AMPL Optimization | INFORMS Annual Meeting 2025

From Classroom to Industry

Modern Optimization with AMPL for Energy, Finance, Supply Chain, and Beyond

Marcos Dominguez
Gleb Belov

Outline

1. Modern Optimization in 2025

   → Industry level experience in Academia

   → Writing Optimization Code + Amplbot

2. AMPL/MP: Complex models can be made easy

   → Automatic Reformulations & Enhanced solvers

What is AMPL?

Mathematical Optimization Engine

→ **Express problems as you think about them**

- Large-scale modeling
- Linear, Non-Linear, Constraint Programming…

→ **Solver interfaces**

- Open-source: HiGHS, Scip, CBC, ipopt,. . .
- Commercial: CPLEX, Gurobi, Xpress, Knitro, COPT . . .

→ **Amplpy** 

→ **Other APIs**

- C++, C#, Java, MATLAB, R

Models aren't just "min cx s.t. Ax=b"

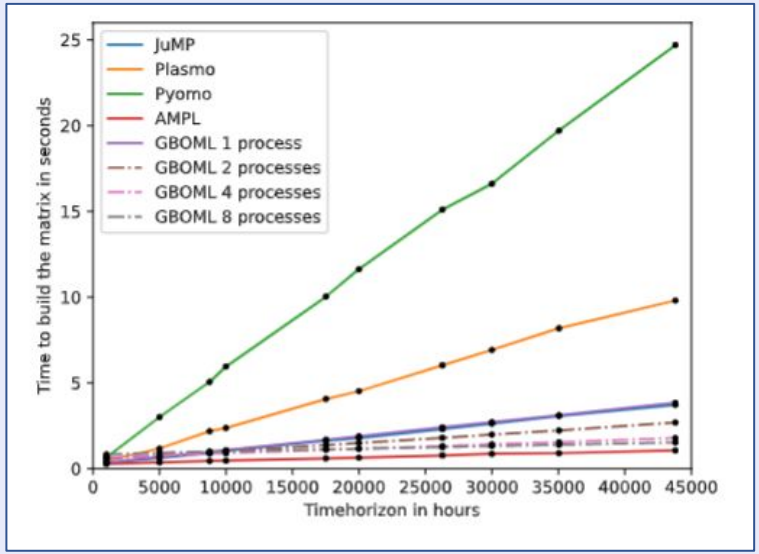
Example: If you are charging a thermal battery, you need to charge at a stable rate:

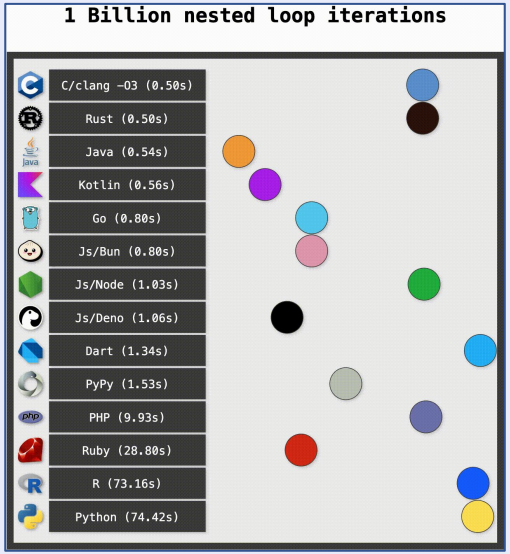
```
set Times ordered;
param MaxChargeRate;

var ChargeRate{t in Times} >= 0 <= MaxChargeRate;

subject to ChargeRateVariationLimit{t in Times: ord(t) >= 2}:
   ChargeRate[t] > 0 && ChargeRate[prev(t)] > 0
        ==>
   abs(ChargeRate[t]-ChargeRate[prev(t)]) <= 10;
```


Third-party benchmarks





Ref: GBOML: A Structure-Exploiting Optimization Modelling Language in Python
Bardhyl Miftari et al. (2023)

GBOML: a structure-exploiting optimization modelling language in Python

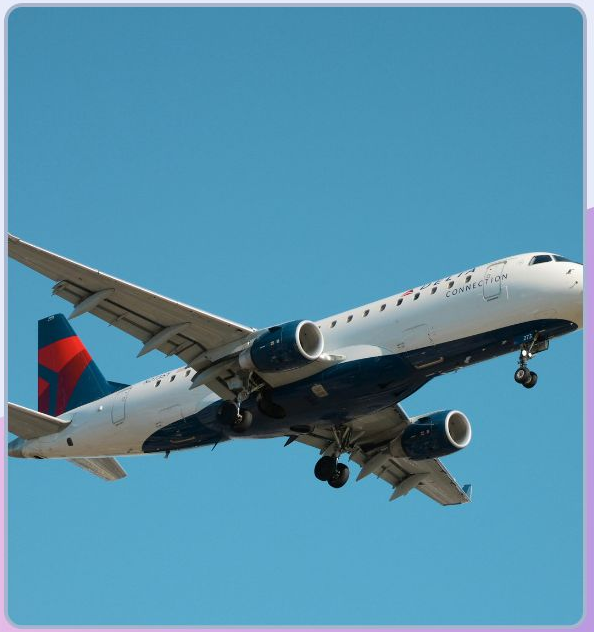
Bardhyl Miftari, Mathias Berger, Guillaume Derval, Quentin Louveaux & Damien Ernst

Pages 227-256 | Received 30 Nov 2022, Accepted 06 Aug 2023, Published online: 08 Sep 2023

Cite this article https://doi.org/10.1080/10556788.2023.2246169 Check for updates

Abstract

Mixed-Integer Linear Programs (MILPs) have many practical applications. Most modelling tools for MILPs fall in two broad categories. Indeed, tools such as algebraic modelling languages allow practitioners to compactly encode models using syntax close to mathematical notation but usually lack support for special structures, while other tools instead provide predefined components that can be easily assembled but modifying or adding new components is difficult. In this work, we present the inner workings of the Graph-Based Optimization Modelling Language (GBOML), an open-source modelling tool implemented in Python combining the strengths of both worlds. GBOML natively supports special structures that can be encoded by a hierarchical hypergraph, offers syntax close to mathematical notation and facilitates the modular construction and reuse of time-indexed models. We detail design choices enabling these features and show that they simplify problem encoding, lead to faster instance generation times and sometimes faster solve times. We benchmark the times taken by GBOML, JuMP, Plasmo, Pyomo and AMPL to generate instances of a structured MILP. We find that GBOML outperforms Plasmo and Pyomo, is tied with JuMP but is slower than AMPL. With parallel model generation, GBOML outperforms JuMP and closes the gap with AMPL. GBOML has the smallest memory footprint.


CONNECTION








Where is AMPL used most?

→ Large scale optimization applications where highly efficient optimization software is essentially indispensable due to **performance requirements**.

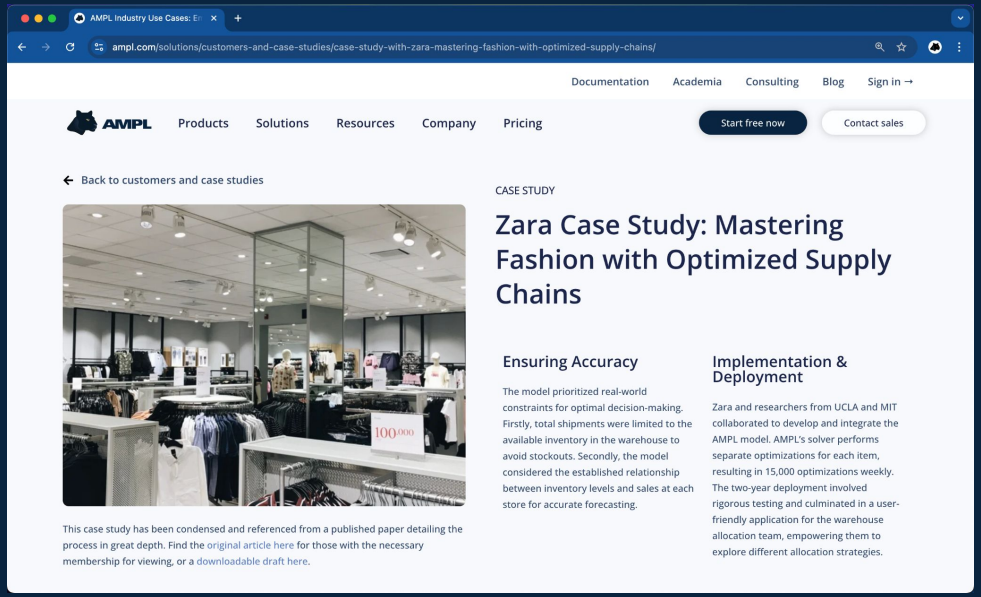
→ Very **complex and detailed models** incorporating complex **business details** and **regulations**.

→ Mission-critical applications where mistakes can be very expensive (e.g., damaging equipment) and **transparent modeling is essential**.

→ Competitive environments where new models must be **quick to develop and deploy**.

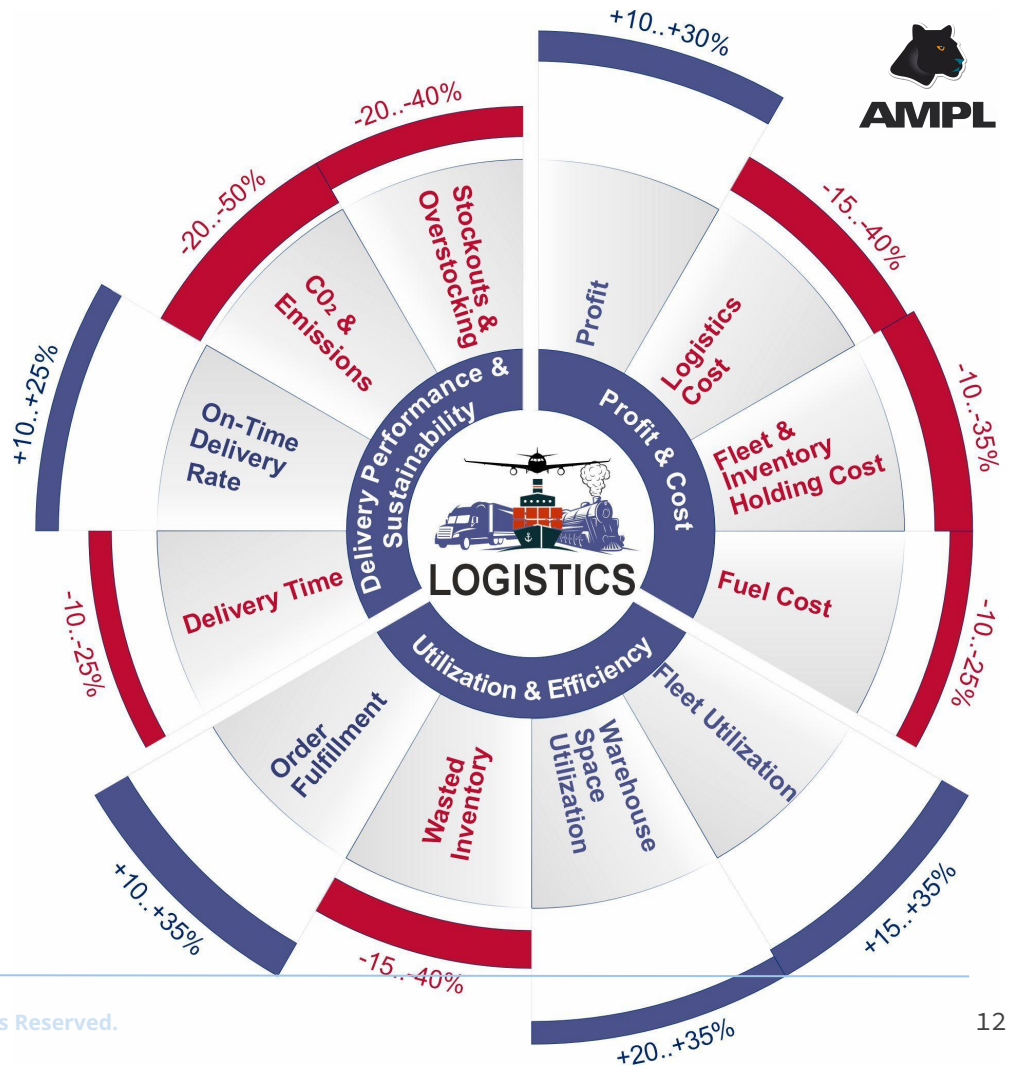
How Optimization Generates Millions💰

Zara Case Study



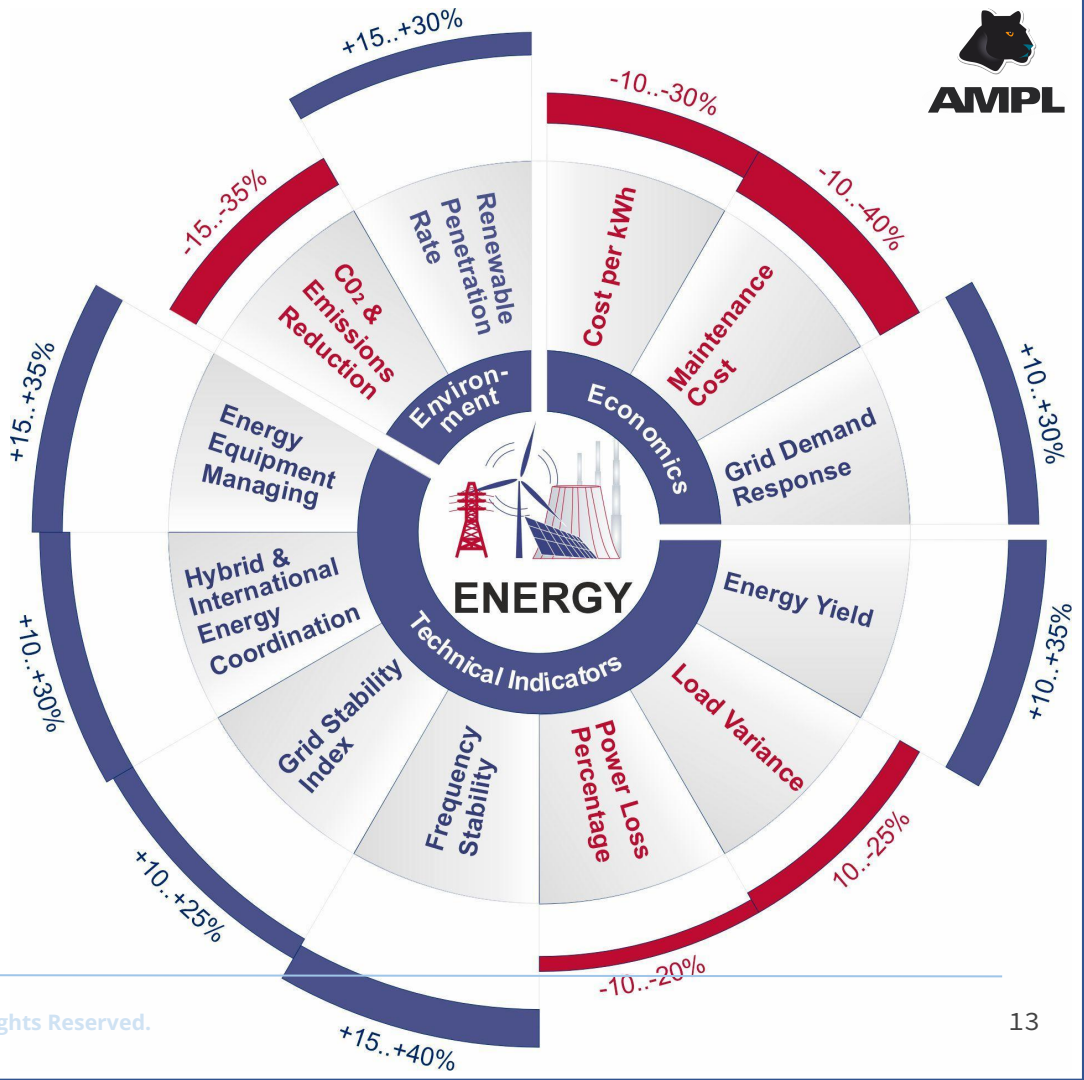
AMPL in Action

Logistics & Supply Chain



AMPL in Action

Finance & Energy

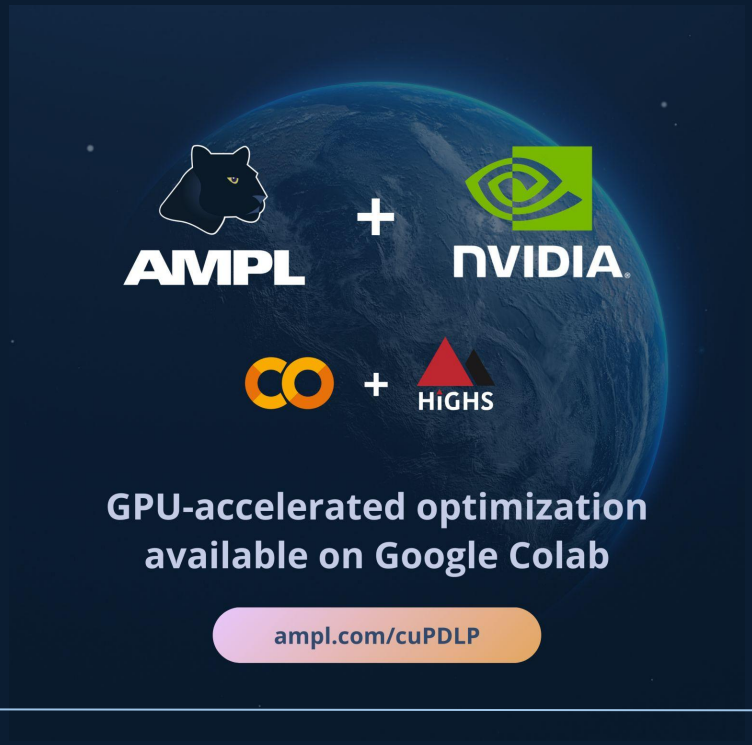


ABB

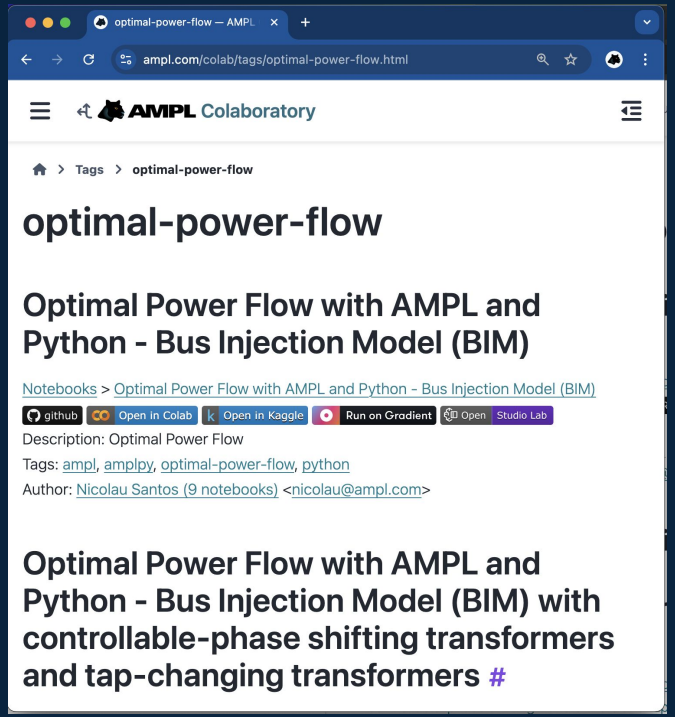
New York ISO
Independent System Operator

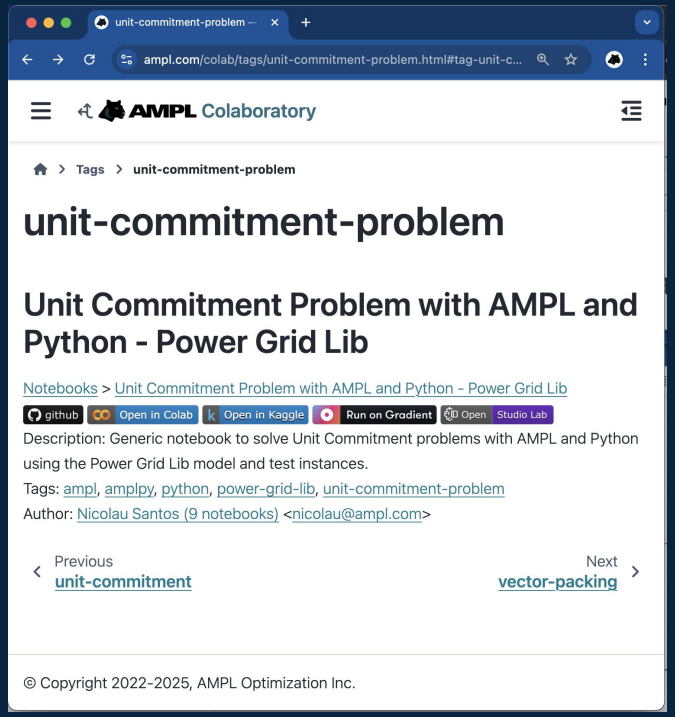
ercot
Your Power. Our Promise.

Solving Energy Models with GPUs *(https://ampl.com/cuPDLP)*

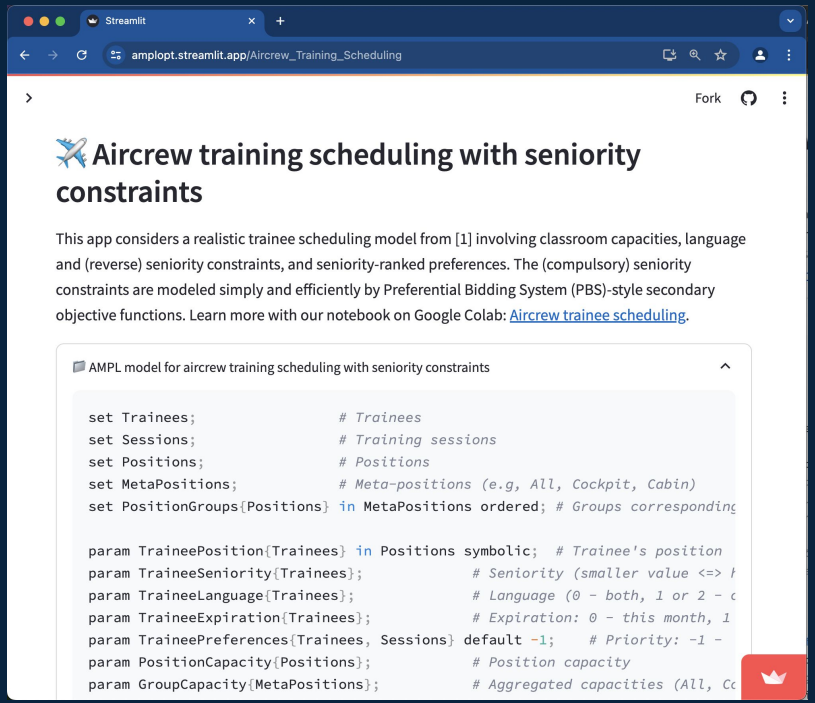


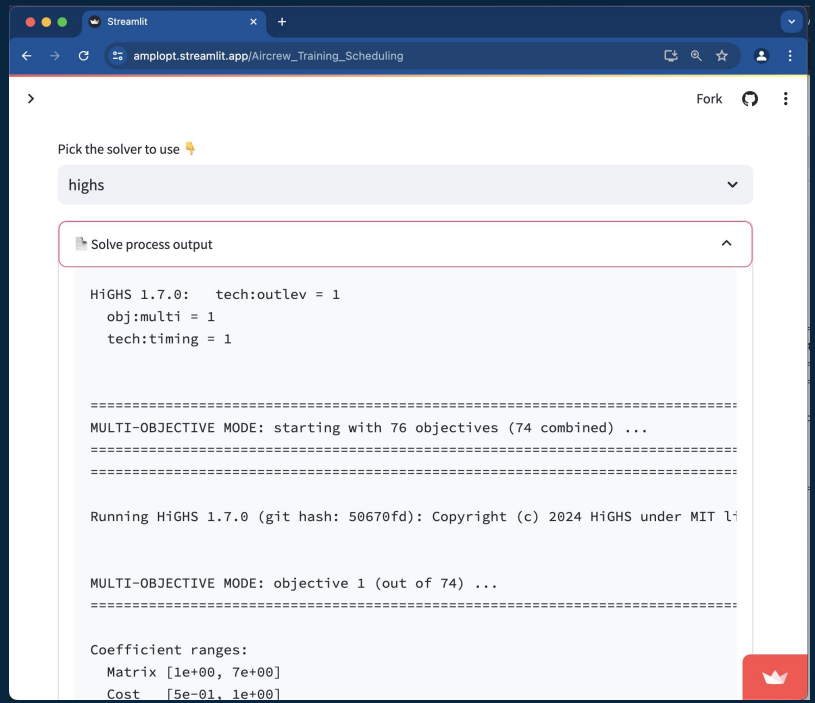
Open-source repo of notebooks (https://ampl.com/colab)





Interactive optimization Apps (https://ampl.com/streamlit)





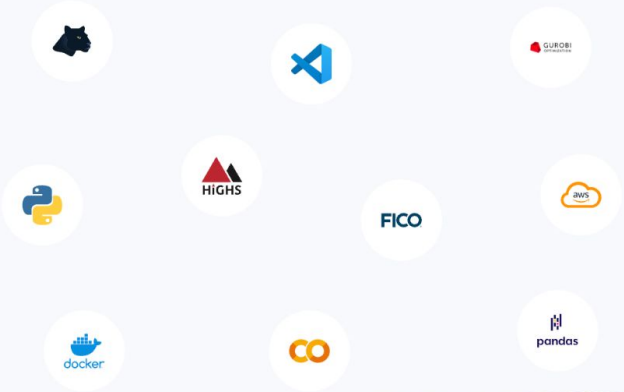
Industry level experience for Academia too

AMPL for Academia

Free Academic Licenses

Industry-leading software for tomorrow's optimization specialists

Empowering the academic community with the same professional modeling system used by global companies. AMPL integrates with Python, VS Code, and modern data stacks – making it easy to go from classroom to production without compromises.



Academic Community License

AMPL for Academics (A4A)

AMPL for Academics (A4A) provides students, faculty, and researchers full AMPL functionality with academic access to select commercial solvers – ideal for coursework, thesis projects,

Academic Teaching License

AMPL for Courses (A4C)

AMPL for Courses (A4C) supports educators in course-based teaching. Easily distributed to entire classes through shareable links and activation codes enabling collaborative use for

Young Professionals License

AMPL Career Starter (ACS)

AMPL Career Starter (ACS) enables recent graduates to take their optimization skills from the classroom to professional practice, bridging the gap between academic study and

AMPL for Academics

Includes:

→ Completely free

→ No size restrictions
→ Full Ampl
+ All Open source solvers
+ select Commercial solvers (Gurobi, CPLEX, Mosek, COPT, Xpress)

Requires:

→ Academic email for access (sign-up at portal.ampl.com)

AMPL for Courses

Includes:

→ Full AMPL + **all** solvers

→ Dynamic cloud licensing by default

1 *uuid* shareable with your course

→ Independent from installation

→ Straightforward Google Colab / Python integration

AMPL Career Starter

AMPLify your career!

Includes:

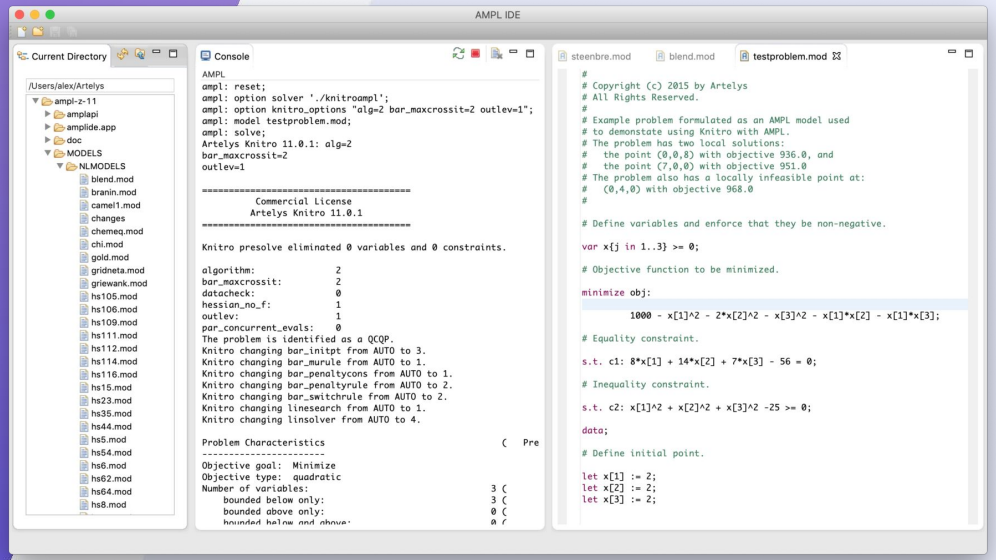
→ Accessible within 24 months post-graduation for 12-month period

→ Full AMPL with open source + commercial solvers

→ Commercial/production use for your place of business

Writing optimization in 2025

- Directly from Google Colab
- From wherever you write Python
- Amplide? (Nope)

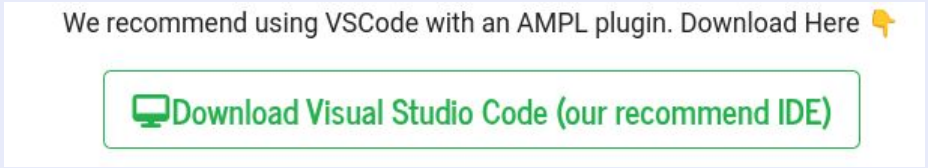




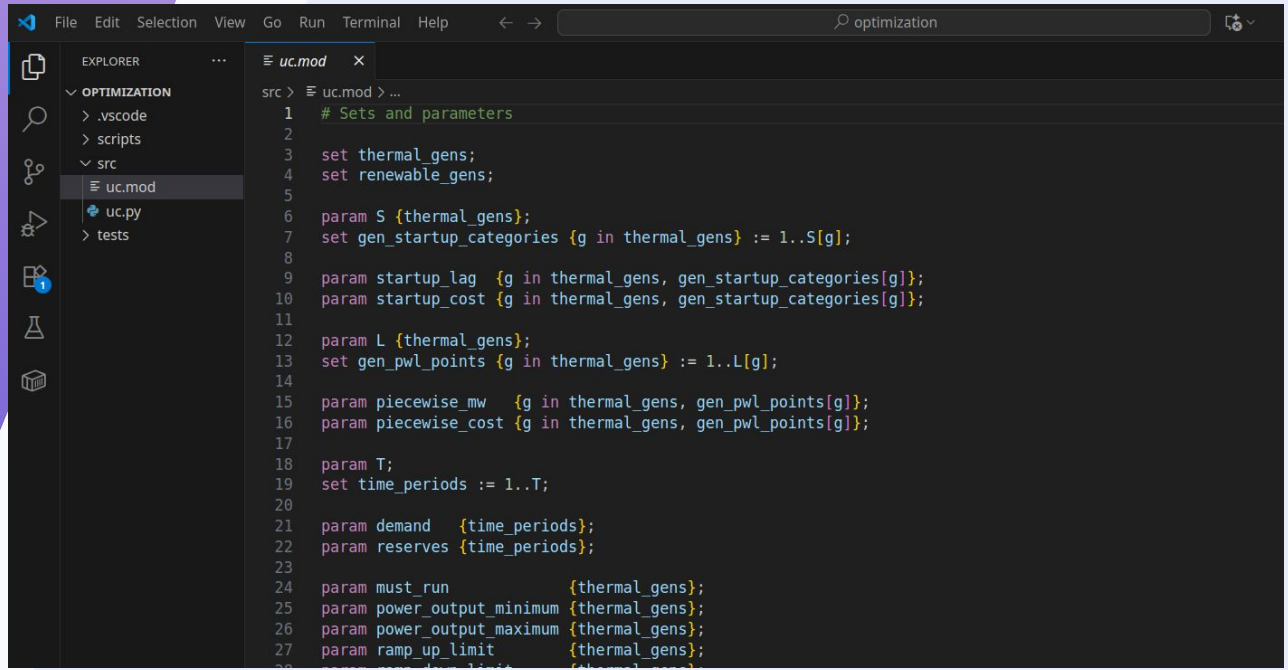

Visual Studio Code Extension

→ Develop Python and Ampl code through Visual Studio Code

→ Check out the **new official plugin**! VS Code Marketplace



Visual Studio Code Extension



Try the amplbot!

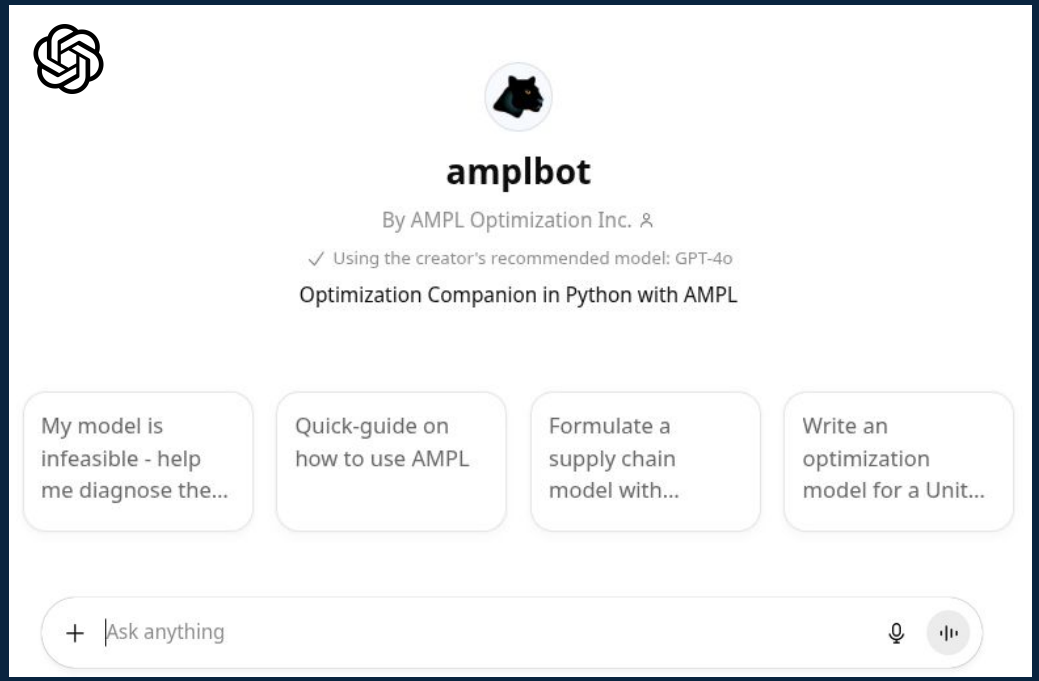
Available at *chatgpt.com*

→ High Level modeling

→ Troubleshooting

→ Explainability

→ Substantial improve over baseline LLMs



Part II

Complex Problems Can Be Made Simple!

Outline Part II

→ Why enhanced modeling?

- User vs. solver perspective

→ Enhanced expression support

- (Automatic) reformulation examples & control options
- MINLP support
- Multi-objective modeling

→ Tools

- Automatic solution checker
- Reformulation Explorer
- Solvers

ChatGPT: “how to formulate a disjunction in MIP?”

1. General Form of a Disjunction

A **disjunction** expresses that **at least one** of several constraint sets must hold:

$$(A_1x \le b_1) \quad \vee \quad (A_2x \le b_2)$$

2. Standard MIP Reformulation Using Binary Variables

To encode this in a linear MIP, introduce a **binary variable** $y \in \{0,1\}$.

Then, we can linearize as:

$$A_1x \le b_1 + M_1y$$
$$A_2x \le b_2 + M_2(1-y)$$

Here:

- M_1, M_2 are **big-M constants** (large enough to “turn off” the constraints when not active).

ChatGPT: “how to formulate a disjunction in MIP?”

```
........ (continued)

7. Example in AMPL (Using Logical Constraints via MP Library)

AMPL MP allows disjunctions directly:

     subject to either_or:

          (x1 <= 5) or (x2 <= 3);
```

→ **As a modeler, which would be your preference?**

Enhanced modeling: solver perspective

Disjunction modeling

→ Most commercial MIP solvers: *indicator constraints*

→ Except Mosek: *disjunctive normal form*

→ Constraint Programming solvers: arbitrary logical formulas

Solvers claim that their specialized input representations are best for them!

AMPL MP Library: new solver interfaces

Motivation

→ Describe optimization models to computers more like you think about them

- Development speed, transparency, maintainability
- Better/more general information for the solver => performance ?

Strategy

→ Extend modeling tools to more natural objective and constraint expressions

→ Build solver interfaces to convert natural expressions to forms that solvers require

Supported Expressions

AMPL operators and functions

- Conditional: **if-then-else**; **==>**, **<==**, **<==>**
- Logical: **or**, **and**, **not**; **exists**, **forall**
- Piecewise linear: **abs**; **min**, **max**; **<<breakpoints; slopes>>**
- Counting: **count**; **atmost**, **atleast**, **exactly**; **numberof**
- Comparison: **>**, **<**, **!=**; **alldiff**
- Complementarity: **complements**
- Nonlinear: *****, **/**, **^**; **exp**, **log**; **sin**, **cos**, **tan**; **sinh**, **cosh**, **tanh**
- Set membership: **in**

Recognizable expressions

- High-order polynomials
- Second-order cones
- Exponential cones (MOSEK driver!)

A basic case: indicator (into inequality)

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

- **Natively supported by many solvers**
- **Important: tight bounds on the compared expression, e.g.:**

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

- **Linearized when not supported, or when desired (option acc:indle=0):**

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


Disjunction: ||, exists

```
    subject to Disjunction:
        (level_A <= 5) || (level_B >= 10);
```

Reformulation for a typical MIP solver using indicators:

```
        var {1..3} aux_result: binary;
        s.t. Term1: aux_result[1] ==> level_A <= 5;
        s.t. Term2: aux_result[2] ==> level_B >= 10;
        s.t. Disj__:
            aux_result[3] ==> (aux_result[1] || aux_result[2]);
        s.t. FixResult: aux_result[3] <==> True;
```

Indicators and OR are linearized if necessary.

==> vs <==> vs <==: when which?

```
minimize TotalCost:
    if (level_A <= 5) || (level_B >= 10) then 1000;  # else 0
```

Reformulation:

```
    var {1..3} aux_result: binary;
    s.t. Term1: aux_result[1] <== level_A <= 5;
    s.t. Term2: aux_result[2] <== level_B >= 10;
    s.t. Disj__:
        aux_result[3] <== (aux_result[1] || aux_result[2]);
    minimize TotalCost__: 1000*aux_result[3];
```

The disjunction is in *negative context* which reverses the implications.

An experiment with max()

From S. Brand et al. (2008), *Flexible, rule-based constraint model linearisation*, citing a radiation model for intensity-modulated radiotherapy, core part: min-cardinality decomposition in C1P matrices:

```
subject to Increment_Constraints {i in ROWS, b in BTIMES}:
    N[b] >= Q[i,1,b]
                + sum {j in 2..n} max(Q[i,j,b] - Q[i,j-1,b], 0);
```

- The `max()` expressions are in negative context: sufficient linearization is

```
var Z {i,j,b} >=0;
s.t. LinearizeNegCtx {i,j,b}:
          Z[i,j,b] >= Q[i,j,b] - Q[i,j-1,b];
subject to Increment_Constraints_Lin {i in ROWS, b in BTIMES}:
    N[b] >= Q[i,1,b]
                + sum {j in 2..n} Z[i,j,b];
```


An experiment with max()

```
subject to Increment_Constraints{i in ROWS, b in BTIMES}:
    N[b] >= Q[i,1,b] + sum{j in 2..n} max(Q[i,j,b] - Q[i,j-1,b], 0);
```

- The `max()` expressions are in negative context: sufficient linearization is

```
var Z {i,j,b} >=0;
s.t. LinearizeNegCtx {i,j,b}: Z[i,j,b] >= Q[i,j,b] - Q[i,j-1,b];
```

- Authors observe that CPLEX 9.1 performs much better with full linearization which includes positive context:

```
var B {i,j,b,1..2} binary;
s.t. Disjunction {i,j,b}: B[i,j,b,1] + B[i,j,b,2] >= 1;
s.t. LinearizePosCtx {i,j,b}:
     B[i,j,b,1]==1 ==> Z[i,j,b] <= Q[i,j,b] - Q[i,j-1,b]
     && B[i,j,b,1]==1 ==> Z[i,j,b] <= 0;
```


Replicate the experiment

		Native max()		Linear max()		Linear max(), full	
		N solved	mean *t*	N solved	mean *t*	N solved	mean *t*
Max. intens. 10	CPLEX 22.1.1	-	-	10	**1.80**	10	2.53
	Gurobi 12.0.3	**10**	**0.81**	10	1.02	10	1.06
	Highs 1.11	-	-	10	**3.33**	10	6.37
Max. intens. 12	CPLEX	-	-	10	**18.47**	10	26.05
	Gurobi	**10**	**6.21**	10	10.27	9	33.50
	Highs	-	-	10	**12.18**	10	20.69

Apple M1 Pro, 1 thread, 10 instances per category, time limit 300s

Discussion

- Observations contrast those from S. Brand et al. (2008):
  - Gurobi:
    - Native max() is best, followed by simple linearization. Full linearization is worst.
  - CPLEX 22.1.1, Highs:
    - Simple (context-aware) linearization is best, as expected.

- Options to control reformulations (dev.ampl.com):
  - `acc:max=4` as expression tree node, if supported
  - `acc:max=2` as general constraint, if supported
  - `acc:max=0` linearize (context-aware by default)
  - `cvt:pre:ctx:max=0` full linearization

Corner Case: Strict Comparison

Consider the above reverse implication:

```
s.t. Term1: aux_result[1] <== level_A <= 5;
```

Which is equivalent to the indicator constraint

```
Term1IndStrict: (aux_result[1]==0) ==> (level_A > 5);
```

Assuming `level_A` is real-valued, what is the meaning of `(level_A > 5)`?

- For the modeler…
- For the chip…

Strict comparison tolerance

```
IndStrict: (aux_result[1]==0) ==> (level_A > 5);
```

AMPL MP converts strict comparisons for MIP solvers into non-strict comparisons using option `cmp:mip:eps` (default value 1e-4):

```
IndNonStrict: (aux_result[1]==0) ==> (level_A >= 5.0001);
```

- A guideline for the value of `cmp:mip:eps` is to set it at least 10x larger as `feastol`, the primal feasibility tolerance (default 1e-6 for most solvers).
- When the comparison has a single context (pos/neg), the strictness can be chosen to avoid tolerance application:

```
minimize TotalCost_NoFinalStrictCmp:
     if (level_A < 5) || (level_B > 10) then 1000;
```


(Dis)equalities

- Implied equality: again, an indicator

  ```
  s.t. IndEq: b==0 ==> 5*x[1] + 2*x[2] == 7;
  ```

- Reverse implication:

  ```
  s.t. IndEqRev0: b==0 <== 5*x[1] + 2*x[2] == 7;
  ```

  - Equivalent to implied disequality

    ```
    s.t. IndDiseq: b==1 ==> 5*x[1] + 2*x[2] != 7;
    ```

    - Reformulation as a disjunction:

      ```
      ... ==> (5*x[1]+2*x[2] < 7 or 5*x[1]+2*x[2] > 7);
      ```

    - *Unary encoding* is applied in some cases

Unary encoding

```
var Final_patt_seq {SLOT_SEQ} in PATTERNS_ALL;    # integer vars

s.t. assign_color_SLV {r in ROUND, s in SLOT, k in PATTERNS: ...}:
     Final_patt_seq[slot_r_cum[r] + s]==k
               ==> exists {i in 1..noSLV}
                  (Color_SLV[slot_r_cum[r] + s,i]==color_spec1[k]
                   and Avail_SLV[slot_r_cum[r] + s,i]);
```

- **When an integer variable compares to several values, unary encoding applies:**

```
var x in 1..3;
```

translates into

```
var unary {1..3} binary;
x == unary[1] + 2*unary[2] + 3*unary[3];
1 == sum {i in 1..3} unary[i];
```

Options `uenc:ratio` and `uenc:negctx` control the choice between reformulations.

Operators count, atleast/atmost/exactly, numberof

Generalize disjunction to count the number of true terms:

```
minimize NumBigDeviations:
     count {i in ASSETS}
          ( abs( weight[i]-benchmark[i] ) > 0.005 );

s.t. LimitBigWeights:
     atmost 5 {i in ASSETS} ( weight[i] > 0.05 );
```

Finance AMPL Colab notebooks: https://colab.ampl.com/tags/finance.html

The Golomb Ruler problem

A *Golomb ruler* is a set of marks at integer positions along a ruler such that no two pairs of marks are the same distance apart.

```
param m default 4;
param n = m*m;

set OrdPairs = {i in 1..m-1, j in 2..m: i<j};

var mark {i in 1..m} in 0..n;
var differences {(i,j) in OrdPairs} in 1..n;

s.t. AssignDiffs {(i,j) in OrdPairs}:
     differences[i,j] == mark[j] - mark[i];

s.t. FixMark1: mark[1] == 0;
s.t. MarksOrdered {i in 1..m-1}: mark[i] <= mark[i+1]-1;
```


The Golomb Ruler problem

```
param DefaultAllDiff default 1;      ## Choose the alldiff

s.t. AllDiff {if DefaultAllDiff}:    ## MIP reform. uses UEnc
    alldiff {(i,j) in OrdPairs} differences[i,j];   ## CP: native

s.t. AllDiffNaive {if not DefaultAllDiff}:
    forall {(i,j) in OrdPairs, (k,l) in OrdPairs:
                          i<k || (i==k && j<l)}
     differences[i,j] != differences[k,l];

s.t. AntiSymm3: mark[2] - mark[1] <= mark[m] - mark[m-1] - 1;

minimize Largest: mark[m];
```


Golomb Ruler

		Native alldiff		alldiff -> Uenc		Naïve alldiff	
m (opt)		nodes/cp	*t*	nodes/cp	*t*	nodes/cp	*t*
7 (25)	IBM ILOG CP	**1693**	**0.02**	96782	2.35	**1693**	**0.02**
	Gurobi	-	-	**985**	2.94	1533	**0.47**
	Highs	-	-	**660**	2.57	1282	**1.26**
9 (44)	IBM ILOG CP	**149818**	**1.82**	3613742	269.85	**149818**	**1.81**
	Gurobi	-	-	**4437**	**69.81**	18463	82.32
	Highs	-	-	**34194**	**154.75**	286620	508.96

Apple M1 Pro, 1 thread

Golomb Ruler OLD

		alldiff		alldiff naive	
m (opt)		nodes/cp	*t*	nodes/cp	*t*
7 (25)	IBM ILOG CP	1693	0.02	1693	0.02
	Gurobi	**985**	2.94	1533	**0.47**
	Highs	**660**	2.57	1282	**1.26**
9 (44)	IBM ILOG CP	149818	1.82	149818	1.81
	Gurobi	**4437**	**69.81**	18463	82.32
	Highs	**34194**	**154.75**	286620	508.96

Apple M1 Pro, 1 thread

Reformulation control essentials (dev.ampl.com)

- Options `acc:max`, `acc:abs`, `cvt:pre:ctx:cos`, etc.
  - Native constraint/expression acceptance and context
- Option `cvt:mip:eps`
  - Real-valued strict comparison tolerance
- Option `cvt:bigM`
  - Default big-M value for unbounded variables, when linearization is necessary or desired. Avoid unbounded variables.
- Options `cvt:compl[:eps]`
  - Complementarity reformulations
- Options `cvt:plapprox:...`
  - Domain and precision of piecewise-linear approximation of nonlinear functions

(MI)NLP support for MP-based solvers

(MI)NLP support for Gurobi, Xpress Global, COPT (NLP only), BaronMP, and SCIP

```
minimize Fchi:
      x[1]^2 - 12*x[1] + 11 + 10*cos(pi*x[1]/2)
    + 8*sin(pi*5*x[1]) - exp(-(x[2] - .5)^2/2)/sqrt(5);
```

- Disabled in COPT by default, set

```
acc:_expr=2 [alg:relax=1]
```


Multiobjective modeling

- Multiobjective models can be simpler for certain problems
  - Some constraints can be moved to the objectives as penalties
  - Lexicographic optimization allows objective ranking
- Example on AMPL Colab: aircrew trainee scheduling with seniority constraints https://colab.ampl.com/tags/multi-objective.html
  - Smaller model and faster solving than the original hard-constrained model

```
maximize ReverseSeniority {e in 1..2, i in I: E[i]==e}:
  sum {t in V[i]: Pr[i, t]==0}
    (S[i] - min {j in I} S[j] + 1) * x[i, t]
  suffix objpriority (2-e)*S_range + 1 + S[i] - min {j in I} S[j];
```


Automatic solution checker

- All MP solvers automatically check every solution
- Tolerance options
  - `sol:chk:feastol`, `sol:chk:feastolrel` (both default 1e-6)
  - `sol:chk:inttol` (default 1e-5) integrality
  - Warning on violation, or abort if option `sol:chk:fail` given
- Checks only original (AMPL-side) and final (solver-side) model items by default
  - Option `sol:chk:mode` can add intermediate reformulated expressions

https://mp.ampl.com/modeling-tools.html#automatic-solution-check

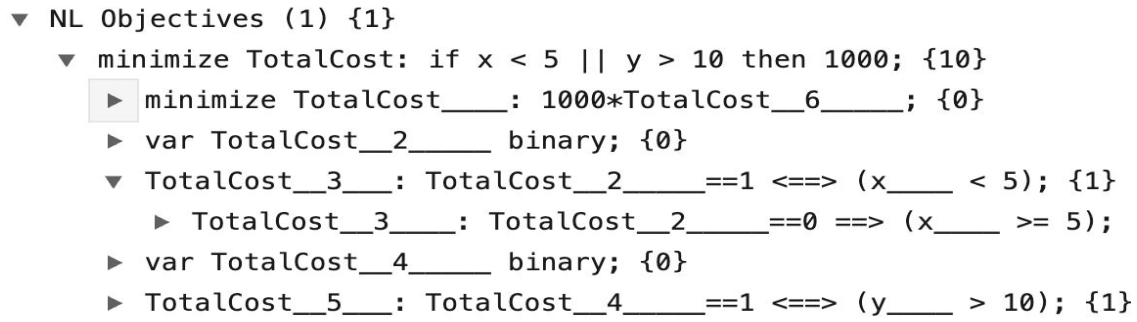
Check solutions yourself!

E.g., in AMPL:

- Objective value(s):
  ```
  _obj
  ```
- Algebraic constraints and variable bounds:
  ```
  min{i in 1 ..  ncons}  con[i].slack    # Should be >= -1e-6
  min{i in 1 .. _nvars} _var[i].slack
  ```
- Logical constraints:
  ```
  if forall {i in 1.._nlogcons} _logcon[i] then 1
  ```
- Complementarities:
  ```
  max{i in 1 .. _nccons} abs(_ccon[i])    # Should be <= 1e-6
  ```


Reformulation Explorer

- Want to know what AMPL MP does to your model?
- Simplest way: option `writeprob=model.lp` or similar
  - Exports solver model in the LP or another format
- To see intermediate reformulation steps, use https://mp.ampl.com/modeling-tools.html#reformulation-explorer.



NL reader and writer libraries

- NL is a low-level model format accepted by AMPL solvers and used by several modeling systems
- NL reader and writer libraries in MP use low-overhead C++ template-based interfaces
- C and Python NL writer wrappers are available for MILP
  - Package nlwpy on PyPi

Which solver to use?

(Originally) Linear solvers

[1] GPU-accelerated algorithms, such as PDLP, are available.

[2] Conic programming: MOSEK supports SOCP and exponential cones, other solvers only SOCP.

	LP	MILP	QP	MIQP	convex (MI)QCP	non-convex (MI)QCP	Conic	MINLP	MP
Gurobi	✅	✅	✅	✅	✅	✅	✅	✅	✅
FICO XPRESS	✅	✅	✅	✅	✅	✅	✅	✅	✅
IBM CPLEX	✅	✅	✅	✅	✅	❌	✅	❌	✅
COPT	✅[1]	✅	✅	✅	✅	❌	✅	❌	✅
MOSEK	✅	✅	✅	✅	✅	❌	✅[2]	❌	✅
NVIDIA cuOpt	✅[1]	✅[1]	❌	❌	❌	❌	❌	❌	✅
HiGHS	✅[1]	✅	✅	❌	❌	❌	❌	❌	✅
CBC	✅	✅	✅	✅	❌	❌	❌	❌	✅
SCIP	✅	✅	✅	✅	✅	✅	✅	✅	✅
GCG	✅	✅	✅	✅	✅	✅	✅	✅	✅

Nonlinear solvers

[3]

Global optimality: while Knitro and Bonmin accept integer variables, they only guarantee local optimum for non-convex models.

[4]

With the MP2NL meta-driver.

	NLP	MINLP	Global [3]	MP
Knitro	✅	✅	❌	✅[4]
BARON	✅	✅	✅	✅
LINDO Global Solver	✅	✅	✅	✅[4]
Octeract	✅	✅	✅	✅[4]
RAPOSa	✅	✅	✅	✅[4]
LGO	✅	❌	✅	✅[4]
CONOPT	✅	❌	❌	✅[4]
LOQO	✅	❌	❌	✅[4]
MINOS	✅	❌	❌	✅[4]
SNOPT	✅	❌	❌	✅[4]
IPOPT	✅	❌	❌	✅[4]
BONMIN	✅	✅	❌	✅[4]
COUENNE	✅	✅	✅	✅[4]

AMPL Constraint Programming solvers

- IBM ILOG CP
- Gecode

CP solvers support most combinatorial MP constructs natively.

Summary

- AMPL+MP: bridge between modelers and solvers
  - Write models more like you think about them
  - Solvers receive the preferred model form

- “Set and forget”: our approach to modern real-world optimization development

References

- **https://ampl.com/academia**
  - Free full-size academic licenses
- **https://mp.ampl.com/model-guide.html**
  - Modeling Guide for MP-based AMPL Solvers
- **https://ampl.com/streamlit**
  - Streamlit App with many examples
- **https://ampl.com/mo-book**
  - New AMPL+Python Book
- **https://ampl.com/colab**
  - Collection of AMPL models in Jupyter Notebooks
- **try.ampl.com**

Thank you!

Visit us @ Booth 221

Technology Showcase: *From Classroom to Industry: Modern Optimization with AMPL*

Tuesday, October 28 | 11:00 AM - 12:15 PM

Building A Level 3 A301

Technical session: *Efficient Data Exchange in Amplpy via Apache Arrow Integration.* Jurgen Lentz

Tuesday, October 28 | 2:45 PM - 4:00 PM

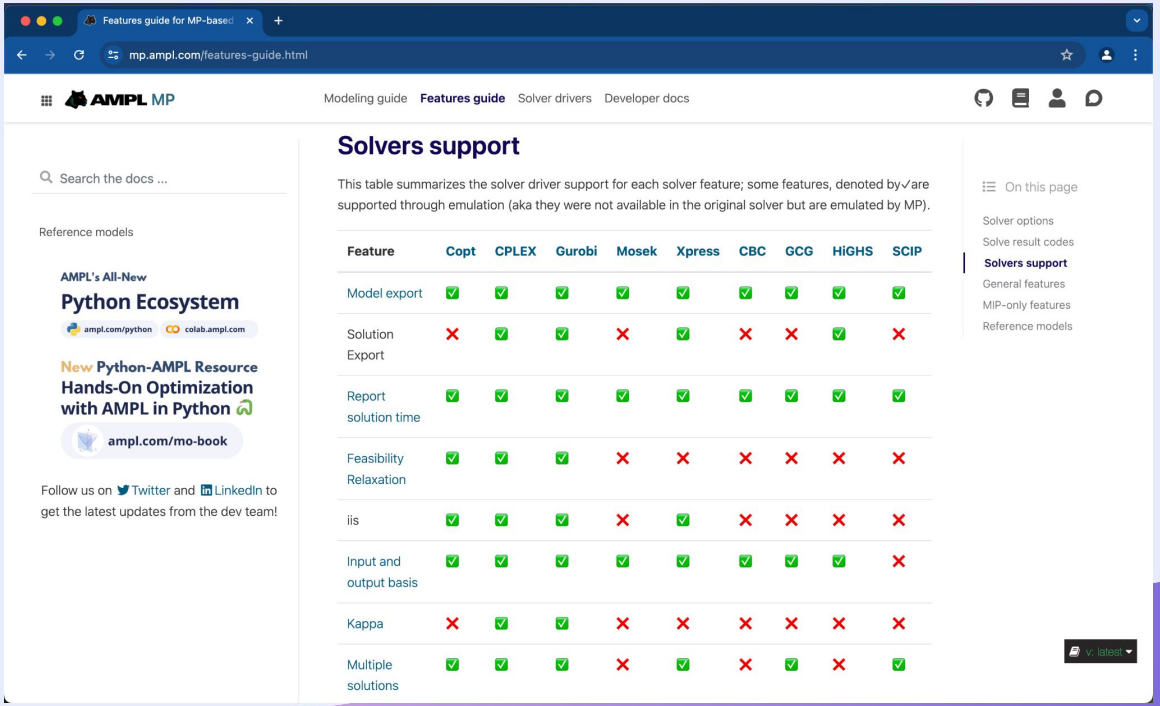
Building B Level 2 B201

Contact us:
academia@ampl.com

AMPL Academic Community:
https://discuss.ampl.com/

Supported Solver Features and Emulated Features

(https://mp.ampl.com/features-guide.html)



Supported Solver Features and Emulated Features: the list goes on…
(https://mp.ampl.com/features-guide.html)