Optimizing Design, Planning and Operation Problems with Discrete Decisions Using Integer Programming

Jim Luedtke

Industrial & Systems Engineering, Wisconsin Institute for Discovery
University of Wisconsin–Madison

MaGIC 2018
Mathematical optimization

Components of a mathematical optimization model

1. **Decision variables**: values we are trying to find
2. **Objective function**: what we want to maximize or minimize
3. **Constraints**: limits on the allowed values of the decision variables

Constraints and objective function are usually written using algebraic functions of the decision variables

**Optimization problem**

Find the values of the decision variables that satisfy the constraints, and give the *best* value of the objective function.
Example: Economic dispatch of power

Power Grid Operation Problem

- Generators are connected to customers via power grid
- How much power to produce from each generator to meet customer demands at minimum cost?
Example: Economic dispatch of power

Decision variables:
- $x_i$: Amount of power to produce from generator $i$
- $\theta_i$: Voltage at bus $i$ in the network
- $f_{ij}$: Flow on line $ij$ in the network

Objective function:
- Minimize sum of generation costs

Constraints (all can be written as linear equations and inequalities):
- Net flow out/in of each node equals its supply/demand (Kirchoff’s first law)
- Flow on a line is proportional to voltage difference of endpoints
- Thermal limits of power flow on each line ($f_{ij} \leq U_{ij}$)
Continuous optimization models

- Decision variables can take on any fractional values
- E.g., flow on a power line can be 2.4325

Solution methods depend on the form of the objective function and constraints

- Best case scenario: all are linear functions of the decision variables ⇒ Linear Programming
- Runner-up: all functions are convex ⇒ Convex Programming
- HUGE linear programming problems (millions of decision variables and constraints) can be solved routinely with modern software
Mixed-integer programming

Many design, planning, and operational problems involve *discrete* decisions

- Turn a generator on or not?
- How many chillers to include in a chemical processing plant?
- Build a warehouse at a candidate location or not?

These types of decisions can be modeled with **mixed-integer programming**

- An optimization model in which some variables are required to take on integer values

\[
\begin{align*}
\min_{x=(x_1,x_2,\ldots,x_n)} & \quad f(x) \\
\text{subject to:} & \quad Ax \leq b \\
& \quad x_j \in \mathbb{Z}, \quad j \in I
\end{align*}
\]

Most important case: Binary variables, \( x_j \) is 0 or 1 ⇒ Can be used to model yes/no decisions
Examples of yes/no decisions

- Facility location: $x_j = 1$ if build a facility in location $j$
- Vehicle routing: $x_{ij} = 1$ if travel from $i$ to $j$ in a route
- Portfolio optimization with fixed transaction costs: $x_i = 1$ if invest in asset $i$
- Sparse regression: $x_i = 1$ if use feature $i$ in regression model
- Machine assignment: $x_{ij} = 1$ if machine $i$ does job $j$
- Network design: $x_{ij} = 1$ if location $i$ and location $j$ are connected
- Scheduling: $x_{it} = 1$ if task $i$ starts at (discrete-time) $t$
Economic dispatch revisited

Suppose turning on a generator incurs a fixed cost – How to model this cost in objective function?

- Add new binary variable: $y_j = 1$ if generator $j$ is switched on
- Add $\text{FixedCost}_j \times y_j$ to the objective function
- Add constraints: $x_j \leq U_j y_j$ (if $y_j = 0$, produce nothing)

Binary variables also useful for modeling “weird’ physical constraints

- E.g., if a generator is switched on, it must produce at least $L_j$ units
- Use same model as above, but add: $x_j \geq L_j y_j$

Binary variables can also be used to model “switching” lines (i.e., disconnecting them)

- Changes the physics constraints that are used
Scheduling using binary variables

Unit commitment problem

- Given a set of time periods $t = 1, \ldots, T$ (e.g., next 168 hours) plan when all generators should be turned on and off

Binary decision variables:

- $y_{jt} = 1$ if generator $j$ is on during period $t$
- $z_{jt} = 1$ if generator $j$ “is switched on” at beginning of period $t$

Can then model minimum up/down constraints:

- When a generator is turned on, it must stay on a minimum number of periods
- When a generator is turned off, it must stay off a minimum number of periods
Consider an integer program with \( n \) binary variables

- “Simple” algorithm: Just try all \( 2^n \) possible solutions

Suppose we can check 1 billion solutions per second:

- \( n = 20 \implies \) enumerate all solutions in 0.001 seconds
- \( n = 30 \implies 1 \) second
- \( n = 50 \implies 13 \) days
- \( n = 60 \implies 37 \) years
- \( n = 75 \implies 1,197,962 \) years
- \( n = 100 \implies 40,196,936,841,332 \) years

Real problems have thousands of binary decision variables \( \Rightarrow \) We need smarter algorithms
Another approach: Heuristics

Heuristic Algorithm
An algorithm that is designed to find a good solution, but which has no guarantee that it will find the best solution, or even find a feasible solution at all.

Popular examples:
- Genetic algorithms
- Tabu search
- Simulated annealing

Pros and Cons of Heuristic Algorithms
+ Typically run fast enough, and in many cases can find near-optimal solutions
- Heuristics need to be specialized for each different problem type
- Might give bad solutions (or none at all), and do not provide any information on how bad your solution might be
Exact algorithms

Exact Algorithm

An algorithm that is guaranteed to find a true optimal solution for any problem instance.

Modern mixed-integer programming software implements exact algorithms for solving general MIP problems

- Any problem that can be written with mix of integer and continuous variables, and linear (or even convex quadratic) constraints and objective
- Algorithm does not need to be specialized to the problem

Key question: Does an exact algorithm find an optimal solution fast enough? (e.g., not true for enumeration)
Exact algorithms

Many special types of mixed-integer programming problems can be solved exactly very efficiently with specialized algorithms

- Shortest path between two nodes in a network
- Maximum weight matching (selection of pairs) of nodes in a network
- Minimum cost assignment of workers to tasks

Commercial MIP solvers: CPLEX, Gurobi, Xpress

- Capable of solving general MIP models with thousands of decision variables and constraints, within seconds or minutes
- Dark magic?
Ingredient 1 of Dark magic: Branch-and-bound

Linear programming relaxation

- If we ignore the integer restrictions, MIP becomes a linear program ⇒ Can be solved very quickly
- More feasible solutions in LP ⇒ Optimal value at least as good as MIP optimal value

After solving LP relaxation:

- If optimal solution satisfies integer restrictions ⇒ It’s optimal to MIP!
- Otherwise, some integer variable has a fractional value in LP solution (e.g., $x_j = 0.5$)
- Then **branch**: Create two new subproblems
  - In one: Add constraint $x_j \leq 0$
  - In other: Add constraint $x_j \leq 1$
- Every solution to MIP is feasible to one of the two subproblems (But not the LP solution!)
Branch-and-bound (cont’d)

Process of dividing solution set is applied recursively

- Creates a list of subproblems: Every solution is feasible to one of them
- If repeated like this, it is just enumeration!

**Dark Magic 1: Bounding**

- Every time we find a feasible solution, obtain an *upper bound* on best known solution
- For every subproblem, solve the LP relaxation ⇒ *lower bound*
- If lower bound > best upper bound ⇒ No need to subdivide!

“Good” bounding ⇒ Avoid explicitly checking almost all solutions

- Good ≡ LP relaxation value ≈ true optimal value
- Unfortunately, this isn’t always true!
Cutting planes are constraints added to a MIP problem that do not change the set of solutions, but cut off solutions from the LP relaxation.

- Theoretically, mixed-integer program can be solved as a linear program with the right cutting planes added.
- In practice, cutting planes improve linear program bound ⇒ Explore (far) fewer subproblems.
- Finding cutting planes is challenging ⇒ Decades of research.
- MIP solvers have leveraged these results very effectively.
Limitation of dark magic

MIP solvers work extremely well for mixed-integer linear problems

- Objective function and all constraints are linear functions of decision variables
- Limited types of nonlinearity also OK (e.g., convex, separable)

Definitely not always true!

- But, with clever modeling, many problems that do not seem linear can be formulated (or closely approximated) with linear constraints
For arbitrary function of one variable:
- Approximate it as a piecewise-linear function
- Use binary variables to choose which “piece” argument lies in
- Can then model function as linear function of new variables
Choosing a discrete pipe sizes

- **a**: Decision variable represents area of pipe
- **r**: Decision variable represents radius of pipe
- Binary variables to choose among discrete types:
  - e.g., \( y_1 = 1 \) if radius = 4, \( y_2 = 1 \) if radius = 5, \( y_3 = 1 \) if radius = 6
  - Choose one size: \( y_1 + y_2 + y_3 = 1 \)
- Radius equation: \( r = 4y_1 + 5y_2 + 6y_3 \)

Nonlinear area equation!

\[
a = \pi r^2
\]

Linear reformulation:

\[
a = \pi(16y_1 + 25y_2 + 36y_2)
\]
Another example problem MIP can help solve

Current Kaggle Competition: Traveling Santa Problem
- Santa has to visit every house in his list (198,000)
- What sequence should Santa visit them in to minimize total distance?
- Same type of problem appears in many real routing problems (UPS, chip manufacturing,...)

Formulation as MIP?
- Binary decision variables: $x_{ij} = 1$ if city $i$ immediately follows city $j$
- Objective is then linear function (sum over all possible connections)
- Linear constraints ensure every city visited exactly once, and no "subtours"
- Competition includes an extra ‘tricky’ cost to make it harder(!!)
<table>
<thead>
<tr>
<th>#</th>
<th>Team Name</th>
<th>Kernel</th>
<th>Team Members</th>
<th>Score</th>
<th>Entries</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prime Mover</td>
<td></td>
<td></td>
<td>1514733.92</td>
<td>3</td>
<td>7h</td>
</tr>
<tr>
<td>2</td>
<td>KMCoders</td>
<td></td>
<td></td>
<td>1515158.25</td>
<td>28</td>
<td>14h</td>
</tr>
<tr>
<td>3</td>
<td>KaizaburoChubachi</td>
<td></td>
<td></td>
<td>1515214.02</td>
<td>11</td>
<td>14h</td>
</tr>
<tr>
<td>4</td>
<td>Farmers peeing further</td>
<td></td>
<td></td>
<td>1515299.37</td>
<td>33</td>
<td>1d</td>
</tr>
<tr>
<td>5</td>
<td>ultragamza</td>
<td></td>
<td></td>
<td>1515338.43</td>
<td>18</td>
<td>1h</td>
</tr>
</tbody>
</table>
Tips for using mixed-integer programming

Use a good solver
- CPLEX, Gurobi, Xpress, SCIP
- Try them out free at neos-server.org

Use a good modeling interface
- Describes model to solver, connects your data
- AMPL, GAMS, JuliaOpt, PuLP (Python), Gurobi (Python)

Use a good modeler
- Hire one of our graduates!
- Or talk to us about collaborations (us=Me, Laura Albert, Alberto Del Pia, Jeff Linderoth, Carla Michini)

Just try it!
Questions?

jim.luedtke@wisc.edu