# Grassland: A Rapid Algebraic Modeling System for Million-variable Optimization


Xihan Li[1], Xiongwei Han[2], Zhishuo Zhou[3], Mingxuan Yuan[2] , Jia Zeng[2] and Jun Wang[1]

[1]University College London

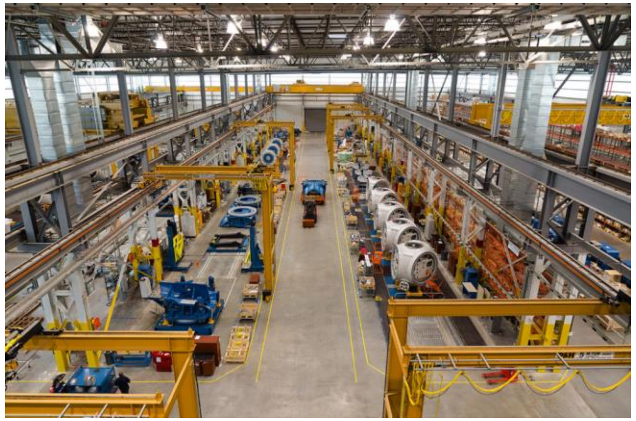
[2]Huawei Noah's Ark Lab

[3]Fudan University


Speaker: Xihan Li

CIKM 2021

# Mathematical Optimization:
# Booster of Global Economy



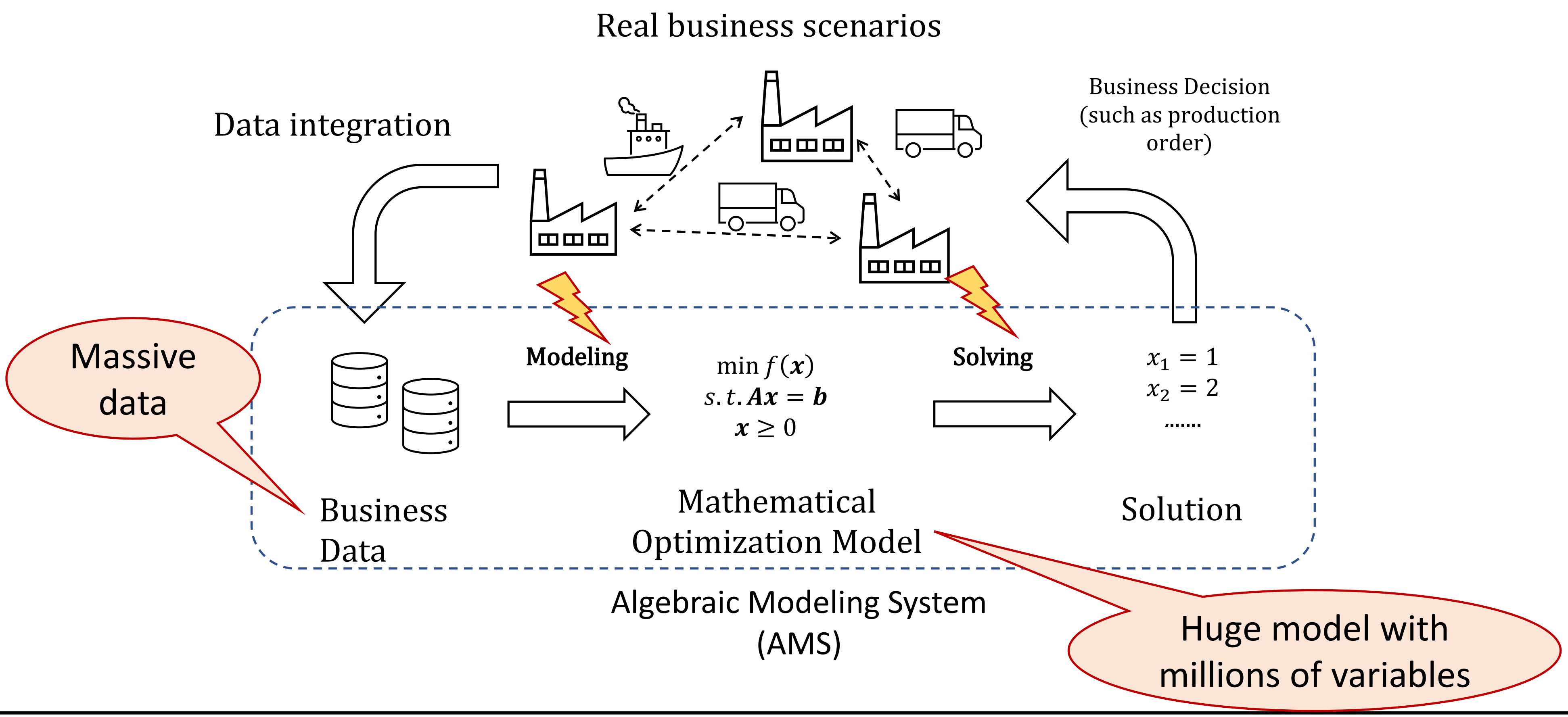
Manufacturing

Energy


Transportation

Logistics

# Mathematical optimization pipeline for practical business decision making scenarios



# Huawei's Supply Chain Scenario: Supply-Demand Analysis

**Demand Products**
**(10K+)**

**Semi-Finished Products (50K+)**


**Key Products**
**(10K+)**

**Demand**
**~$10^6$**

**Supply**
**~$10^5$**

**Complex BOM Relationship**

**Complex business rules like replacement and priority**

Sudden change of supply and demand

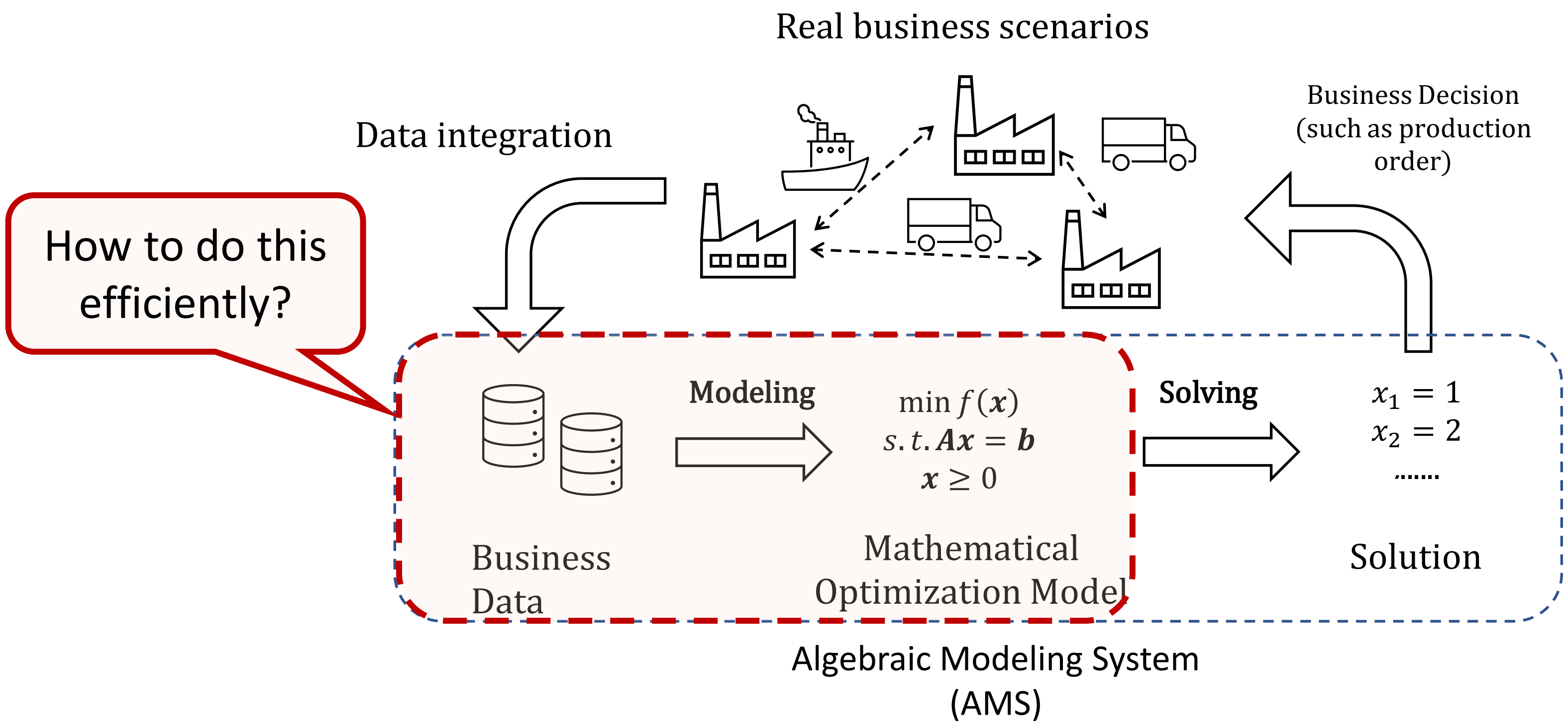
**Supply-Demand Analysis**
(modeling and solving of mathematical optimization models with millions of variables, has to be completed **in minutes**)

Assist

**Final Decisions**
Purchase
Produce
Transport
Replacement
……

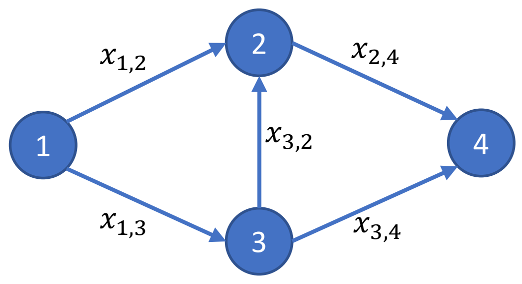
- Demand fulfillment ratio
- Bottleneck products/raw materials
- ……

UCL HUAWEI 復旦大學 FUDAN UNIVERSITY




1. Efficient modeling: A fast algorithm to instantiate millions of linear constraints in optimization

# A simple mathematical optimization example: balance constraint of network flow model




$V = \{1, 2, 3, 4\}$

$E = \begin{bmatrix} (1,2) \\ (1,3) \\ (2,4) \\ (3,2) \\ (3,4) \end{bmatrix}$

$S = [1, 0, 0, -1]$

constraint i: $\sum_{(i,j)\in E} x_{i,j} - \sum_{(j,i)\in E} x_{j,i} = s_i, \forall i \in V$

Iterate through all $i \in V$

i $= 1$, iterate through all $(1, j) \in E$ and $(i, 1) \in E$
constraint 1: $x_{1,2} + x_{1,3} = 1$

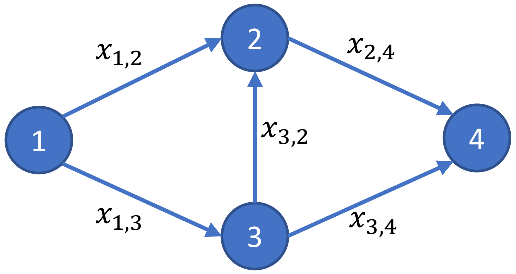
i $= 2$, iterate through all $(2, j) \in E$ and $(i, 2) \in E$
constraint 2: $x_{2,4} - x_{1,2} - x_{3,2} = 0$

i $= 3$, iterate through all $(3, j) \in E$ and $(i, 3) \in E$
constraint 3: $x_{3,2} + x_{3,4} - x_{1,3} = 0$

i $= 4$, iterate through all $(4, j) \in E$ and $(i, 4) \in E$
constraint 4: $-x_{2,4} - x_{3,4} = -1$

$\boldsymbol{O}(|\boldsymbol{V}||\boldsymbol{E}|)$ time complexity!
Cannot work when we have millions of vertices and edges

# Efficient Model Instantiation




$$V = \{1, 2, 3, 4\}$$

$$E = \begin{bmatrix} (1,2) \\ (1,3) \\ (2,4) \\ (3,2) \\ (3,4) \end{bmatrix}$$

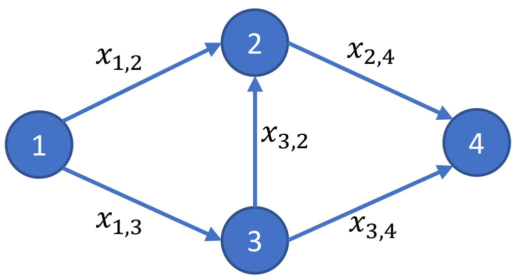
$$S = [1,0,0,-1]$$

- constraint 1: $+x_{1,2} + x_{1,3} = 1$
- constraint 2: $+x_{2,4} - x_{1,2} - x_{3,2} = 0$
- constraint 3: $+x_{3,2} + x_{3,4} - x_{1,3} = 0$
- constraint 4: $-x_{2,4} - x_{3,4} = -1$

constraint i: $+\sum_{(i,j)\in E} x_{i,j} - \sum_{(j,i)\in E} x_{j,i} = s_i, \forall i \in V$

# Efficient Model Instantiation

constraint i: $+\sum_{(i,j)\in E} x_{i,j} - \sum_{(j,i)\in E} x_{j,i} = s_i, \forall i \in V$




$$+\sum_{(i,j)\in E} x_{i,j}$$
$$-\sum_{(j,i)\in E} x_{j,i}$$

$$+x_{1,2} \quad +x_{1,3} \quad +x_{2,4} \quad +x_{3,2} \quad +x_{3,4}$$
$$-x_{1,2} \quad -x_{1,3} \quad -x_{2,4} \quad -x_{3,2} \quad -x_{3,4}$$

$V = \{1, 2, 3, 4\}$

$$E = \begin{bmatrix} (1,2) \\ (1,3) \\ (2,4) \\ (3,2) \\ (3,4) \end{bmatrix}$$

$S = [1,0,0,-1]$

constraint 1: $+x_{1,2} \quad +x_{1,3} \qquad = 1$

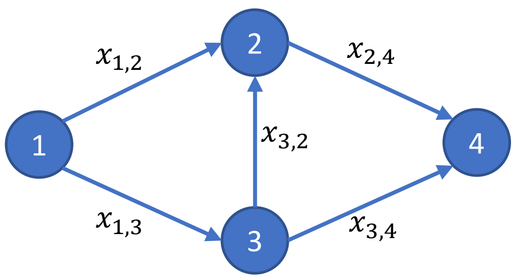
constraint 2: $-x_{1,2} \quad +x_{2,4} \quad -x_{3,2} \qquad = 0$

constraint 3: $-x_{1,3} \quad +x_{3,2} \quad +x_{3,4} \qquad = 0$

constraint 4: $-x_{2,4} \quad -x_{3,4} \qquad = -1$

$\boldsymbol{O(|E|)}$ time complexity!

# Parallelization of the model instantiation algorithm




$V = \{1, 2, 3, 4\}$

$$E = \begin{bmatrix} (1,2) \\ (1,3) \\ (2,4) \\ (3,2) \\ (3,4) \end{bmatrix}$$

$S = [1,0,0,-1]$

constraint i: $+\sum_{(i,j)\in E} x_{i,j} - \sum_{(j,i)\in E} x_{j,i} = s_i, \forall i \in V$

$$E_{out} = \begin{bmatrix} (1, 2) \\ (1, 3) \\ (2, 4) \\ (3, 2) \\ (3, 4) \end{bmatrix}, E_{in} = \begin{bmatrix} (1, 2) \\ (3, 2) \\ (1, 3) \\ (2, 4) \\ (3, 4) \end{bmatrix}$$

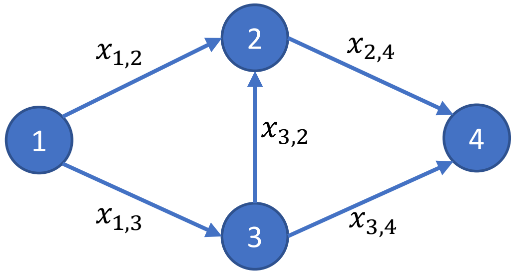
Data partition

CPU 1: $E^1_{out} = \begin{bmatrix} (1, 2) \\ (1, 3) \\ (2, 4) \end{bmatrix}, E^1_{in} = \begin{bmatrix} (1, 2) \\ (3, 2) \end{bmatrix}$

CPU 2: $E^2_{out} = \begin{bmatrix} (3, 2) \\ (3, 4) \end{bmatrix}, E^2_{in} = \begin{bmatrix} (1, 3) \\ (2, 4) \\ (3, 4) \end{bmatrix}$

# Parallelization of the model instantiation algorithm

constraint i: $+\sum_{(i,j)\in E} x_{i,j} - \sum_{(j,i)\in E} x_{j,i} = s_i, \forall i \in V$




$$V = \{1,2,3,4\}$$

$$E = \begin{bmatrix} (1,2) \\ (1,3) \\ (2,4) \\ (3,2) \\ (3,4) \end{bmatrix}$$

$$S = [1,0,0,-1]$$

CPU 1

$+x_{1,2}$ $+x_{1,3}$ $+x_{2,4}$

$-x_{1,2}$ $-x_{3,2}$

CPU 2

$+x_{3,2}$ $+x_{3,4}$

$-x_{1,3}$ $-x_{2,4}$ $-x_{3,4}$

CPU 1

c1: $+x_{1,2}$ $+x_{1,3}$ $= 1$

c2: $-x_{1,2}$ $+x_{2,4}$ $-x_{3,2}$ $= 0$

CPU 2

c3: $-x_{1,3}$ $+x_{3,2}$ $+x_{3,4}$ $= 0$

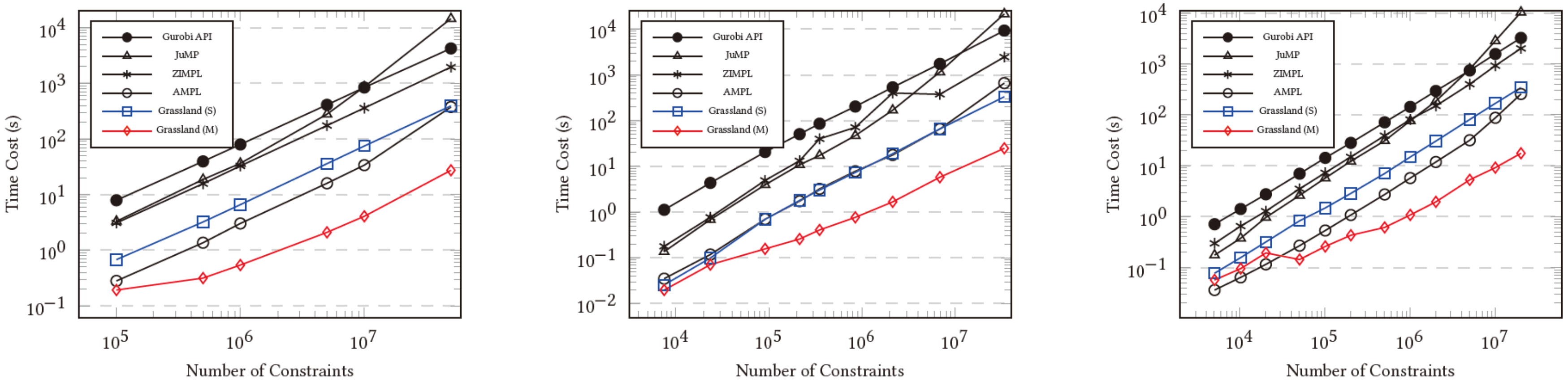
c4: $-x_{2,4}$ $-x_{3,4}$ $= -1$

# Offline benchmark result

| | P-Median | Offshore Wind Farming | Food Manufacture I |
|---|---|---|---|
| Gurobi Py API | 410.20 | 533.71 | 744.39 |
| JuMP | 278.08 | 169.08 | 789.86 |
| ZIMPL | 174.00 | 400.47 | 399.16 |
| AMPL | 15.94 | 17.71 | 31.65 |
| Grassland (S) | 35.91 | 18.85 | 80.83 |
| Grassland (M) | **2.09** | **1.67** | **5.28** |

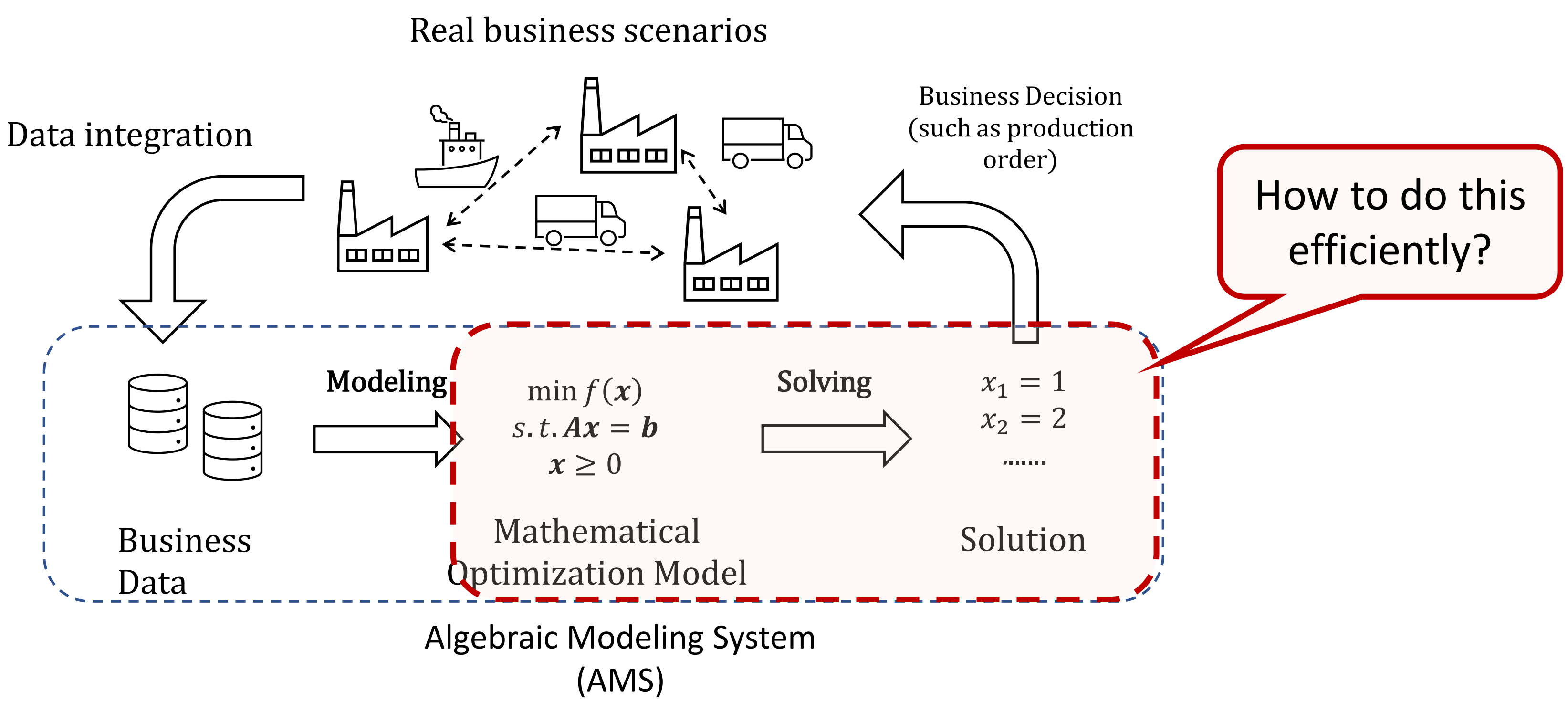
**Table 1: Model Instantiation Benchmark. Total time (in seconds) to process the model definition and produce the output file in CPLEX LP format.**

6-10x speedup over leading commercial modeling software in multi-thread setting




(a) (b) (c)

**Figure 6: Offline model instantiation benchmark on (a) P-Median (b) Offshore Wind Farming (c) Food Manufacture I.**




2. Efficient solving: sequential decomposition of large-scale business optimization model

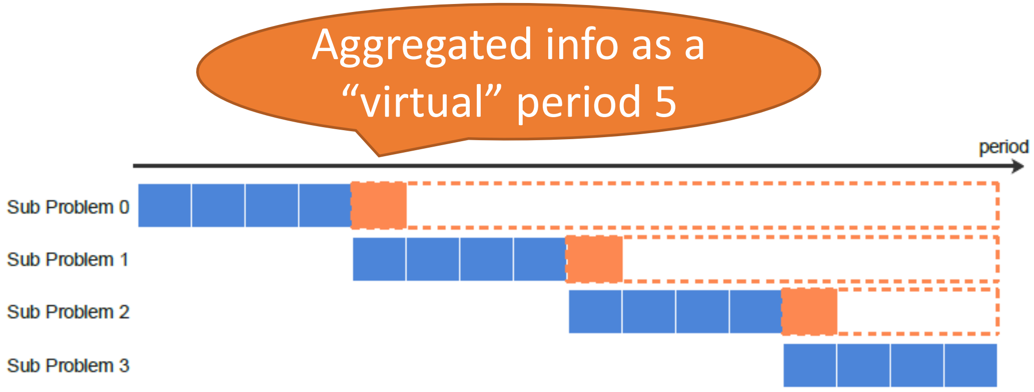
# Why business optimization models are so large?

- Sequential, dynamic decision making
- The number of variables are in direct proportion to the decision horizons $T$
  - E.g., when we have 10K decision variable in a single period, we will need 10K * 100 = 1M variables for 100 periods ($T = 100$)
- Can we partition the periods (rolling horizon)?
  - Yes, but the decision will be very short-sighted, thus global optimality will be lost

$$
\begin{aligned}
\mathbf{A}_1 \mathbf{x}_1^T &= \mathbf{b}_1 \\
\mathbf{A}_2 [\mathbf{x}_1, \mathbf{x}_2]^T &= \mathbf{b}_2 \\
\cdots & \\
\mathbf{A}_T [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T]^T &= \mathbf{b}_T
\end{aligned}
$$

# Forward Rolling Horizon

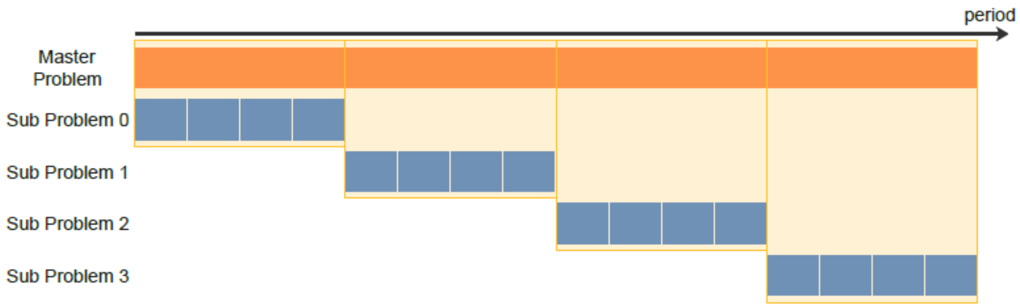
- Divide the model into h sub-models
- Add “aggregated information” to the end of the sub-model.
  - E.g., when we solve the sub-problem in 1-4 period, we aggregate the information in 5-16 period into 1 period, and attach it to the end of the sub-problem as a “virtual” period 5.




(a) Forward RH (FRH), [8]

# Our Proposed Method: Guided Rolling Horizon

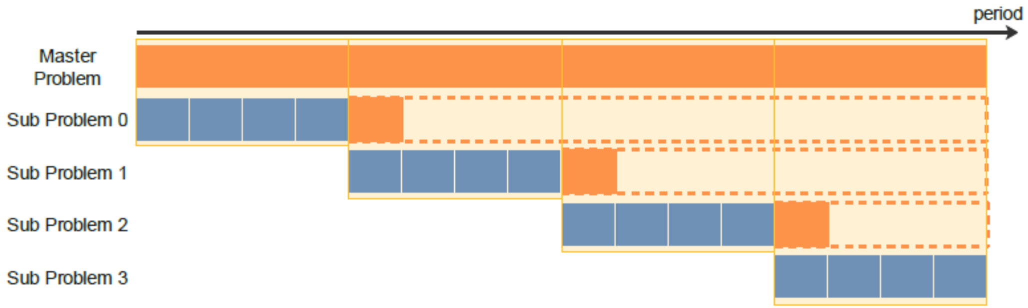
- First, aggregate all data into h periods
- Then, solve a “master problem” of h periods with aggregated data
- Finally, solve h sub-problems sequentially. Add a soft constraint that the sub-problem should be aligned with the master problem as much as possible.




(b) Guided RH

# Guided Forward Rolling Horizon

- Forward Rolling Horizon + Guided Rolling Horizon




(c) Guided FRH

# Fine-tuning of approximated solutions

- In reality, we are more concerned about the decision variables in first several periods, since they need to be executed soon.
- When we have an approximated feasible solution, we can “release” the variables in first several periods and fix the others, and re-optimize the model in the full horizon.

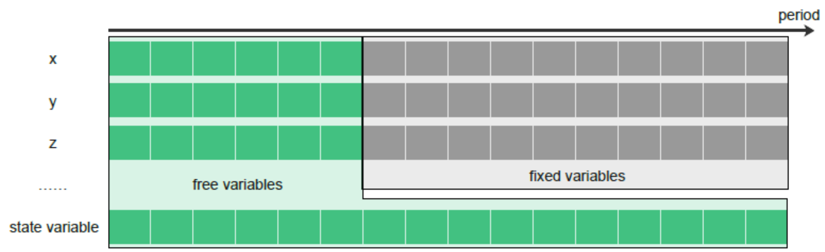
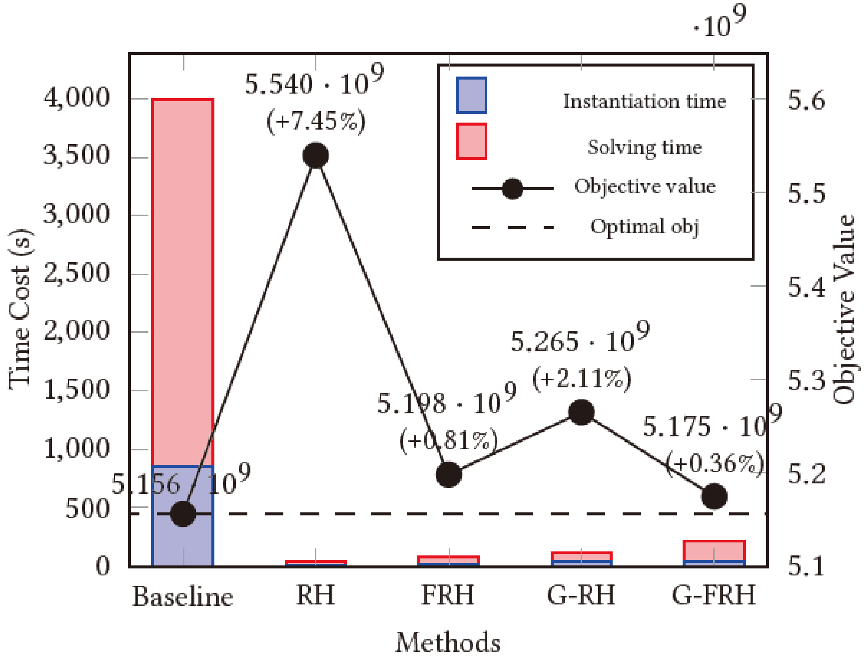
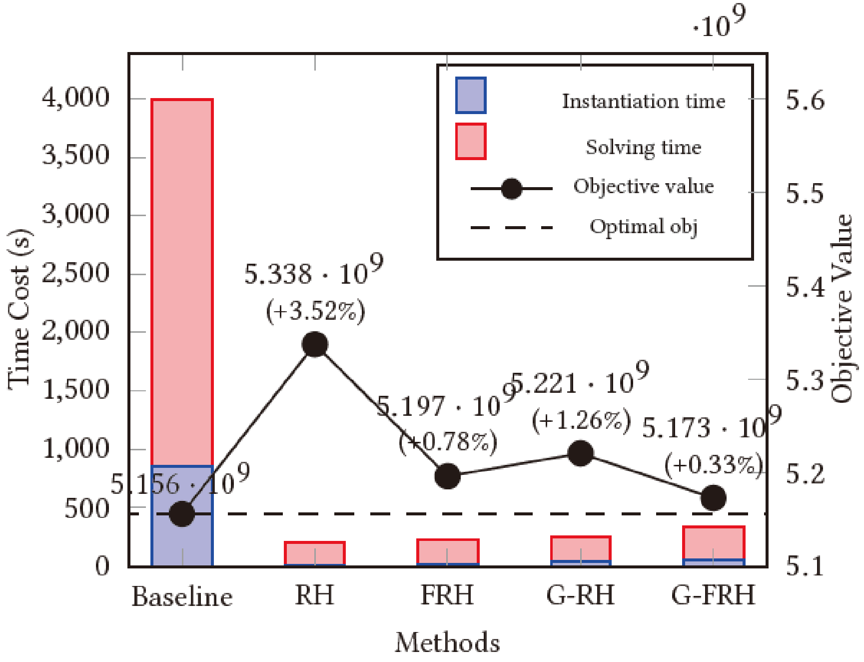



Figure 5: The fine-tuning procedure. The grey variables are fixed while the green variables are to be re-optimized. State variables whose value are determined by other variables keep free in the whole sequence.

# Online experiments on Huawei supply chain scenario



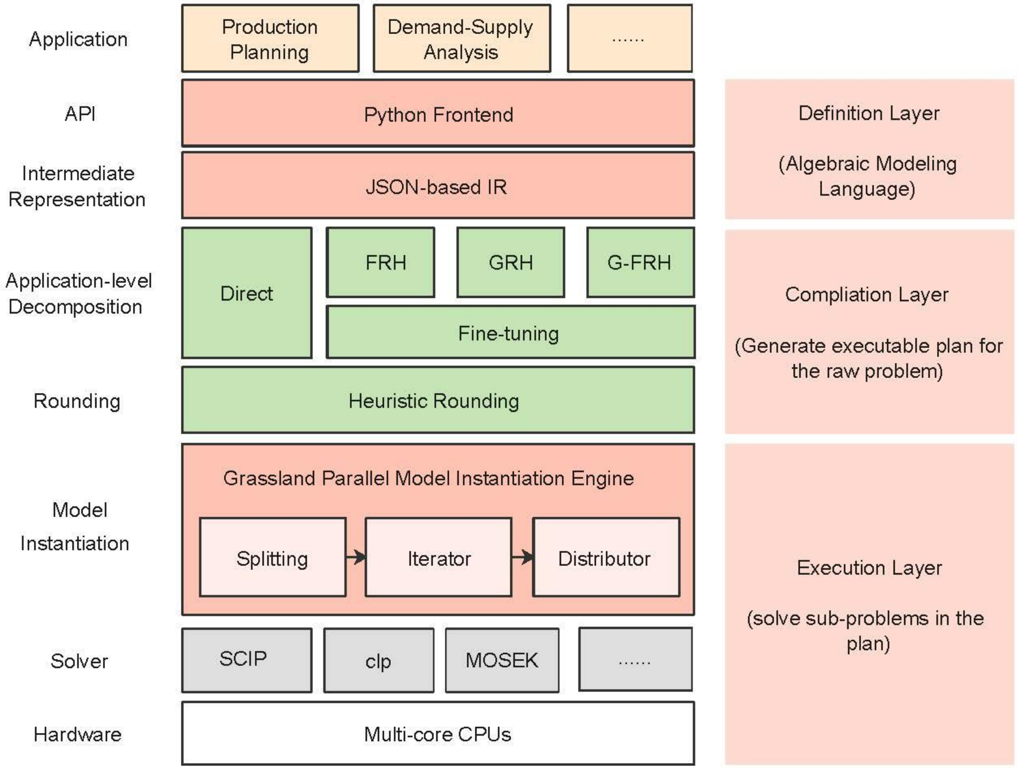

20x speed acceleration (4000s → 200s) with optimality loss of only 3.6‰




The optimality loss can be further reduced to 3.3‰ with fine-tuning

# Grassland in business practice

- Cooperated with Huawei Cloud, Grassland will be integrated as the default modeling system of Huawei OptVerse AI Solver
    - https://www.huaweicloud.com/product/modelarts/optverse.html



# Thank you!

Xihan Li
Department of Computer Science,
University College London
xihan.li@cs.ucl.ac.uk
https://snowkylin.github.io