# Beyond CPLEX and Optim

## A Guide to Optimization Algorithms and Software

Jamie Fairbrother[1]

[1]STOR-i Centre for Doctoral Training, Lancaster University

STOR-i Seminar, 2016-06-02

excellence with impact

# Disclaimer

This talk is not an exhaustive overview.

# Problem Types and Examples

# Mixed Integer Linear Programs (MILPs) I

$$\begin{aligned}
\underset{x \in \mathbb{R}^n}{\text{minimize}}\ & c^T x \\
\text{subject to}\ & Ax \leq b \\
& x_i \in \{0,1\},\ \text{for } i \in \mathcal{F} \\
& x_i \in \mathbb{R}_+,\ \text{for } i \notin \mathcal{F}
\end{aligned}$$

$$\text{where } c \in \mathbb{R}^n,\ A \in \mathbb{R}^{m \times n},\ b \in \mathbb{R}^m$$
$$\mathcal{F} \subseteq \{1, \ldots, n\}$$

**Special cases:**

- Linear Program (LP) when $\mathcal{F} = \emptyset$
- Integer Program (IP) when $\mathcal{F} = \{1, \ldots, n\}$

# Mixed Integer Linear Programs (MILPs) II

**Algorithms:**

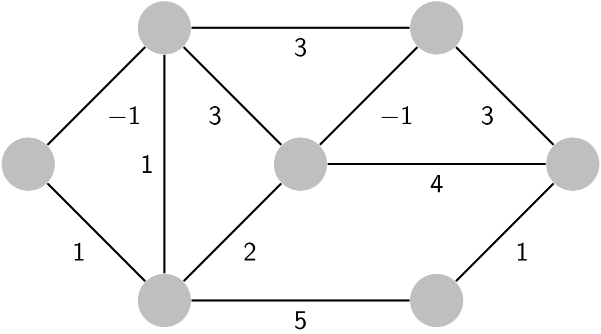
- **LP**: Simplex, interior point
- **MILP**: Branch-and-bound, cutting plane, heuristics

**Problem contexts:**
Knapsack, Networks, Regression, Portfolio Selection, Markov Decision Processes etc.

# Example: $k$-partition problem I

Input:

- An undirected graph $G = (V, E)$
- A (rational) weight $w_e$ for each edge $e \in E$

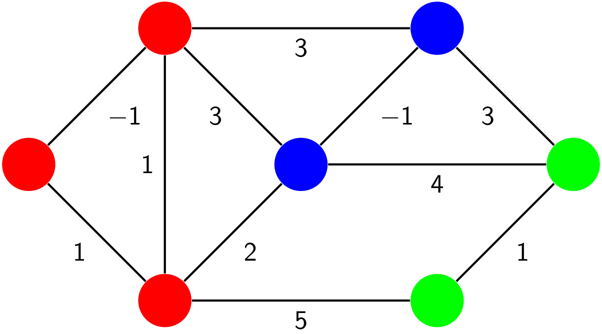



Aim: Partition nodes of a graph into $k$ or fewer clusters such that the total weight of edges whose ends are in the same cluster is minimized

# Example: $k$-partition problem I

Input:

- An undirected graph $G = (V, E)$
- A (rational) weight $w_e$ for each edge $e \in E$




Aim: Partition nodes of a graph into $k$ or fewer clusters such that the total weight of edges whose ends are in the same cluster is minimized

# Example: $k$-partition problem II

IP formulation:

$$
\begin{array}{lrll}
\min & \sum_{e \in E} w_e y_e & & \\
\text{s.t.} & \sum_{c=1}^{k} x_{vc} = 1 & & (v \in V) \\
& y_{uv} \geq x_{uc} + x_{vc} - 1 & & (\{u,v\} \in E,\ c = 1,\ldots,k) \\
& x_{uc} \geq x_{vc} + y_{uv} - 1 & & (\{u,v\} \in E,\ c = 1,\ldots,k) \\
& x_{vc} \geq x_{uc} + y_{uv} - 1 & & (\{u,v\} \in E,\ c = 1,\ldots,k) \\
& x_{vc} \in \{0,1\} & & (v \in V,\ c = 1,\ldots,k) \\
& y_{uv} \in \{0,1\} & & (\{u,v\} \in E).
\end{array}
$$

$$
\text{where } x_{vc} = \begin{cases} 1 & \text{if node } v \text{ assigned to subset c} \\ 0 & \text{otherwise} \end{cases}
$$

$$
y_{uv} = \begin{cases} 1 & \text{if nodes } u,\ v \text{ in same subset} \\ 0 & \text{otherwise} \end{cases}
$$

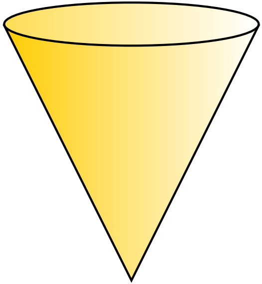
# Example: $k$-partition problem III

**Applications:**

- VLSI circuit design (Barahona et al 1988)
- Statistical physics (Barahona et al 1988)
- Scheduling (Carlson and Nemhauser 1966)
- Telecommunications (Resend and Pardalos 2008)
- Numerical linear algebra
- Statistical clustering/data mining

# Second-order cone programs (SOCPs) I

$$
\begin{aligned}
\underset{x}{\text{minimize}}\ & c^T x \\
\text{subject to}\ & \\
\|A_i x + b_i\|_2 &\le c_i^T x + d_i \qquad \text{for } i = 1, \ldots, m \\
Fx &\le g
\end{aligned}
$$



# Second-order cone programs (SOCPs) II

**Special cases:**

- Quadratic Programs (QP)

$$\begin{aligned} \underset{x \geq 0}{\text{minimize}} \ & \frac{1}{2} x^T P_0 x + q_0^T x + r_0 \\ \text{subject to } & Ax \leq b \end{aligned}$$

- Quadratically Constrained Quadratic Programs (QCQP)

$$\begin{aligned} \underset{x}{\text{minimize}} \ & \frac{1}{2} x^T P_0 x + q_0^T x + r_0 \\ \text{subject to } & \frac{1}{2} x^T P_i x + q_i^T x + r_i \leq 0 \text{ for } i = 1, \dots, p. \end{aligned}$$

where $q_i \in \mathbb{R}^n$ and $P_i \in \mathbb{S}^n$ are positive semidefinite (PSD)

# Second-order cone programs (SOCPs) III

**Extension:** Mixed integer second-order cone program (MISOCP)

$$
\begin{aligned}
&\underset{x}{\text{minimize}}\ c^T x \\
&\text{subject to} \\
&\|A_i x + b_i\|_2 \leq c_i^T x + d_i \qquad \text{for } i = 1, \ldots, m \\
&Fx \leq g \\
&x_i \in \{0, 1\},\ \text{for } i \in \mathcal{F} \\
&x_i \in \mathbb{R},\ \text{for } i \notin \mathcal{F}
\end{aligned}
$$

which has as special cases mixed integer quadratic programs (MIQP) and mixed integer quadratically constrained quadratic programs (MIQCQP).

**Algorithms**
QP: quadratic simplex, conjugate gradient
QCQP/SOCP: interior point (homogenous self-dual method)

**Applications**
Portfolio selection (Markowitz model), robust regression, robust linear programming, maxima/sums of Euclidean norms (e.g facility location)

# Example: Sparse least-squares regression I

Let $X \in \mathbb{R}^{n \times p}$ be the design matrix, and $y \in \mathbb{R}^n$ the response vector.

$$\begin{aligned} \underset{\beta \in \mathbb{R}^p}{\text{minimize}} \ & \|y - X\beta\|_2^2 \\ \text{subject to} \ & \|\beta\|_0 \leq k \end{aligned}$$

where $\|\beta\|_0 = \#\{i : |\beta_i| > 0\}$.

# Example: Sparse least-squares regression II

Introduce variables $z_i$ to indicate whether component $i$ is used:

$$
\begin{aligned}
\underset{\beta,z}{\text{minimize}} \quad & \|y - X\beta\|_2^2 \\
\text{subject to} \quad & \sum_{i=1}^{p} z_i \leq k \\
& -\mathcal{M}z_i \leq \beta_i \leq \mathcal{M}z_i \text{ for } i = 1, \ldots, p \\
& z_i \in \{0, 1\} \text{ for } i = 1, \ldots, p \\
& \beta \in \mathbb{R}^p
\end{aligned}
$$

which is an MIQP.

# Semidefinite Programs (SDPs) I

**Recall:** A symmetric matrix $A \in \mathbf{S}^n$ is called *positive semidefinite* (psd) if one of the following equivalent conditions holds:

- A $= P^T P$ for some $P \in \mathbb{R}^{n \times n}$
- $x^T A x \geq 0$ for all $x \in \mathbb{R}^n$
- $A$ has non-negative eigenvalues

If $A$ is psd we write $A \succcurlyeq 0$

# Semidefinite Programs (SDPs) II

$$\begin{aligned} \underset{x}{\text{minimize}}\ & c^T x \\ \text{subject to}\ & x_1 F_1 + \ldots + x_n F_n + G \succcurlyeq 0 \\ & Ax = b, \end{aligned}$$

where $G, F_1, \ldots, F_n \in \mathbf{S}^k$.

**Algorithms:** interior point

**Applications:** combinatorial optimization, robust optimization

# Example: $k$-partition relaxation I

For a given partition, let $X \in \{0,1\}^{n \times n}$ be the matrix such that for each $u, v \in V$:

$$X_{uv} = \begin{cases} 1 & \text{if } u, v \text{in same subset} \\ 0 & \text{otherwise} \end{cases}$$

then, it can be shown that:

$$X \succcurlyeq 0$$
$$kX - J \succcurlyeq 0$$

where $J$ is the all-ones $n \times n$ matrix.

# Example: $k$-partition relaxation II

This motivates that following SDP relaxation:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{u,v \in V} X_{uv} \\
\text{subject to} \quad & X_{vv} = 1 \text{ for all } v \in V \\
& kX - J \succcurlyeq 0 \\
& X \succcurlyeq 0.
\end{aligned}
$$

# Non-linear programs

Two different types of problem:

- Unconstrained/box-constrained
- Constrained

# Unconstrained/Box Constrained NLPs

$$
\begin{aligned}
&\underset{x}{\text{minimize}}\ f(x)\\
&\text{subject to}\\
&\qquad l_i \leq x_i \leq u_i \qquad \text{for } i = 1, \ldots, n.
\end{aligned}
$$

**Algorithms:**

Gradient-free: Nelder-Mead, simulated-annealing, Baysian global optimization, Brent (1D bisection-based)

Gradient based: Gradient-descent, Conjugate gradient, BFGS, L-BFGS, Newton

# Constrained NLPs

$$
\begin{aligned}
&\underset{x}{\text{minimize}}\; f(x) \\
&\text{subject to} \\
&\qquad f_i(x) \leq 0 \qquad \text{for } i = 1, \ldots, k, \\
&\qquad h_i(x) = 0 \qquad \text{for } i = 1, \ldots, l,
\end{aligned}
$$

**Algorithms:** Interior Point, Lagrangian, COBYLA (gradient-free), Heuristics

**Applications:** MLEs, experimental design, physical problems, (all of the above)

# Solvers

# Modelling and Solver Landscape

User-friendliness

Modelling Tools

High-level language bindings

Solver API (C/C++)

Control and Flexibility

"One-shot" optimizers

# "One-shot" optimizers

- Some languagess provide "one-shot" functions for solving simple problem classes:
  - *optim* in R for solving unconstrained NLPs
  - *linprog* and *quadprog* in Matlab for solving LPs and QPs
- These functions are convenient for simple problems but much less well-developed than solver APIs
  - Bad at modelling complicated problems
  - Limited customizability
  - Slow
- Use dedicated solver packages for difficult research problems!

# Considerations for choosing a Solver API I

- Speed
- Open source/commercial
- Modelling capabilities
- Solver customization:
  - Available algorithms
  - Warm-starts
  - Call-backs

# Considerations for choosing a Solver API II

- Ease of use:
    - Interface
    - Installation
    - Language bindings
- Support

# Solvers and Problem-Types

| Solver | Open/Commercial | LP | MILP | SOCP | SDP | NLP |
|---|---|---|---|---|---|---|
| CPLEX | Commerical | X | X | X | | |
| Gurobi | Commercial | X | X | X | | |
| CLP/CBC | Open | X | X | | | |
| GLPK | Open | X | X | | | |
| Mosek | Commercial | X | X | X | X | X |
| Ipopt | Open | X | | | | X |
| NLOpt | Commercial | | | | | X |

Table : Solver modelling capabilities

- All of these solvers have interfaces for Python, R, Julia and Matlab.

# CPLEX

- Developed by Robert Bixby and released commercially in 1988.
- State-of-the-art solver for LPs, MILPs, SOCPs and SOCIPs.
- Free academic licence available
- Features: problem modification, warm-starts, multithread-support, call-backs

# Gurobi

GUROBI
OPTIMIZATION

- Gurobi is named after its founders: Zonghao **Gu**, Edward **Ro**thberg and Robert **Bi**xby and released in 2009.
- State-of-the-art solver for LPs, MILPs, SOCPs and SOCIPs.
- Free academic licence easily available
- Features: problem modification, warm-starts, multithread-support, call-backs

# COIN Solvers (CLP, and CBC)

- COIN-OR is an initiative founded in 2000 promoting the development of open source software for OR
- COIN-OR has a LP solver **CLP** and a MILP solver **CBC**
- Mature, well-supported libraries which are easily available
- Not under active development

# GLPK

- GNU Linear Programming Kit (GLPK) was first released in 2000
- Mature, well-supported libraries which are easily available
- GNU MathProg modelling language
- Not as fast as COIN-OR solvers

# Mosek

- First launched in 1997, MOSEK is a tool for solving a variety of optimization problems
- Free academic licence available
- LP and MILP solvers not as fast as CPLEX and Gurobi, but supports solution of SOCP, SDP and other convex problems (using interior point methods)

# Ipopt

- IPopt (**I**nterior **P**oint **o**ptimization) is a COIN-OR project for solving constrained non-linear (convex) optimization problems
- Mature and well-supported library

# NLOpt

- NLOpt (**N**on-**L**inear **O**ptimization), initially released in 2008, provides a variety of algorithms for solving constrained and unconstrained non-linear optimization problems
- Algorithms for gradient-based and gradient free optimization
- Algorithms for local and global optimization

# Remarks and Other Solvers

- Commercial solvers tend to be faster and more reliable
- Open solvers more transparent (can view source)

**MILP and Conic Solvers**: SCIP (open, MILP), SCS (open, SOCP, SDP), ECOS (open, SOCP)
**Non-linear Solvers**: KNITRO (commercial, NLP)

# Modelling Tools

# Modelling Languages (MLs)

- Implementing an optimization using a solver API can be tedious and error prone
- MLs are tools which make modelling much more intuitive
- Allow one to code a structured problem in a way which represents the algebraic model
- Independent of problem input data
- Often independent of solver

# AMPL Example

```
set NUTR ordered;
set FOOD ordered;

param cost{FOOD} >= 0;
param n_min{NUTR} >= 0, default 0;
param n_max{i in NUTR} >= n_min[i], default Infinity;
param amt{NUTR,FOOD} >= 0;

var Buy{j in FOOD} integer;

minimize Total_Cost: sum{j in FOOD} cost[j] * Buy[j];

subject to Diet {i in NUTR}:
n_min[i] <= sum{j in FOOD} amt[i,j]*Buy[j] <= n_max[i];
```

Listing 1: Diet problem in AMPL

# JuMP Example

```
m = Model()

@defVar(m, minNutrition[i] <= nutrition[i=1:numCategories] <= maxNutrition[i])
@defVar(m, buy[i=1:numFoods] >= 0)

@setObjective(m, Min, dot(cost, buy))

for j = 1:numCategories
  @addConstraint(m, sum{nutritionValues[i,j]*buy[i], i=1:numFoods} == nutrition[j])
end

status = solve(m)

```

Listing 2: A Diet problem in JuMP (Julia)

## Open vs. Commercial MLs

- Commercial MLs are stand-alone pieces of software and can lack flexibility
- Academic licences are available but can be quite restrictive
- Open source MLs are usually embedded in existing programming languages[1]
- This additional flexibility can make them seem less user-friendly

| | |
|---|---|
| Commercial MLs: | Lindo, Xpress-Mobel, MPL, AMPL, AIMMS |
| Open MLs: | MathProg, FlopC++, Pyomo (Python), JuMP (Julia), CVX[2] (Matlab/Octave) |

[1] MathProg is stand-alone

[2] Tool primarily for convex optimization

# Final Remarks

# Tips

- Providing solvers with initial solutions can greatly improve running time and output
- More specific solvers are faster as they can exploit problem structure
- If problem seems intractible, try reparameterising or a different formulation
- Solver-independent modelling tools are less error-prone, more readable and portable

# Conclusions

- Dedicated solvers are more suitable than “one-shot” optimizers for difficult research problems
- Solvers libraries are typically written in C/C++ but have bindings for high-level languages like R and Julia
- Choice of solver primarily determined by problem type, but other factors such as speed and licence are important
- Modelling tools simplify the task of implementing optimization problems