AMPL Optimization | IFORS 2026

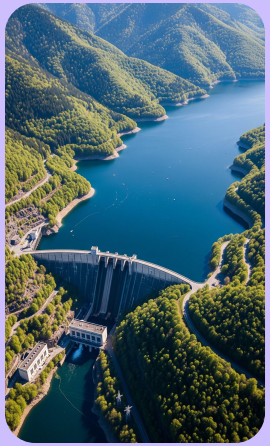
Getting Optimization into Production with AMPL

From Prototype Models to Operational Decision Systems

[Github Repo]

Juan Bohorquez
Technical Account Manager

Marcos Dominguez
Senior Software Engineer






Agenda

1 Introduction

Integrating optimization in a Decision Support System: challenges and opportunities

2 Workshop

- Energy problem statement
- Simple formulation *-[explainability]*
- Refine formulation *-[adaptability]*

Real World Challenges: Rich non-linear syntax, automatic reformulation, solver speed, & deployment

3 Closing notes

Concluding remarks, open Q&A, and session wrap-up.

Overview

What is AMPL?

AMPL is a system modeling and solving large, complex business decisions.

A modern, comprehensive environment designed to help teams formulate, test, and deploy mathematical models at scale.

1

Expressive Modeling Language

A clean, expressive syntax tailored specifically to mathematical optimization problems.

2

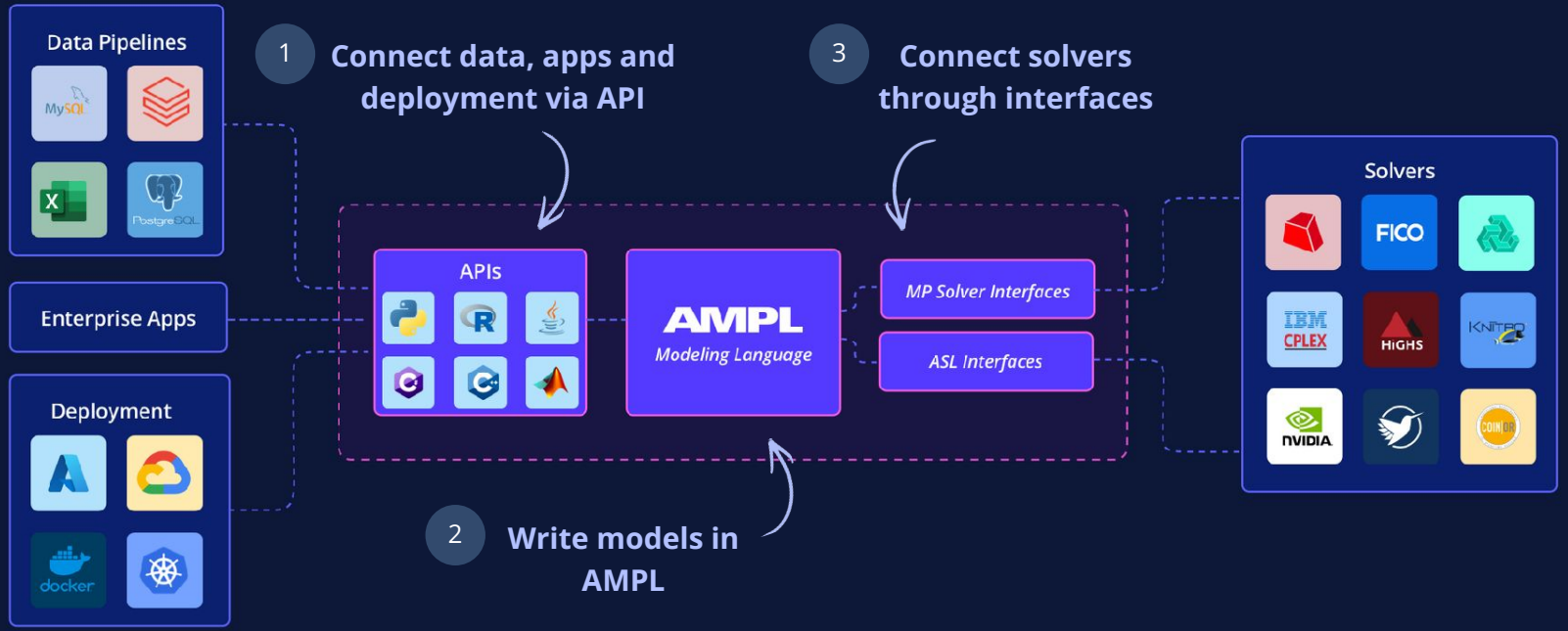
In-House Solver Interfaces

Connect directly to solvers including Gurobi, CPLEX, Xpress, COPT, HiGHS, Knitro, Mosek...

3

Easy Lifecycle Integration

Quick to learn, build, iterate, and seamlessly connect or collaborate with your team.


Data Pipelines
Enterprise Apps
Deployment
1
Connect data, apps and deployment via API
3
Connect solvers through interfaces
APIs
AMPL
Modeling Language
MP Solver Interfaces
ASL Interfaces
Solvers
2
Write models in AMPL

AMPL

Who uses AMPL?

Organizations with mission-critical, large-scale operational environments that demand mathematical precision in production.

01

Large-Scale Optimization

Organizations solving large, complex operational optimization problems.

02

High-Speed Decisions

Environments where critical operational decisions must be fast, reliable, and repeatable.

03

Production Systems

Complex constraints and large models running consistently in enterprise production.

Airline Scheduling

Assign crews based on seniority, preferences, and regulations while minimizing disruption and cost in time-sensitive scheduling and recovery systems.

Crew Constraints
Seniority, individual preferences, and industry regulations.

Disruption & Cost
Minimizing overall operational disruption and recovery costs.

Time-Sensitive Recovery
Enabling real-time, highly efficient scheduling adjustments.

Electricity Markets

New York ISO
Independent System Operator

Optimize power generation and bidding strategies across multiple assets while satisfying operational constraints and market rules.

Asset Optimization
Maximizing generation efficiency and market bidding across units.

Operational Constraints
Satisfying physical limitations, transmission limits, and market rules.

Commodities Trading

Optimize trading and portfolio positions across commodities while managing risk, exposure limits, and market constraints under tight time windows.

Risk & Exposure Limits
Active risk mitigation and constraint enforcement across portfolios.

Tight Time Windows
Executing operational decisions in seconds to capture market value.

Problem Statement

Context CLICK HERE

In liberalized electricity markets, large generation companies must operate a **diverse fleet** to maximize profit while respecting strict physical and environmental constraints.

Managing a portfolio of hydro, solar, wind, thermal, and batteries in a medium term horizon is a non-trivial task.

Power Generation Portfolio Optimization CLICK HERE

The core objective is to maximize **Producer Surplus**:

Spot Market Revenue - Variable Costs

The Project: Create a solution for medium-term decisions including plant generation levels, reservoir management, and physical constraint handling.

Generation Portfolio and Data

Generation Portfolio

- **3 Reservoir Hydro:** Two are chained (cascaded river use)
- **8 Run-of-river** plants
- **4 Thermal** generators
- **Up to 4 Solar** plants (scenario-dependent)

Multiple Scenarios (5)

Uncertainty Sources:

Market price, hydro inflows, and renewable availability drawn from historical records.

1-year horizon with hourly resolution.

Model Scale: Each scenario involves at least ~480,000 variables and ~650,000 constraints.

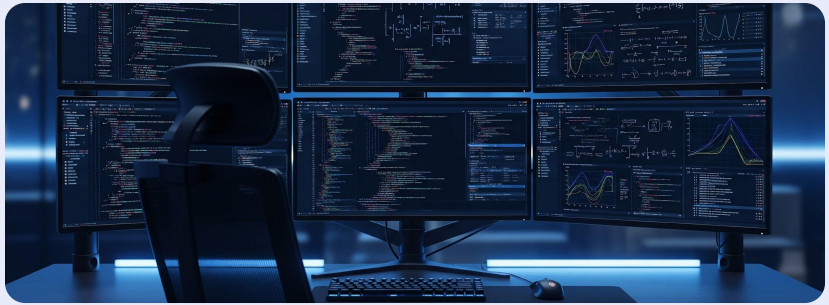
GROUNDED IN REALITY

Portfolio Topology: Generation assets and hydraulic relationships (upstream/downstream chains) are based on a real-world energy portfolio from an actual power system.

Step 1. Moving Beyond Hardcoded Models

INITIAL MODEL STRUCTURE

```
set PLANTS := {Solar1, Thermal1, Res1};
param hours := 24;
# Data hardcoded in .ampl file
... (1 reservoir, 1 solar, 1 thermal) ...
```



The Challenge

- Difficulty incorporating full portfolio
- Inflexible for new scenarios (BESS/New plants)
- Manual modification required for evaluations

THE SOLUTION

Decouple Model from Data

Implement robust, indexed formulation of sets to allow for seamless portfolio scaling and scenario testing.

Step 1. Moving Beyond Hardcoded Models

```
# 🚩 No sections. Everything is mixed together.

var g_t {1..24} >= 0; # 🚩 Terrible, cryptic naming
param c {1..24};
var g_s {1..24} >= 0;
param r_s {1..24};

# 🚩 Hardcoded scalars mixed into declarations
param eff = 0.85;

var v {0..24} >= 0; # 🚩 Hardcoded time indices (1..24)
var q {1..24} >= 0;
var sp {1..24} >= 0;
param p {1..24};

maximize z: # 🚩 Meaningless objective name
sum{t in 1..24} (p[t] * (g_t[t] + g_s[t] + (eff*q[t])))
- sum{t in 1..24} (c[t]*g_t[t])
- sum{t in 1..24} 250*sp[t]); # 🚩 Hardcoded penalty inside objective

subject to c1 {t in 1..24}: g_t[t] <= 100;# 🚩 Unhelpful constraint names
subject to c2 {t in 1..24}: g_s[t] <= r_s[t];

subject to c3 {t in 1..24}:
# 🚩 Clunky index math (t-1) and hardcoded unit conversions
v[t] = v[t-1] + 0.0036 * (15 - q[t] - sp[t]);

# 🚩 Initial/Final conditions hardcoded as brittle constraints
subject to c4: v[0] = 0.7 * 500;
subject to c5: v[24] = 350;

# 🚩 Clunky subset bounding for ramps
subject to c6 {t in 2..24}: g_t[t] - g_t[t-1] <= 40;
```

Step 1. Moving Beyond Hardcoded Models

```
# ✅ Clear sections separating structure, data, and logic.

### SETS
# ✅ Ordered sets replace brittle, hardcoded indices like {1..24}.
set PERIODS ordered;

### PARAMETERS
# ✅ Verbose, self-documenting naming conventions.
param thermal_cost {PERIODS}; param solar_resource {PERIODS}; param inflows {PERIODS}; param market_price {PERIODS};

# ✅ Scalars parameterized with safe cascading defaults. No hardcoded values.
param hydro_efficiency > 0 default 0.85; param delta_t default 0.0036; param penalty_spillage >= 0 default 250;
param max_thermal_cap >= 0 default 100; param ramp_limit >= 0 default 40;

### OBJECTIVE
# ✅ Meaningful objective name and parameterized penalties.
maximize Total_Profit: sum{t in PERIODS} (market_price[t] * (Thermal_Gen[t] + Solar_Gen[t] + (hydro_efficiency * TurbiningFlow[t])))
- sum{t in PERIODS} (thermal_cost[t] * Thermal_Gen[t]) - sum{t in PERIODS} (penalty_spillage * SpillageFlow[t]);

### CONSTRAINTS
# ✅ Special ordered set functions (first, prev) eliminate clunky t-1 math.
subject to Water_Balance {t in PERIODS}: ReservoirStorage[t] = (if t = first(PERIODS) then initial_storage else
ReservoirStorage[prev(t)])
+ delta_t * (inflows[t] - TurbiningFlow[t] - SpillageFlow[t]);

# ✅ Final condition uses last() function, robust to horizon changes.
subject to Final_Storage_Target: ReservoirStorage[last(PERIODS)] = target_storage;

# ✅ Clean subset bounding using ord() to safely manage ramps.
subject to Thermal_Ramp_Up {t in PERIODS: ord(t) > 1}: Thermal_Gen[t] - Thermal_Gen[prev(t)] <= ramp_limit;
```

Complete initial model.

Step 2: Navigating Real-World Complexity

The Challenges

- **Technology Gaps:** Need to evaluate BESS and complex thermal constraints (ramps, minimum online time, etc).
- **Modeling Limits:** Simple constant dam efficiency relationships are insufficient for realistic production factors.
- **Uncertainty:** High variability in real-world operational environments.

The AMPL Logic Solutions

- **Flexible Syntax:** Easy BESS logic inclusion via native AMPL logic syntax, and incremental indexing for scenario management.
- **Handling Non-Linearity:** Use of PWL or estimated polynomial relationships.
- **Binary Decisions:** Adding binaries for UC-like logic and operation rules.

Advanced Capabilities: Leverage AI chatbots for PWL modeling and MP reformulations (e.g., Big-M) to simplify complex binary constraints.

Optional: Explore model reformulation to see what is sent to solver.

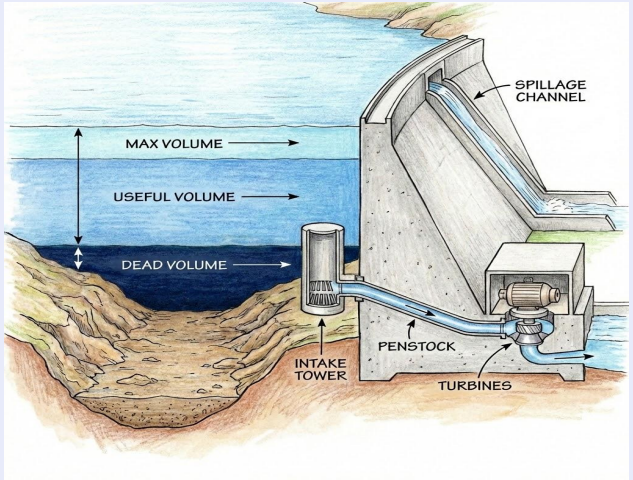
Reservoir Efficiency

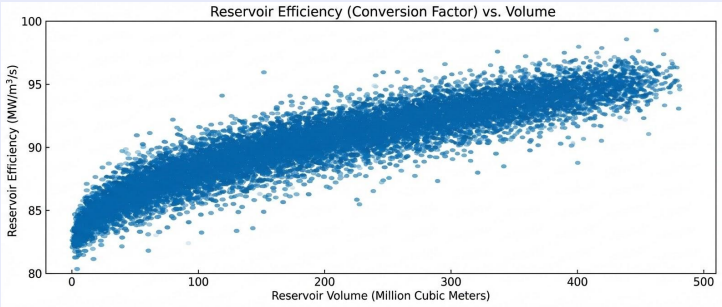
Reservoir Efficiency Dynamics

Efficiency (conversion factor) is dependent on the reservoir volume. This relationship is unique to each dam's design.

- **High Water Jump:** Some designs prioritize elevation head.
- **High Flow:** Others are optimized for large volume river flows.
- **Floor Shape:** Geometry affects bathymetry curves.

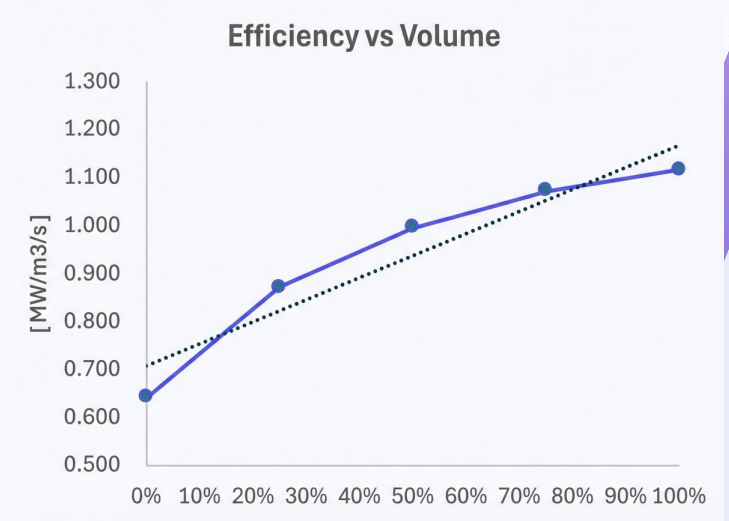
General Intuition: *Increased volume typically leads to higher energy production efficiency.*





Step 2.a: Efficiency (Conversion Factor)

Key Insight: Stakeholders know efficiency is very affected by volume in one of the 3 reservoirs.



AMPL Code Modification

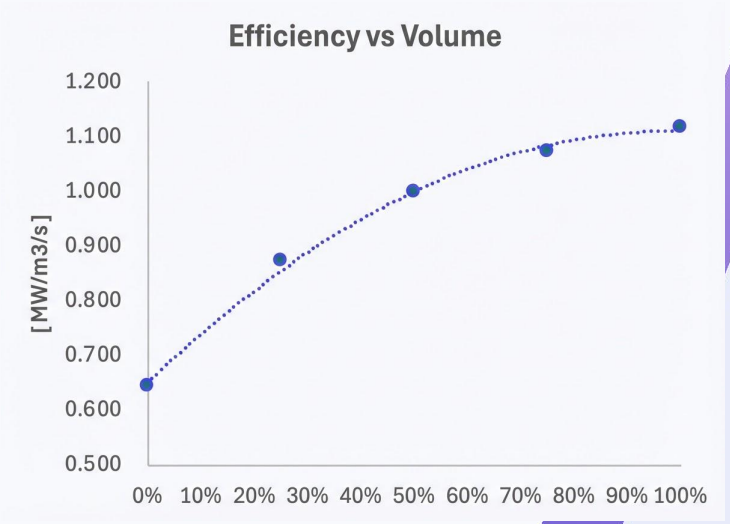
```
# Simple PWL definition in AMPL

param npiece {PWL_GEN} integer >= 1;
param breakpoints {g in PWL_GEN, p in 1..npiece[g]};
param slopes {g in PWL_GEN, p in 1..npiece[g]+1};
param intercept {PWL_GEN};

# Piecewise Linear Efficiency
var Eff_PWL {g in PWL_GEN, t in PERIODS} =
   (<<{p in 1..npiece[g]} breakpoints[g,p];
      {p in 1..npiece[g]+1} slopes[g,p]>>
      Reservoir[g,t]) + intercept[g];
```

Implementation Strategy: Transition from constant factors to PWL relationships effortlessly with AMPL's native syntax.

Complete model comparison.

Step 2.a: Efficiency (Conversion Factor)

Key Question: But, can we do better? Exploring non-linear formulations for higher precision.



AMPL Code Modification

```
# Simple QUAD coefficients definition in AMPL
param quad_coeff {g in QUAD_GEN, p in 1..3};

# Quadratic Relationship

var Eff_QUAD {g in QUAD_GEN, t in PERIODS} =
      quad_coeff[g,1]*(Reservoir[g,t])^2 +
      quad_coeff[g,2]*Reservoir[g,t] +
      quad_coeff[g,3];
```

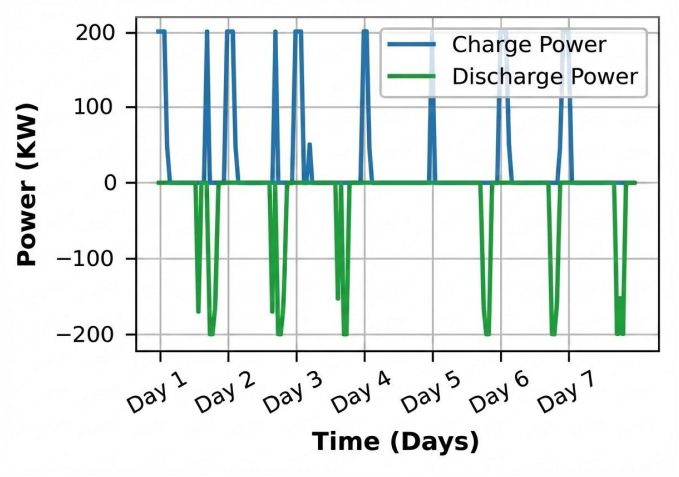
Performance Advantage: Non-linear formulations can solve **faster than PWL** using modern local solvers like Gurobi's latest `optimalitytarget=1` or dedicated NLP solvers like Knitro.

Complete model comparison.

Step 2.b: Include BESS in portfolio

Mathematical Shift: From non-convex NLP to non-convex MINLP (MIQCP).

BESS Charge and Discharge Cycles



AMPL Code Modification

```
# Variables for BESS inclusion
var Charge {BESS, PERIODS} >= 0;      # MW
var Discharge {BESS, PERIODS} >= 0;   # MW
var SoC {g in BESS, t in PERIODS} >= 0 <= bess_capacity[g];
var IsCharging {BESS, PERIODS} binary;
var IsDischarging {BESS, PERIODS} binary;

# New constraints
subject to ChargingStateExclusion {g in BESS, t in PERIODS}:
   IsCharging[g, t] + IsDischarging[g, t] <= 1;

subject to SoC_Balance {g in BESS, t in PERIODS}:
   SoC[g, t] = SoC[g, prev(t)]
               + (Charge[g, t] * (bess_efficiency[g]^0.5))
               - (Discharge[g, t]/(bess_efficiency[g]^0.5));
```

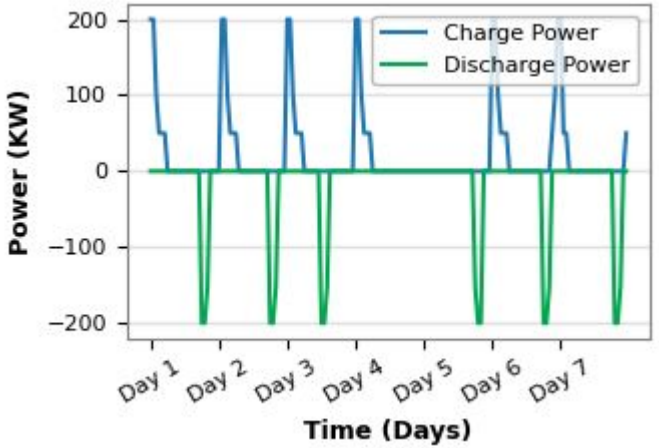
Solver Note: Local solvers will no longer work due to discrete variables. Global solvers are required to navigate the non-convex MINLP space.

Complete model comparison.

Step 2.b: Include BESS in portfolio

Strategic Goal: Adding custom operation rules to preserve battery lifetime.

BESS Charge and Discharge Cycles



AMPL Code Modification (Battery Rules)

```
# Preserve battery lifetime
subject to MinChargingPeriods {g in BESS, t in PERIODS}:
   (IsCharging[g,t]==1 and IsCharging[g, prev(t)]==0)
==> forall{tp in t..t+min_periods[g]} IsCharging[g,tp]==1;


# Avoid too small charges
subject to ChargeMaxLimit {g in BESS, t in PERIODS}:
   Charge[g, t] <= max_bess_power[g] * IsCharging[g, t];

subject to ChargeMinLimit {g in BESS, t in PERIODS}:
   Charge[g, t] >= min_charging_power[g] * IsCharging[g, t];
```

Optimization Impact: Operational constraints prevent constant switching and micro-charging, significantly extending the BESS lifecycle.

Complete model comparison.

AMPL/MP Solver Library: high-level expressions supported

Combinatorial and logical

- Conditional operators: `if-then-else; ==> [else], <== [else], <==>`
- Logical operators: `or, and, not; exists, forall`
- Piecewise-linear functions: `abs; min, max; <<breakpoints; slopes>>`
- Counting operators: `count; atmost, atleast, exactly; numberof`
- Relational and comparison operators: `>(=), <(=), (!)=; alldiff`
- Complementarity operator: `complements`
- Set membership: `in`

Nonlinear

- Nonlinear operators and functions: `*, /, ^; exp, log;` trigonometric, hyperbolic
- High-order polynomials
- Conic algebra

Much more than expressions...

A basic reformulation case: indicator

```
var b binary;
   s.t. Indicator: b==1 ==> 5*x[1] + 2*x[2] <= 0;
```

- Important: tight bounds on the compared expression, e.g.:

```
   var {1..2} x >= 0 <= 15;
```

- Linearized when not supported, or when desired:

```
   s.t. BigM: 5*x[1] + 2*x[2] <= 105*(1-b);
```

Directly supported by many solvers

Paintshop color change scheduling problem

Different solvers might prefer own ways to receive these constraints!

```
s.t. ColorMAXSpec {s in SLOTS,
                   p in SKID_TYPE_PATTERNS[slot_skid_type[s]]:
                   color_spec_MAX[p] in COLORS_MAX}:
    (slot_pattern[s] == p) ==>                                  # Pattern allocated, then …
        (exists {cm in COLOR_MAX_MACHINES}
            (slot_color_spec_MAX[s, cm] == color_spec_MAX[p]
                && slot_color_MAX_machine_available[s, cm]));
```

Paintshop color change scheduling problem

Different solvers might prefer own ways to receive these constraints!

```
s.t. ColorMAXSpec {s in SLOTS,
                   p in SKID_TYPE_PATTERNS[slot_skid_type[s]]:
                   color_spec_MAX[p] in COLORS_MAX}:
    (slot_pattern[s] == p) ==>                                  # Pattern allocated, then …
        (exists {cm in COLOR_MAX_MACHINES}
            (slot_color_spec_MAX[s, cm] == color_spec_MAX[p]
                && slot_color_MAX_machine_available[s, cm]));
```

Paintshop color change scheduling problem

Different solvers might prefer own ways to receive these constraints!

```
s.t. ColorMAXSpec {s in SLOTS,
                   p in SKID_TYPE_PATTERNS[slot_skid_type[s]]:
                   color_spec_MAX[p] in COLORS_MAX}:
    (slot_pattern[s] == p) ==>                                  # Pattern allocated, then …
        (exists {cm in COLOR_MAX_MACHINES}
            (slot_color_spec_MAX[s, cm] == color_spec_MAX[p]
                && slot_color_MAX_machine_available[s, cm]));
```

Paintshop color change scheduling problem

Different solvers might prefer own ways to receive these constraints!

```
s.t. ColorMAXSpec {s in SLOTS,
                   p in SKID_TYPE_PATTERNS[slot_skid_type[s]]:
                   color_spec_MAX[p] in COLORS_MAX}:
    (slot_pattern[s] == p) ==>                                  # Pattern allocated, then …
        (exists {cm in COLOR_MAX_MACHINES}
            (slot_color_spec_MAX[s, cm] == color_spec_MAX[p]
                && slot_color_MAX_machine_available[s, cm]));
```

High-level vs Linearization experiment

A paintshop color change scheduling problem with 5 pattern types, 4 skid types, and 50 slots, tested with two major MIP solvers.

	Solver 1	Solver 2
High-level (default)	3.87 s	4.22 s
Full linearization	3.17 s	6.70 s

AMPL MP uses a solver's high-level modeling by default.

Performance difference can increase on larger instances.

https://colab.ampl.com/tags/color-change-scheduling.html

Step 3: Managing Project Restrictions

Challenges & Barriers

- **Computation Time:** Strict limits on model execution speed.
- **Optimality Gap:** Precise requirements for solution quality.
- **Budgetary Constraints:** Financial limits for the final solution.

The Strategy

- **Solves:** Evaluation and tuning of solvers.
- **Parameter Tuning:** Ensuring operational limits.
- **Ad-hoc Strategies:** “Close to the solver” API available to fine tune algorithms..
- **Solver Choice:** Cost/benefit multi-solver analysis. More than 20 solvers compatible.

Strict limit of 2 minutes per scenario and preference of solving all scenarios if possible

Strategic Solution: Implement rigorous **solver tuning** and evaluation to maintain high-quality results within fixed budgets.

Step 3.a: Solver Benchmarking

Strategy: Pick the right solver tool for your model type.

Advantage: **AMPL model remains unchanged** across different solvers.

AMPL Code: Switch Solvers & Options

```
options = {"solver": "gurobi",
"mp_options": "optimalitytarget=1"}
ampl.solve(**options)
-----------
ampl.option["solver"] = "highs"
ampl.option["mp_options"] = "outlev=1"
ampl.solve()
-----------
ampl.solve(solver="xpress",
mp_options="outlev=0 lim:time=300")
-----------
ampl.set_option("solver", "knitro")
ampl.solve()
```

Solver	Base-LP	Local NLP	NLP	BESS	BESS+rules	UC thermal
Com. Best*	00:20	00:35	03:39	03:39	04:53	**10:07***
Com. Worst*	00:34	**10:20***	**09:40***	**11:23***	**11:29***	**12:01***
Com. Avg*	00:27	**04:45***	**06:39***	**07:31***	**08:11***	**11:04***
SCIP	01:01	-	**10:23***	-	-	-
HiGHS	00:26	-	-	-	-	-

*** Grouped commercial solvers including: Gurobi, Cplex, Xpress and Knitro**
*** Hit time limit in one or more scenario.**
** Results may differ depending on hardware used.

Business Decision Trade-offs: Removing thermal UC constraints might be necessary to meet budget and time goals. **Stakeholders must take this decision.**

Step 3b. Solver Tuning

Options Exploration

```
options = {"solver": "gurobi", "mp_options":
"outlev=1 mipgap=0.01 mipfocus=1 obbt=1
timelimit=120"}
ampl.solve(**options)
```

Final Comp. Time: 02:30 - achieved by rescaling the model

Access Auto-Tuner trough AMPL

```
options = {"solver": "gurobi", "mp_options":
"outlev=1 mipgap=0.01 timelimit=120
tunebase=tuning_results tunetimelim=14400
logfile=tuninglog.log"}
ampl.solve(**options)
```

Final Comp. Time: 03:37

Suggested Autotuner Parameters:

```
OBBT=1 VarBranch=3 Cut Passes=5
```

Performance Optimization: Besides tuning the selected solver, other alternatives to improve performance include **scaling** the model, **reformulating** when possible and **warm starting** the solution with special focus on binaries.

Step 3c. Ad-hoc solution strategies

Real life problems come in real life sizes!

- Information about the model structure can lead to heuristics or to decomposition methods
- Solution can be “good enough” and solution process could be stopped

Solver-level functionalities via ampls

```
import amplpy_gurobi as ampls
# Model building logic
model = ampl.to_ampls(SOLVER);
model.set_callback(cb)
model.optimize()
ampl.import_ampls_solution(model)
```

A callback can:

Receive information during the solution process and take actions (inject solutions, cut nodes, stop the solution process, ..)

Performance Optimization: A low-level control of the solver while retaining the convenience of AMPL’s syntax and solver agnosticity is possible

Step 3c. Ad-hoc solution strategies

```
class StoppingCallback(ampls.GenericCallback):
    def run(self):
        if t != ampls.Where.MIPSOL: return 0
        # Each time we see a new solultion, check its value;
        obj = self.get_obj()
        print(f"Objective value {obj}")
        if obj != self.last_obj:
            if self.last_obj_time is not None:
                print(f"Improvement detected")
            self.last_obj_time=time.time()
            self.last_obj = obj
        else:
            if time.time() - self.last_obj_time > MAX_TIME:
                print(f"No improvement, terminating.")
                return -1
        return 0
```

- Executed in many stages of the solution process
- Has the ability to stop the execution of the solver and return the current solution

Step 4: Integration and deployment

Challenges & Barriers

- **Multi-Source Integration:** input data comes from a variety of sources
- **Specific Formats:** flexible results format required for dashboarding and integration.
- **Cloud Scalabiltiy:** real life deployments often leverage containerized cloud environments.

The Integration Solution

- **Python Ecosystem:** full access to AMPL features via amplpy
- **Non-python Ecosystem:** most programming languages supported via API
- **Cloud Licensing:** flexible AMPL dynamic licensing for cloud.
- **Deployment Partners:** **NextMV** integrates AMPL to execute, scale and create decision apps based on AMPL

Strategic Integration: Leveraging the **Python ecosystem via amplpy** ensures seamless data pipeline integration, automated cloud scaling, and flexible API delivery for decision-making.

Step 4: Deployment leveraging APIs

- **Python packages for AMPL and solvers:** installation in any environment where python is available is as simple as:

```
python -m pip install amplpy                    # Install amplpy
python -m amplpy.modules install highs gurobi   # Install solvers
```

- **.NET core packages for AMPL and solvers:** installation via NuGet locally or on Azure via

  AMPL.Api

  AMPL.Module.highs.linux64, AMPL.Module.gurobi.linux64, …

- **Tutorial on azure functions:** https://ampl.com/api/latest/dotnet/deployment.html

Mainstream deployment platforms supported. Leveraging mainstream deployment techniques to avoid skills fragmentation

Step 5: Gaining Trust and Adoption

Challenge & Barriers

- **Trust Gap:** Decision makers may not trust or use the model.
- **Knowledge Loss:** Models can be lost in forgotten repositories.
- **Onboarding:** High learning curve for new team members.

The Solutions

- **Analysis:** Snapshot 'what-if' features and sensitivity analysis.
- **Explanations:** Clear infeasibility logic and feasrelax functionality.
- **Comparison:** Multiple solutions for expert evaluation.

Trust-Building Syntax: AMPL's clear syntax acts as self-documenting code, significantly accelerating the onboarding process for new members **from months to just days**.

amplbot: your AMPL expert

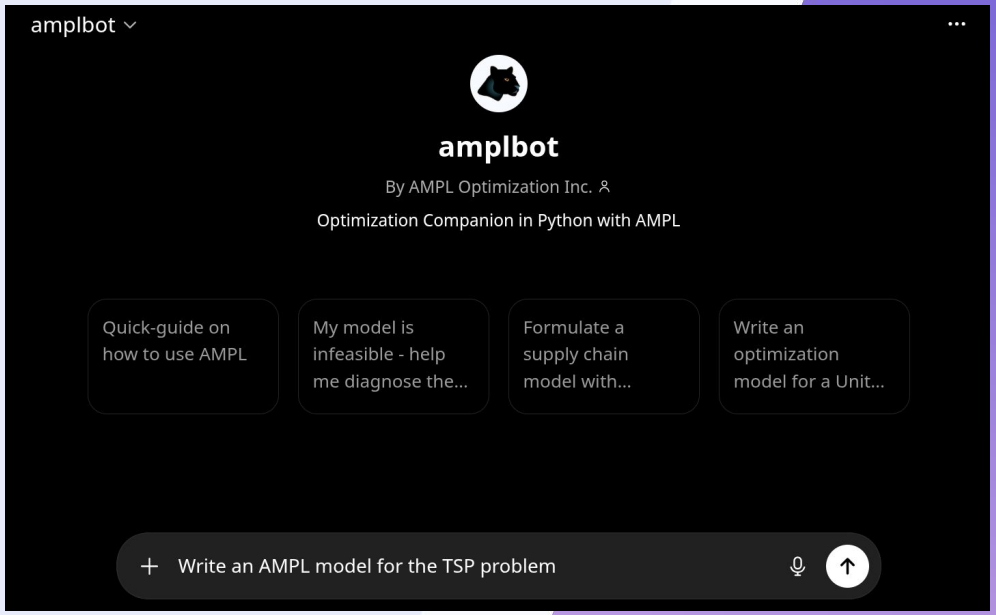
Custom AMPL assistant as LLM to enhance Optimization

→ Aware of latest documentation, features, and **amplpy**

→ Troubleshooting and Explainability

→ Instant onboarding

Ideal for beginners and veterans



Different amplbot flavors

Workshop Summary: Overcoming Challenges

Step / Challenge Faced	Lesson Learned	AMPL Feature
Hardcoded Models	Always formulate the model for scalability and production use cases.	**AMPL clear syntax**
Real World Complexity	Problems change quickly, dynamic requirements. Adapt fast	**MP formulations, logic functions, etc.**
Project Restrictions	Use the right tool, test your options, tune the solution, use multiple data.	**Solver benchmarking and tuning**
Gaining Trust and Adoption	Include stakeholders and decision makers from the start. Keep updated docs.	**Snapshot features, self-documented models**
Technology Integration Wall	Build for integration (API, cloud, latency, etc.)	**Easy deployment and integration**

Closing: From Models to Operational Systems

Transitioning to a system isn't just about math; it's about architecture, integration, and trust.

Final Message

AMPL is the infrastructure that allows optimization to live and breathe in production environments.

+ Operational support, ecosystem, and AI integration.

The AMPL Advantage

Bridging the gap between optimization challenges and enterprise-ready solutions.

Available Licenses

ampl.com/academia

AMPL for Academics (A4A)

Free AMPL license, available with our APIs, data connectors, and other key integrations.

All commercial and open source solvers included

AMPL for Courses (A4C)

AMPL and commercial solver licenses for free with shareable links and activation codes that enables collaborative use for coursework.

AMPL Career Starter

AMPL and commercial solver licenses for free use in industry applications for 1 year.

Available to recent graduates in last 2 years.

Gurobi IBM CPLEX FICO

Professional Services

Energy-Specific Services + other industries

In-house energy experts to develop end-to-end energy models.

Customized support plans.

Training and consulting to get the most out of your energy sources.

Consulting & Training

Model audits and enhancements.

Solver tuning and benchmarking.

On-site or virtual training sessions for onboarding or upskilling.

Model Development

In-house experts to build custom models, including full model translation (legacy systems, open source).

Thank You!

Do you have any questions?

Free for Academia

ampl.com/academia

academia@ampl.com