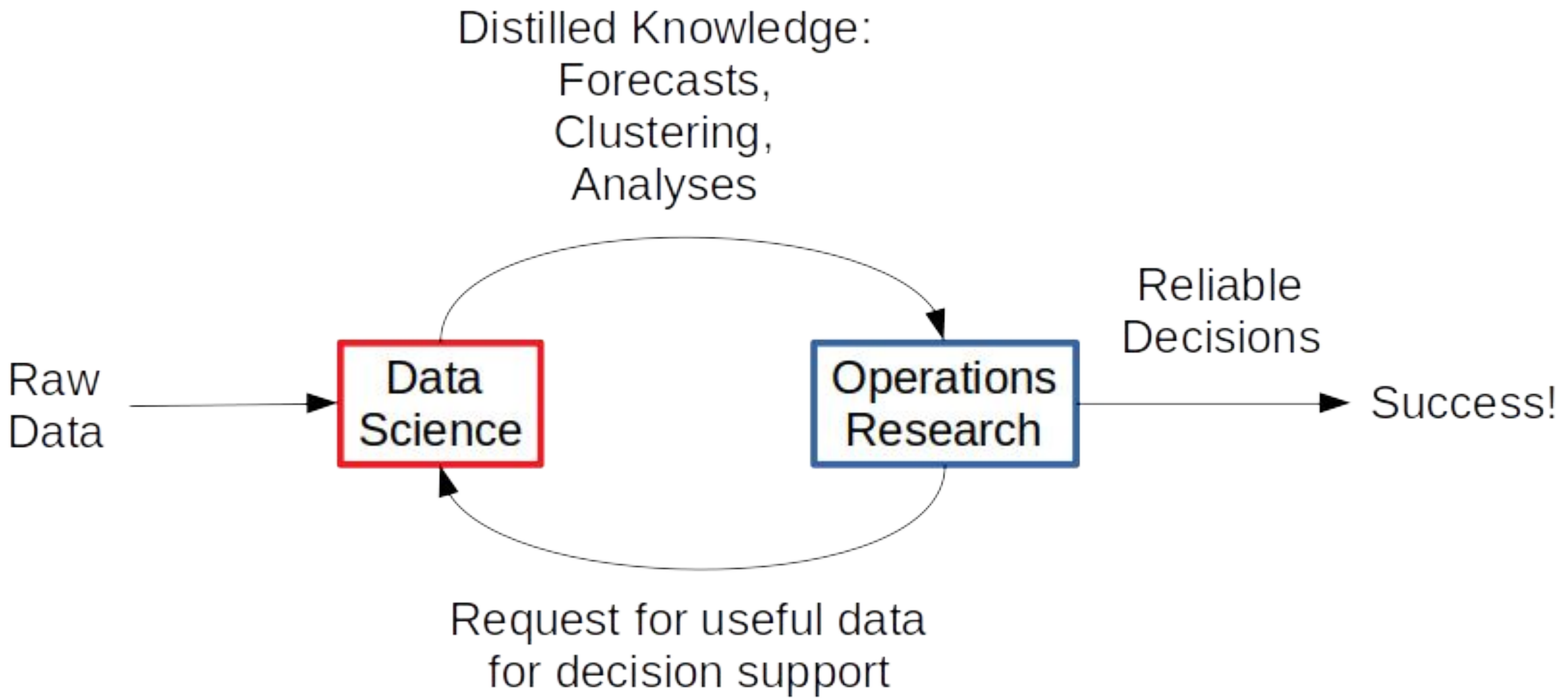
Operations Research: what is it?

- Branch of Analytics for **decision support**
  - Clarify problems through modelling
  - Reliably recommend good decisions
  - Answer complex questions (what if ? why? how?...)

- Also known as Prescriptive Analytics, Management Science (MS), **(Combinatorial) Optimization**, Decision Science

- **The best way to exploit Data Science!**

Data Science & Operations Research Synergy



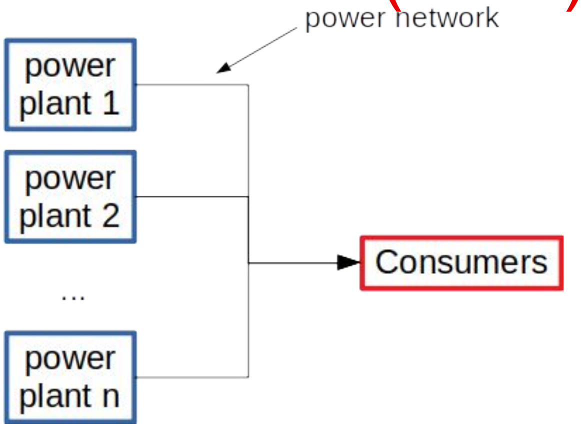
Example problem: Unit Commitment Problem (UCP)

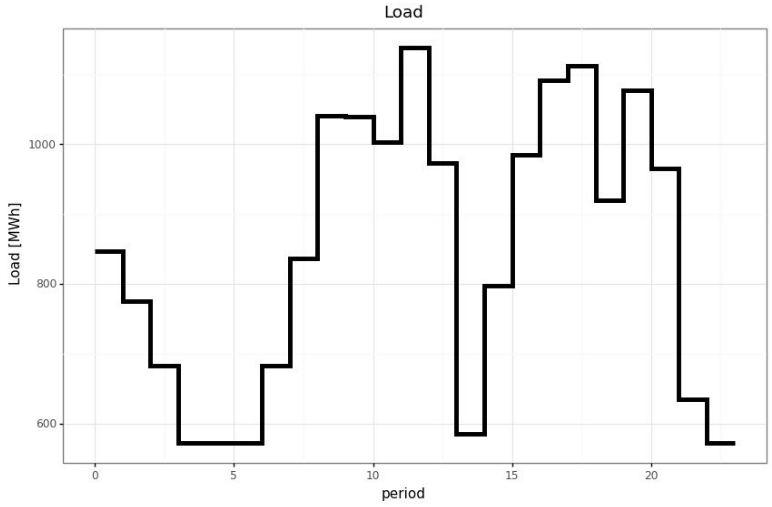
Simple **Power System**:

- Power plants ("**units**") deliver electricity to satisfy consumers' demand, aka "load"
- Hourly Horizon of 24h
- No storage, simple bus network

Goal: schedule ("**commit**") power plants to satisfy load at minimum cost

How to solve that?





Mathematical Programming

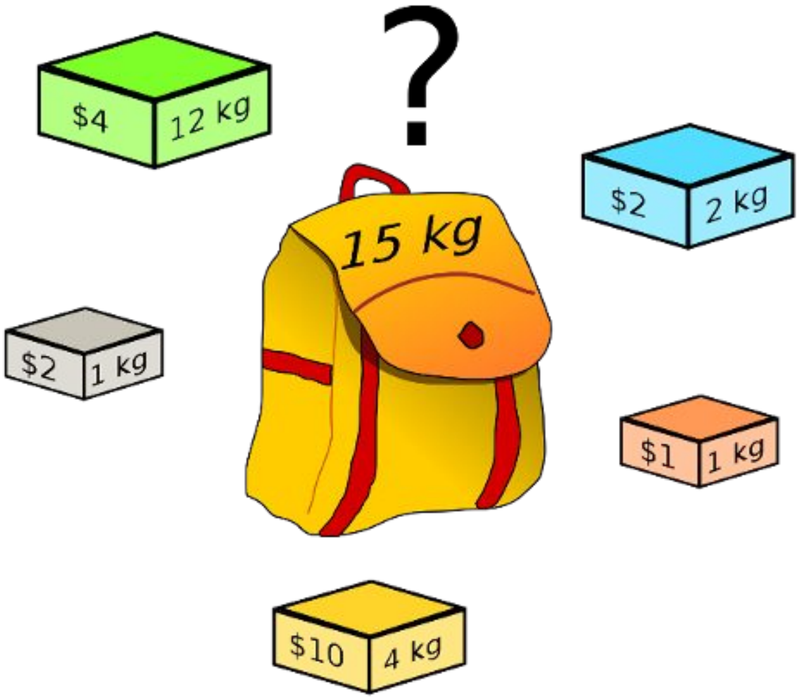
A simple problem: Knapsack

Given

- A set of items i in I, each with a weight w_i and a price p_i
- A knapsack with limited weight capacity C

Select items to put in the knapsack such that:

- The total items' value is maximized
- The total items' weight does not exceed the knapsack's capacity



Knapsack: Mathematical Program

$$\max \sum_{i \in I} p_i x_i$$

$$\sum_{i \in I} w_i x_i \leq C$$

$$x_i \in \{0, 1\} \quad \forall i \in I$$

Where x_i=1 iff item *i* is in the knapsack

Constrained Optimization Problems

Optimization sense

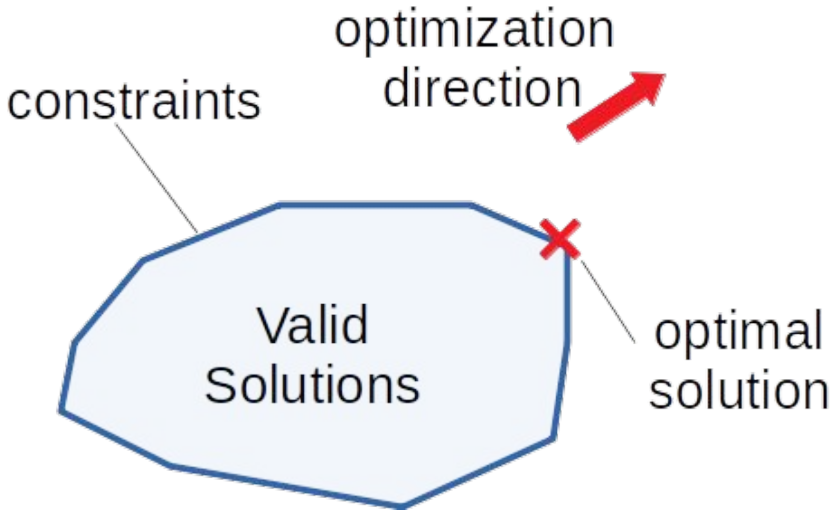
objective/goals

variables/decisions

$$\max \quad f(x)$$

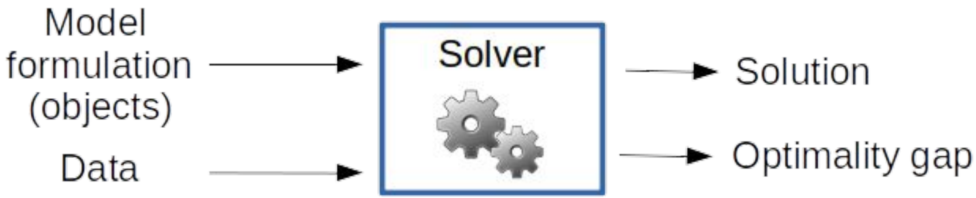
$$g_j(x) \le 0 \qquad \forall j \in J$$

constraints/rules



General-purpose solver for Mathematical Programs

- Reads **Mathematical Programs formulations**
- Combines **exact methods and (meta) heuristics** to efficiently find **optimal solutions**
- Provide an **optimality gap**: (upper) estimate of distance from true optimum
  - Solution with 100K€ profit and 1% optimality gap means true optimal profit is at most 101k€



Example: Knapsack with PuLP/CBC

- PuLP: modelling library/solver interface
- CBC: math.prog. solver
- Both open source and from COIN-OR

Just `pip install pulp` required

Knapsack: general-purpose modelling

Math formulation:

$$\max \sum_{i \in I} p_i x_i$$

$$\sum_{i \in I} w_i x_i \leq C$$

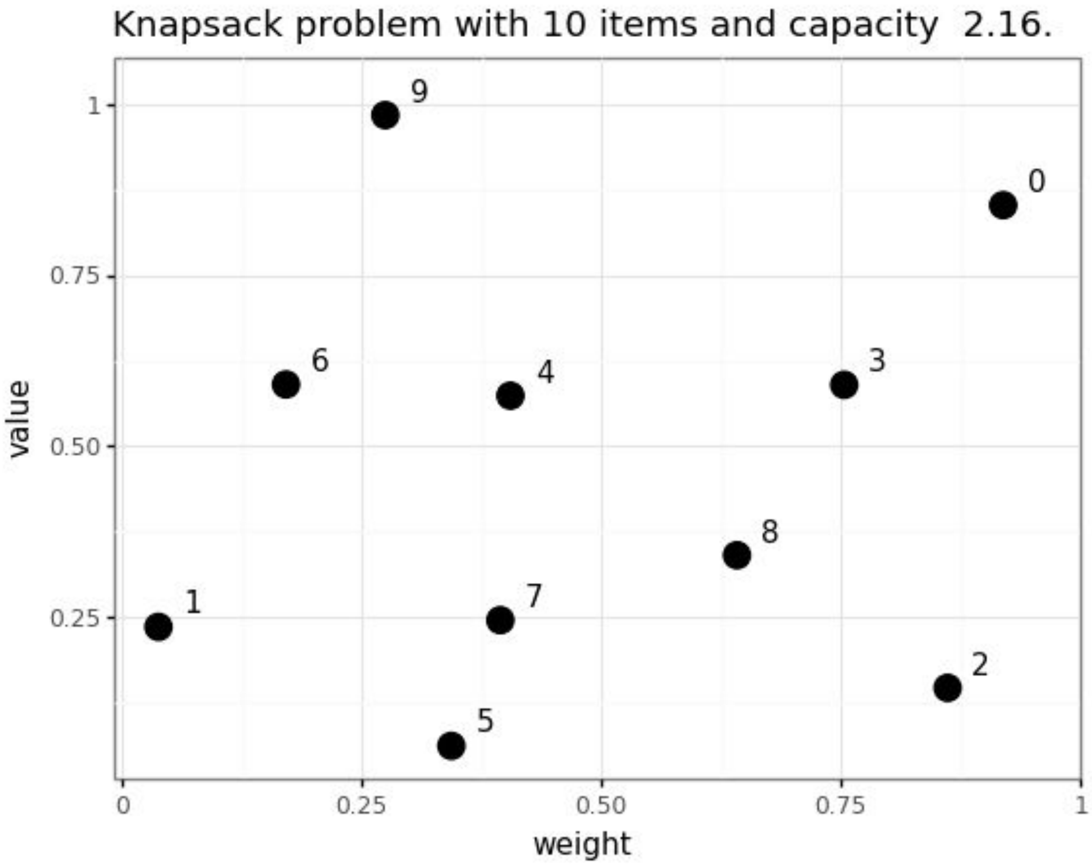
$$x_i \in \{0, 1\} \quad \forall i \in I$$

Python code:

```python
def make_knapsack_model(items_data: DataFrame, capacity:float) -> LpProblem:
"""
    items_data: items dataframe with columns `weight` and `value`
    capacity: knapsack capacity
"""
    I = list(range(len(items_data)))

    model = LpProblem("Knapsack",sense=LpMaximize)

    x = [LpVariable(cat=LpBinary, name=f"x_{i}") for i in I]

    total_value = lpSum(items_data["value"][i] * x[i] for i in I)
    total_weight = lpSum(items_data["weight"][i] * x[i] for i in I)

    model.addConstraint(total_weight <= capacity, "capacity_constraint")

    model.setObjective(total_value)

    return model
```

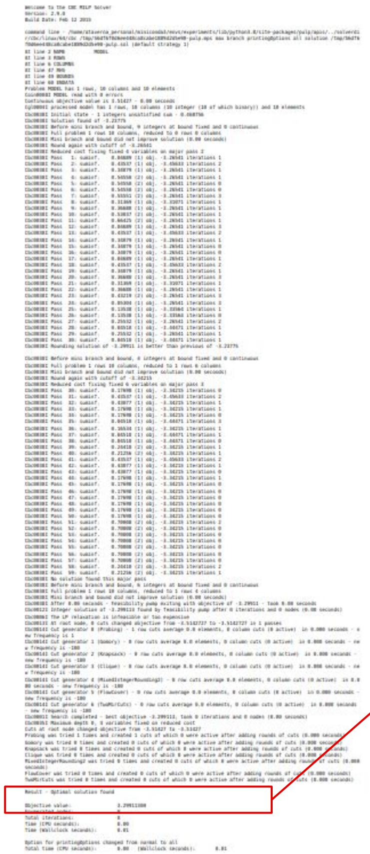
Knapsack: example

Which items do I pick 🤔?



Knapsack: invoking CBC with method `.solve()`

Logs:


Welcome to the CBC MILP Solver
Version: 2.9.0
Build Date: Feb 12 2015

Problem MODEL has 1 rows, 10 columns and 10 elements

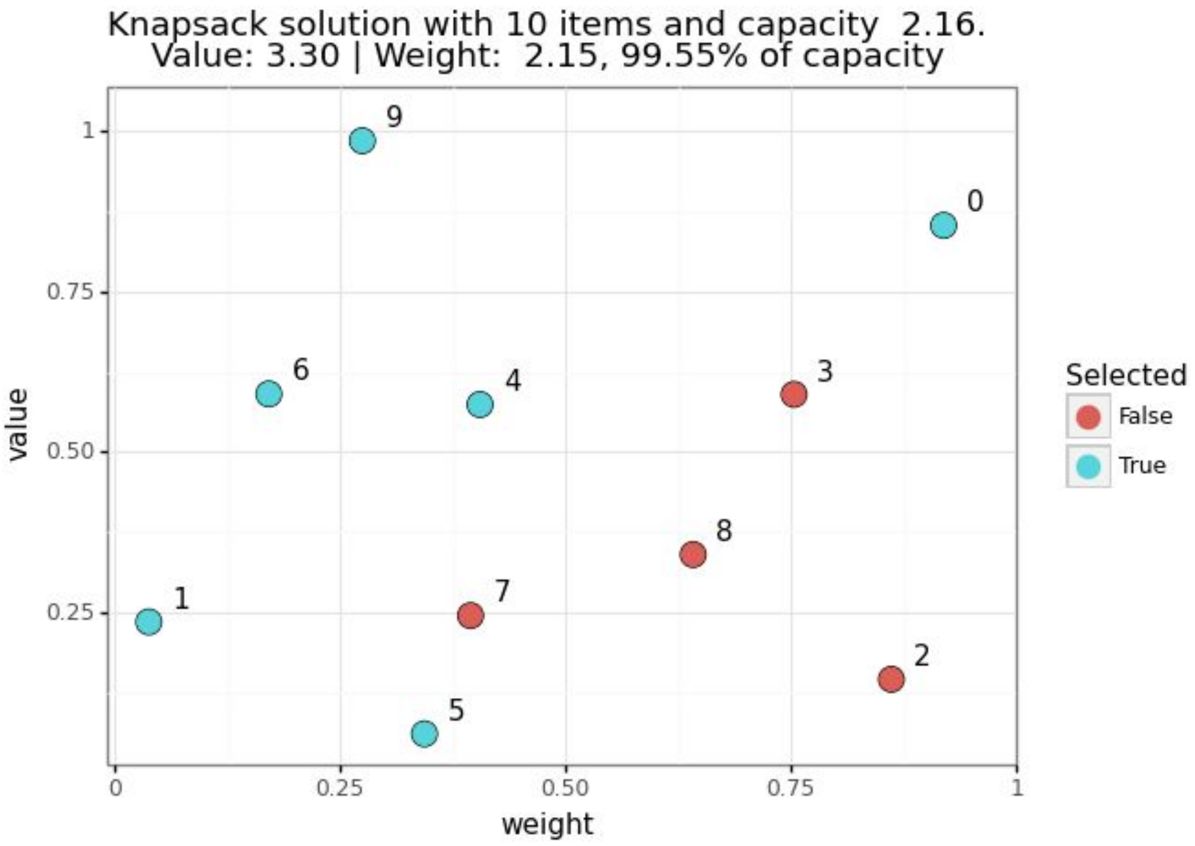
Result - Optimal solution found

Objective value: 3.29911300

The solver found a solution and *proved* it is optimal!

```
Result - Optimal solution found

Objective value:                3.29911300
```

Knapsack: solution



Knapsack: interpretability /explainability/what-if simulation

Possible questions about, e.g. item 8:

- Why wasn’t 8 picked?
- What would happen if 8 was in the knapsack?

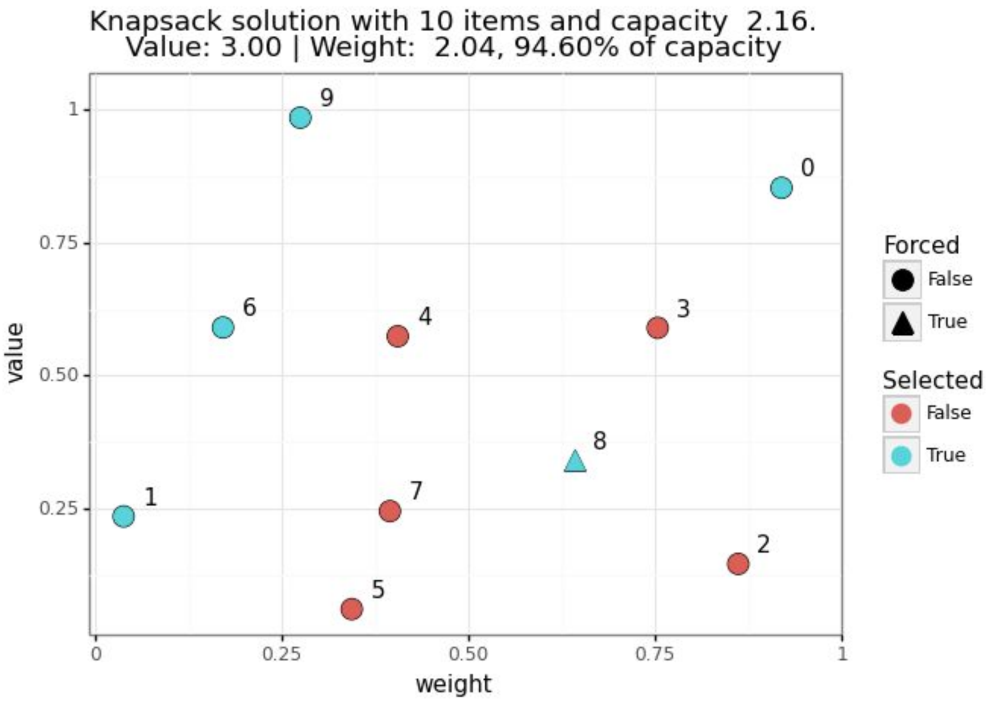
To answer add the constraint $x_8 = 1$ in the model and solve it again!

Knapsack with item 8 forced

Item 8 in knapsack yields 3.0-3.3 = -0.3 profit loss compared to optimal solution.

That's why it was not picked.

To put item 8 in the knapsack, I should ask for a payoff of at least 0.3 .



From Knapsack to practical problems

For decision problems, with Mathematical Programming you can:

- model problems with general language
- get high-quality solutions
- interpret and trust models

But can it scale to practical problems with realistic size 🤔?

The Traveling Salesman Problem and the Curse of Dimensionality

The Traveling Salesman Problem

The Traveling Salesman Problem is one of the most intensively studied problems in computational mathematics. These pages are devoted to the history, applications, and current research of this challenge of finding the shortest route visiting each member of a collection of locations and returning to your starting point.

The TSP is NP-hard.

A TSP with *n* cities has *n!* solutions to search through!

With 60 cities the number solutions is close to the amount of atoms in the observable universe (10^{80}).

Source: www.math.uwaterloo.ca/tsp/

New hopes: Modern Machine Learning (2021)

Challenge's goal:

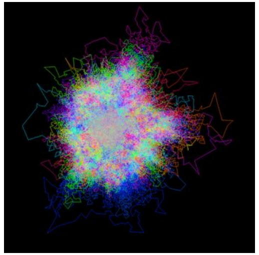
- encourage the development of **innovative approaches** based on **ML, Deep Learning, Computer Vision** …
- … that **outperform traditional Operations Research methods** …
- … for routing problems

Link here

“New hopes”: Operations Research (2021)

The Amazon Last Mile Challenge was actually won by the “traditional” heuristics developed by well-known TSP experts.

They also solved TSP for ~1.3bln stars in the observable universe! (see pic)



Combinatorics and Optimization » News » 2021 » August »

Bill Cook's team announced as winner of the Amazon Last Mile Routing Research Challenge

TUESDAY, AUGUST 10, 2021

C&O professor Bill Cook, Stephan Held (University of Bonn) and Keld Helsgaun (Roskilde University, Denmark) have been announced as winners of the "Amazon Last Mile Routing Research Challenge". Their team, *Just Passing Through*, received a cash prize of $100,000.

The Last Mile Routing Research Challenge was organized by

Alex Kontorovich
@AlexKontorovich

I didn't mention the best part: This was supposed to be a competition for machine learning. You had 12 hrs to train your net, then 3 to compute. Bill's winning team "trained" for 10 secs (downloaded data), then waited 11:59:50 to start computing (no ML!).

ML&OR: Integrate, don't compete! /1

*Is **combinatorial optimization the right domain for ML?** Classical solvers are extremely mature and very powerful. It is not clear there's a huge improvement by applying ML, whereas **there could be other domains [in combinatorial optimization] where ML could shine brighter.***

-- Vinod Nair, DeepMind
Panel discussion "Machine Learning in Combinatorial Optimization"
*ML4CO **NeurIPS 2021***

ML&OR: Integrate, don't compete! /2

We have seen ML cannot simply replace Optimization*, and that may have been the expectation when the hype was at his peak. Now we understand how we can integrate these techniques into each other, and* ***use ML inside solvers.***
I find this much more interesting than seeing these as competing technologies.

-- Timo Berthold, FICO Xpress Solver
An idea shared by most if not all members in the panel.
Panel discussion "Machine Learning in Combinatorial Optimization"
ML4CO ***NeurIPS 2021***

From Knapsack to practical problems

For decision problems, with Mathematical Programming you can:

- model problems with general language
- get high-quality solutions
- interpret and trust the models

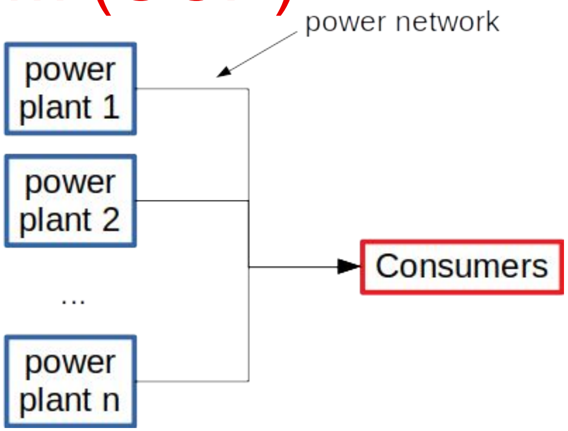
But can it scale to practical problems with realistic size 🤔?

YES!!!

Mathematical Programming

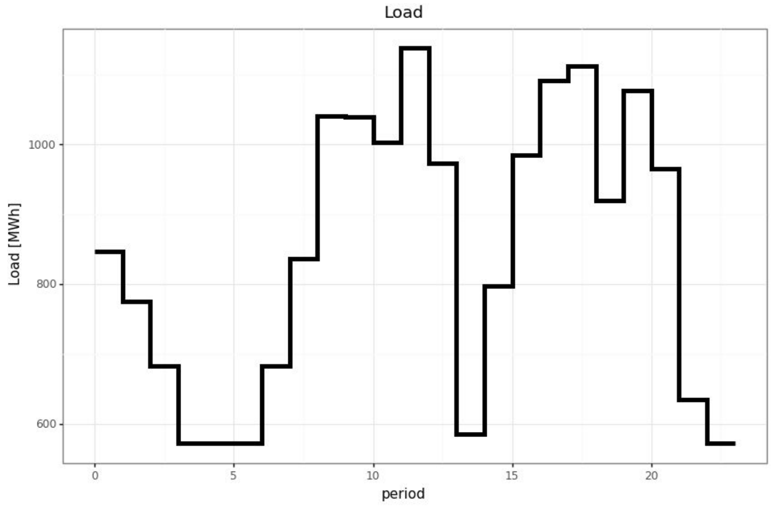
for Unit Commitment Problem

Example: Unit Commitment Problem (UCP)



Simple **Power System**:

- Power plants ("**units**") deliver electricity to satisfy consumers' demand, aka "load"
- Hourly Horizon of 24h
- No storage, simple bus network



Goal: schedule ("**commit**") power plants to satisfy load at minimum cost

Thermal Power Plants (TPPs)

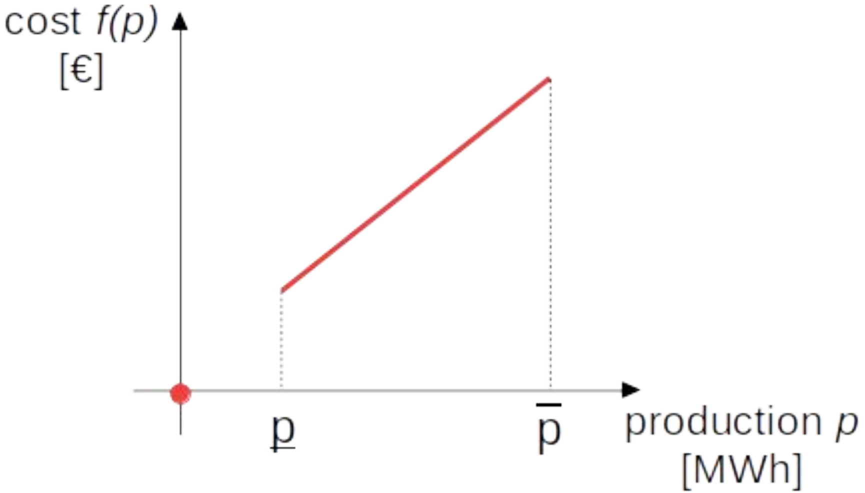
production level

state on/off

$$f(p) = lp + cs$$

$$\text{s.t.} \quad \underline{p}s \leq p \leq \bar{p}s, \quad s \in \{0, 1\}$$

- Thermal power plant: gas,coal, nuclear
- **When on**, production is between $\underline{p}$ and $\bar{p}$.
  When off, production is zero.
- **“cooling/warming” period:** when a plant switches state, it needs to wait a few hours before switching again



UCP Model - math vs code

$$
\begin{aligned}
\min \sum_{i\in I, t\in T} &(l_i p_{it} + c_i s_{it}) + \\
&\sum_{t\in T} c_{EIE} EIE_t + \sum_{t\in T} c_{ENP} ENP_t
\end{aligned}
$$

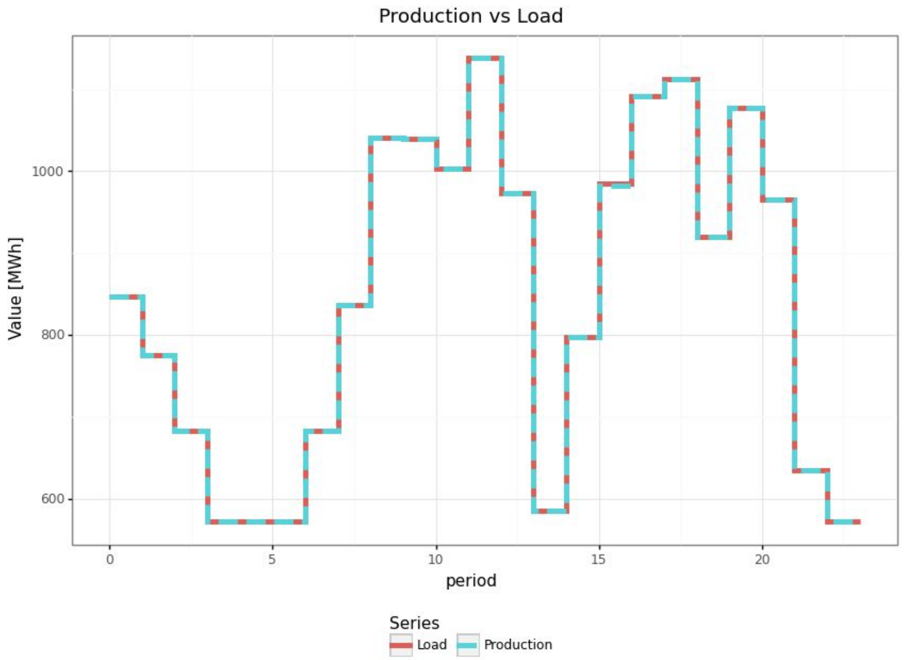
$$
\begin{aligned}
\text{s.t. } & \underline{p}_i s_{it} \le p_{it} \le \bar{p}_i s_{it} && \forall i \in I, t \in T \\
& u^+_{it} \ge s_{it} - s_{i(t-1)} && \forall i \in I, t \in T : t > 0 \\
& u^-_{it} \ge s_{i(t-1)} - s_{it} && \forall i \in I, t \in T : t > 0 \\
& s_{it} \ge \sum_{t' \in \max(0, t-\tau^+_i+1)\ldots t} u^+_{it'} && \forall i \in I, t \in T \\
& s_{it} \le 1 - \sum_{t' \in \max(0, t-\tau^-_i+1)\ldots t} u^-_{it'} && \forall i \in I, t \in T \\
& \sum_{i\in I} p_{it} + \mathrm{ENP}_t = D_t + \mathrm{EIE}_t && \forall t \in T \\
& s_{it}, u^-_{it}, u^+_{it} \in \{0, 1\} && \forall i \in I, t \in T \\
& p_{it}, EIE_t, ENP_t \in \mathbb{R}^+_0 && \forall i \in I, t \in T
\end{aligned}
$$

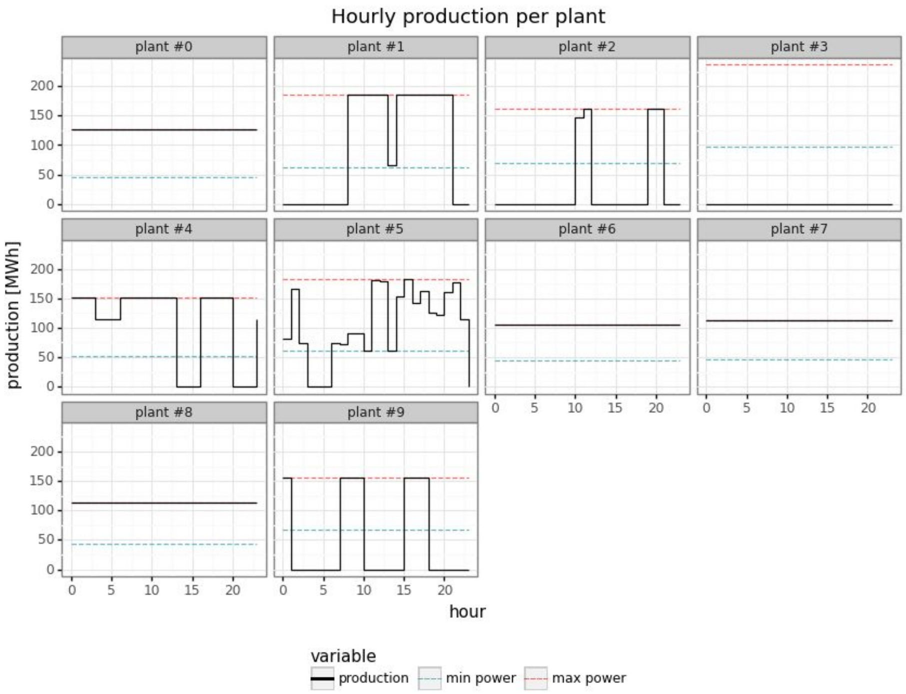
```python
def create_model(data: UCPData) -> MathematicalProgram:
    TPP = data.thermal_plants
    Plants = TPP["plant"].to_list()
    Time = data.loads["period"].values

    model = LpProblem("UCP", sense=LpMinimize)

    ### VARIABLES
    p = {(plant, t): LpVariable(f"p_{plant}_{t}", lowBound=0) for (plant, t) in product(Plants, Time)}

    s = LpVariable.dict("s", (Plants, Time), cat=LpBinary)

    up = LpVariable.dict("up", (Plants, Time), cat=LpBinary)
    dn = LpVariable.dict("dn", (Plants, Time), cat=LpBinary)

    EIE = LpVariable.dict("EIE", Time, lowBound=0)
    ENP = LpVariable.dict("ENP", Time, lowBound=0)

    ### CONSTRAINTS
    def_up = {
        (plant, t): add_constraint(model, up[plant, t] >= s[plant, t] - s[plant, t - 1], f"def_up_{plant}_{t}")
        for (plant, t) in product(Plants, Time)
        if t > 0
    }

    def_down = {
        (plant, t): add_constraint(model, dn[plant, t] >= s[plant, t - 1] - s[plant, t], f"def_dn_{plant}_{t}")
        for (plant, t) in product(Plants, Time)
        if t > 0
```

...

```python
    ### OBJECTIVE
    thermal_production_cost = lpSum(
        l_cost * p[plant, t] + c_cost * s[plant, t]
        for ((plant, l_cost, c_cost), t) in product(TPP[["plant", "l_cost", "c_cost"]].itertuples(index=False), Time)
    )

    demand_mismatch_cost = lpSum(data.c_EIE * EIE[t] + data.c_ENP * ENP[t] for t in Time)
    total_production_cost = thermal_production_cost + demand_mismatch_cost

    model.setObjective(total_production_cost)
```

Example solution, 10 TPPs x 24h
(optimal in 0.67s with PuLP/CBC)

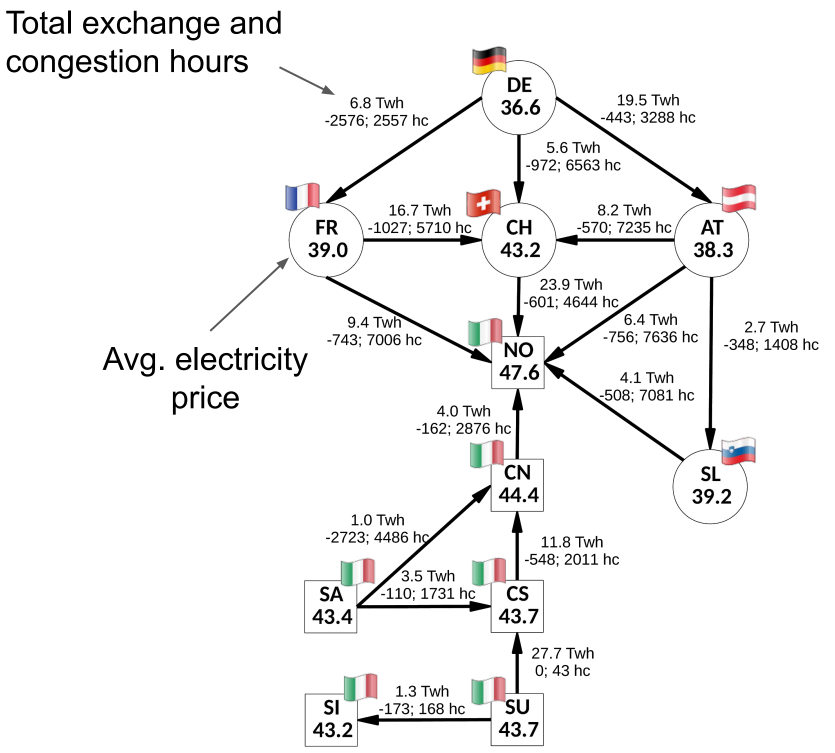




Large-scale Example: Electricity Market Simulation

CASE:

- Simulate the ideal/optimal state (equilibrium) of electricity markets for one year at hourly resolution
- Used to benchmark/ evaluate cross-country **energy policies**
  - Examples here and here



Example simulation summary for Italian market zones and neighbours

Large-scale Example: Electricity Market Simulation

CASE:

- For Italy and neighbours:
  - $\simeq$120 power plants to turn on or off
  - 8760 hours
  - Plus network details etc…
  - Total:
    - 1.8 mln variables
    - 2 mln constraints

SOLUTION SPACE SIZE:

$\simeq$120,000 binary variables

(hourly on/off state of power plants)

=

$2^{120,000}$ combinations

(FIY atoms in observable universe $\simeq 2^{250}$)

Plus other 1.6 mln variables
and 2 mln constraints…

Large-scale Example: Electricity Market Simulation

SOLUTION:

- **≥99% optimality**
- "No code" implementation (GAMS) with limited parallelization on **8-core 32GB RAM requires** $\simeq$ **one hour**.
  - Huge time savings if python/java were used with "full" parallelization

Where's the trick?

SOLUTION SPACE SIZE:

$\simeq$120,000 binary variables

(hourly on/off state of power plants)

=

$2^{120,000}$ combinations

(FIY atoms in observable universe $\simeq 2^{250}$)

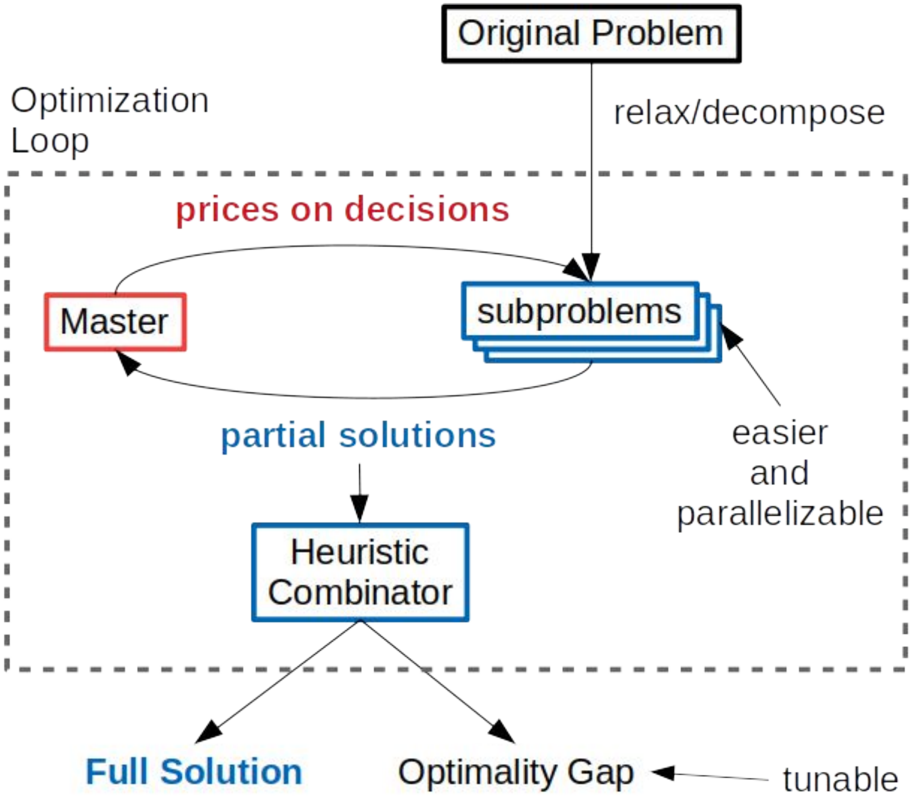
Plus other 1.6 mln variables
and 2 mln constraints…

Decomposition

The decomposition trick

When a problem gets too big:

- decompose **Original Problem** into easier, **independent subproblems**
- **“master model”** provides **“prices on decisions”** to **subproblems** to better coordinate
- Use a “**heuristic combinator**” to combine **subproblems’ partial solutions** into a **“full solution” feasible and optimal for the Original Problem**
- **Iterate until convergence** (desired optimality gap)



Decomposition, Lagrangian-style

Given original problem P:

- Write **coupling constraints as penalties** in the objective function, **weighted by lagrangian prices λ**
- constraints violated ⇒ **b-A^{T}x>0**
  ⇒ the objective worsens
- **Relaxed model R(λ)** is easier than P and **decomposes in independent subproblems**

$$P:\ \min\ c^{\top}x$$
$$A^{\top}x \geq b$$
$$x \in \mathcal{X}$$

Coupling constraints

Non-coupling constraints

$$R(\lambda):\ \min_{x\in\mathcal{X}}\ c^{\top}x + \lambda^{\top}(b - A^{\top}x)$$

original objective

weighted penalties

Master problem: finding good prices

$$D : \max_{\lambda \geq 0} \varphi(\lambda) = \overbrace{\min_{x \in \mathcal{X}} c^\top x + \lambda^\top (b - A^\top x)}^{R(\lambda)}$$

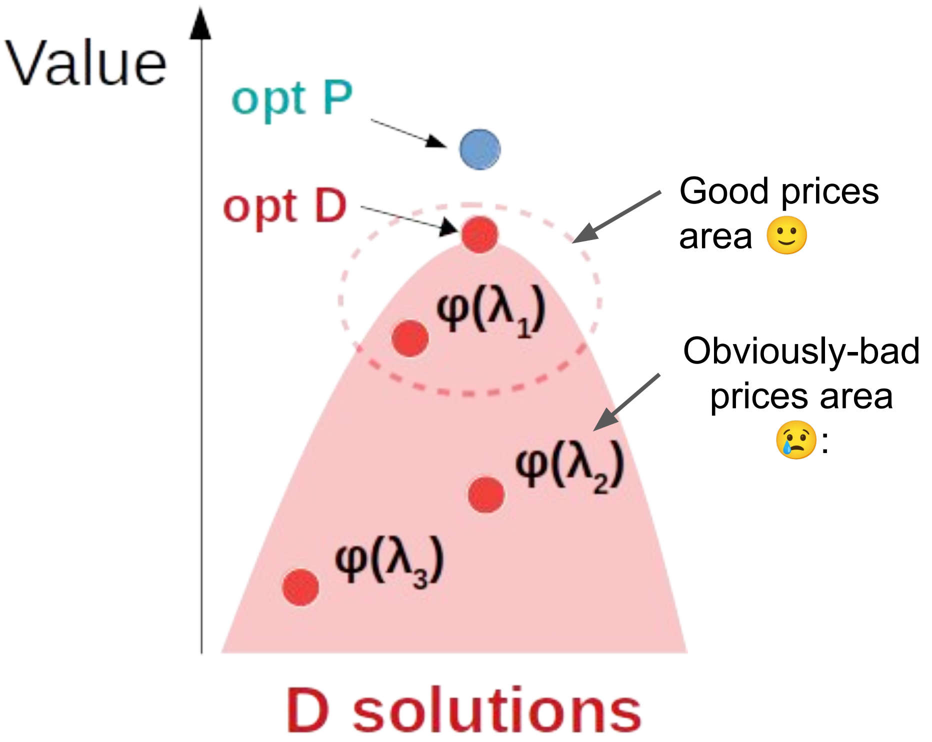
Let opt S := optimal value of problem S.
Let φ(**λ**) = opt R(**λ**).

To find good prices solve the Lagrangian Dual Problem

D: **λ*** = arg max φ(**λ**)

rationale:

- $\max_{\lambda}$ φ(**λ**) ≤ opt P
- Obviously-bad prices yield relaxations with optimum “too good”
  φ(**λ**) << opt P
  by violating relaxed constraints.
- Conversely, good prices yield
  φ(**λ**) ~ opt P



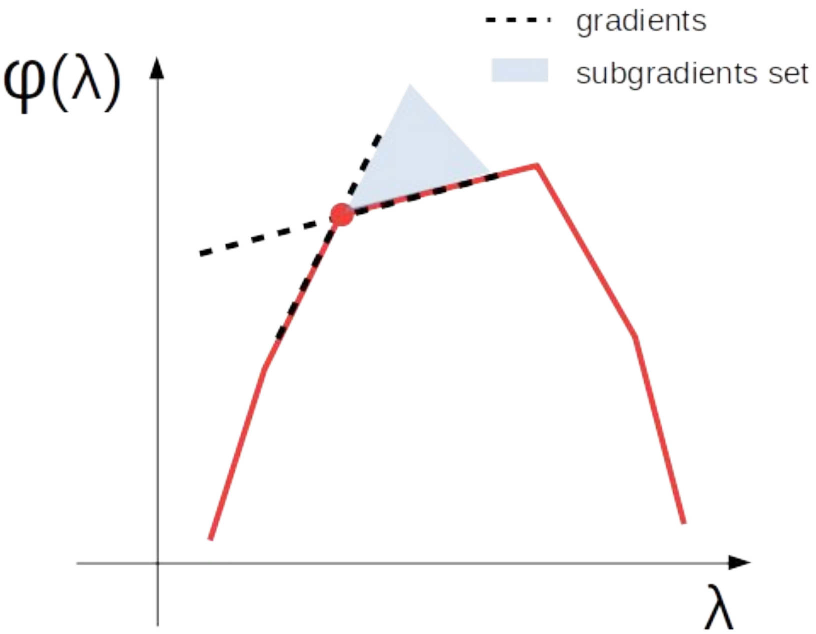
Solving the Lagrangian Dual

$$D: \max_{\lambda \geq 0} \varphi(\lambda) = \min_{x \in \mathcal{X}} c^\top x + \lambda^\top (b - A^\top x)$$

- **φ(λ) is convex, piece-wise and non-smooth** (not differentiable everywhere)
- use ***subgradients***, which can under or over estimate the "actual gradient"
  - Optimization gets harder than with smooth functions!
- A subgradient of φ at **λ** is

  $$(b - A^T \mathbf{x}_\lambda)$$

  where $\mathbf{x}_\lambda$ is a solution of R(**λ**), i.e. the violations of the relaxed constraints at $\mathbf{x}_\lambda$



Non-Smooth Optimization: Subgradient Descent

Typical Subgradient Descent:

- "Like" Gradient Descent, but with SubGradients (g_n)
- **Uses momentum/**"deflection" to reduce "zig-zagging"
- **Polyak's step size rule**: step size (s_n) depends on a **parameter** β_n and an **over-estimation** $\varphi\hat{}_n$ of the optimum $\max_{\lambda} \varphi(\boldsymbol{\lambda})$

$$g_n = b - A^{\top} x_n$$

$$d_n = \alpha_n g_n + (1 - \alpha_n) d_{n-1}$$

$$s_n = \beta_n \frac{\hat{\varphi}_n - \varphi(\lambda_n)}{\|d_n\|^2}$$

$$\lambda_{n+1} = \max(0, \lambda_n - s_n d_n)$$

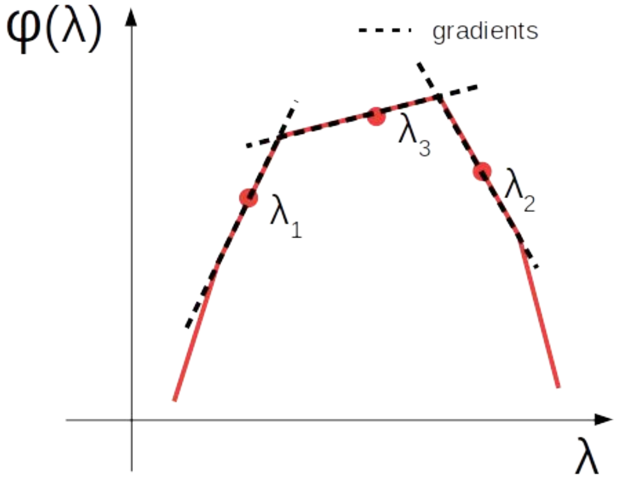
Non-Smooth optimization: Cutting Plane (CP)

- combine **partial solutions x_n** and **subgradients (b-Ax_n)** to iteratively construct a **piece-wise linear approximation** of φ

- Implemented as Mathematical Program

subgradient

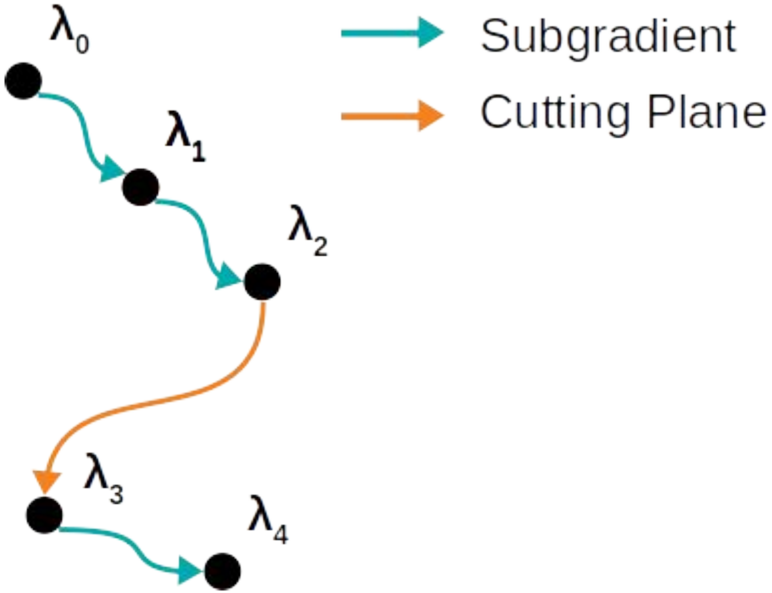
$$\max \; z$$
$$z \le c^\top x_{n'} + \lambda^\top (b - A^\top x_{n'}) \quad n' \in 1..n$$
$$\lambda \ge 0$$
$$z \in \mathbb{R}$$



Non-Smooth Optimization: combining methods

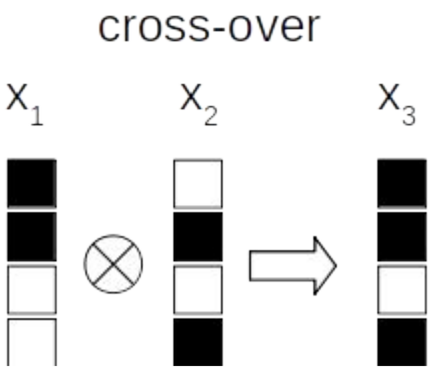
Subgradient and Cutting Plane are often combined to exploit complementary strengths:

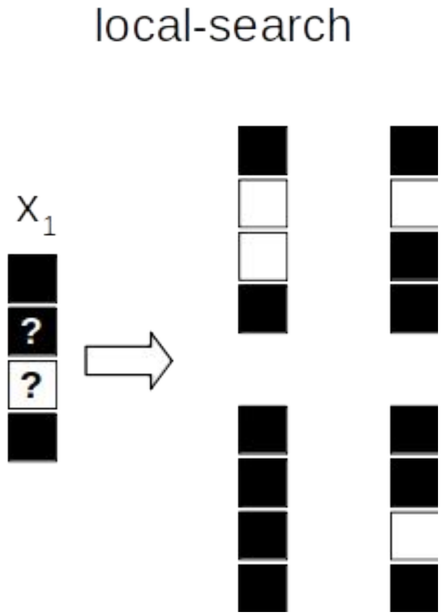
- Subgradient Descent: “local”, performs small steps that improve the current solution
- Cutting Plane: ”global”, larger steps to explore the solution space, but can jump to worse solutions



Heuristic Combination

- Exploit subproblems solutions $\{\mathbf{x}_n\}$
- Combine $\{\mathbf{x}_n\}$ into optimal and feasible solutions for P.
  Examples:
  - "Cross-over"
  - Local search
  - …
- **Can be implemented with minimum effort and high efficiency with PuLP**





Lagrangian Decomposition

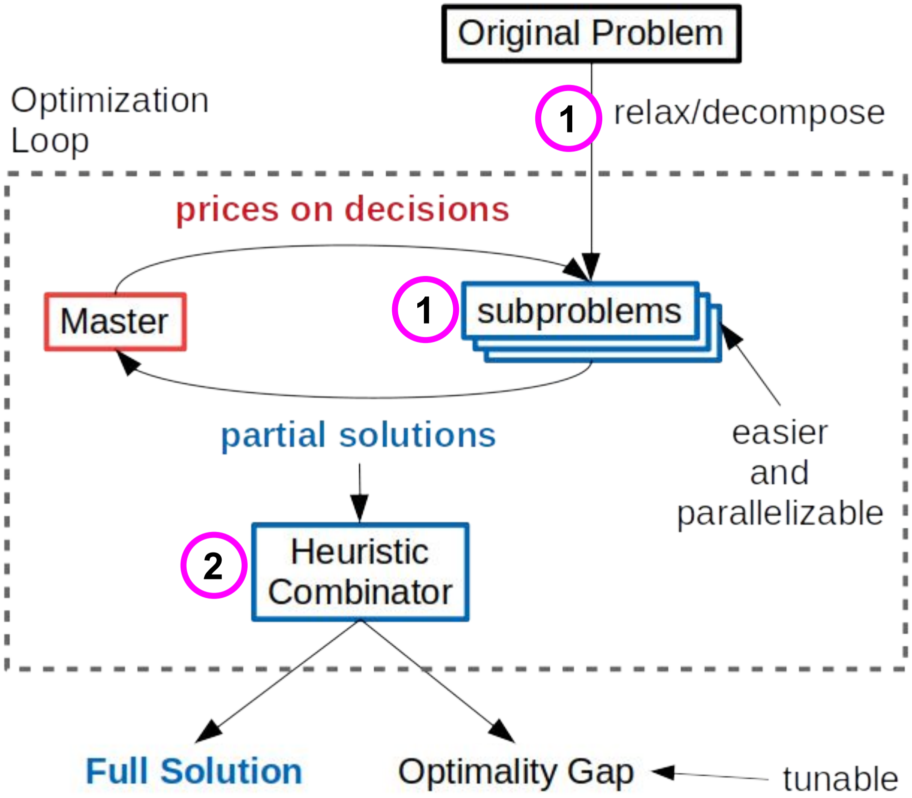
For Unit Commitment Problem

Lagrangian Decomposition for UCP

To apply Lagrangian Decomposition to UCP we need to specify:

1. How to relax the original problem/ formulate the subproblems
2. The heuristic combinator to construct a full solution from the partial solutions.

The rest, including the Master Problem, is (usually) problem independent.

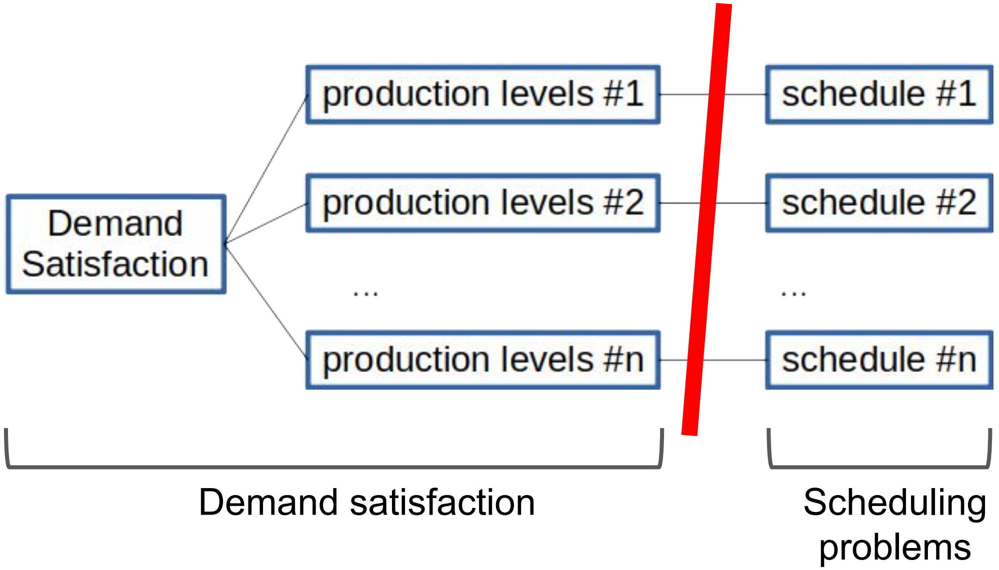


Relaxation for UCP

- Separate each plant's schedule from its production levels
- **Constraint violation:** a power plant produces while being off or does not produce enough while being on
- Two groups of subproblems:
  - a demand satisfaction problem, continuous, easy
  - Independent, very easy scheduling problems, one for each power plant

both implemented in PuLP

#*i*: power plant id



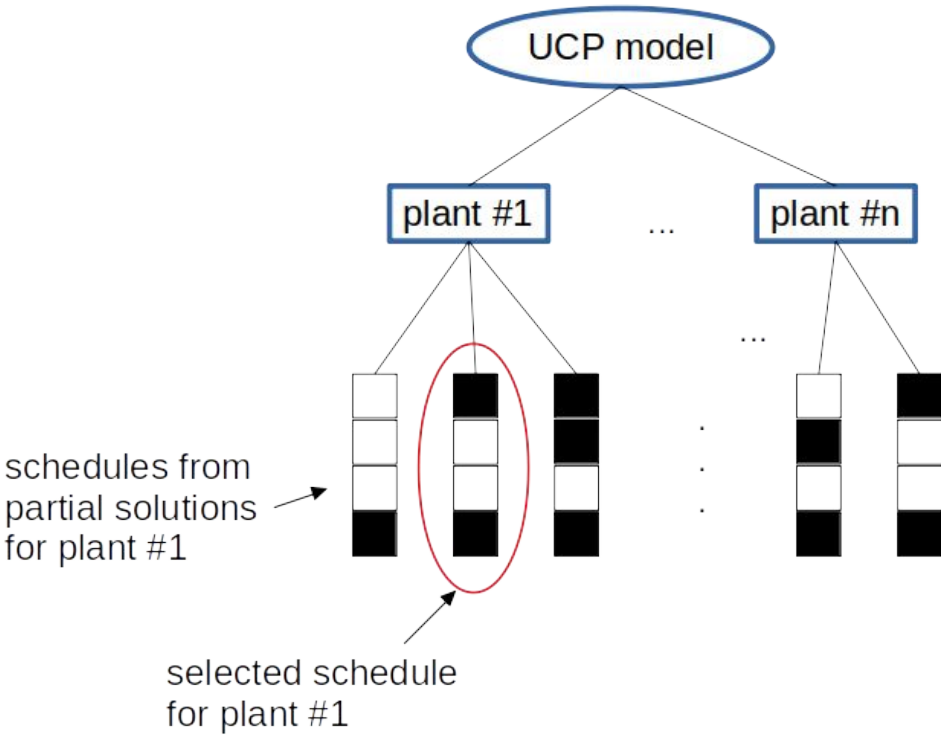
Combinatorial Heuristic for UCP

1. **Combination:** select for each power plant a schedule among the ones in partial solutions, then

2. **Local search:** re-optimize the overall schedule in few selected hours having high electricity prices
   - "Electricity prices" from Pulp/CBC

Both steps implemented in PuLP

Combination



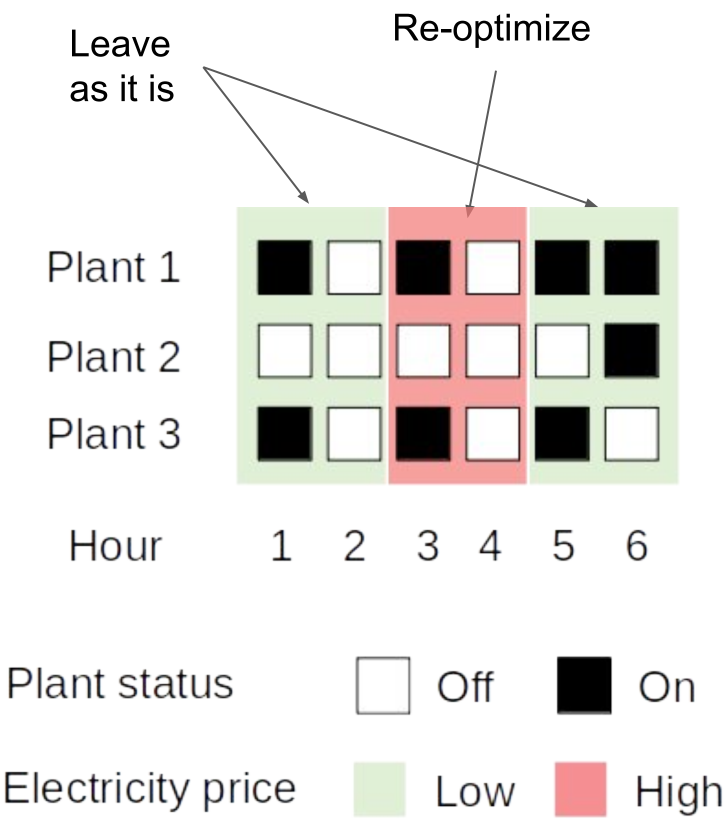
Combinatorial Heuristic for UCP

1. **Combination:** select for each power plant a schedule among the ones in partial solutions, then

2. **Local search:** re-optimize the overall schedule in few selected hours having high electricity prices
   – "Electricity prices" from Pulp/CBC

Both steps implemented in PuLP

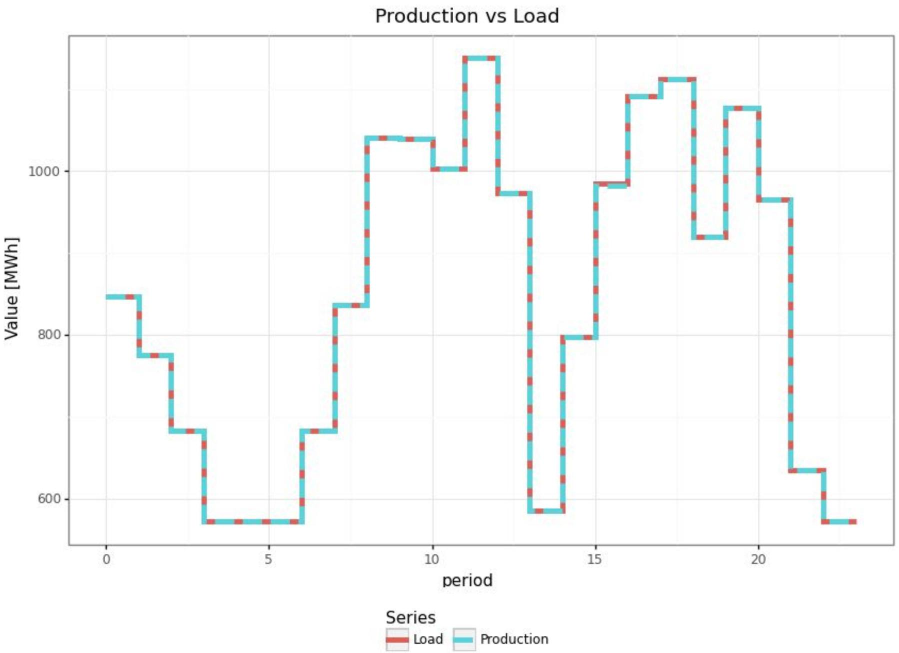
Local search

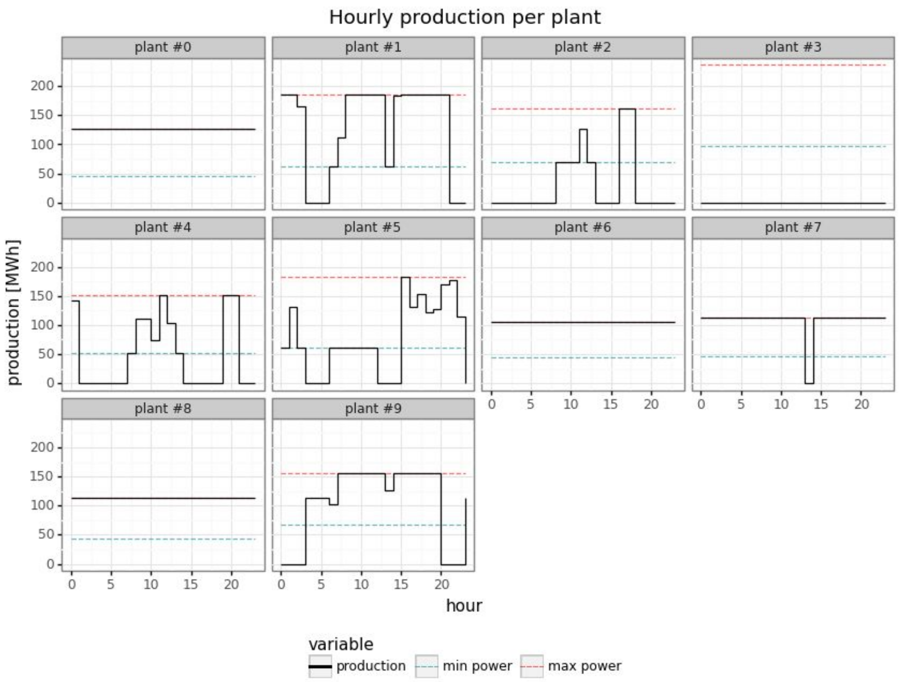


UCP 10TPPsx24h with Lagrangian Decomposition

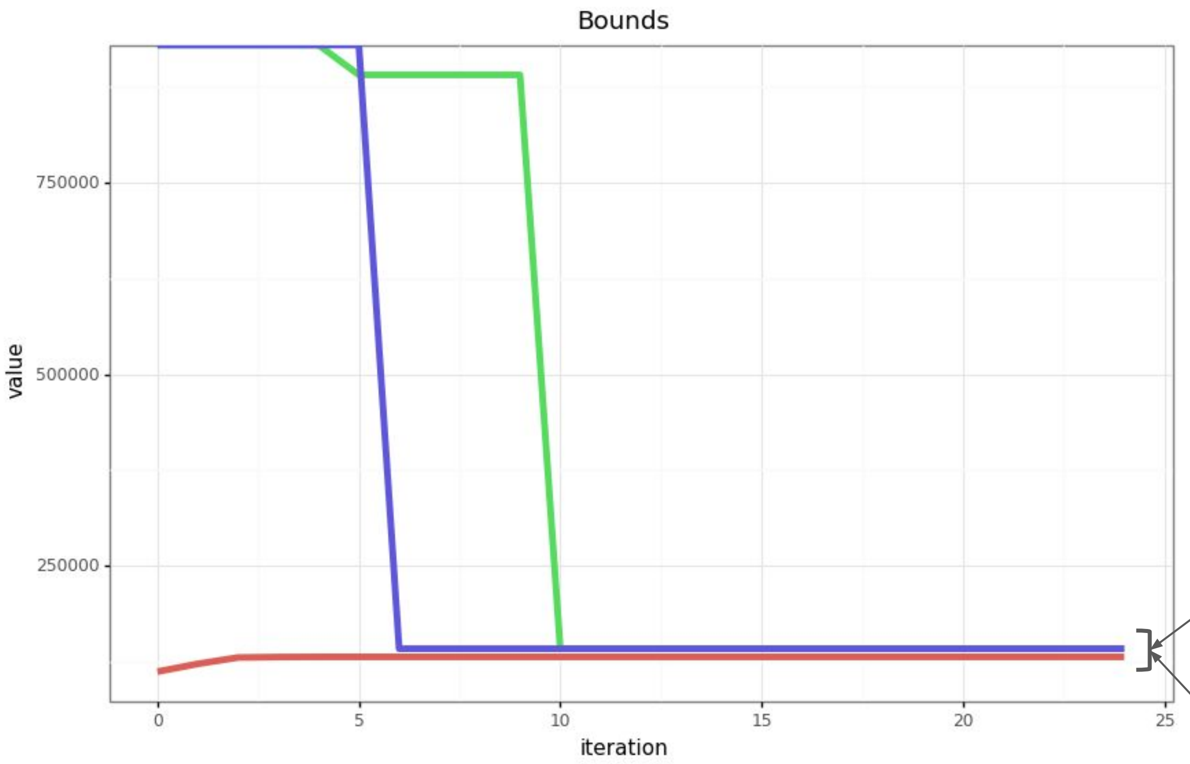
Final Solution with 7% optimality gap (2.3% from actual optimum)

Final solution found at iteration #5





Bounds and optimality gap



A: Current best φ(**λ**)

B: Current best Full Solution

C: Cutting Plane/"Master" bound (variable *z*)

Optimality Gap:
100x(1- (B/A))

Cutting Plane "convergence gap":
100x(1- (C/A))

Take-away message

Today, **only with OR** you **already** can:

1. Solve complex large-scale optimization problems
2. Identify and clarify problems to your stakeholders and users
3. Revolutionize decision processes for the better!

STOP WAITING, START OPTIMIZING NOW!

Extra: more ML/OR Synergy

Example: Knapsack problem reduction via clustering

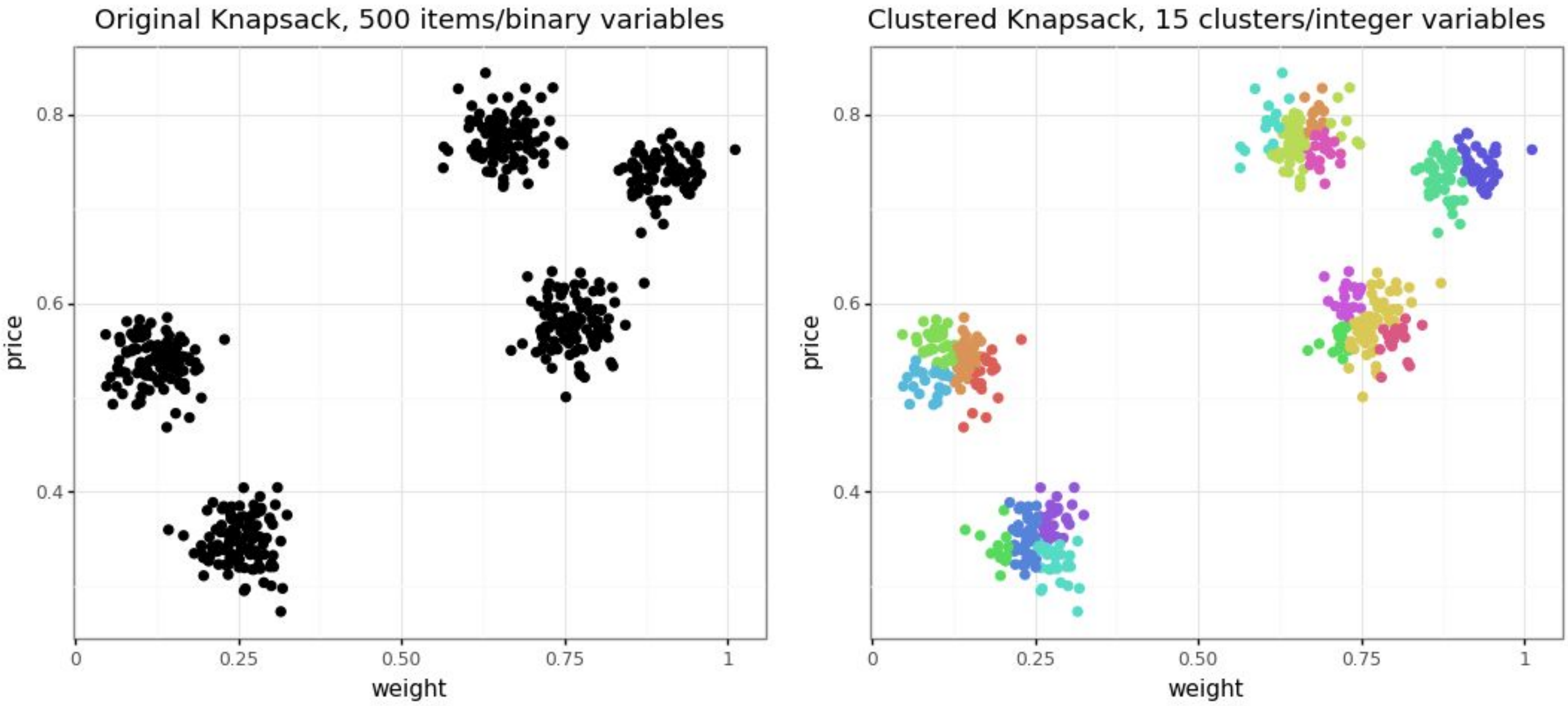
- knapsack problem with **large number of items *N***
- If **items are similar** in weight ***w*** and value ***p***, they can be **clustered**!
- Have **a variable *z* for each cluster** *j* to **count** how many of its items are put in the knapsack
- Smaller model, better representation!

$$\max \sum_{i=1}^{N} p_i x_i$$
$$\sum_{i=1}^{N} w_i x_i \leq W$$
$$x_i \in \{0,1\} \quad \forall i \in 1 \dots N$$

$$\Downarrow$$

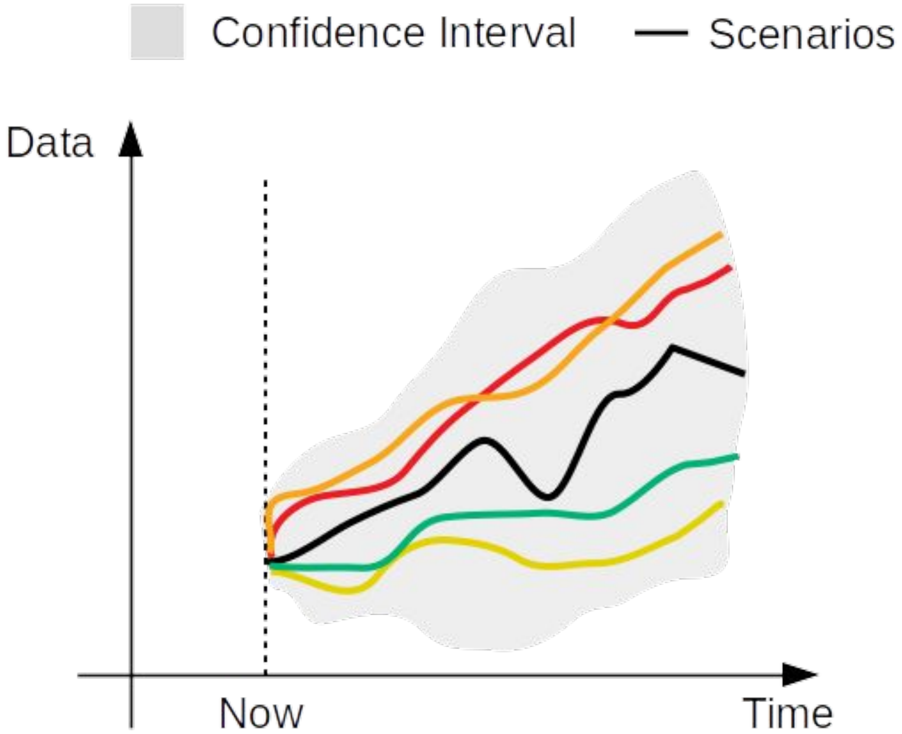
$$\max \sum_{j=1}^{K} \bar{p}_j z_j$$
$$\sum_{j=1}^{K} \bar{w}_i z_j \leq W$$
$$z_j \in \{0,1,\dots M_j\} \quad \forall j \in 1 \dots K$$

Example: Clustered Knapsack

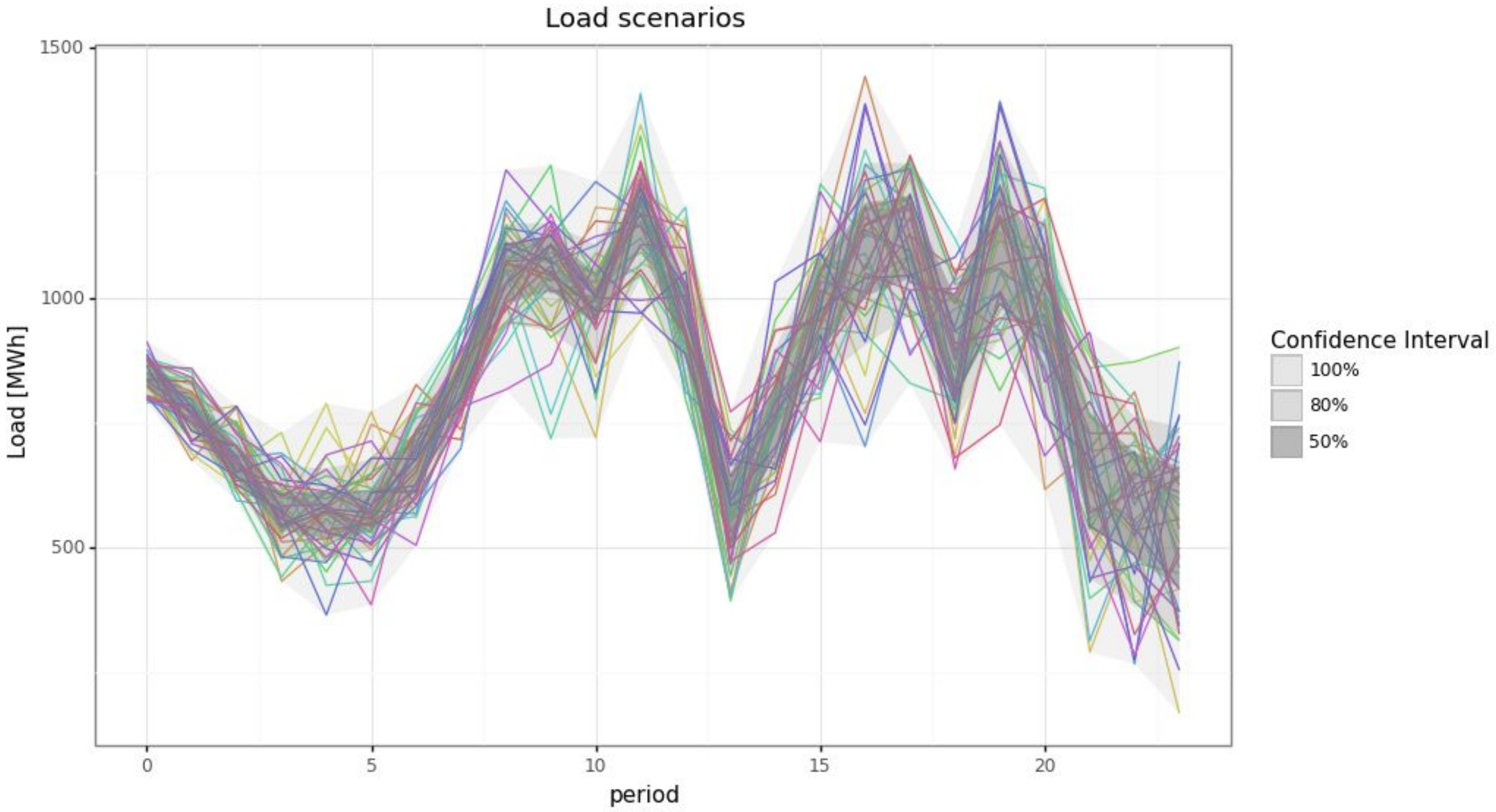


Example: exploiting Probabilistic Forecasts

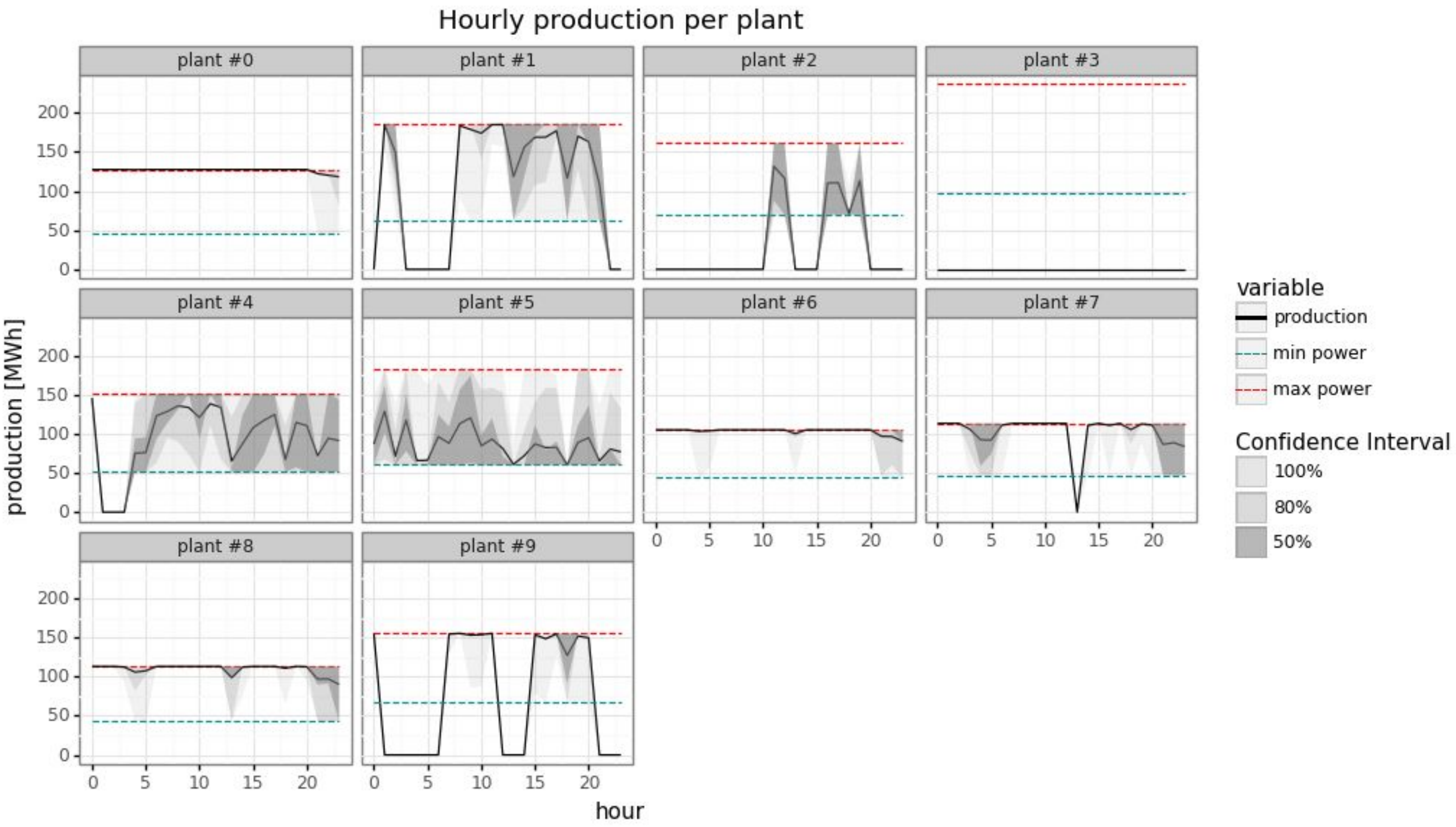
- Forecast = expectation + confidence interval
- Future will differ from expectation!
  - Errors!
  - Cannot optimize just for the expected case!
- Make solution more **robust**:
  - Sample different scenarios
  - optimize **expected outcome across all the scenarios**



Stochastic UCP: Uncertain Load



Stochastic UCP: Uncertainty-aware schedules



Stochastic UCP: comparison with Deterministic UCP