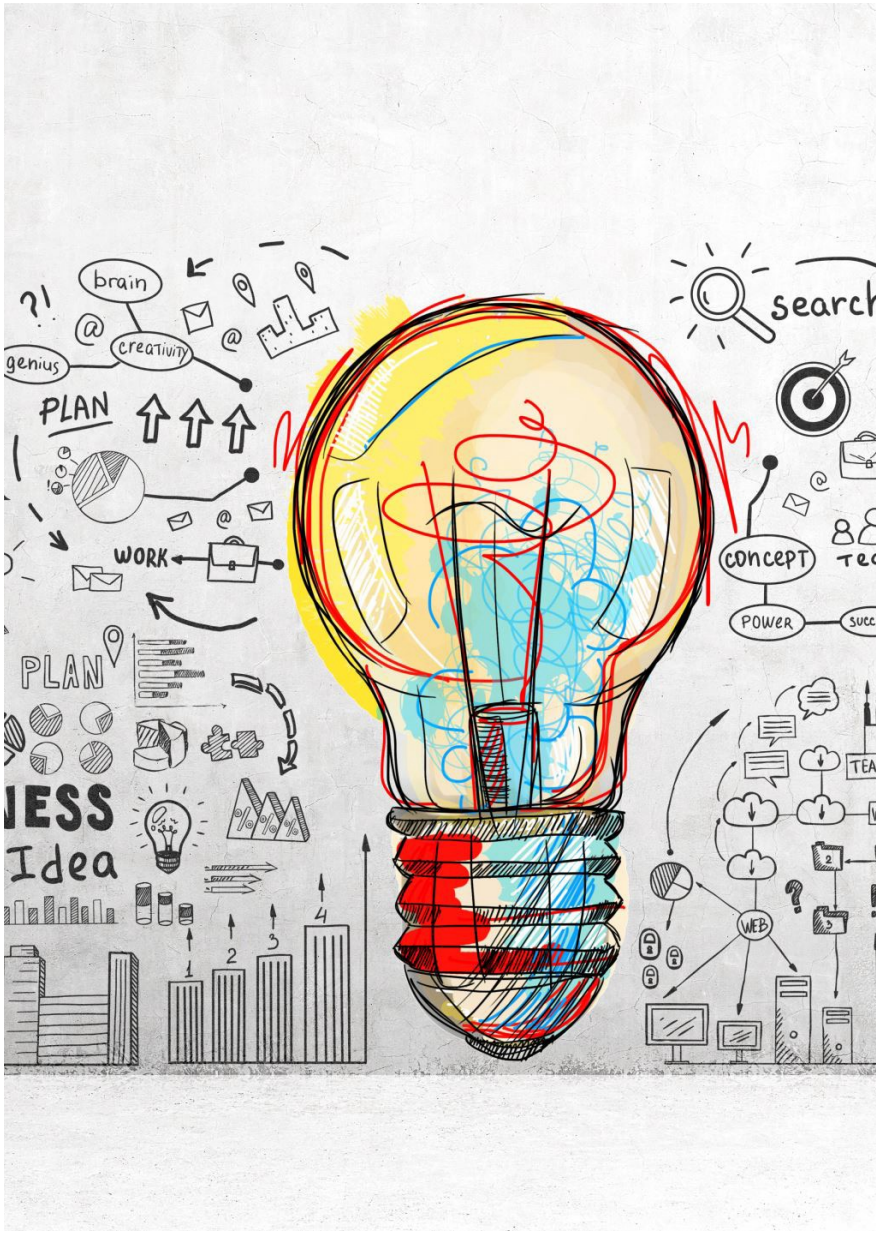


DEVELOPING OPTIMAL SOLUTIONS FOR ORGANIZATIONAL AND BUSINESS NEEDS USING OR (OPERATIONS RESEARCH) AND AI (ARTIFICIAL INTELLIGENCE)

DAY 1: Introduction to optimization using Linear Programming, Integer Programming and Constraint Programming

Prof. Christos G. Gkogkos

University of Ioannina, Dept. of Informatics and Telecommunications

https://github.com/chgogos/INTRO_OR_AI

INTRODUCTION TO OPTIMIZATION

- Descriptive analytics (Statistics)
- Predictive analytics (A.I.=Artificial Intelligence)
- Prescriptive analytics (O.R.=Operations Research)

O.R. and A.I. are complementary. Predictive models can inform prescriptive models, and that combination is where the real power lies

https://doi.org/10.1287/orms.2022.04.14

MATHEMATICAL PROGRAMMING

- Problem description → Mathematical Modeling → Mathematical Programming → Solve using solver (e.g., Gurobi, IBM ILOG CPLEX, Google OR-Tools, HIGHS, …)→ Evaluate results → Reiterate
- Mathematical Programming is a branch of O.R. (Operations Research)

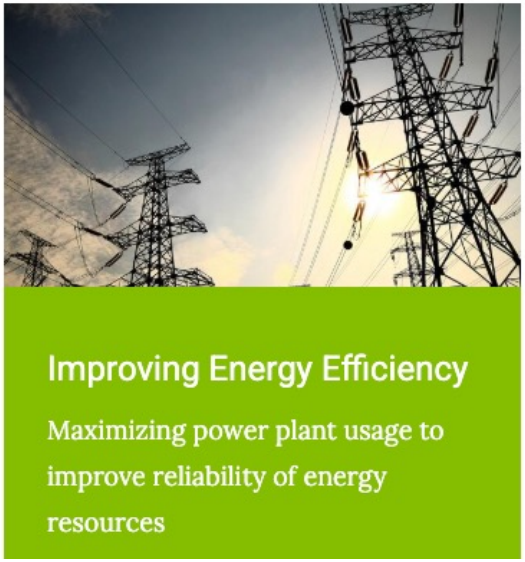
State of Mathematical Optimization Report, 2021

https://cdn.gurobi.com/wp-content/uploads/2022/08/Report-StateOfMathematicalOptimizationReport.pdf

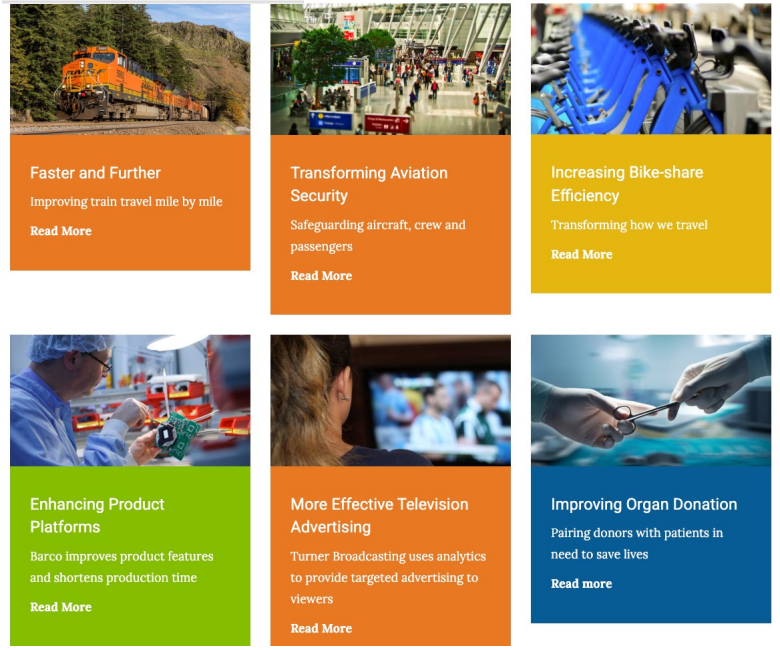
85% of Fortune 500 companies use optimization

O.R. & ANALYTICS SUCCESS STORIES

INFORMS = Institute for Operations Research and the Management Sciences



informs Smarter Decisions for a Better World https://www.informs.org/Impact/O.R.-Analytics-Success-Stories



A SIMPLE EXAMPLE OF MODELING: INVESTMENT OPPORTUNITIES

An amount of 2,000,000 € will be allocated across four investment opportunities, labelled A, B, C, and D. A minimum of 200,000 € and a maximum of 1,000,000 € must be invested in each opportunity. The combined investment in A and B must not exceed 1,500,000 €. The projected returns are 9%, 11%, 5%, and 8% for opportunities A, B, C, and D, respectively.

$$\text{Maximize } Z = 0.09x_A + 0.11x_B + 0.05x_C + 0.08x_D$$

s.t.

$$x_A + x_B + x_C + x_D = 2{,}000{,}000 \quad (1)$$

$$x_A \geq 200{,}000 \quad (2)$$

$$x_B \geq 200{,}000 \quad (3)$$

$$x_C \geq 200{,}000 \quad (4)$$

$$x_D \geq 200{,}000 \quad (5)$$

$$x_A \leq 1{,}000{,}000 \quad (6)$$

$$x_B \leq 1{,}000{,}000 \quad (7)$$

$$x_C \leq 1{,}000{,}000 \quad (8)$$

$$x_D \leq 1{,}000{,}000 \quad (9)$$

$$x_A + x_B \leq 1{,}500{,}000 \quad (10)$$

SOLUTION FOR THE SIMPLE INVESTMENT EXAMPLE USING OR-TOOLS

```python
from ortools.linear_solver import pywraplp

solver = pywraplp.Solver.CreateSolver('SCIP')

# Decision variables
x_A = solver.NumVar(200_000, 1_000_000, 'x_A')
x_B = solver.NumVar(200_000, 1_000_000, 'x_B')
x_C = solver.NumVar(200_000, 1_000_000, 'x_C')
x_D = solver.NumVar(200_000, 1_000_000, 'x_D')

# Constraints
solver.Add(x_A + x_B + x_C + x_D == 2_000_000)

solver.Add(x_A + x_B <= 1_500_000)

# Objective
solver.Maximize(0.09 * x_A + 0.11 * x_B + 0.05 * x_C + 0.08 * x_D)

# Solve
status = solver.Solve()

if status == pywraplp.Solver.OPTIMAL:
    print(f'Optimal solution found:')
    print(f'x_A = {x_A.solution_value():,.2f}')
    print(f'x_B = {x_B.solution_value():,.2f}')
    print(f'x_C = {x_C.solution_value():,.2f}')
    print(f'x_D = {x_D.solution_value():,.2f}')
    print(f'Maximum Return = {solver.Objective().Value():,.2f}')
else:
    print('The problem does not have an optimal solution.')
```

```
Optimal solution found:
x_A = 500,000.00
x_B = 1,000,000.00
x_C = 200,000.00
x_D = 300,000.00
Maximum Return = 189,000.00
```

PYTHON BASICS AND BEYOND!

- Lists, sets, tuples, dictionaries
- Iterations over lists & dictionaries, enumerate
- Functions
- Assertions
- OOP (classes, methods)
- Comprehensions
- f-strings
- lambdas
- Read data from text files, write data to text files
- Measure execution time of code
- Debugging

- https://learnxinyminutes.com/python/
- Python As Fast as Possible - Learn Python in ~75 Minutes - https://www.youtube.com/watch?v=VchuKL44s6E
- https://marko-knoebl.github.io/slides/python-all-en.html
- https://programming-25.mooc.fi/

PACKAGE MANAGERS + VIRTUAL ENVIRONMENTS

- UV: modern high performance Python package manager (10-100x faster than pip)
- UV can act as a replacement for the traditional tools pip and virtualenv (also for conda, poetry)
- UV provides better cross-platform consistency than its peers
- UV installation instructions: https://docs.astral.sh/uv/getting-started/installation/

```
$ uv init prj1
Initialized project `prj1` at `/Users/a_user/dev_git_repos/prj1`
$ cd prj1
$ tree -a
.
├── main.py
├── pyproject.toml
├── README.md
└── uv.lock
$ uv add ortools
Using CPython 3.11.11
Creating virtual environment at: .venv
Resolved 11 packages in 682ms
Prepared 3 packages in 6.94s
Installed 10 packages in 84ms
 ...
+ ortools==9.12.4544
 ...
```

main.py

```python
import sys
import ortools

# Display Python version
print(f"Python version: {sys.version}")

# Display OR-Tools version
print(f"OR-Tools version: {ortools.__version__}")

```

```
$ uv run main.py
Python version: 3.11.11 (main, Dec 19 2024,
14:23:18) [Clang 18.1.8 ]
OR-Tools version: 9.12.4544
```

EXAMPLE: SEND + MORE = MONEY

```python
from ortools.sat.python import cp_model

model = cp_model.CpModel()

s = model.new_int_var(1, 9, "S")  # cannot be 0
e = model.new_int_var(0, 9, "E")
n = model.new_int_var(0, 9, "N")
d = model.new_int_var(0, 9, "D")
m = model.new_int_var(1, 9, "M")  # cannot be 0
o = model.new_int_var(0, 9, "O")
r = model.new_int_var(0, 9, "R")
y = model.new_int_var(0, 9, "Y")

letters = [s, e, n, d, m, o, r, y]

model.add_all_different(letters)

# SEND + MORE = MONEY
send = 1000 * s + 100 * e + 10 * n + d
more = 1000 * m + 100 * o + 10 * r + e
money = 10000 * m + 1000 * o + 100 * n + 10 * e + y

model.add(send + more == money)

solver = cp_model.CpSolver()
status = solver.solve(model)

if status == cp_model.FEASIBLE or status == cp_model.OPTIMAL:
    print(f"S = {solver.value(s)}")
    print(f"E = {solver.value(e)}")
    print(f"N = {solver.value(n)}")
    print(f"D = {solver.value(d)}")
    print(f"M = {solver.value(m)}")
    print(f"O = {solver.value(o)}")
    print(f"R = {solver.value(r)}")
    print(f"Y = {solver.value(y)}")
    print()
    print(f"{solver.value(send)} + {solver.value(more)} = {solver.value(money)}")
```

```
$ uv run cryptarithmetic.py
S = 9
E = 5
N = 6
D = 7
M = 1
O = 0
R = 8
Y = 2

9567 + 1085 = 10652
```

```
  SEND
+ MORE
= MONEY

  9567
+ 1085
= 10652
```

How about:

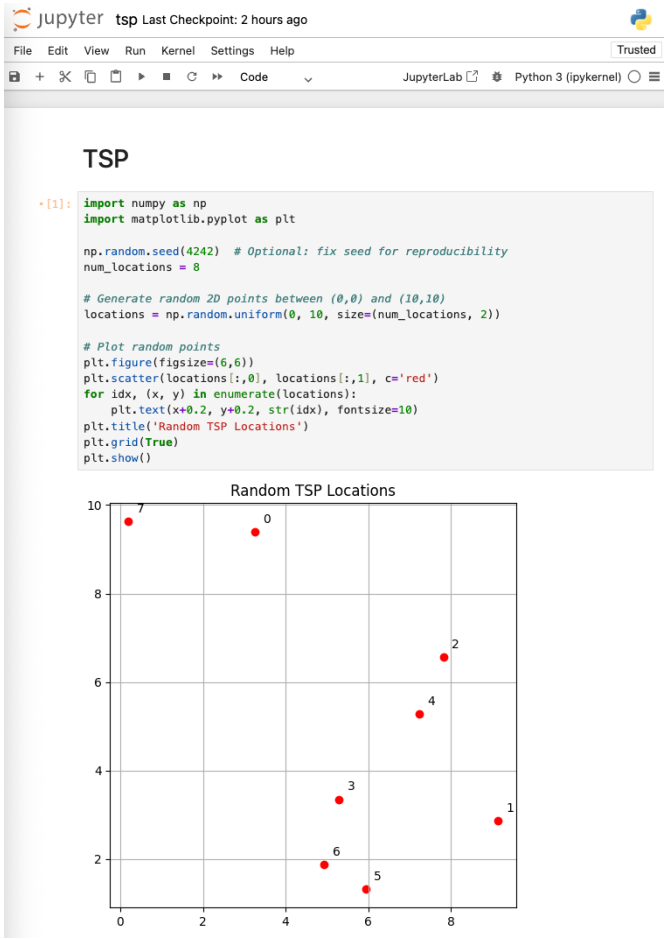
TWO + TWO = FOUR

?

JUPYTER NOTEBOOK

- Jupyter notebook is a popular tool for interactive computing, data analysis, and visualization

```
$ uv init prj2
Initialized project `prj2` at `/Users/a_user/dev_git_repos/prj2`
$ cd prj2
$ uv add jupyter ortools matplotlib numpy
...
$ uv run jupyter notebook
[I 2025-04-28 11:50:20.242 ServerApp] http://localhost:8888/tree?token=c96782dbcf41a97121a3bb5382da454b1a6d2375da26e0e7
[I 2025-04-28 11:50:20.242 ServerApp] http://127.0.0.1:8888/tree?token=c96782dbcf41a97121a3bb5382da454b1a6d2375da26e0e7
[I 2025-04-28 11:50:20.242 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
...
```



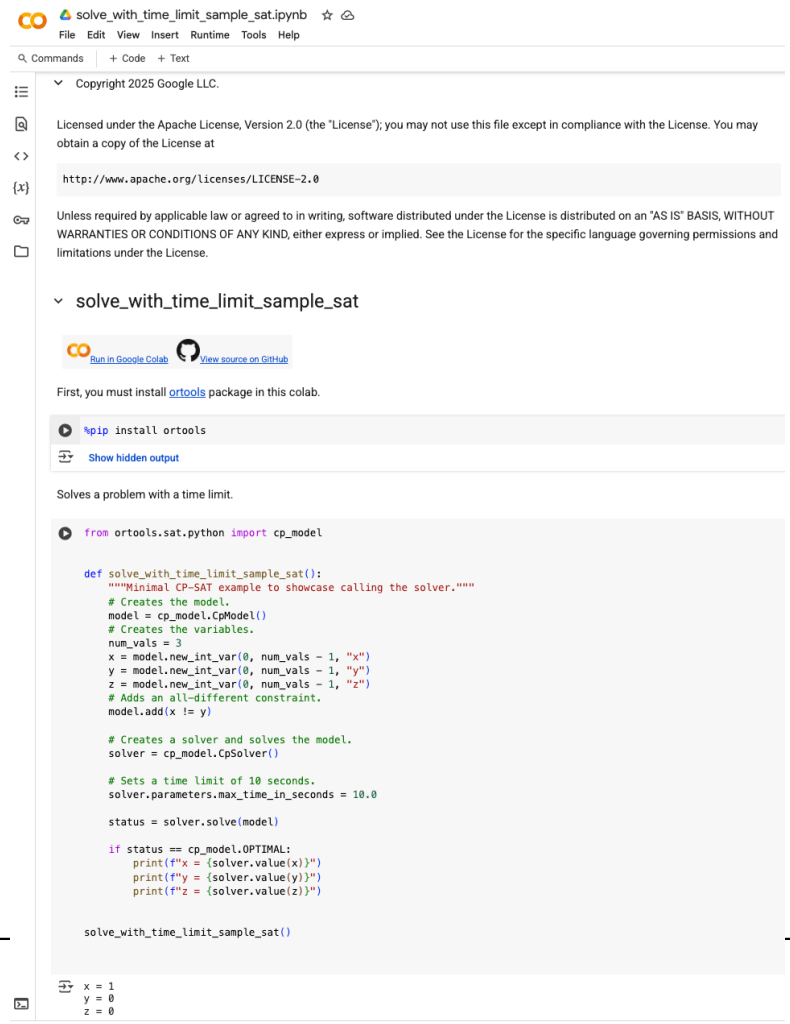
VSCODE + PYTHON

- https://marko-knoebl.github.io/slides/python-beginner-collection-en.html - VS Code for Python

GOOGLE COLAB

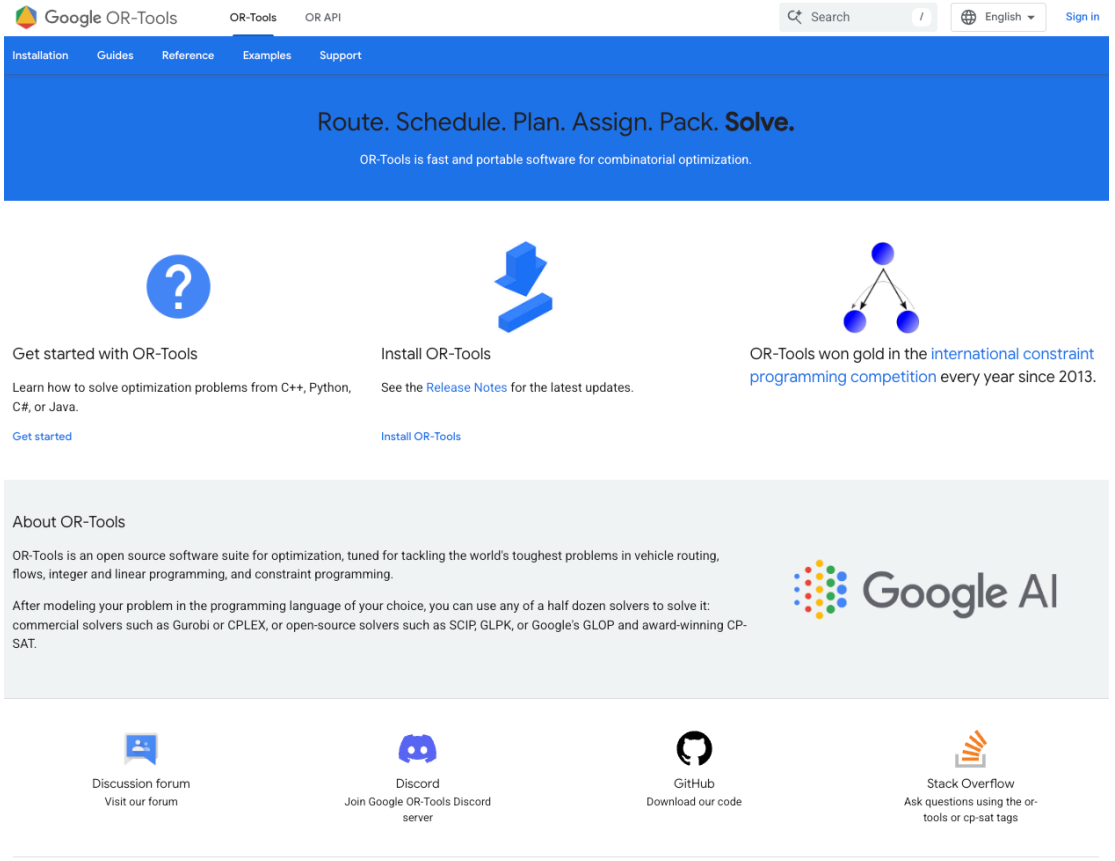
- Google's Colab is a cloud based Jupyter Notebook service that requires no setup to use and provides **free access to computing resources**
- Google colab notebooks can be saved into Google Drive

https://colab.research.google.com/github/google/or-tools/blob/stable/examples/notebook/sat/solve_with_time_limit_sample_sat.ipynb



GOOGLE ORTOOLS - RESOURCES

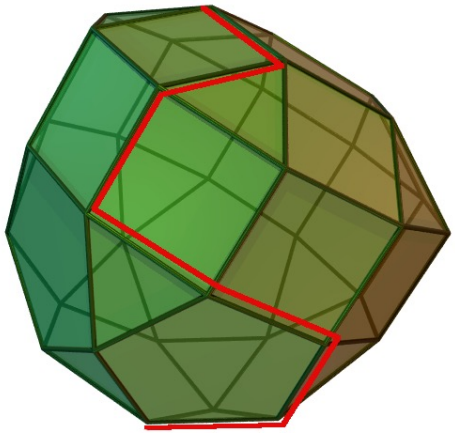
- https://developers.google.com/optimization
- https://github.com/or-tools/awesome_or-tools
- https://d-krupke.github.io/cpsat-primer/
- https://kunlei.github.io/python-ortools-notes/



INTRODUCTION TO OPTIMIZATION USING LINEAR PROGRAMMING, INTEGER PROGRAMMING AND CONSTRAINT PROGRAMMING

Model and solve decision support problems using Google's OR-Tools and Python

LINEAR PROGRAMMING



- Linear Programming (LP) is used a lot in practice
- LP entails formulation (modeling) and then solving of linear programs
- Basic elements of a Linear Program:
  - **Decision variables** (assume continuous values)
  - **Constraints** (functions should be linear over decision variables)
  - **Objective function** (either maximize or minimize of a linear function over the decision variables)
- The general formulation of a mathematical program:

$$\begin{aligned} \max \quad & f(x_1, x_2, \dots, x_n) \\ s.t. \quad & g_i(x_1, x_2, \dots, x_n) \leq b_i \quad \forall i = 1, \dots, m \\ & x_j \in \mathbb{R} \quad \forall j = 1, \dots, n \end{aligned}$$

- Concrete example of a problem instance:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1 + 2x_2 \leq 6 \\ & 2x_1 + x_2 \leq 6 \\ & x_1 \geq 0, x_2 \geq 0 \end{aligned}$$

LINEAR PROGRAMMING – COMPACT LP FORMULATIONS

- Most problem instances, in practice, have big (or huge!) numbers of variables and constraints
- Compact formulations enhance readability by use of **indices** (e.g., i, j, k), the **summation** symbol (Σ) and the **for all** symbol (∀)
- Example of compact formulation:

$$
\begin{aligned}
\min \quad & \sum_{j \in J} c_j x_j \\
\text{s.t.} \quad & \sum_{j \in J} a_{ij} x_j \geq b_i, \quad \forall i \in I \\
& x_j \geq 0, \quad \forall j \in J
\end{aligned}
$$

where x_j are the decision variables for each $j \in J$,
c_j is the cost coefficient for each $j \in J$,
a_{ij} is the coefficient of variable j in constraint i,
b_i is the right-hand side value for constraint i,
I, J are finite index sets.

LINEAR PROGRAMMING – ALGORITHMS & SOLVERS

- Effective algorithms exist for solving LPs:
  - Simplex method (Dantzig, 1947) and its variants, i.e., Revised Simplex, Dual Simplex, Primal-Dual Simplex
  - Interior point methods (Karmarkar, 1984), i.e., Barrier
  - ~~Ellipsoid method (Khachiyan, 1979)~~
  - First-order methods (for large LPs), i.e., Primal-Dual Hybrid Gradient
- Commercial and open-source software provide SoTA implementations of LP solvers:
  - Gurobi
  - IBM ILOG CPLEX
  - FICO XPRESS
  - MOSEK
  - ORTools (GLOP, PDLP, and 3rd party LP solvers)
  - HIGHS (LP solvers)
  - SCIP Optimization Suite (SoPlex)
  - COIN-OR (CLP)
  - GLPK

- The relative performance of algorithms and solvers can vary by orders of magnitude on a single instance.
- No single algorithm and solver is uniformly better than others.
- Crossover (enabled by default) increases solve time, sometimes substantially.
- PDLP can solve to low precision sometimes 10 times faster than to high precision.

https://developers.google.com/optimization/lp/lp_advanced

LP EXAMPLE – DIET PROBLEM (PROBLEM STATEMENT)

You are tasked with planning a daily diet using 5 available food items. The goal is to minimize the total cost while ensuring that the diet provides at least the required daily amounts of 3 essential nutrients: **Calories**, **Protein**, and **Vitamins**. Each food item has a known cost per unit and provides specific amounts of these nutrients. The **nutritional requirements** are at least 1,200 calories per day, at least 50 grams of protein per day and at least 5 grams of vitamins per day.

Food	**Cost per Unit (€)**	**Calories (per unit)**	**Protein (grams per unit)**	**Vitamins (grams per unit)**
Food 1	2	300	5	1
Food 2	1.5	200	10	2
Food 3	3	400	8	3
Food 4	2	250	6	2
Food 5	4	500	12	4

Nutritional Information for Each Food Item

DIET PROBLEM (FORMULATION X 2)

Food	Cost per Unit (€)	Calories (per unit)	Protein (grams per unit)	Vitamins (grams per unit)
Food 1	2	300	5	1
Food 2	1.5	200	10	2
Food 3	3	400	8	3
Food 4	2	250	6	2
Food 5	4	500	12	4

$$\min \quad Z = 2x_1 + 1.5x_2 + 3x_3 + 2x_4 + 4x_5$$

$$\begin{aligned} \text{s.t.} \quad & 300x_1 + 200x_2 + 400x_3 + 250x_4 + 500x_5 \geq 1200 \\ & 5x_1 + 10x_2 + 8x_3 + 6x_4 + 12x_5 \geq 50 \\ & x_1 + 2x_2 + 3x_3 + 2x_4 + 4x_5 \geq 5 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \end{aligned}$$

$$\min \quad Z = \mathbf{c}^T \mathbf{x}$$

$$\text{s.t.} \quad A\mathbf{x} \geq \mathbf{b}$$

$$\mathbf{x} \geq 0$$

where

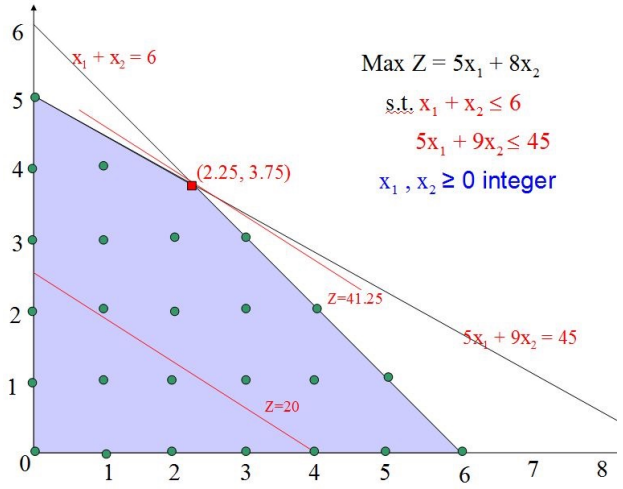
$$\mathbf{c} = \begin{pmatrix} 2 \\ 1.5 \\ 3 \\ 2 \\ 4 \end{pmatrix}, \quad A = \begin{pmatrix} 300 & 200 & 400 & 250 & 500 \\ 5 & 10 & 8 & 6 & 12 \\ 1 & 2 & 3 & 2 & 4 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 1200 \\ 50 \\ 5 \end{pmatrix}$$

DIET PROBLEM (SOLUTION USING ORTOOLS GLOP)

```python
from ortools.linear_solver import pywraplp

# Create the solver
solver = pywraplp.Solver.CreateSolver('GLOP')

# Variables (continuous)
x1 = solver.NumVar(0.0, solver.infinity(), 'x1')
x2 = solver.NumVar(0.0, solver.infinity(), 'x2')
x3 = solver.NumVar(0.0, solver.infinity(), 'x3')
x4 = solver.NumVar(0.0, solver.infinity(), 'x4')
x5 = solver.NumVar(0.0, solver.infinity(), 'x5')

# Objective function
solver.Minimize(2 * x1 + 1.5 * x2 + 3 * x3 + 2 * x4 + 4 * x5)

# Constraints
solver.Add(300 * x1 + 200 * x2 + 400 * x3 + 250 * x4 + 500 * x5 >= 1200)  # Calories
solver.Add(5 * x1 + 10 * x2 + 8 * x3 + 6 * x4 + 12 * x5 >= 50)  # Protein
solver.Add(x1 + 2 * x2 + 3 * x3 + 2 * x4 + 4 * x5 >= 5)  # Vitamins

# Solve the problem
status = solver.Solve()

# Results
solution = {}
if status == pywraplp.Solver.OPTIMAL:
    solution = {
        'x1': x1.solution_value(),
        'x2': x2.solution_value(),
        'x3': x3.solution_value(),
        'x4': x4.solution_value(),
        'x5': x5.solution_value(),
        'objective': solver.Objective().Value()
    }

print(solution)
```

```python
from ortools.linear_solver import pywraplp

# Create the solver
solver = pywraplp.Solver.CreateSolver('GLOP')

# Data (costs, nutritional values)
c = [2, 1.5, 3, 2, 4]  # Cost per unit for each food
A = [
    [300, 200, 400, 250, 500],  # Calories
    [5, 10, 8, 6, 12],          # Protein
    [1, 2, 3, 2, 4]             # Vitamins
]
b = [1200, 50, 5]  # Minimum nutritional requirements

# Variables (continuous)
x = [solver.NumVar(0.0, solver.infinity(), f'x{i+1}') for i in range(5)]

# Objective function: Minimize c^T * x
objective = solver.Objective()
for i in range(5):
    objective.SetCoefficient(x[i], c[i])
objective.SetMinimization()

# Constraints: A * x >= b
for i in range(3):  # 3 constraints
    constraint = solver.Constraint(b[i], solver.infinity())
    for j in range(5):  # 5 foods
        constraint.SetCoefficient(x[j], A[i][j])

# Solve the problem
status = solver.Solve()

# Results
solution = {}
if status == pywraplp.Solver.OPTIMAL:
    solution = {
        'x': [x[i].solution_value() for i in range(5)],
        'objective': solver.Objective().Value()
    }

print(solution)
```

INTEGER PROGRAMMING



- In some problems, variables must only assume **integer values**
- **Integer Programming (IP)** is the process of formulating and solving problems with integer variables
- An IP is actually an ILP (Integer Linear Program)
- **MIP** (Mixed Integer Programming) and **BIP** (Binary Integer Programming) are special cases of IP
- Non-Linear Programs (e.g., using the quadratic function over a decision variable in the objective or the constraints) is another class of problems (NLPs)

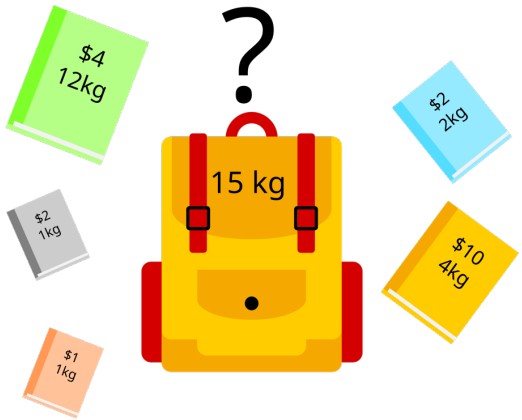
INTEGER PROGRAMMING – ALGORITHMS & SOLVERS

- Effective* algorithms exist for solving IPs:
  - Branch and Bound (B&B) – solves linear relaxations and systematically partitions the problem space (branching) while using bounds (bounding) to prune subproblems
  - Branch and Cut (B&C) – an extension to B&B that adds cutting planes (valid inequalities) during the search to tighten the LP relaxation and speed up convergence
- Commercial and open-source software provide SoTA implementations of IP solvers:
  - Gurobi
  - IBM ILOG CPLEX
  - FICO XPRESS
  - ORTools (uses 3rd party IP solvers such as SCIP and CBC)
  - HIGHs (MIP solver)
  - SCIP Optimization Suite (SCIP)
  - COIN-OR (CBC)
  - GLPK (supports both LP and IP)

*although, less effective than their LP counterparts

IP EXAMPLE: THE 0/1 KNAPSACK PROBLEM

The 0/1 Knapsack Problem is a classic combinatorial optimization problem where, given a set of items each with a weight and a value, the goal is to determine the most valuable subset of items to include in a knapsack without exceeding its weight capacity



$$\begin{aligned} \max \quad & \sum_{i=1}^{n} v_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \{0,1\}, \quad i = 1, 2, \ldots, n \end{aligned}$$

where:

- v_i = value (or profit) of item i,
- w_i = weight of item i,
- W = capacity of the knapsack.

There are special purpose algorithms for the knapsack problem that can solve it efficiently (e.g., Dynamic Programming)

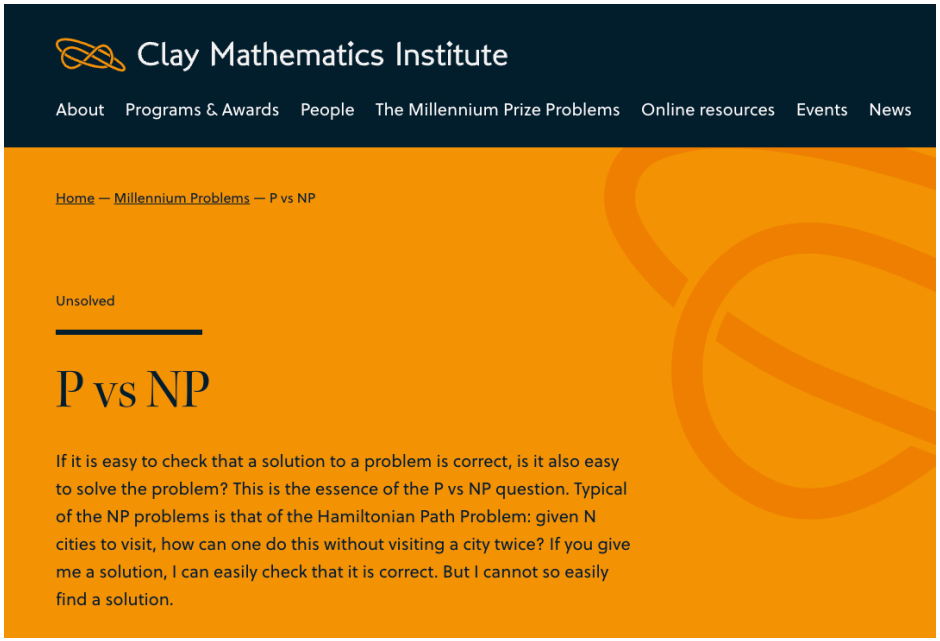
THE 0/1 KNAPSACK PROBLEM – FORMULATION

Item	1	2	3	4	5
Value ($)	4	2	1	2	10
Weight (kg)	12	1	1	2	4

$$
\begin{aligned}
\max \quad & 4x_1 + 2x_2 + x_3 + 2x_4 + 10x_5 \\
\text{s.t.} \quad & 12x_1 + x_2 + x_3 + 2x_4 + 4x_5 \leq 15 \\
& x_i \in \{0, 1\}, \quad \forall i = 1, \ldots, 5
\end{aligned}
$$

```python
from ortools.linear_solver import pywraplp

solver = pywraplp.Solver.CreateSolver("SCIP")

# Binary Decision Variables
x = []
for i in range(5):
    x.append(solver.IntVar(0, 1, f"x{i+1}"))

# Objective function
solver.Maximize(4 * x[0] + 2 * x[1] + x[2] + 2 * x[3] + 10 * x[4])

# Constraint
solver.Add(12 * x[0] + x[1] + x[2] + 2 * x[3] + 4 * x[4] <= 15)

status = solver.Solve()

if status == pywraplp.Solver.OPTIMAL:
    print("Solution:")
    for i in range(5):
        print(f"x{i+1} = {int(x[i].solution_value())}")
    print(f"Maximum objective value = {solver.Objective().Value()}")
else:
    print("No optimal solution found.")
```

P VS NP PROBLEMS



https://www.claymath.org/millennium/p-vs-np/

"I can't find an efficient algorithm, but neither can all these famous people."

An example of an NP problem is the **Subset Sum Problem (SSP)**: Given a set of integers and a target sum T, is there a subset whose sum equals T?

SSP is directly related to the 0/1 knapsack problem

CONSTRAINT PROGRAMMING

"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."

Eugene Freuder, 1996

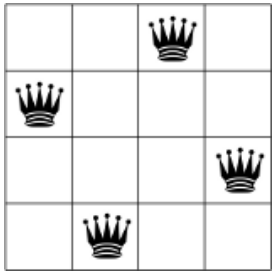
- **Constraint Programming (CP)** is a paradigm for solving combinatorial problems by stating constraints that must be satisfied
- Core idea: The problem is modelled using variables that have finite domains (e.g., $x \in \{1, 2, 3, 4\}$, $y \in \{1 \dots 10\}$, $z = \{2, 4, 6, 8, 10\}$) and constraints (e.g., $3x \neq y$, $x * y = z$)
- The solving process:
  - Constraint propagation (reduce domains by enforcing constraints)
  - Search (explore assignments using backtracking if necessary)
- CP can be used for feasibility problems and for optimization problems
- CP enables a more flexible modelling framework than LP and IP, as it supports general combinatorial and logical constraints beyond linear forms

CONSTRAINT PROGRAMMING – ALGORITHMS & SOLVERS

- Several algorithms have been developed for solving Constraint Programming problems:
  - Arc consistency (e.g., AC-3, AC-4, AC-2001)
  - Global constraint propagators (e.g., all-different, if-then, cumulative)
  - Backtracking search and Backjumping
  - Branch and Bound (B&B)
  - Heuristics (e.g., MRV, LCV)
  - Lazy Clause Generation (LCG)
  - Large Neighborhood Search (LNS)
- Commercial and open-source software provide SoTA implementations of CP solvers:
  - IBM ILOG CPLEX (CP-Optimizer)
  - FICO XPRESS (XPRESS-CP)
  - ORTools (CP, CP-SAT)
  - Choco Solver
  - Gecode
  - Chuffed

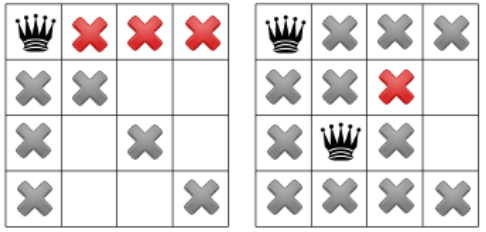
CP EXAMPLE – THE N-QUEENS PROBLEM

How can N queens be placed on an NxN chessboard so that no two of them attack each other?

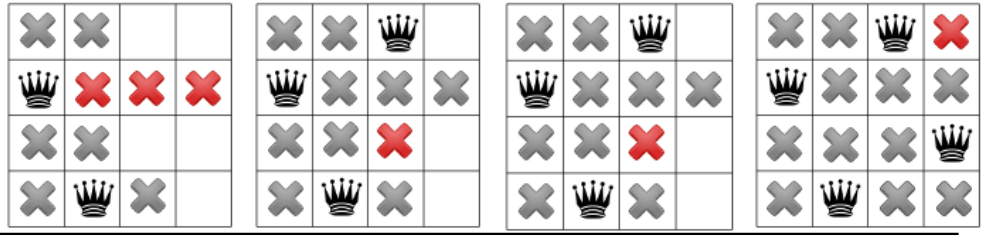


https://developers.google.com/optimization/cp/queens

- N-Queens is not an optimization problem
- Propagation + backtracking



backtrack occurs here!



THE N-QUEENS PROBLEM – FORMULATION

```python
from ortools.constraint_solver import pywrapcp

def main(board_size):
    solver = pywrapcp.Solver("n-queens")

    # Variables: The array index is the column, and the value is the row.
    queens = [solver.IntVar(0, board_size - 1, f"x{i}") for i in range(board_size)]

    # All rows must be different.
    solver.Add(solver.AllDifferent(queens))

    # No two queens can be on the same diagonal.
    solver.Add(solver.AllDifferent([queens[i] + i for i in range(board_size)]))
    solver.Add(solver.AllDifferent([queens[i] - i for i in range(board_size)]))

    db = solver.Phase(queens, solver.CHOOSE_FIRST_UNBOUND, solver.ASSIGN_MIN_VALUE)

    solver.NewSearch(db)
    while solver.NextSolution():
        for i in range(board_size):
            for j in range(board_size):
                if queens[j].Value() == i:
                    print("Q", end=" ")
                else:
                    print("_", end=" ")
            print()
        print()
    solver.EndSearch()

main(8)
```

1 of the 92 solutions for the 8 x 8 board

```
Q _ _ _ _ _ _ _
_ _ _ _ _ _ Q _
_ _ _ _ Q _ _ _
_ _ _ _ _ _ _ Q
_ Q _ _ _ _ _ _
_ _ _ Q _ _ _ _
_ _ _ _ _ Q _ _
_ _ Q _ _ _ _ _
```

A different formulation, that solves the problem using IP, can be found at:
https://kunlei.github.io/python-ortools-notes/ip-nqueens.html

THE N-QUEENS PROBLEM – IMPLEMENTATION USING CP-SAT

```python
from ortools.sat.python import cp_model


def solve(board_size):
    # Create the solver
    model = cp_model.CpModel()

    # Variables: The array index is the column, and the value is the row.
    queens = [model.new_int_var(0, board_size - 1, f"x{i}") for i in range(board_size)]

    # All rows must be different.
    model.add_all_different(queens)

    # No two queens can be on the same diagonal.
    model.add_all_different([queens[i] + i for i in range(board_size)])
    model.add_all_different([queens[i] - i for i in range(board_size)])

    # Solve the model.
    solver = cp_model.CpSolver()
    solver.solve(model)

    # Display the first solution found and terminate.
    all_queens = range(board_size)
    for i in all_queens:
        for j in all_queens:
            if solver.value(queens[j]) == i:
                # There is a queen in column j, row i.
                print("Q", end=" ")
            else:
                print("_", end=" ")
        print()


if __name__ == "__main__":
    solve(8)
```

This code displays only the first solution

```
Q _ _ _ _ _ _ _
_ _ _ _ _ _ Q _
_ _ _ _ Q _ _ _
_ _ _ _ _ _ _ Q
_ Q _ _ _ _ _ _
_ _ _ Q _ _ _ _
_ _ _ _ _ Q _ _
_ _ Q _ _ _ _ _
```

HANDS-ON ACTIVITIES

Resource Allocation

Workforce Planning

Cryptarithmetic puzzle

HANDS-ON – RESOURCE ALLOCATION – THE DIET PROBLEM

1. Find what the "Stigler Diet" is by consulting either the Web or Open AI's ChatGPT
2. Navigate through the OR-Tools web site and **locate the code** that using LP finds the optimal solution for the "Stigler diet"
3. Execute locally the code, using GLOP (i.e., the Linear Programming solver developed as part of ORTools) and another LP solver (e.g., CLP of COIN-OR) and compare the results
4. What would have been the cost of the optimal solution in current money value (you can use https://data.bls.gov/cgi-bin/cpicalc.pl) ?

Extra

- Visit: https://www.feedcalculator.org/
- Read: Dantzig, George B. "The diet problem." Interfaces 20.4 (1990): 43-47.

HANDS-ON – WORKFORCE PLANNING

- Assume that we operate a department store and we know the number of employees needed for each day of the week. Each employee must work for five consecutive days. How many employees should we hire for each one of the 7 shifts:
  - Monday to Friday = ?
  - Tuesday to Saturday = ?
  - Wednesday to Sunday = ?
  - Thursday to Monday = ?
  - Friday to Tuesday = ?
  - Saturday to Wednesday = ?
  - Sunday to Thursday = ?
- The goal is to mininimize the number of employees hired

Mon	Tue	Wed	Thu	Fri	Sat	Sun
90	80	100	60	80	150	140

Tip: Use 7 decision variables, the first one should assume the value of the number of employees that are hired in shift Monday to Friday, the second one should assume the value of the number of employees that are hired in shift Tuesday to Saturday, etc.

HANDS-ON – WORKFORCE PLANNING – FORMULATION

$$
\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \\
\text{s.t.} \quad & x_1 + x_4 + x_5 + x_6 + x_7 \geq 90 \\
& x_1 + x_2 + x_5 + x_6 + x_7 \geq 80 \\
& x_1 + x_2 + x_3 + x_6 + x_7 \geq 100 \\
& x_1 + x_2 + x_3 + x_4 + x_7 \geq 60 \\
& x_1 + x_2 + x_3 + x_4 + x_5 \geq 80 \\
& x_2 + x_3 + x_4 + x_5 + x_6 \geq 150 \\
& x_3 + x_4 + x_5 + x_6 + x_7 \geq 140 \\
x_i \geq 0 \quad & \forall i = 1, \ldots, 7
\end{aligned}
$$

HANDS-ON – CRYPTARITHMETIC PUZZLES

Solve the following cryptarithmetic puzzle:

NINE - FOUR = FIVE

The constraints are the following:

- All letters map to distinct digits
- Leading letters cannot be zero
- The subtraction must hold numerically

Use ORTools CP-SAT to find a single solution (if it exists)

Do you believe that a solution exists for the cryptarithmetic puzzle:

BASE × BALL = GAMES

MANY OTHER OPTIMIZATION APPROACHES EXIST

- Gradient Decent (GD) & Stochastic Gradient Descent (SGD)
- Dynamic Programming (DP)
- Simulated Annealing (SA)
- Tabu Search (TS)
- Genetic Algorithms (GA)
- Particle Swarm Optimization (PSO)
- Ant Colony Optimization (ACO)
- Iterated Local Search (ILS)
- Variable Neighborhood Search (VNS)
- Reinforcement Learning (RL)
- Quantum Annealing (QA)
- Quantum Unconstrained Binary Optimization (QUBO)

REFERENCES

- The Burrito Optimization Game, https://doi.org/10.1287/orms.2022.04.14
- The state of the Mathematical Optimization Report 2021, https://cdn.gurobi.com/wp-content/uploads/2022/08/Report-StateOfMathematicalOptimizationReport.pdf
- CP-SAT Primer, https://d-krupke.github.io/cpsat-primer/
- Freuder, E. C. (1997). In pursuit of the holy grail. Constraints, 2, 57-61.