

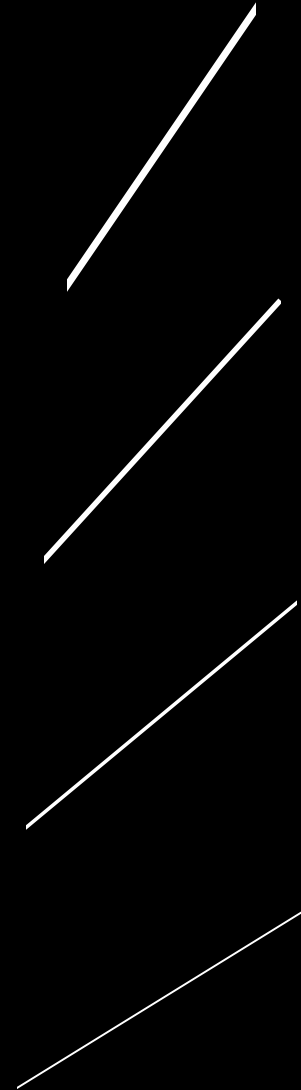
# Democratizing Optimization Modeling: Status, Challenges and Future Directions

—  
Segev Wasserkrug,

Senior Technical Staff Member, IBM Research - Israel

Research Associate, Faculty of Decision and Data Sciences,  
Technion, Israel Institute of Technology

**IBM Research**



IBM does not endorse any third parties referenced in the presentation

# Mathematical optimization can provide significant benefits in many use cases and industries

## Well-Documented Optimization ROI Cases

2 Chilean Forestry firms*	Timber Harvesting	\$20M/yr + 30% fewer trucks
UPS*	Air Network Design	\$40M/yr + 10% fewer planes
South African Defense*	Force/Equip Planning	\$1.1B/yr
Motorola*	Procurement Mgmt	\$100M-150M/yr
Samsung Electronics*	Semiconductor Mfg	50% reduction in cycle times
SNCF (French RR)*	Scheduling & Pricing	\$16M/yr rev + 2% lower op ex
Continental Airlines*	Crew Re-scheduling	\$40M/yr
AT&T*	Network Recovery	35% reduction spare capacity
Grant Mayo van Otterloo*	Portfolio Optimization	\$4M/yr
Pepsi Bottling Group	Production Sourcing	\$6M inv reduction + 2% fewer miles
Fonterra	Dairy Distribution	\$15M annual savings
NA Brewing Company	Mfg Sourcing + Distribution	\$150M/yr transportation savings
US Water Products Mfg	Inventory Optimization	\$6.2M working capital reduction

\*Franz Edelman Competition Finalists, Science of Better, <http://www.scienceofbetter.org>, Published Case Studies

# Requires creating an optimization model for the business problem

This model is to determine the order of a set of custom computers to be processed on an assembly line. Once the order is assigned, it is kept from start to finish. The custom computers have different lists of components to be contained, which are given in the array "computer".

The ordering of the computers is constrained by the following assembly rules for each component:

- 1) There must be a minimum number of computers in a row that need this component ("minSeq");
- 2) There is an upper bound on the number of computers in a row that can have that component;
- 3) Each component also has a list of illegal followers ("illegalFollowers") so that the next computer cannot have a component which appears in the illegal followers list for this component.

“Natural Language” Description

# Model in modeling Language

```
dvar int order[AllComputers] in AllComputers;

/* -----
 * Constraints
 * ----- */

subject to
{
  allDifferent(order);

  // Min/Max sequences
  forall (c in UsedComponentTypes) {
    forall ( p in 1..nComputers - components[c].maxSeq )
      // If there are maxSeq # of component c in a row starting from position p to p+maxSeq-1,
      // => the (p+ maxSeq)th computer must not contain component c.
      (sum(s in p..p+components[c].maxSeq-1) (order[s] in componentInComputer[c]) == components[c].maxSeq)
      =>
      //not (order[p+components[c].maxSeq] in componentInComputer[c]);
      order[p+components[c].maxSeq] not in componentInComputer[c];

      // The components in the 1st computer must appear at least minSeq # of times in a row.
      (order[1] in componentInComputer[c])
      =>
      ((sum( s in 1..components[c].minSeq) (order[s] in componentInComputer[c])) >= components[c].minSeq);

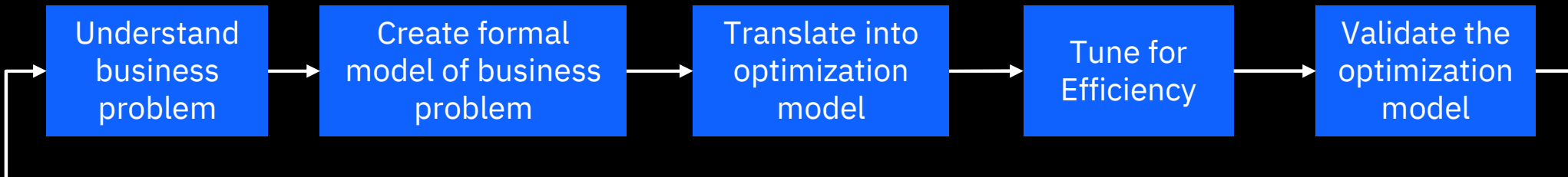
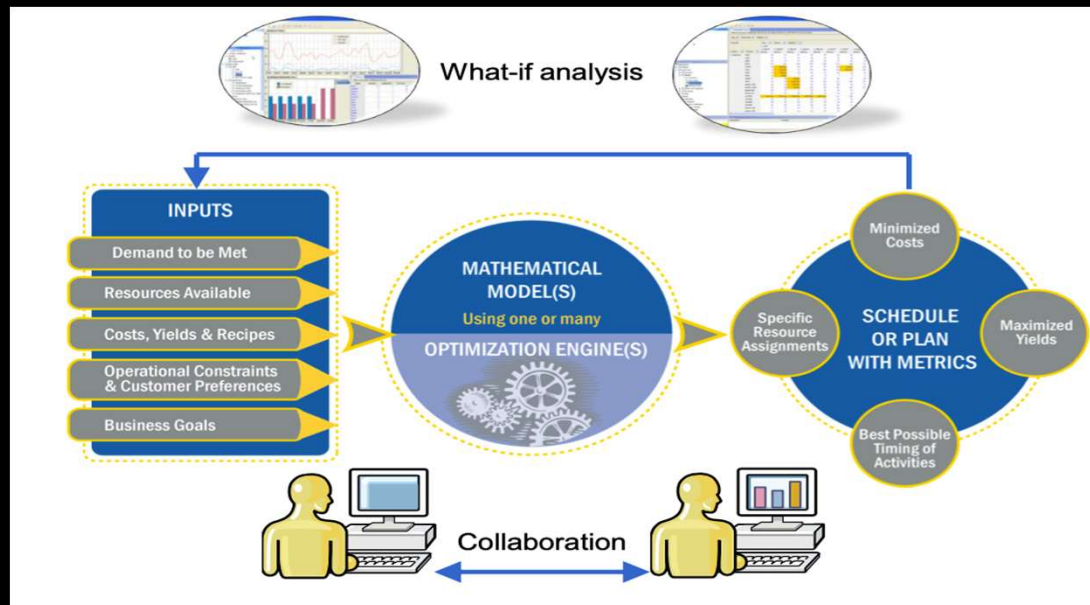
      forall ( p in 1..nComputers-components[c].minSeq )
        // Every component that is not in computer p but appears in computer p+1
        // must appear minSeq # of times in a row from p+1 to p+minSeq.
        (((order[p] not in componentInComputer[c])
          && (order[p+1] in componentInComputer[c])) =>
          (sum(s in p+1..p+components[c].minSeq) (order[s] in componentInComputer[c]))
            == components[c].minSeq);
  };

  forall (c in HasIllegalFollowers) // for component c that has an illegal follower,
  forall( p in 1..nComputers-1) // for computer p
  forall( c2 in UsedComponentTypes : c2 in components[c].illegalFollowers)
```

# Mathematically

$$x^* = \arg \min_{x \in \mathcal{X} \cap \Omega(p)} f(x, p)$$

# Current Optimization Modeling Process



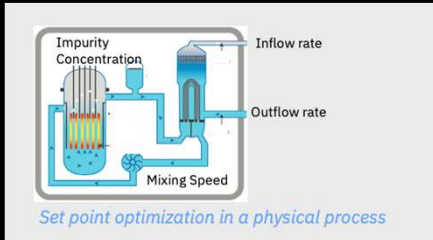
# Comparing optimization modeling paradigms

Approach	Pros	Cons
Custom algorithm	Provides best solution for specific problem	<ul style="list-style-type: none"> <li>• Most difficult to create</li> <li>• Very difficult to customize with new constraints.</li> </ul>
Mathematical Programming (e.g. MILP)	<ul style="list-style-type: none"> <li>• Provides optimality gap</li> <li>• Extremely efficient when modelled correctly and tuned</li> </ul>	<ul style="list-style-type: none"> <li>• Requires rare optimization modeling expertise to model and tune</li> </ul>
Metaheuristics	<ul style="list-style-type: none"> <li>• Easy to create running model</li> <li>• Can be significantly more efficient than MILP for some problem types (e.g., VRP)</li> </ul>	<ul style="list-style-type: none"> <li>• Requires deep understanding of specific algorithm to provide good solution</li> <li>• No theoretical optimality gap</li> <li>• Often less suitable for continuous decision variables</li> </ul>
Markov Decision Processes/Reinforcement Learning	<ul style="list-style-type: none"> <li>• Higher level abstraction (states/actions/rewards)</li> <li>• Inherent handling of uncertainty</li> </ul>	<ul style="list-style-type: none"> <li>• Curse of dimensionality/extensive hyperparameter and algorithm tuning</li> <li>• Incorporating constraints</li> <li>• Inherent uncertainty makes explainability more difficult</li> </ul>



# AI Can Significantly Simplify Model Creation but Introduces new Problems

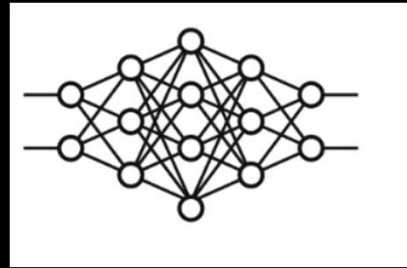
# Constraint Learning – learning difficult to model constraints from data (Subramanian et. al. 2019, Maragno et. al. 2021,...)



$$D = \{(x_i, p_i, z_i)\}$$

- $x_i$  - historical decision
- $p_i$  - historical uncontrollable
- $z_i$  - historical objective and constraints outcome

Raw material Inflow rate	Utility Water Flow rate	Mixing Speed	Active Ingredient Concentration	Impurity Concentration	Outflow Rate
97.7170000000001	17.8940000000000	11.47	5.4156	10.293	46.494
75.947	83.5579999999999	15.686	15.112	17.555	16.236
91.9179999999999	38.299	31.239	11.59	11.71	57.995
16.973	81.07	73.962	8.1577	17.289	53.614



$$\begin{aligned}
 & \min_{\mathbf{x}} \phi(\mathbf{y}_L) \\
 & \text{s.t. } \mathbf{y}_1 = f_1(\mathbf{x}_1), \\
 & \mathbf{y}_l = f_l(\mathbf{z}_{l-1}, \mathbf{x}_l), \quad \forall l = 2, \dots, L, \\
 & \mathbf{A}_l \mathbf{y}_l + \mathbf{B}_l \mathbf{z}_l \leq \mathbf{b}_l, \quad \forall l = 1, \dots, L-1, \\
 & \underline{\mathbf{x}}_l \leq \mathbf{x}_l \leq \bar{\mathbf{x}}_l, \quad \forall l = 1, \dots, L, \\
 & \underline{\mathbf{y}}_l \leq \mathbf{y}_l \leq \bar{\mathbf{y}}_l, \quad \forall l = 1, \dots, L, \\
 & \underline{\mathbf{z}}_l \leq \mathbf{z}_l \leq \bar{\mathbf{z}}_l, \quad \forall l = 1, \dots, L-1,
 \end{aligned}$$

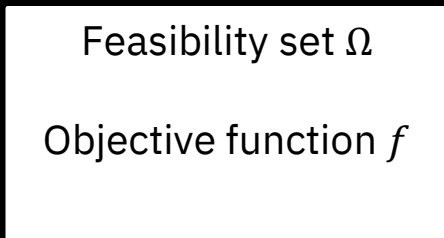
Introduces inaccuracies different from the one in traditional ML which need to be accounted for

## Example Problem – World Food Palatability [Peters et. al. 2021]

- Provide optimal food basket to developing countries
- Objective: Minimize costs
- Explicit constraints: Nutritional composition
- Hard to model constraints: Palatability
  - Create regression function using data from historical baskets to palatability score  $h(x)$
  - Add constraint:  $h(x) \geq pthres$

# Problem formulation

”Actual”  
optimization  
formulation



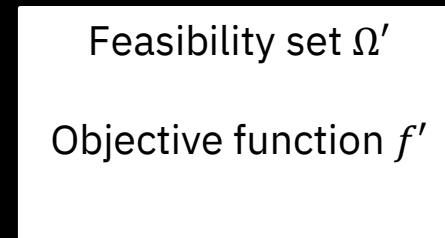
$$x^* = \arg \min_{x \in \mathcal{X} \cap \Omega(p)} f(x, p)$$

Desired outcome

$D$



Data driven  
optimization  
formulation



$$x'^* = \arg \min_{x \in \mathcal{X} \cap \Omega'(p)} f'(x, p)$$

Obtainable outcome

How can we ensure that  $x'^*$  is a good solution for the actual problem?

SE

Please create an OPL optimization model for vehicle routing



Here is a simple OPL optimization model for a vehicle routing problem:

This model uses binary decision variables  $y[i]$  to represent whether customer  $i$  is visited and integer decision variables  $x[i][j]$  to represent the order in which customers are visited. The constraints ensure that each customer is visited at most once, customers with demand are visited, and the total demand served by each vehicle does not exceed its capacity. The objective is to minimize the number of customers visited.

```
// Data Declarations
int n = ...; // number of customers
int m = ...; // number of vehicles
int Q = ...; // capacity of each vehicle
int demand[1..n] = ...; // demand for each customer
int x[1..n][1..n] = ...; // x[i][j] = 1 if customer i is served before customer j
int y[1..n] = ...; // y[i] = 1 if customer i is visited

// Decision Variables
dvar boolean y[1..n];
dvar int x[1..n][1..n];

// Constraints
subject to {
  // At most one customer is served before another customer
  forall (i in 1..n)
    sum(j in 1..n) x[i][j] + sum(j in 1..n) x[j][i] == 1;

  // Only customers with demand are visited
  forall (i in 1..n)
    y[i] <= demand[i];

  // Total demand served by each vehicle does not exceed its capacity
  forall (i in 1..n, j in 1..n)
    sum(k in 1..m) x[i][j] * y[i] <= Q;
}

// Objective
minimize
  sum(i in 1..n) y[i];
```

SE

Can you add a constraint saying that all routes should begin and end in a given location?



Yes, to add a constraint that all routes should begin and end at a given location, you can use the following code:

This constraint ensures that all routes start and end at the same location (specified by `start` and end`). The x[start][i] + x[i][end] expression represents the total number of times customer i is served (as either the first or last customer on a route), and the y[i] represents the binary decision of whether customer i is visited. The constraint x[start][i] + x[i][end] == y[i] says that a customer i is only visited if it is either the first or last customer on a route.`

```
// Data Declarations
int n = ...; // number of customers
int m = ...; // number of vehicles
int Q = ...; // capacity of each vehicle
int demand[1..n] = ...; // demand for each customer
int start = ...; // starting location
int end = ...; // ending location
int x[1..n][1..n] = ...; // x[i][j] = 1 if customer i is served before customer j
int y[1..n] = ...; // y[i] = 1 if customer i is visited

// Decision Variables
dvar boolean y[1..n];
dvar int x[1..n][1..n];

// Constraints
subject to {
  // At most one customer is served before another customer
  forall (i in 1..n)
    sum(j in 1..n) x[i][j] + sum(j in 1..n) x[j][i] == 1;

  // Only customers with demand are visited
  forall (i in 1..n)
    y[i] <= demand[i];

  // Total demand served by each vehicle does not exceed its capacity
  forall (i in 1..n, j in 1..n)
    sum(k in 1..m) x[i][j] * y[i] <= Q;

  // All routes start and end at the same location
  forall (i in 1..n)
    x[start][i] + x[i][end] == y[i];
}

// Objective
minimize
  sum(i in 1..n) y[i];
```

SE

Can you give me a version where also the fuel is minimised?



Sure! Here's a version of the optimization model that also minimizes the fuel consumption:



This model minimizes the fuel consumption, which is represented by the total distance traveled between all pairs of customers  $i$  and  $j$  ( $\text{dist}[i][j]$ ) that are visited in the specified order ( $\text{x}[i][j]$ ). The other constraints from the previous model are unchanged, ensuring that only customers with demand are visited, that the total demand served by each vehicle does not exceed its capacity, and that all routes start and end at the same location.

```
// Data Declarations
int n = ...; // number of customers
int m = ...; // number of vehicles
int Q = ...; // capacity of each vehicle
int demand[1..n] = ...; // demand for each customer
int start = ...; // starting location
int end = ...; // ending location
int dist[1..n][1..n] = ...; // distance between each pair of customers
int x[1..n][1..n] = ...; // x[i][j] = 1 if customer i is served before customer j
int y[1..n] = ...; // y[i] = 1 if customer i is visited

// Decision Variables
dvar boolean y[1..n];
dvar int x[1..n][1..n];

// Constraints
subject to {
    // At most one customer is served before another customer
    forall (i in 1..n)
        sum(j in 1..n) x[i][j] + sum(j in 1..n) x[j][i] == 1;

    // Only customers with demand are visited
    forall (i in 1..n)
        y[i] <= demand[i];

    // Total demand served by each vehicle does not exceed its capacity
    forall (i in 1..n, j in 1..n)
        sum(k in 1..m) x[i][j] * y[i] <= Q;

    // All routes start and end at the same location
    forall (i in 1..n)
        x[start][i] + x[i][end] == y[i];
}

// Objective
minimize
    sum(i in 1..n) sum(j in 1..n) dist[i][j] * x[i][j];
```

SE

Can you remove the constraint that only customers with demand are visited?



Sure! Here's a version of the optimization model where the constraint that only customers with demand are visited is removed:

In this model, the constraint  $y[i] \leq \text{demand}[i]$  has been removed, so all customers can potentially be visited, regardless of whether they have demand or not. The other constraints from the previous models are unchanged, ensuring that the total demand served by each vehicle does not exceed its capacity, and that all routes start and end at the same location.

```
// Data Declarations
int n = ...; // number of customers
int m = ...; // number of vehicles
int Q = ...; // capacity of each vehicle
int demand[1..n] = ...; // demand for each customer
int start = ...; // starting location
int end = ...; // ending location
int dist[1..n][1..n] = ...; // distance between each pair of customers
int x[1..n][1..n] = ...; // x[i][j] = 1 if customer i is served before customer j
int y[1..n] = ...; // y[i] = 1 if customer i is visited

// Decision Variables
dvar boolean y[1..n];
dvar int x[1..n][1..n];

// Constraints
subject to {
  // At most one customer is served before another customer
  forall (i in 1..n)
    sum(j in 1..n) x[i][j] + sum(j in 1..n) x[j][i] == 1;

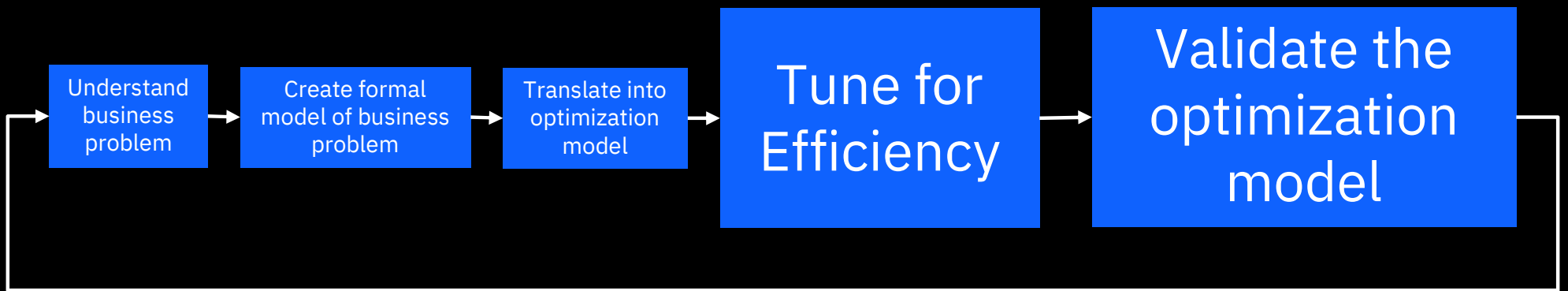
  // Total demand served by each vehicle does not exceed its capacity
  forall (i in 1..n, j in 1..n)
    sum(k in 1..m) x[i][j] * y[i] <= Q;

  // All routes start and end at the same location
  forall (i in 1..n)
    x[start][i] + x[i][end] == y[i];
}

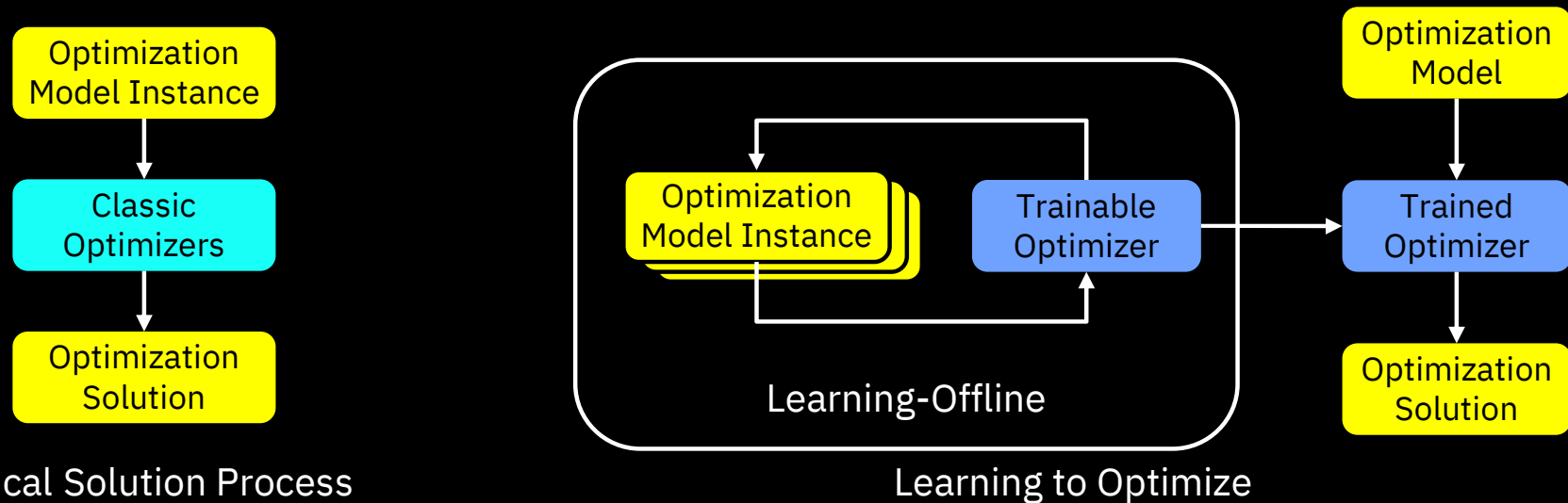
// Objective
minimize
  sum(i in 1..n) sum(j in 1..n) dist[i][j] * x[i][j];
```



# Shift in Problem Focus



# Tune for efficiency - Learning to optimize



(Adapted from “Learning to Optimize: A Primer and A Benchmark” Chen et. al 2022)

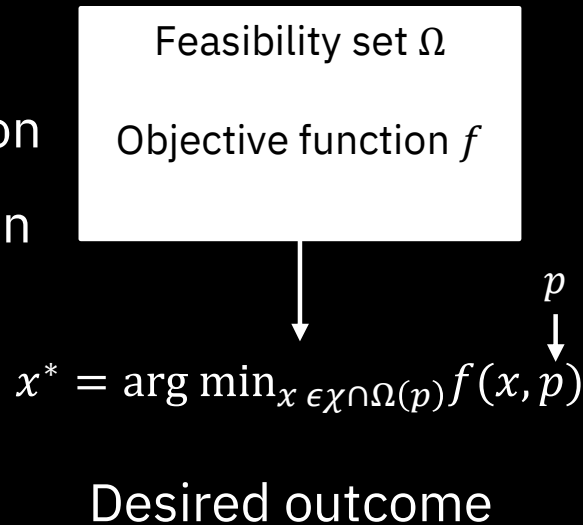
- Node selection [He et al., 2014], [Song, Lanka, Zhao, et al., 2018], Variable selection [Nair et al., 2020], [Nair et al., 2020], Cutting planes selection [Baltean-Lugojan et al., 2018], [Tang et al., 2019], Primal heuristic selection [Khalil, Dilkina, et al., 2017], [Khalil, Dilkina, et al., 2017],...
- ECOLE - *Extensible Combinatorial Optimization Learning Environments* : <https://www.ecole.ai/>

# Model Verification for Constraint Learning

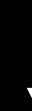
Joint work with Orit Davidovich, Parikshit Ram and Dharmashankar Subramanian

# Problem formulation

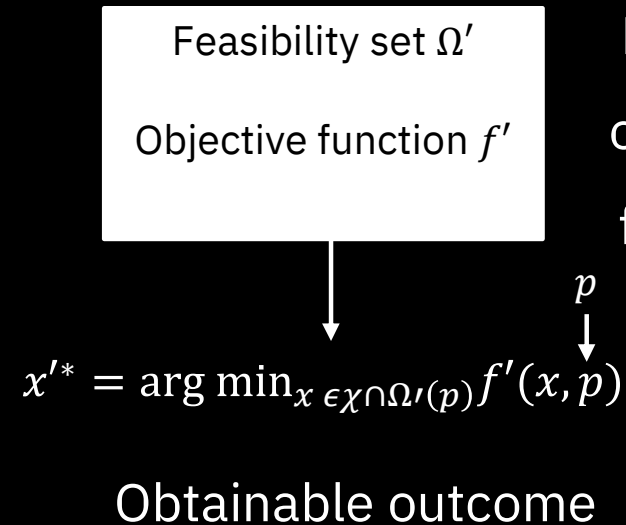
”Actual”  
optimization  
formulation



$D$

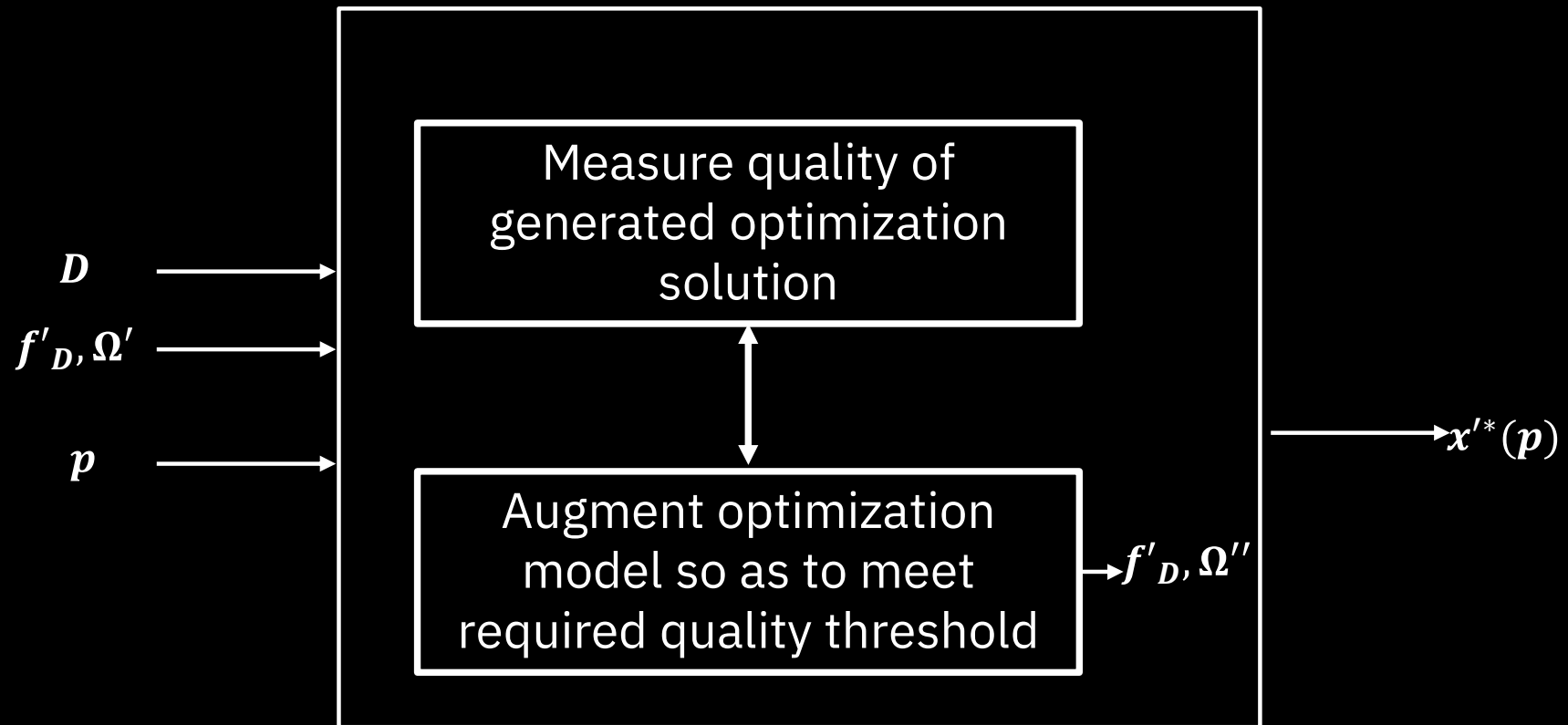


Data driven  
optimization  
formulation



How can we ensure that  $x'^*$  is a good solution for the actual problem?

# Ensuring improvements of generated models



# Approach

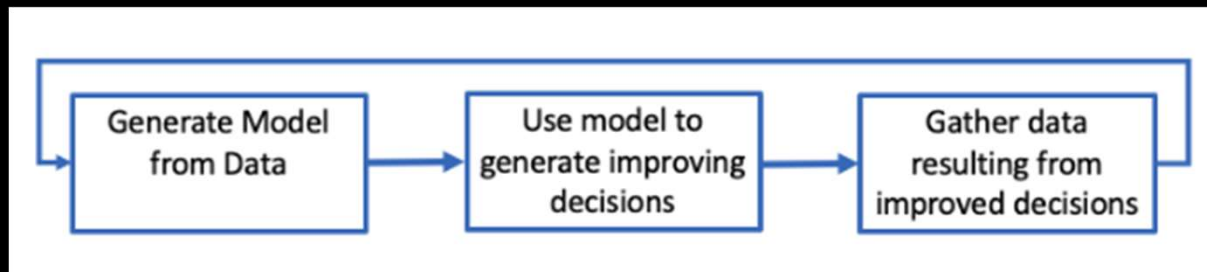
## What we want:

$$\Pr(x'^* \in \Omega'(p)) \geq (1 - \delta_1)$$

$$\Pr(f(x'^*, p) - f(x^*, p) \leq \varepsilon) \geq (1 - \delta_2)$$

## Instead:

$$\Pr(f(x_0, p) - f(x'^*, p) \geq \varepsilon) \geq (1 - \delta_2)$$



# Probability of Improvement (POI)

Uncertainty on  $f$  represented by a **Gaussian Process**

$f|D \text{ data} \sim GP$

$\Rightarrow$  Value @ Point:  $f(x, p) \sim \mathcal{N}(\mu(x, p), \sigma^2(x, p))$

$\Rightarrow$  **Probability of Improvement:**  $\Pr[f(x_0, p) - f(x'^*, p) \geq \epsilon]$

# Weighted Probability of Improvement

Prior belief on  $f$

Posterior:  $f | D \sim GP_{f|D}$

Marginalize  $\Rightarrow$  Weighted Probability of Improvement

$$\Pr[f(x_0, p) - f(x'^*, p) \geq \epsilon] = \sum_{D' \subseteq D} \Pr_{GP_{f|D'}}[f(x_0, p) - f(x'^*, p) \geq \epsilon]$$

Similarly , probability of constraint satisfaction (PoCS):  $\Pr[x'^*(p) \in \Omega(p)]$



# PoI Experimental Results

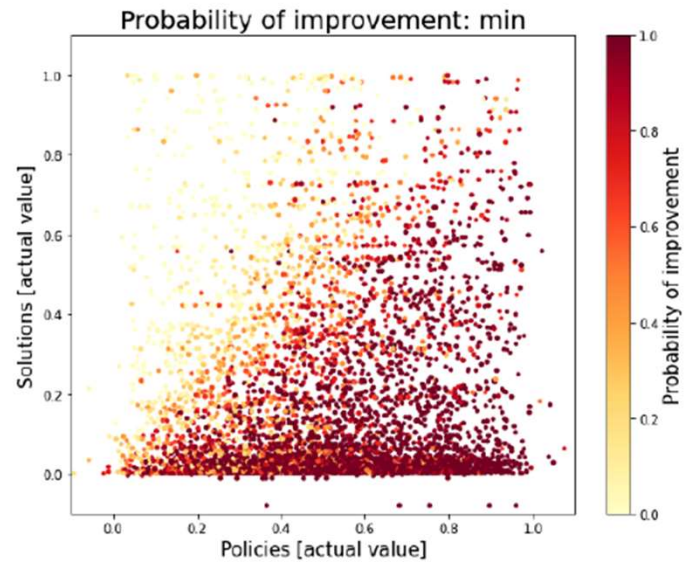


Figure 3: PoI calculated for 5593 solution-policy pairs associated with 1718 randomly generated triplets  $(f, D, \Omega)$ .

	$f(\hat{x}^*) < f(x_0)$	$f(\hat{x}^*) > f(x_0)$
PoI > 0.5	4201	146
PoI < 0.5	513	733

Table 1: PoI results summary.

# Model Augmentation

## Data Sufficiency

Optimal solution  $\epsilon$ -“close” to existing dataset

## Model Fidelity

Learnt objective is  $\delta$ -“accurate” in vicinity of solution

$\epsilon$  and  $\delta$  can be tuned to achieve desired PoI/PoCS

Can be used to enhance trust: user can ensure that solution is close to previously existing solutions and uncontrollables

# Augmented optimization formulation

$$x'^* = \arg \min_{x \in \mathcal{X} \cap \Omega(p)} f'(x, p)$$

subject to:

*Data Sufficiency Constraints*

*Model Fidelity Constraints*

- Can be formulated as Mixed Integer Linear Program (MILP) when original problem is MILP
- Theoretical guarantee: In some settings, can provide a bound for  $\Pr(f(x'^*, p) - f(x^*, p) \leq \varepsilon) \geq (1 - \delta_2)$

# Results on WFP Problem

Palatability Threshold	OptiCL w/o TR		OptiCL w/ TR		DSMF w PoCS $\geq 0.95$ (Ours)	
	Obj	GR	Obj	GR	Obj	GR
0.50	3213 $\pm$ 1.27	0.01 $\pm$ 0.0003	3431 $\pm$ 0.12	0.55 $\pm$ 0.0002	<b>3296 <math>\pm</math> 23.49</b>	0.62 $\pm$ 0.04
0.60	3244 $\pm$ 19.09	0.09 $\pm$ 0.09	3443 $\pm$ 4.053	0.54 $\pm$ 0.02	<b>3358 <math>\pm</math> 30.9</b>	0.69 $\pm$ 0.02
0.70	3405 $\pm$ 21.64	0.64 $\pm$ 0.02	3535 $\pm$ 11.91	0.69 $\pm$ 0.01	<b>3685 <math>\pm</math> 175.96</b>	0.73 $\pm$ 0.02
0.75	3576 $\pm$ 75.41	0.68 $\pm$ 0.06	3669 $\pm$ 59.32	0.7 $\pm$ 0.03	<b>3914 <math>\pm</math> 127.33</b>	0.76 $\pm$ 0.01

Table 2: Objective (Obj – food basket cost, *lower is better*) and Ground Truth Constraint Values (GR – *should be above the Palatability Threshold*) with 95% Confidence Interval for the Three Algorithms. (GR constraint-satisfying solutions in **Green**, the constraint-violating solutions in **Red**, best constraint-satisfying objective value is in **boldface**.)

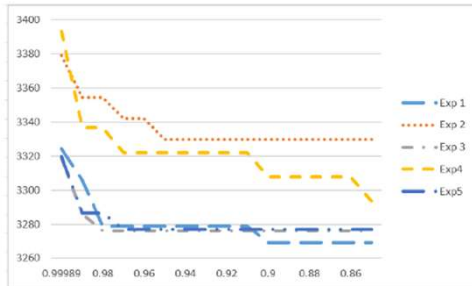


Figure 4: Objective PoCS tradeoff for Threshold 0.5

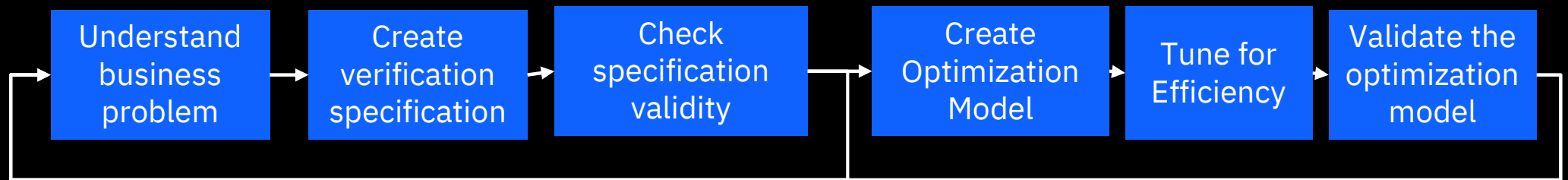
	False Positive	False Negative
PoCS $\geq 0.8$	1.09%	9.78%
PoCS $\geq 0.5$	4.37%	3.93%

Table 3: Accuracy of PoCs Estimate

# Validation First Simplified Optimization Modeling

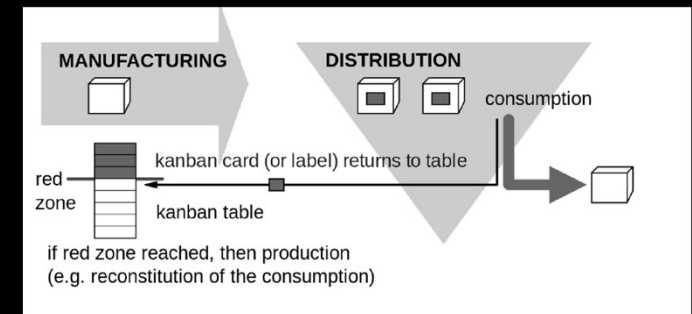
Joint work with Yishai Feldman and Aviad Sela

# Validation first process



# Supply chain inventory replenishment

- Given the demand forecast, user wants to find the optimal  $(s,S)$  order policy
- Objective: Minimize lost sales and inventory holding costs
- Constraints: Quantity conservation



# User has relevant data in spreadsheet

1	Date	Demand
2	01/02/2020	500
3	02/02/2020	300
4	03/02/2020	200
5	04/02/2020	600
6	05/02/2020	700
7	06/02/2020	100
8	07/02/2020	200
9	08/02/2020	100
10	09/02/2020	300
11	10/02/2020	400
12	11/02/2020	500
13	12/02/2020	100
14	13/02/2020	50



# User adds cells for policy, initial inventory and costs

s	350
S	450

Inventory (State)
1000

18	Inventory cost (per item)	1	Stockout Cost (per item)	5
----	---------------------------	---	--------------------------	---

## User adds formulas for calculating objective and intermediate values

- Stockouts:  $=\text{MAX}(0, B3 - C3)$
- Remaining inventory:  $=\text{MAX}(C3 - B3, 0)$
- Cost per day:  $=G3 * \$B\$18 + \$D\$18 * F3$
- Order amount based on current (s,S) values:  $=\text{IF}(C3 < \$B\$19, \$B\$20 - C3, 0)$

# User can calculate costs with different values, validating solution

Inventory cost (per item)	1 Stockout Cost (per item)		5				
Date	Demand (State)	Inventory (State)	Restocking (Action)	Cost (Reward)	Stockouts	Excess Inventory	Order amount
01/02/2020	500	1000	0	500	0	500	0
02/02/2020	300	500	0	200	0	200	0
03/02/2020	200	200	250	0	0	0	250
04/02/2020	600	250	200	1750	350	0	200
05/02/2020	700	200	250	2500	500	0	250
06/02/2020	100	250	200	150	0	150	200
07/02/2020	200	350	0	150	0	150	0
08/02/2020	100	150	300	50	0	50	300
09/02/2020	300	350	0	50	0	50	0
10/02/2020	400	50	400	1750	350	0	400
11/02/2020	500	400	0	500	100	0	0
12/02/2020	100	0	450	500	100	0	450
13/02/2020	50	450	0	400	0	400	0
				8500			
Inventory cost (per item)	1 Stockout Cost (per item)		5				
\$	350						
\$	450						

Inventory cost (per item)	1 Stockout Cost (per item)		5				
Date	Demand (State)	Inventory (State)	Restocking (Action)	Cost (Reward)	Stockouts	Excess Inventory	Order amount
01/02/2020	500	1000	0	500	0	500	0
02/02/2020	300	500	0	200	0	200	0
03/02/2020	200	200	0	0	0	0	0
04/02/2020	600	0	1500	3000	600	0	1500
05/02/2020	700	1500	0	800	0	800	0
06/02/2020	100	800	0	700	0	700	0
07/02/2020	200	700	0	500	0	500	0
08/02/2020	100	500	0	400	0	400	0
09/02/2020	300	400	0	100	0	100	0
10/02/2020	400	100	1400	1500	300	0	1400
11/02/2020	500	1400	0	900	0	900	0
12/02/2020	100	900	0	800	0	800	0
13/02/2020	50	800	0	750	0	750	0
				10150			
Inventory cost (per item)	1 Stockout Cost (per item)		5				
\$	200						
\$	1500						

# Spreadsheets formulas automatically translated into optimization model

```
minimize E16;
```

```
subject to {
```

```
  E16 == E3 + E4 + E5 + E6 + E7 + E8 + E9 + E10 + E11 + E12 + E13 + E14 + E15;
```

```
  E3 == G3 * B18 + D18 * F3;
```

```
  E4 == G4 * B18 + D18 * F4;
```

```
...
```

```
  F3 == max1(0, B3 - C3);
```

```
  G3 == max1(C3 - B3, 0);
```

```
  F4 == max1(0, B4 - C4);
```

```
  G4 == max1(C4 - B4, 0);
```

```
...
```

```
  C4 == G3 + D3;
```

```
  C5 == G4 + D4;
```

```
...
```

```
  (C3 <= B19 - 1 => H3 == B20 - C3) && (C3 >= B19 => H3 == 0);
```

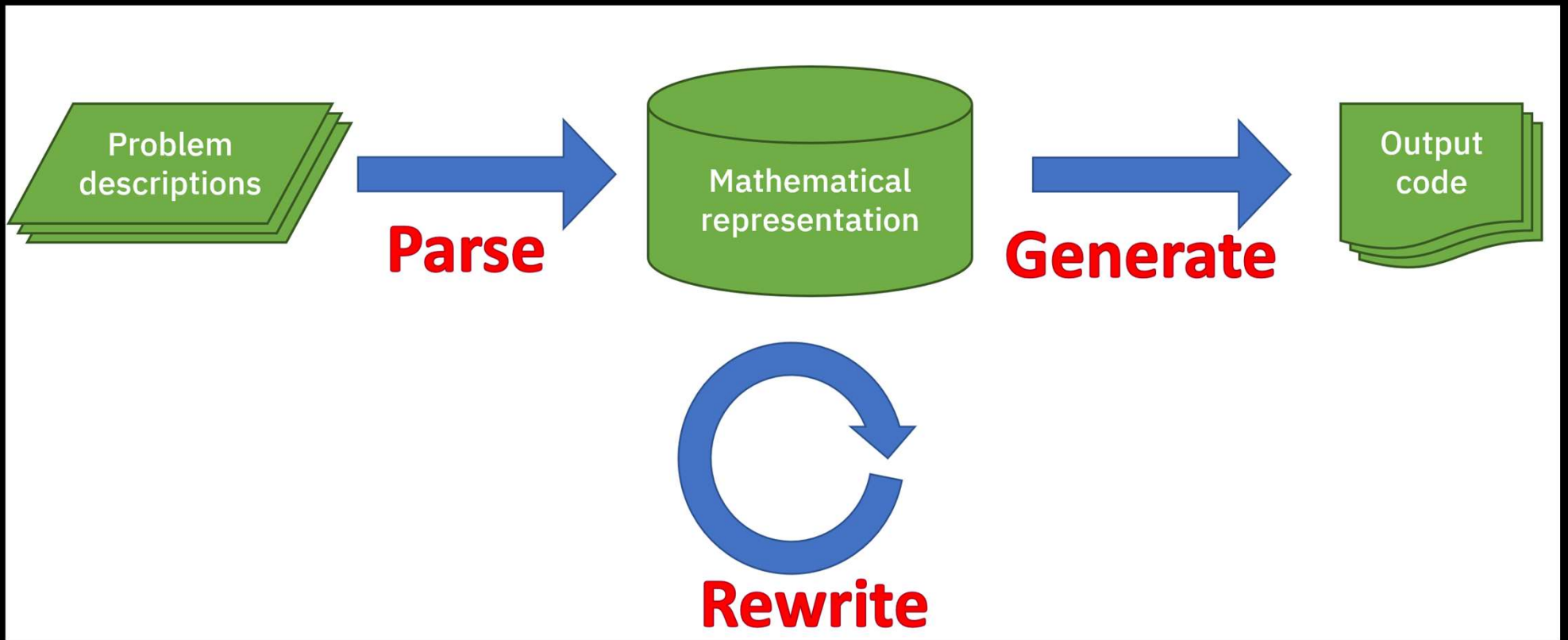
```
  (C4 <= B19 - 1 => H4 == B20 - C4) && (C4 >= B19 => H4 == 0);
```

```
...
```

```
}
```

**=IF (C3<\$B\$19, \$B\$20-C3, 0)**

# Solution Approach



# Mathematical representation

- High expressive power
  - Similar to first-order logic, or a functional programming language
- Powerful and extensible reasoning mechanisms
  - Rewrite rules
    - Implement transformations for specific targets
  - Constraint propagation
    - Discover information that is implicit in the specification
      - Precise data types; decision variables; variable domains
- Multiple types of inputs: Functional Python, spreadsheet, ...
- Multiple types of outputs can be generated: OPL, docplex, OptaPlanner, ...

## Rewriting rules example: If-then-else operator

$$x = \text{if } a \geq b \text{ then } c \text{ else } d$$

OPL doesn't allow  $?$ : operator on decision variables



Semantics of if-then-else operator

$$a \geq b \supset x = c \quad \wedge \quad a < b \supset x = d$$

OPL doesn't allow strict comparisons on decision variables



$a, b$ : integer

$$a \leq b - 1 \supset x = d$$

# Summary

- AI can significantly simplify the creation of an optimization model, but raises new issues in model validation and efficient model solving
- Presented works on how to validate learnt constraints, and how a validation first approach can significantly simplify the end-to-end model creation



# Future work

- Validation for constraint learning: More efficient model augmentation
- Validation first approach: extending mathematical representation and rewrite rules to cover larger set of optimization problems
- Extending learning to optimize capabilities to extend efficient model solving capabilities for meta-heuristic optimizers
- Utilizing large language models as part of an end-to-end optimization model generation pipeline
- Utilization of transformer architecture and attention mechanism as part of decision-making pipelines

# Backup Slides

# Theoretical Guarantees

Under the following assumptions

- $x^* \in \text{Conv}(X(p))$  (where  $X(p) \equiv \{x_i: \langle x_i, p_i, z_i \rangle \in s(p)\}$ )
- $\forall \langle x_i, p_i, z_i \rangle \in S(p)$  such that  $x_i \in X(p)$ ,  $|z_i - \hat{f}(x_i, p_i)| \leq \epsilon_f$   
(Enforced by model sufficiency for  $\sigma = \max$ )
- $f(x, \cdot)$  is  $L_f^P$  lipschitz and  $\hat{f}(x, \cdot)$  is  $L_{\hat{f}}^P$  Lipschitz for all  $x \in \mathcal{X}$
- $f(\cdot, p)$  is  $L_f^X$  lipschitz and  $\hat{f}(\cdot, p)$  is  $L_{\hat{f}}^X$  Lipschitz

# Theoretical Guarantee

Let:

$$x'^* = \arg \min_{(x \in \mathcal{X} \cap \Omega(p) \cap \text{Conv}(X(p))) \cup (MF-C')} f(x, p)$$

Then:

$$f(x'^*, p) - f(x^*, p) \leq 2 \left( \delta_p \left( L_f^P + L_{\hat{f}}^P \right) + \epsilon_f \left( L_f^P + L_{\hat{f}}^P \right) \Delta(p) \right)$$

$$\text{Where: } \Delta(p) = \min_{x_i \in X(p)} \max_{x_j \in X(p)} \|x_i - x_j\|_2$$

Bound could be extremely loose and may not be useful in practice

- $\Delta(p)$  could be very large (depending both on  $\delta_p$  and  $|X(p)|$ )
- The smaller are  $\delta_p$  and  $|X(p)|$ , the less likely it is that

If we can calculate probability that  $x^* \in \text{Conv}(X(p))$ , we can calculate

$$\Pr(f(x'^*, p) - f(x^*, p) \leq 2 \left( \delta_p \left( L_f^P + L_{\hat{f}}^P \right) + \epsilon_f \left( L_f^P + L_{\hat{f}}^P \right) \Delta(p) \right))$$

# Data sufficiency

- Input  $p$  should be “close enough” to historical uncontrollables:

$$S(p) \equiv \{\langle x_i, p_i, x_i \rangle \in D: d^p(p, p_i) \leq \delta_p\}$$

for some distance function  $d^p$

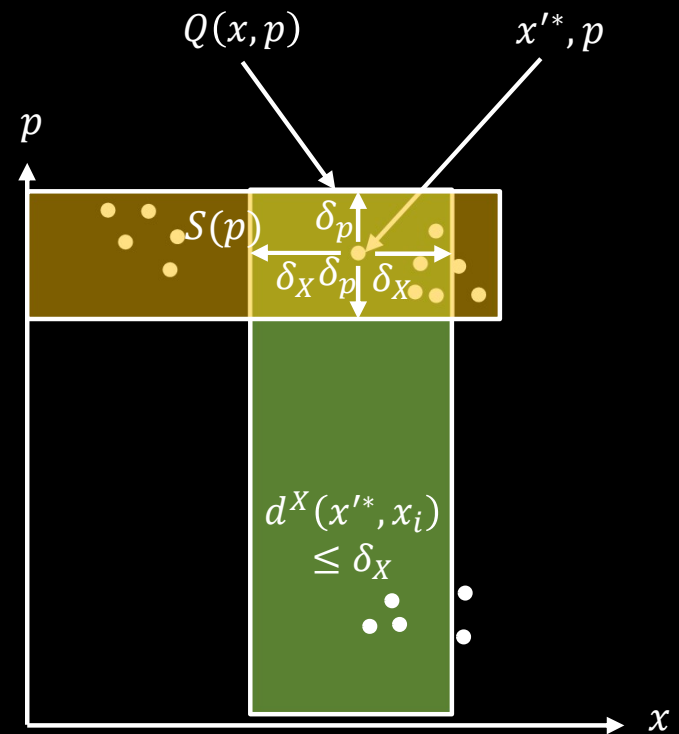
- Optimal decision should be “close enough” both to historical decisions and historical uncontrollable

$$Q(x, p) \equiv \{\langle x_i, p_i, z_i \rangle \in S(p): d^X(x', x_i) \leq \delta_X\}$$

for some distance function  $d^X$

- Add constraint ensuring that optimal decision and uncontrollable “sufficiently close” to enough

$$\text{points in the data } \frac{|Q(x, p)|}{|D|} \geq \epsilon_S$$



# Model Fidelity

- Learnt objective needs to be accurate enough in region where input uncontrollable and optimal decision are

$$\sigma(\{|z_i - \hat{f}(x_i, p_i)|: \langle x_i, p_i, z_i \rangle \in Q(x'^*, p)\}) \leq \epsilon_M^f$$

where  $\sigma$  is some statistic (max, average, etc.)

- Better to test on the holdout dataset used to train  $\hat{f}$
- Different accuracy measures could be used

# Augmented optimization formulation

$$x'^* = \arg \min_{x \in \mathcal{X} \cap \Omega(p)} f'(x, p)$$

subject to:

$$\frac{|Q(x, p)|}{|D|} \geq \epsilon_S \quad (DS - C)$$

$$\sigma(\{|z_i - \hat{f}(x_i, p_i)| : \langle x_i, p_i, z_i \rangle \in Q(x'^*, p)\}) \leq \epsilon_M^f \quad (MF - C)$$

- Parameterized by  $\delta_p, \delta_X, \epsilon_S, \epsilon_f^M$   
Can be used to enhance trust: user can ensure that solution is close to previously existing solutions and uncontrollables
- Can be formulated as Mixed Integer Linear Program (MILP) when original problem is MILP
- Theoretical guarantee: In some settings, can provide a bound for

$$\Pr(f(x'^*, p) - f(x^*, p) \leq \epsilon) \geq (1 - \delta_2)$$