CPLEX Implementation Roadmap

From Model to Solution

A Complete Guide for Technical Users

I. Prerequisites & Preparation

System Requirements

Before beginning installation, ensure your system meets the following requirements:

- **Operating System:** Windows 10/11, Linux, or macOS
- **Python Version:** Python 3.7 or higher (Python 3.9+ recommended)
- **Development Environment**: Jupyter Notebook, or Visual Studio Code

Before You Begin

License Acquisition

You will need a valid AMPL license to use CPLEX. License types include:

- **Free Trial License:** Available for evaluation purposes
- **Community Edition License:** For educational institutions and research
- **Commercial License:** Full-featured license for commercial use

Your license will be provided as a UUID (Universally Unique Identifier) that looks like:

```
12345678-1234-1234-1234-123456789abc
```

Choosing Your Installation Method

AMPL and CPLEX can be installed using the following method:

- **amplpy with Python Modules (Recommended):** Best for Python users who want seamless integration with Python workflows. Quick installation via pip with automatic dependency management.
- **For other language APIs (R, MATLAB, C++, Java)**, see https://dev.ampl.com/ampl/install.html

Note: *Both methods provide full CPLEX functionality. The choice depends on your preferred development environment.*

II. Installation

Installation via amplpy (Recommended)

This method provides the fastest path to getting started with AMPL and CPLEX in Python environments.

Step 1: Install amplpy

Open your terminal or command prompt and run:

```
python -m pip install amplpy --upgrade
```

Common Issues:

- **Permission denied:** Add `--user` flag or use sudo on Linux/macOS
- **Proxy issues:** Configure pip proxy with `--proxy http://user:pass@server:port`

Step 2: Install CPLEX Module

Once amplpy is installed, add the CPLEX solver module:

```
python -m amplpy.modules install cplex
```

What Gets Installed:

- The latest version of CPLEX solver optimized for your platform
- Required libraries and dependencies
- Integration files for seamless AMPLpy operation

Installation Verification:

Modules are installed in your Python environment's site-packages directory. To verify your installed modules run:

```
python -m amplpy.modules installed
```

Step 3: Activate License

Activate your AMPL license using your UUID:

```
python -m amplpy.modules activate <license-uuid>
```

Replace `<license-uuid>` with your actual license UUID (without angle brackets).

License Verification:

- Upon successful activation, you'll see a confirmation message
- The license file will be stored in your AMPL configuration directory
- Your license type and limitations will be displayed

Verification Checkpoint:

- Check installation by running: `python -c "import amplpy; print(amplpy.__version__)"`
- You should see the amplpy version number printed
- To test the AMPL license, you can run in your terminal: `python -m amplpy.modules run ampl -vvq`

III. Verification & Testing

Test the complete workflow with a simple optimization problem:

Simple Linear Programming Example:

Create model file (test_model.mod)

```
# Define variables
var x >= 0;
var y >= 0;
# Objective function
maximize profit: 3*x + 2*y;
# Constraints
subject to labor: 2*x + y <= 100;
subject to materials: x + y <= 80;
```

Access and solve through Python

```
# from amplpy import AMPL

# Create AMPL instance
ampl = AMPL()

# Load model
ampl.read("test_model.mod")

# Set solver and solve
ampl.option["solver"] = "cplex"
ampl.solve()

# Check solve status
if ampl.solve_result == "solved":
    # Get objective value
    profit = ampl.get_objective("profit")
    print(f"Optimal profit: {profit.value()}")

    # Get variable values
    x_val = ampl.get_variable("x").value()
    y_val = ampl.get_variable("y").value()
    print(f"x = {x_val}, y = {y_val}")
else:
    print(f"Solve failed: {ampl.solve_result}")
```

Expected Output:

CPLEX should solve the problem and display:

```
Optimal profit: 180.0
x = 20.0, y = 60.0
```

Interpretation:

- **Solver Status:** optimal solution (problem solved successfully)

Number	String	Interpretation
0-99	solved	Optimal solution found
100-199	solved?	Optimal solution indicated, but error likely
200-299	infeasible	Constraints cannot be satisfied
300-399	unbounded	Objective can be improved without limit
400-499	limit	Stopped by a limit that you set (such as on iterations)
500-599	failure	Stopped by an error condition in the solver routines

- **Objective Value:** 180 (maximum profit achieved)
- **Decision Variables:** x=20, y=60 (optimal production quantities)

Troubleshooting If Solve Fails:

- **License error:** Verify license activation (check license UUID), for additional troubleshooting refer to https://dev.ampl.com/help/dynamic-license-troubleshooting.html.
- **Solver not found:** Ensure CPLEX is in PATH and properly installed through the amplpy module
- **Syntax errors:** Check for typos in the model formulation

IV. Working with CPLEX

Understanding CPLEX Options

CPLEX provides hundreds of parameters to control solver behavior. Understanding and using these options is key to achieving optimal performance. Some optimization specialists dedicate significant time to mastering CPLEX's full parameter space. This guide covers the most impactful options for typical users—focusing on the 80/20 of practical usage.

For comprehensive CPLEX documentation, see: https://www.ibm.com/docs/en/icos/22.1.2

For the complete list of CPLEX options available through AMPL, see: https://dev.ampl.com/solvers/cplex/options.html

Major Categories of Options:

- **Algorithm Selection:** Choose between primal simplex, dual simplex, barrier, network optimizer, or concurrent methods
- **MIP Parameters:** Control cuts, heuristics, branching strategies, and node selection for mixed-integer programs
- **Tolerances:** Set feasibility, optimality, and integrality tolerances
- **Resource Limits:** Time limits, iteration limits, memory limits
- **Output Control:** Display settings, logging options, solution file generation

Commonly Used Options:

- `mipgap`: Relative MIP gap tolerance (default 0.0001 or 0.01%)
- `timelimit`: Maximum solution time in seconds
- `threads`: Number of parallel threads (0 = automatic)
- outlev: Control solver verbose (0=off, 1=on)
- mipdisplay: Control MIP progress display (0=none, 1=normal, 2=detailed). It's necessary to activate outlev=1
- `presolve`: Presolve level (0=off, 1=on)
- `mipfocus`: Useful in scheduling problems, set to 1 to improve finding feasible solutions

Setting Parameters

CPLEX options can be set in multiple ways depending on your workflow.

Via Python/AMPLpy

```
from amplpy import AMPL
ampl = AMPL()
ampl.setOption('solver', 'cplex')
ampl.setOption('cplex_options', 'mipgap=0.01 timelimit=300')
ampl.solve()
```

Algorithm Selection

Choosing the right algorithm can dramatically impact solution time. CPLEX offers several optimization algorithms for different problem types.

Linear Programming Algorithms:

- `Primal Simplex` *(lpmethod=1)*
- `Dual Simplex` *(lpmethod=2)*
- `Barrier Method` *(lpmethod=4)*: Best for large, sparse problems; uses interior-point method
- `Network Optimizer` *(lpmethod=3)*: Specialized for pure network flow problems

Mixed-Integer Programming Strategies:

- `Emphasis on Optimality` *(mipemphasis=2)*: Proves optimality rigorously (may take longer)
- `Emphasis on Feasibility` *(mipemphasis=1)*: Finds good feasible solutions quickly
- `Emphasis on Moving Best Bound` *(mipemphasis=3)*: Improves bound quickly
- `Emphasis on Hidden Feasible Solutions` *(mipemphasis=4)*: For problems where finding feasible solutions is difficult

Advanced CPLEX Features

Infeasibility Analysis (IIS)

When a model is infeasible, CPLEX can identify an Irreducible Inconsistent Subsystem (IIS) - a minimal set of constraints that cannot be satisfied simultaneously. To return consistent IIS results, it is recommended to set AMPL's presolve (distinct from CPLEX presolve) to 0, through `ampl.option["presolve"] = 0`

Detecting Infeasibility:

```
ampl = AMPL()
ampl.read("infeasible_model.mod")

ampl.option["solver"] = "cplex"
ampl.option["cplex_options"] = "iisfind=1"
if ampl.solve_result == "infeasible":
    var_iis, con_iis = ampl.get_iis()
    print(var_iis, con_iis)
```

Interpreting IIS Output:

- CPLEX will report which constraints are in the IIS
- Review these constraints to identify modeling errors or conflicting requirements
- Common causes: typos, incorrect bounds, mutually exclusive requirements

Resolving Infeasibilities:

- Relax one or more constraints in the IIS
- Adjust bounds or right-hand sides
- Add slack/surplus variables with penalties in objective

Solution Pool (Multiple Solutions):

Generate and store multiple near-optimal solutions:

```
ampl.option["cplex_options"] = "populatelim=10 poolstub=solutions"
```

This creates alternative solutions that may be valuable for scenario analysis.

Lazy Constraints:

For problems with many constraints where only a subset is active at optimality, lazy constraints can improve performance by adding constraints dynamically during the branch-and-bound process. This requires using AMPL's ampls (AMPL solver libraries) library or callback functions.

Performance Tuning

Memory Management:

- `treememory`: Tree memory limit in MB
- `nodefile`: Control when to write node files to disk

Parallel Processing:

```
ampl.option["cplex_options"] = "threads=8 parallelmode=1"
```

- `threads=0`: Automatic (uses all available cores)
- `threads=N`: Use N threads
- `parallelmode=1`: Deterministic mode (reproducible results)
- `parallelmode=-1`: Opportunistic mode (faster but non-deterministic)

Tuning Tool:

CPLEX includes an automatic parameter tuning tool that experiments with different parameter settings:

```
ampl.option["cplex_options"] = "tunefile=results.prm tunetimelimit=3600"
```

The tuner will report recommended parameter settings for your specific problem.

Warm Starting

Provide initial solution values to help CPLEX start the branch-and-bound process:

```
from amplpy import AMPL

ampl = AMPL()
ampl.read("facility.mod")
# ... set data ...

# Set initial values for binary variables
# (from previous solve or heuristic)
ampl.get_variable('open').set_values({
    'F1': 1,
    'F2': 0,
    'F3': 1
})

# Solve with warm start
ampl.option["solver"] = "cplex"
ampl.solve()
```

V. Practical Examples

End-to-End Examples

Example 1: Linear Programming - Production Planning

Problem Description:

A factory produces two products, A and B. Each product requires labor and materials. The goal is to maximize profit while respecting resource constraints.

Model File (production.mod):

```
# Sets and Parameters
set PRODUCTS;

param profit {PRODUCTS};
param labor_required {PRODUCTS};
param material_required {PRODUCTS};
param labor_available;
param material_available;

# Decision Variables
var produce {PRODUCTS} >= 0;

# Objective Function
maximize total_profit:
    sum {p in PRODUCTS} profit[p] * produce[p];

# Constraints
subject to labor_limit:
    sum {p in PRODUCTS} labor_required[p] * produce[p] <= labor_available;

subject to material_limit:
    sum {p in PRODUCTS} material_required[p] * produce[p] <=
material_available;
```

Python Solution:

```
from amplpy import AMPL
import pandas as pd

# Create AMPL instance
ampl = AMPL()

# Load model
ampl.read("production.mod")

# Prepare data
products_df = pd.DataFrame({
```

```
    'profit': [50, 40],
    'labor_required': [2, 1],
    'material_required': [1, 2]
}, index=[1, 2])
products_df.index.name = 'PRODUCTS'

# Send data to AMPL
ampl.set_data(products_df, 'PRODUCTS')
ampl.get_parameter('labor_available').set(100)
ampl.get_parameter('material_available').set(80)

# Configure solver
ampl.option["solver"] = "cplex"
ampl.option["cplex_options"] = "mipdisplay=1"

# Solve
ampl.solve()

# Check solve status
if ampl.solve_result == "solved":
    # Get objective value
    obj = ampl.get_objective('total_profit')
    print(f"Maximum profit: ${obj.value():.2f}")

    # Get variable values as DataFrame
    production = ampl.get_variable('produce').get_values().to_pandas()
    print("\nOptimal production plan:")
    print(production)

    # Check constraint slack
    labor_slack = ampl.get_constraint('labor_limit').slack()
    material_slack = ampl.get_constraint('material_limit').slack()
    print(f"\nUnused labor: {labor_slack:.2f} hours")
    print(f"Unused materials: {material_slack:.2f} units")
else:
    print(f"Optimization failed: {ampl.solve_result}")
```

Expected Results:

```
CPLEX 22.1.2: optimal solution; objective 2800
2 simplex iterations
Maximum profit: $2800.00

Optimal production plan:
```

```
   produce.val
1           40
2           20

Unused labor: 0.00 hours
Unused materials: 0.00 units
```

Example 2: Mixed-Integer Programming - Facility Location

Problem Description:

Determine which facilities to open and how to allocate customers to facilities to minimize total cost (fixed facility costs plus transportation costs).

Model File (facility.mod):

```
# Sets
set FACILITIES;
set CUSTOMERS;

# Parameters
param fixed_cost {FACILITIES};
param transport_cost {FACILITIES, CUSTOMERS};
param demand {CUSTOMERS};
param capacity {FACILITIES};

# Decision Variables
# Binary: 1 if facility is open, 0 otherwise
var open {FACILITIES} binary;

# Continuous: fraction of customer demand served by facility
var allocate {FACILITIES, CUSTOMERS} >= 0, <= 1;

# Objective Function
# Minimize total cost
minimize total_cost:
    sum {f in FACILITIES} fixed_cost[f] * open[f] +
    sum {f in FACILITIES, c in CUSTOMERS}
        transport_cost[f,c] * demand[c] * allocate[f,c];

# Constraints
# Each customer must be fully served
subject to customer_demand {c in CUSTOMERS}:
    sum {f in FACILITIES} allocate[f,c] = 1;

# Facility cannot serve more than its capacity if open
subject to facility_capacity {f in FACILITIES}:
    sum {c in CUSTOMERS} demand[c] * allocate[f,c] <= capacity[f] * open[f];
```

Key MIP Features:

- **Binary variables:** Model yes/no facility opening decisions
- **Linking constraints:** Can only allocate to open facilities
- **Branching:** CPLEX explores different combinations of facility openings

Python Solution:

```
from amplpy import AMPL
import pandas as pd

# Create AMPL instance
ampl = AMPL()
ampl.read("facility.mod")

# Prepare facility data
facilities_df = pd.DataFrame({
    'fixed_cost': [10000, 12000, 15000],
    'capacity': [1000, 1200, 1500]
}, index=['F1', 'F2', 'F3'])
facilities_df.index.name = 'FACILITIES'

# Prepare customer data
customers_df = pd.DataFrame({
    'demand': [200, 300, 250, 400]
}, index=['C1', 'C2', 'C3', 'C4'])
customers_df.index.name = 'CUSTOMERS'

# Prepare transportation costs
transport_df = pd.DataFrame({
    'C1': [5, 7, 9],
    'C2': [8, 4, 6],
    'C3': [6, 7, 4],
    'C4': [9, 5, 3]
}, index=['F1', 'F2', 'F3'])
transport_df.index.name = 'FACILITIES'
transport_df.columns.name = 'CUSTOMERS'

# Send directly to AMPL
# Send data to AMPL
ampl.set_data(facilities_df, 'FACILITIES')
ampl.set_data(customers_df, 'CUSTOMERS')
ampl.param['transport_cost'] = transport_df



# Configure solver for MIP
```

```
ampl.option["solver"] = "cplex"
ampl.option["cplex_options"] = " ".join([
    "mipgap=0.01",       # Accept 1% optimality gap
    "mipemphasis=1",     # Emphasize finding good solutions quickly
    "timelimit=300"      # 5 minute time limit
])

# Solve
ampl.solve()

# Process results
if ampl.solve_result == "solved":
    print(f"Total cost: ${ampl.get_objective('total_cost').value():.2f}")

    # Which facilities are open?
    open_facilities = ampl.get_variable('open').get_values().to_pandas()
    open_facilities = open_facilities[open_facilities['open.val'] > 0.5]
    print(f"\nOpen facilities: {list(open_facilities.index)}")

    # Customer allocations
    allocations = ampl.get_variable('allocate').get_values().to_pandas()
    allocations = allocations[allocations['allocate.val'] > 0.01]
    print("\nCustomer allocations:")
    print(allocations)
else:
    print(f"Optimization failed: {ampl.solve_result}")
```

Expected Results:

```
CPLEX 22.1.2: optimal solution; objective 18350
Total cost: $18350.00
Open facilities: ['F2']
Customer allocations:
              allocate.val
index0 index1
F2     C1                1
       C2                1
       C3                1
       C4                1
```

Interpreting MIP Gap:

The MIP gap indicates solution quality:

- Gap = 0.01 (1%): Solution is within 1% of proven optimal
- Gap = 0.00: Proven optimal solution
- Larger gaps: May want to increase time limit or adjust solver parameters

Example 3: Large-Scale Problem - Network Optimization

Performance Considerations:

For problems with millions of variables and constraints, consider:

- **Memory management:** Set appropriate work memory and tree limits

```
ampl.option["cplex_options"] = "treememory=8000 nodefile=2"
```

- **Output control:** Reduce display frequency to avoid I/O bottlenecks

```
ampl.option["cplex_options"]= "mipdisplay=3 mipinterval=1000"
```

- **Result management:** Write solutions to files instead of displaying

```
ampl.option["cplex_options"] = "writesolution=solution.sol"
```

VI. Troubleshooting & Best Practices

Common Issues & Solutions

Installation Failures

- **Problem:** pip install fails with permission errors
  - **Solution:** Use `--user` flag or create virtual environment
- **Problem:** Module download fails
  - **Solution:** Check internet connection and proxy settings
- **Problem:** Platform-specific binary not available
  - **Solution:** Use Portal download method for your specific platform

License Activation Problems

- **Problem:** Invalid UUID error
  - **Solution:** Verify UUID is correctly copied (no extra spaces or characters)
- **Problem:** License expired
  - **Solution:** Contact AMPL support to renew or upgrade license
- **Problem:** License limits exceeded
  - **Solution:** Check problem size against license limitations; consider upgrading

Memory and Performance Issues

- **Problem:** Out of memory errors
  - **Solution:** Increase work memory limit and enable node files:

```
ampl.option["cplex_options"] = "treememory=16000 nodefile=2"
```

- **Problem:** Slow solve times
  - **Solution:** Use tuning tool or adjust emphasis parameters

```
ampl.option["cplex_options"] = "tunefile=results.prm tunetimelimit=3600"
```

- **Problem:** Numerical instability
  - **Solution:** Scale problem coefficients, tighten tolerances, or use barrier method

Best Practices

Model Organization and File Structure

- Use descriptive names for sets, parameters, and variables
- Add comments to explain model formulation and assumptions
- Organize related models in project directories

Version Control

- Use Git or similar version control for model files
- Track changes to model formulation over time
- Document major model revisions in commit messages

Documentation Standards

- Maintain a README file describing model purpose and usage
- Document parameter units and valid ranges
- Keep a changelog of model modifications
- Include example data files for testing

Performance Benchmarking

- Record solve times for standard test cases
- Monitor solution quality metrics (gap, objective value)
- Test model scalability with increasing problem sizes

- Compare different solver option configurations

VII. Resources & Next Steps

Official Documentation

- **AMPL Documentation:** https://ampl.com/resources/
- **AMPL Book (Free PDF):** https://ampl.com/resources/the-ampl-book/
- **AMPLpy Documentation:** https://amplpy.readthedocs.io/
- **CPLEX Documentation:** https://www.ibm.com/docs/en/icos/22.1.2
- **AMPL CPLEX Documentation:** https://dev.ampl.com/solvers/cplex/index.html

Community & Support

- **AMPL Forum:** Ask questions and share knowledge with the AMPL community
- **AMPL Support:** support@ampl.com for technical assistance
- **Example Models:** Browse AMPL model repository for real-world examples: https://colab.ampl.com/

Advanced Topics to Explore

- **Stochastic Programming:** Model uncertainty with scenarios
- **Robust Optimization:** Optimize under worst-case scenarios
- **Multi-Objective Optimization:** Balance competing objectives
- **Decomposition Methods:** Benders, Dantzig-Wolfe for large-scale problems
- **Custom Callbacks:** Advanced solver control for specialized algorithms

Additional Resources

- **AMPL YouTube Channel:** Video tutorials and webinars

End of the CPLEX Implementation Roadmap

For additional support, contact AMPL Optimization