Mathematical Programming: Modelling and Software

Leo Liberti

LIX, École Polytechnique, France

The course

Title:	**Mathematical Programming: Modelling and Software**
Code:	INF572 (DIX) / ISC610A (ISIC)
Teacher:	Leo Liberti (`liberti@lix.polytechnique.fr`)
Assistants:	Sonia Cafieri (`cafieri@lix.polytechnique.fr`), Antonio Mucherino (`mucherino@lix.polytechnique.fr`), Giacomo Nannicini (`giacomon@lix.polytechnique.fr`), Fabien Tarissan (`tarissan@lix.polytechnique.fr`)
Timetable INF572:	tue 23,30/9, 7,14/10, 4,18,25/11, 2,9/12 0830-1000 (SI34/72), 1015-1200 (SI34); exam 16/12
Timetable ISC610A:	thu 6,10,13,27/11, 4,11,18/12; exam 8/1
URL INF572:	`http://www.lix.polytechnique.fr/~liberti/teaching/inf572`
URL ISC610A:	`http://www.lix.polytechnique.fr/~liberti/teaching/isc610a`

Contents

Introduction

Example: Set covering

There are 12 possible geographical positions $A_1, \ldots, A_{12}$ where some discharge water filtering plants can be built. These plants are supposed to service 5 cities $C_1, \ldots, C_5$; building a plant at site j ($j \in \{1, \ldots, 12\}$) has cost c_j and filtering capacity (in kg/year) f_j; the total amount of discharge water produced by all cities is 1.2×10^{11} kg/year. A plant built on site j can serve city i if the corresponding (i, j)-th entry is marked by a '*' in the table below.

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}
C_1	*		*		*		*	*				*
C_2		*	*			*			*		*	*
C_3	*	*				*	*			*		
C_4		*		*			*	*		*		*
C_5				*	*	*			*	*	*	*
c_j	7	9	12	3	4	4	5	11	8	6	7	16
f_j	15	39	26	31	34	24	51	19	18	36	41	34

What is the best placement for the plants?

Example: Sudoku

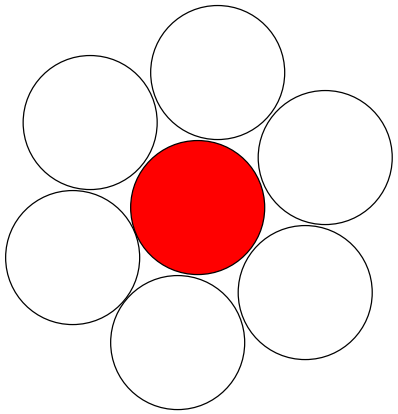
Given the Sudoku grid below, find a solution or prove that no solution exists

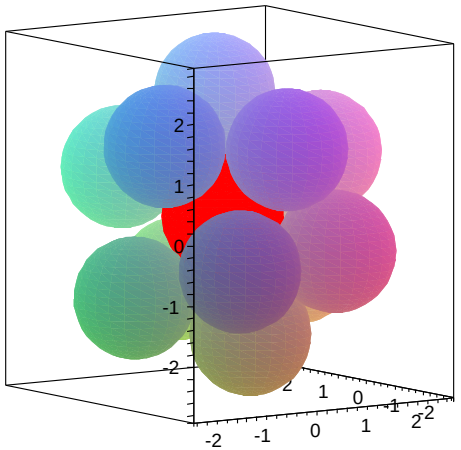
2								1
	4	1	9		2	8	6	
5	8						2	7
			5	1	3			
				9				
			7	8	6			
3	2	6					4	9
	1	9	4		5	2	8	
8								6

Example: Kissing Number

How many unit balls with disjoint interior can be placed adjacent to a central unit ball in $\mathbb{R}^d$?

In $\mathbb{R}^2$



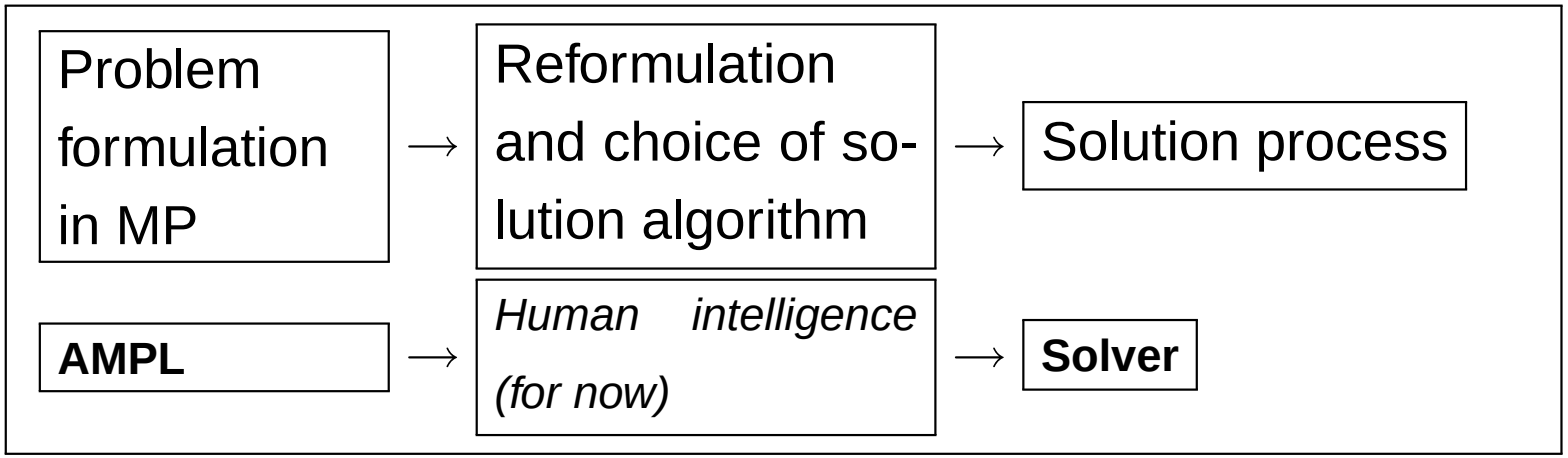


In $\mathbb{R}^3$

($D = 3$: problem proposed by Newton in 1694, settled by [Schütte and van der Waerden 1953] and [Leech 1956])

Mathematical programming

- The above three problems seemingly have *nothing* in common!
- Yet, there is a *formal language* that can be used to describe all three: **mathematical programming** (MP)
- Moreover, the MP language comes with a rich supply of solution algorithms so that problems can be solved right away



Modelling questions

Asking yourself the following questions should help you get started with your MP model

- The given problem is usually a particular *instance* of a *problem class*; you should model the whole class, not just the instance (replace given numbers by parameter symbols)
- What are the decisions to be taken? Are they logical, integer or continuous?
- What is the objective function? Is it to be minimized or maximized?
- What constraints are there in the problem? Beware — some constraints may be “hidden” in the problem text

If expressing objective and constraints is overly difficult, go back and change your variable definitions

Set covering 1

Let us now consider the Set Covering problem

What is the problem class?

- We replace the number 12 by the parameter symbol n, the number 5 by m and the number 1.2×10^{11} by d
- We already have symbols for costs (c_j) and capacities (f_j), where $j \leq n$ and $i \leq m$
- We represent the asterisks by a 0-1 matrix $A = (a_{ij})$ where $a_{ij} = 1$ if there is an asterisk at row i, column j of the table, and 0 otherwise

Set covering 2

What are the decisions to be taken?

- The crucial text in the problem is *what is the best placement for the plants?*; i.e. we need to **place each plant at some location**
  1. geographical placement on a plane? (continuous variables)
  2. yes/no placement? ("should the j-th plant be placed here?" — logical 0-1 variables)
- Because the text also says *there are n possible geographical positions...*, it means that for each position we have to decide **whether or not to build a plant** there
- For each of geographical position, introduce a **binary variable** (taking 0-1 values):

$$\forall j \leq n \quad x_j \in \{0, 1\}$$

Set covering 3

What is the objective function?

- In this case we only have the indication *best placement* in the text
- Given our data, two possibilities exist: cost (minimization) and filtering capacity (maximization)
- However, because of the presence of the parameter d, it wouldn't make sense to have more aggregated filtering capacity than d kg/year
- Hence, the objective function is the cost, which should be minimized:

$$\min \sum_{j \leq n} c_j x_j$$

Set covering 4

What are the constraints?

- The total filtering capacity must be at least d:

$$\sum_{j \leq n} f_j x_j \geq d$$

- Each city must be served by at least one plant:

$$\forall i \leq m \quad \sum_{j \leq n} a_{ij} x_j \geq 1$$

- Because there are no more constraints in the text, this concludes the first modelling phase

Analysis

- What category does this mathematical program belong to?
  - Linear Programming (LP)
  - Mixed-Integer Linear Programming (MILP)
  - Nonlinear Programming (NLP)
  - Mixed-Integer Nonlinear Programming (MINLP)
- Does it have any notable mathematical property?
  - If an NLP, are the functions/constraints convex?
  - If a MILP, is the constraint matrix Totally Unimodular (TUM)?
  - Does it have any apparent symmetry?
- **Can it be reformulated to a form for which a fast solver is available?**

Set covering 5

- The objective function and all constraints are *linear forms*
- All the decision variables are *binary*
- Hence the problem is a MILP
- **Good solutions** can be obtained via heuristics (e.g. local branching, feasibility pump, VNS, Tabu Search)
- **Exact solution** via Branch-and-Bound (solver: CPLEX)
- No need for reformulation: CPLEX is a fast enough solver
- CPLEX 11.0.1 solution: $x_4 = x_7 = x_{11} = 1$, all the rest 0 (i.e. build plants at positions 4,7,11)
- Notice the paradigm | model & solver $\rightarrow$ solution |

Since there are many solvers already available, | *solving the problem* | reduces to | **modelling the problem** |

AMPL Basics

AMPL

- AMPL means “A Mathematical Programming Language”
- AMPL is an implementation of the Mathematical Programming language
- Many solvers can work with AMPL
- AMPL works as follows:
  1. translates a user-defined model to a low-level formulation (called *flat form*) that can be understood by a solver
  2. passes the flat form to the solver
  3. reads a solution back from the solver and interprets it within the higher-level model (called *structured form*)

Model, data, run

- AMPL usually requires three files:
  - the *model* file (extension `.mod`) holding the MP formulation
  - the *data* file (extension `.dat`), which lists the values to be assigned to each parameter symbol
  - the *run* file (extension `.run`), which contains the (imperative) commands necessary to solve the problem
- The model file is written in the MP language
- The data file simply contains numerical data together with the corresponding parameter symbols
- The run file is written in an imperative C-like language (many notable differences from C, however)
- Sometimes, MP language and imperative language commands can be mixed in the same file (usually the run file)

To run AMPL, type `ampl < problem.run` from the command line

An elementary run file

- Consider the set covering problem, suppose we have coded the model file (`setcovering.mod`) and the data file (`setcovering.dat`), and that the CPLEX solver is installed on the system
- Then the following is a basic `setcovering.run` file

```
# basic run file for setcovering problem
model setcovering.mod; # specify model file
data setcovering.dat;  # specify data file
option solver cplex;   # specify solver
solve;                 # solve the problem
display cost;          # display opt. cost
display x;             # display opt. soln.
```

Set covering model file

```
# setcovering.mod
param m integer, >= 0;
param n integer, >= 0;
set M := 1..m;
set N := 1..n;

param c{N} >= 0;
param a{M,N} binary;
param f{N} >= 0;
param d >= 0;

var x{j in N} binary;

minimize cost: sum{j in N} c[j]*x[j];
subject to capacity: sum{j in N} f[j]*x[j] >= d;
subject to covering{i in M}: sum{j in N} a[i,j]*x[j] >= 1;
```

Set covering data file

```
param m := 5;
param n := 12;
param  :   c   f :=
       1   7  15
       2   9  39
       3  12  26
       4   3  31
       5   4  34
       6   4  24
       7   5  51
       8  11  19
       9   8  18
      10   6  36
      11   7  41
      12  16  34 ;
param a:   1  2  3  4  5  6  7  8  9 10 11 12 :=
      1    1  0  1  0  1  0  1  1  0  0  0  0
      2    0  1  1  0  0  1  0  0  1  0  1  1
      3    1  1  0  0  0  1  1  0  0  1  0  0
      4    0  1  0  1  0  0  1  1  0  1  0  1
      5    0  0  0  1  1  1  0  0  1  1  1  1 ;
param d := 120;
```

AMPL+CPLEX solution

```
liberti@nox$ cat setcovering.run | ampl
ILOG CPLEX 11.010, options: e m b q use=2
CPLEX 11.0.1: optimal integer solution; objective 15
3 MIP simplex iterations
0 branch-and-bound nodes
cost = 15
x [*] :=
 1  0
 2  0
 3  0
 4  1
 5  0
 6  0
 7  1
 8  0
 9  0
10  0
11  1
12  0
;
```

Formal definition of MP language

Formal languages

- An *alphabet* is any well-defined set A
- A *word* is any finite sequence of letters of A; the empty word is denoted by ϵ
- The set of words is denoted by A^*
- A *formal language* is a subset L of A^*
- The main problem linked to formal languages is: **given a word ℓ, does it belong to L?** (i.e. determine whether a given "sentence" is correctly formulated w.r.t. the language)
- Main tool: *context-free grammars* (CFGs)

Grammars

- Informally A grammar is a set of rules used to break down a sentence in its constituent parts
- *For example, the English language*:

sentence	:	subj_obj predicate
subj_obj	:	article noun
		noun
predicate	:	verb complements
	:	verb
complements	:	complement complements
	:	complement
complement	:	subj_obj
	:	time_complement
	:	space_complement
	:	...

Grammars

article	:	*the*
	:	*a*
noun	:	*aardvark*
	:	*Aalto*
	:	...
	:	*zulu*

Potentially, each English sentence can be decomposed in its logical components

The sentence *a zulu watches the aardvark* would be analysed as:

sentence $\longrightarrow$ **subj_obj predicate** $\longrightarrow$
article noun verb complements $\longrightarrow$
a zulu watches **complement** $\longrightarrow$
a zulu watches **subj_obj** $\longrightarrow$
a zulu watches **article noun** $\longrightarrow$
a zulu watches the aardvark

Grammars

- Aim Given a sentence, decide whether it belongs to a language via a grammar
- For example, the sentence *aardvark zulu a the watches* is made up of English words but **is not English**
- The grammar allows us to decide this:
- *zulu* would be classified as a **verb** because it appears in the **predicate** after **subj_obj** — but then no rule **verb** $\longrightarrow$ *zulu* exists
- the procedure stops before all words have been analysed
- $\Rightarrow$ the sentence does not belong to the language

If letters in strings represent stored states, a grammar is like a set of imperative statements whose application does not follow a fixed succession

Context free grammars

- A *CFG* is a quadruplet (N, Σ, R, S)
- $\Sigma \subseteq L$ is a finite set of *terminal symbols*
- N is a finite set of *nonterminal symbols* s.t. $\Sigma \cap N = \emptyset$ and $S \in N$
- R is a relation of $N \times (N \cup \Sigma)^*$ such that $\exists w \in (N \cup \Sigma)^* \ ((S, w) \in R)$
- (ν, w) is written $\nu : w$ for all $\nu \in N$, and w is a *parsing expression* (PEs); these are defined recursively as follows:
  1. terminal or nonterminal symbols and empty string ϵ (*atomic* PEs):
  2. given any PE $\bar{e}, \tilde{e}$, the following are PEs:
     - $\bar{e}\tilde{e}$ (sequence), $\bar{e}|\tilde{e}$ (or)
     - $\bar{e}*$ (zero-or-more) equivalent to $\bar{e}\ \mu|\epsilon$ (for $\mu \in \Sigma$),
     - $\bar{e}+$ (one-or-more) equivalent to $\bar{e}\ \bar{e}*$
     - $\bar{e}?$ (optional) equivalent to $\bar{e}|\epsilon$
     - $\&\bar{e}$ (and), $!\bar{e}$ (not)
     - square brackets are used to indicate precedence when necessary

Non-atomic parsing expressions

Assume $\Sigma = \{a, \ldots, z, \_\}$

- (Sequence) ν : *cat* matches $\nu =$ *cattle* but not *tomcat*
- (Or) ν : *cat*|*dog* matches $\nu \in \{$*catalysis*, *dogbert*$\}$ but not *my_cat_is_a_dog*
- (Zero-or-more) ν : *ca*[*t*]$*$ matches $\nu \in \{$*cane*, *catalogue*, *cattle*$\}$, but not *copper*
- (One-or-more) ν : *ca*[*t*]$+$ matches $\nu \in \{$*catalogue*, *cattle*$\}$ but not *cane* or *copper*
- (Optional) ν : *ca*[*t*]$?$*e* matches $\nu \in \{$*case*, *category*$\}$ but not *catters* or *cars*
- (And) ν : *cat*$\&$*ego* matches $\nu =$ category but not *cattle*
- (Not) ν : *cat*!*ego* matches $\nu =$ *cattle* but not *category*

Mathematical expressions

- Assume $\Sigma = \{$[0-9], [a-zA-Z], ., (,), *, /, +, -$\}$,
  $N = \{$float, variable, leaf, value, product, sum, expression$\}$.
- The following CFG (call it $\mathfrak{A}$) recognizes arithmetical expressions with the operators +,-,*,/ acting on floating point numbers and variables of the form *name number* (such as e.g. *x1*)

integer	:	[0-9]$+$
float	:	[0-9]$*$.[0-9]$+$
number	:	integer|float
variable	:	[a-z]$+$ [0-9] $*$
leaf	:	number|variable
value	:	leaf| '(' expression ')'
product	:	value[['*' | '/']value] $*$
sum	:	product[['+' | '-']product] $*$
expression	:	sum

- It is easy to extend $\mathfrak{A}$ to also recognize the power operator and transcendental functions such as $\exp, \log, \sin, \cos$

Quantified expressions

- Assume $\Sigma = \{\sum, \prod, \forall, \in, \downarrow, \leq, =, \geq, :\} \cup \mathfrak{A}$,
  $N = \{$index, set, param, ivar, ileaf, prod1, sum1, iprod, isum, iexpr, qatom, qlist, catom, clist, quantifier$\}$
- The following CFG (call it $\mathfrak{B}$) recognizes mathematical expressions quantified over given sets

indexatom	:	[a-zA-Z]$*$ [$\downarrow$ integer]? !ϵ
index	:	[index ’,’]? indexatom
set	:	[a-zA-Z]$+$ [$\downarrow$ index]?
param	:	[a-zA-Z]$+$ [$\downarrow$ index]?
ivar	:	variable[$\downarrow$ index]?
ileaf	:	float|param|ivar
ival	:	ileaf|[’(’]? iexpr [’)’]?
prod1	:	ival[[’*’ | ’/’]ival]$*$
sum1	:	prod1[[’+’ | ’-’]prod1]$*$

iprod	:	$\prod \downarrow$ quantifier ival|prod1
isum	:	$\sum \downarrow$ quantifier ival|sum1
iexpr	:	isum
catom	:	iexpr [$\leq$ | $=$ | $\geq$] iexpr
clist	:	[clist [$\wedge$|$\vee$]]? catom
qatom	:	$\forall$ index $\in$ set
qlist	:	[qlist ’,’]? qatom
quantifier	:	qlist [’:’ clist]?

- An iexpr $f \downarrow q$ is written x_q
- Extension to power and transcendental functions equally easy

MP Language

- $\Sigma = \{\mathbb{Z}, \min, \max\} \cup \mathfrak{B}$, $N =$ $\{$objective, constraint, integrality, objlist, conlist, intlist$\}$
- The following CFG (call it $\mathfrak{M}$) recognizes mathematical programming formulations

objective	:	[quantifier]? $[\min \mid \max]$ iexpr
objlist	:	objlist objective
constraint	:	[quantifier]? catom
conlist	:	conlist constraint
integrality	:	[quantifier]? ivar $\in \mathbb{Z}$
intlist	:	intlist integrality

Example: set covering

$$
\begin{array}{lrcl}
\min & \sum_{j \in N} c_j x_j & & \\
 & \sum_{j \in N} f_j x_j & \geq & d \\
\forall i \in M & \sum_{j \in N} a_{ij} x_j & \geq & 1 \\
\forall j \in N & x_j & \geq & 0 \\
\forall j \in N & x_j & \leq & 1 \\
\forall j \in N & x_j & \in & \mathbb{Z}
\end{array}
$$

$M = \{1, \ldots, m\}$, $N = \{1, \ldots, n\}$

- $\forall j \in N$: quantifier
- $c_j x_j$: iexpr
- $\sum_{j \in N} c_j x_j$: iexpr
- $x_j \geq 0$: catom
- $\forall j \in N x_j \leq 1$: constraint
- $\forall j \in N x_j \in \mathbb{Z}$: integrality

- objlist only has one objective which has an empty quantifier and a fairly simple iexpr consisting of a quantified sum whose argument is a product
- conlist has two constraints, the second of which is quantified
- intlist only has one (quantified) integrality element

MP language implementations

Software packages implementing (sub/supersets of the) MP language:

- **AMPL (our software of choice, mixture of MP and near-C language)**
  - commercial, but student version limited to 300 vars/constrs is available from `www.ampl.com`
  - quite a lot of solvers are hooked to AMPL
- GNU MathProg (subset of AMPL)
  - free, but only the GLPK solver (for LPs and MILPs) can be used
  - it is a significant subset of AMPL but not complete
- GAMS (can do everything AMPL can, but looks like COBOL — ugh!)
  - commercial, limited demo available from `www.gams.com`
  - quite a lot of solvers are hooked to GAMS
- Zimpl (free, C++ interface, linear modelling only)
- LINDO, MPL, . . . (other commercial modelling/solution packages)

AMPL Grammar

AMPL MP Language

- There are 5 main entities: sets, parameters, variables, objectives and constraints
- In AMPL, each entity has a name and can be quantified
  - `set` *name* [{*quantifier*}] *attributes* `;`
  - `param` *name* [{*quantifier*}] *attributes* `;`
  - `var` *name* [{*quantifier*}] *attributes* `;`
  - `minimize|maximize` *name* [{*quantifier*}]`:` *iexpr* `;`
  - `subject to` *name* [{*quantifier*}]`:` *iexpr* `<=` | `=` | `>=` *iexpr* `;`
- Attributes on sets and parameters is used to validate values read from data files
- Attributes on vars specify integrality (`binary`, `integer`) and limit constraints (`>= lower`, `<= upper`)
- Entities indices: square brackets (e.g. `y[1]`, `x[i,k]`)
- The above is the basic syntax — there are some advanced options

AMPL data specification

In general, syntax is in map-like form; a

```
param p{i in S} integer;
```

is a map $S \to \mathbb{Z}$, and each pair (domain, codomain) must be specified:

```
param p :=
  1  4
  2 -3
  3  0;
```

The grammar is simple but tedious, best way is learning by example or trial and error

AMPL imperative language

- `model` *model_filename.mod* `;`
- `data` *data_filename.dat* `;`
- `option` *option_name literal_string*, ... `;`
- `solve ;`
- `display` [{*quantifier*}]`:` *iexpr* `;` / `printf` (syntax similar to C)
- `let` [{*quantifier*}] *ivar* `:=`*number*`;`
- `if (`*clist*`) then` { *commands* } [`else` {*commands*}]
- `for` {*quantifier*} {*commands*} / `break;` / `continue;`
- `shell '`*command_line*`';` / `exit` *number*`;` / `quit;`
- `cd` *dir_name*`;` / `remove` *file_name*`;`
- In all output commands, screen output can be redirected to a file by appending > *output_filename.txt* before the semicolon
- These are basic commands, there are some advanced ones

Reformulation commands

- `fix` [{*quantifier*}] *ivar* [`:=`*number*]`;`
- `unfix` [{*quantifier*}] *ivar*`;`
- `delete` *entity_name*`;`
- `purge` *entity_name*`;`
- `redeclare` *entity_declaration*`;`
- `drop/restore` [{*quantifier*}] *constr_or_obj_name*`;`
- `problem` *name*[{*quantifier*}] [`:`*entity_name_list*] `;`
- This list is not exhaustive

Solvers

Solvers

In order of solver reliability / effectiveness:

1. **LPs**: use an LP solver ($O(10^6)$ vars/constrs, fast, e.g. CPLEX, CLP, GLPK)
2. **MILPs**: use a MILP solver ($O(10^4)$ vars/constrs, can be slow, e.g. CPLEX, Symphony, GLPK)
3. **NLPs**: use a local NLP solver to get a local optimum ($O(10^4)$ vars/constrs, quite fast, e.g. SNOPT, MINOS, IPOPT)
4. **NLPs/MINLPs**: use a heuristic solver to get a good local optimum ($O(10^3)$, quite fast, e.g. BONMIN, MINLP_BB)
5. **NLPs**: use a global NLP solver to get an (approximated) global optimum ($O(10^3)$ vars/constrs, can be slow, e.g. COUENNE, BARON)
6. **MINLPs**: use a global MINLP solver to get an (approximated) global optimum ($O(10^3)$ vars/constrs, can be slow, e.g. COUENNE, BARON)

Not all these solvers are available via AMPL

Solution algorithms (linear)

- **LPs**: *(convex)*
  1. simplex algorithm (non-polynomial complexity but very fast in practice, reliable)
  2. interior point algorithms (polynomial complexity, quite fast, fairly reliable)
- **MILPs**: *(nonconvex because of integrality)*
  1. *Local* (heuristics): Local Branching, Feasibility Pump [Fischetti&Lodi 05], VNS [Hansen et al. 06] (quite fast, reliable)
  2. *Global*: Branch-and-Bound (exact algorithm, non-polynomial complexity but often quite fast, heuristic if early termination, reliable)

Solution algorithms (nonlinear)

- **NLPs**: *(may be convex or nonconvex)*
  1. *Local*: Sequential Linear Programming (SLP), Sequential Quadratic Programming (SQP), interior point methods (linear/polynomial convergence, often quite fast, unreliable)
  2. *Global*: spatial Branch-and-Bound [Smith&Pantelides 99] (ε-approximate, nonpolynomial complexity, often quite slow, heuristic if early termination, unreliable)
- **MINLPs**: *(nonconvex because of integrality and terms)*
  1. *Local* (heuristics): Branching explorations [Fletcher&Leyffer 99], Outer approximation [Grossmann 86], Feasibility pump [Bonami et al. 06] (nonpolynomial complexity, often quite fast, unreliable)
  2. *Global*: spatial Branch-and-Bound [Sahinidis&Tawarmalani 05] (ε-approximate, nonpolynomial complexity, often quite slow, heuristic if early termination, unreliable)

LP example: .mod

```
# lp.mod
param n integer, default 3;
param m integer, default 4;
set N := 1..n;
set M := 1..m;
param a{M,N};
param b{M};
param c{N};

var x{N} >= 0;
minimize objective: sum{j in N} c[j]*x[j];
subject to constraints{i in M} :
  sum{j in N} a[i,j]*x[j] <= b[i];
```

LP example: .dat

```
# lp.dat
param n := 3; param m := 4;
param c                 :=
            1  1
            2 -3
            3 -2.2 ;
param b                 :=
            1 -1
            2  1.1
            3  2.4
            4  0.8 ;
param a : 1    2    3  :=
      1   0.1  0   -3.1
      2   2.7 -5.2  1.3
      3   1    0   -1
      4   1    1    0 ;
```

LP example: .run

```
# lp.run

model lp.mod;
data lp.dat;
option solver cplex;
solve;
display x;
```

LP example: output

```
CPLEX 11.0.1: optimal solution; objective -11.30153
0 dual simplex iterations (0 in phase I)
x [*] :=
1  0
2  0.8
3  4.04615
;
```

MILP example: .mod

```
# milp.mod
param n integer, default 3;
param m integer, default 4;
set N := 1..n;
set M := 1..m;
param a{M,N};
param b{M};
param c{N};

var x{N} >= 0, <= 3, integer;
var y >= 0;
minimize objective: sum{j in N} c[j]*x[j];
subject to constraints{i in M} :
  sum{j in N} a[i,j]*x[j] - y <= b[i];
```

MILP example: .run

```
# milp.run

model milp.mod;
data lp.dat;
option solver cplex;
solve;
display x;
display y;
```

MILP example: output

```
CPLEX 11.0.1: optimal integer solution; objective
0 MIP simplex iterations
0 branch-and-bound nodes
x [*] :=
1  0
2  3
3  3
;
y = 2.2
```

NLP example: .mod

```
# nlp.mod
param n integer, default 3;
param m integer, default 4;
set N := 1..n;
set M := 1..m;
param a{M,N};
param b{M};
param c{N};
var x{N} >= 0.1, <= 4;
minimize objective:
  c[1]*x[1]*x[2] + c[2]*x[3]^2 + c[3]*x[1]*x[2]/x[3];
subject to constraints{i in M diff {4}} :
  sum{j in N} a[i,j]*x[j] <= b[i]/x[i];
subject to constraint4 : prod{j in N} x[j] <= b[4];
```

NLP example: .run

```
# nlp.run
model nlp.mod;
data lp.dat;
## only enable one of the following methods
## 1: local solution
option solver minos;
# starting point
let x[1] := 0.1;
let x[2] := 0.2; # try 0.1, 0.4
let x[3] := 0.2;
## 2: global solution (heuristic)
#option solver bonmin;
## 3: global solution (guaranteed)
#option solver boncouenne;
solve;
display x;
```

NLP example: output

```
MINOS 5.51: optimal solution found.
47 iterations, objective -38.03000929
Nonlin evals: obj = 131, grad = 130, constrs = 131,
x [*] :=
1  2.84106
2  1.37232
3  0.205189
;
```

MINLP example: .mod

```
# minlp.mod
param n integer, default 3;
param m integer, default 4;
set N := 1..n;
set M := 1..m;
param a{M,N};
param b{M};
param c{N};
param epsilon := 0.1;
var x{N} >= 0, <= 4, integer;
minimize objective:
  c[1]*x[1]*x[2] + c[2]*x[3]^2 + c[3]*x[1]*x[2]/x[3] +
  x[1]*x[3]^3;
subject to constraints{i in M diff {4}} :
  sum{j in N} a[i,j]*x[j] <= b[i]/(x[i] + epsilon);
subject to constraint4 : prod{j in N} x[j] <= b[4];
```

MINLP example: .run

```
# minlp.run
model minlp.mod;
data lp.dat;

## only enable one of the following methods:
## 1: global solution (heuristic)
#option solver bonmin;
## 2: global solution (guaranteed)
option solver boncouenne;

solve;
display x;
```

MINLP example: output

```
bonmin: Optimal
x [*] :=
1  0
2  4
3  4
;
```

Sudoku

Sudoku: problem class

What is the problem class?

- The class of all sudoku grids
- Replace $\{1, \ldots, 9\}$ with a set K
- Will need a set $H = \{1, 2, 3\}$ to define 3×3 sub-grids
- An “instance” is a partial assignment of integers to cases in the sudoku grid
- We model an empty sudoku grid, and then *fix* certain variables at the appropriate values

Modelling the Sudoku

- Q: *What are the decisions to be taken?*
- A: Whether to place an integer in $K = \{1, \ldots, 9\}$ in the case at coordinates (i, j) on the square grid ($i, j \in K$)
- **We might try integer variables** $y_{ij} \in K$
- Q: *What is the objective function?*
- A: There is no "natural" objective; we might wish to employ one if needed
- Q: *What are the constraints?*
- A: For example, the first row should contain all numbers in K; hence, we should express a constraint such as:
  - if $y_{11} = 1$ then $y_{1\ell} \neq 1$ for all $\ell \geq 1$;
  - if $y_{11} = 2$ then $y_{1\ell} \neq 2$ for all $\ell \geq 2$;
  - . . . (for all values, column and row indices)

Sudoku constraints 1

- In other words,

$$\forall i, j, k \in K, \ell \neq j \ (y_{ij} = k \rightarrow y_{i\ell} \neq k)$$

- *Put it another way*: a constraint that says "all values should be different"
- In *constraint programming* (a discipline related to MP) there is a constraint

$$\forall i \in K \ \text{AllDiff}(y_{ij} \mid j \in K)$$

that *asserts* that all variables in its argument take different values: **we can attempt to implement it in MP**

- A set of distinct values has the *pairwise distinctness property*: $\forall i, p, q \in K \ y_{ip} \neq y_{iq}$, which can also be written as:

$$\forall i, p < q \in K \quad |y_{ip} - y_{iq}| \geq 1$$

Sudoku constraints 2

- We also need the same constraints in each column:

$$\forall j, p < q \in K \quad |y_{pj} - y_{qj}| \geq 1$$

- . . . and in some appropriate 3×3 sub-grids:
  1. let $H = \{1, \ldots, 3\}$ and $\alpha = |K|/|H|$; for all $h \in H$ define $R_h = \{i \in K \mid i > (h-1)\alpha \wedge i \leq h\alpha\}$ and $C_h = \{j \in K \mid j > (h-1)\alpha \wedge j \leq h\alpha\}$
  2. show that for all $(h, l) \in H \times H$, the set $R_h \times C_l$ contains the case coordinates of the (h, l)-th 3×3 sudoku sub-grid
- Thus, the following constraints must hold:

$$\forall h, l \in H, i < p \in R_h, j < q \in C_l \quad |y_{ij} - y_{pq}| \geq 1$$

The Sudoku MINLP

- The whole model is as follows:

$$
\left.
\begin{array}{rrcl}
\min & 0 & & \\
\forall i, p < q \in K & |y_{ip} - y_{iq}| & \geq & 1 \\
\forall j, p < q \in K & |y_{pj} - y_{qj}| & \geq & 1 \\
\forall h, l \in H, i < p \in R_h, j < q \in C_l & |y_{ij} - y_{pq}| & \geq & 1 \\
\forall i \in K, j \in K & y_{ij} & \geq & 1 \\
\forall i \in K, j \in K & y_{ij} & \leq & 9 \\
\forall i \in K, j \in K & y_{ij} & \in & \mathbb{Z}
\end{array}
\right\}
$$

- This is a nondifferentiable MINLP
- MINLP solvers (BONMIN, MINLP_BB, COUENNE) can't solve it

Absolute value reformulation

- This MINLP, however, can be linearized:

$$|a - b| >= 1 \iff a - b >= 1 \vee b - a >= 1$$

- For each $i, j, p, q \in K$ we introduce a binary variable $w_{ij}^{pq} = 1$ if $y_{ij} - y_{pq} \geq 1$ and 0 if $y_{pq} - y_{ij} \geq 1$
- For all $i, j, p, q \in K$ we add constraints
  1. $y_{ij} - y_{pq} \geq 1 - M(1 - w_{ij}^{pq})$
  2. $y_{pq} - y_{ij} \geq 1 - M w_{ij}^{pq}$

  where $M = |K| + 1$
- This means: if $w_{ij}^{pq} = 1$ then constraint 1 is active and 2 is *always* inactive (as $y_{pq} - y_{ij}$ is always greater than $-|K|$); if $w_{ij}^{pq} = 0$ then 2 is active and 1 inactive
- Transforms problematic absolute value terms into added binary variables and linear constraints

The reformulated model

- The reformulated model is a MILP:

$$
\begin{array}{rlcl}
 & \min \quad 0 & & \\
\forall i, p < q \in K & y_{ip} - y_{iq} & \geq & 1 - M(1 - w_{ip}^{iq}) \\
\forall i, p < q \in K & y_{iq} - y_{ip} & \geq & 1 - M w_{ip}^{iq} \\
\forall j, p < q \in K & y_{pj} - y_{qj} & \geq & 1 - M(1 - w_{pj}^{qj}) \\
\forall j, p < q \in K & y_{qj} - y_{pj} & \geq & 1 - M w_{pj}^{qj} \\
\forall h, l \in H, i < p \in R_h, j < q \in C_l & y_{ij} - y_{pq} & \geq & 1 - M(1 - w_{ij}^{pq}) \\
\forall h, l \in H, i < p \in R_h, j < q \in C_l & y_{pq} - y_{ij} & \geq & 1 - M w_{ij}^{pq} \\
\forall i \in K, j \in K & y_{ij} & \geq & 1 \\
\forall i \in K, j \in K & y_{ij} & \leq & 9 \\
\forall i \in K, j \in K & y_{ij} & \in & \mathbb{Z}
\end{array}
$$

- It can be solved by CPLEX; however, it has $O(|K|^4)$ binary variables on a $|K|^2$ cases grid with $|K|$ values per case ($O(|K|^3)$ in total) — often an effect of **bad modelling**

A better model

- *In such cases, we have to go back to variable definitions*
- One other possibility is to define **binary variables** $\forall i, j, k \in K\ (x_{ijk} = 1)$ **if the** (i, j)**-th case has value** k, and 0 otherwise
- Each case must contain exactly one value

$$\forall i, j \in K \sum_{k \in K} x_{ijk} = 1$$

- For each row and value, there is precisely one column that contains that value, and likewise for cols

$$\forall i, k \in K \sum_{j \in K} x_{ijk} = 1 \quad \wedge \quad \forall j, k \in K \sum_{i \in K} x_{ijk} = 1$$

- Similarly for each $R_h \times C_h$ sub-square

$$\forall h, l \in H, k \in K \sum_{i \in R_h, j \in C_l} x_{ijk} = 1$$

The Sudoku MILP

- The whole model is as follows:

$$
\left.
\begin{array}{rrcl}
\min & 0 & & \\
\forall i \in K, j \in K & \sum_{k \in K} x_{ijk} & = & 1 \\
\forall i \in K, k \in K & \sum_{j \in K} x_{ijk} & = & 1 \\
\forall j \in K, k \in K & \sum_{i \in K} x_{ijk} & = & 1 \\
\forall h \in H, l \in H, k \in K & \sum_{i \in R_h, j \in C_l} x_{ijk} & = & 1 \\
\forall i \in K, j \in K, k \in K & x_{ijk} & \in & \{0,1\}
\end{array}
\right\}
$$

- This is a MILP with $O(|K|^3)$ variables
- Notice that there is a relation $\forall i, j \in K \; y_{ij} = \sum_{k \in K} k x_{ijk}$ between the MINLP and the MILP
- *The MILP variables have been derived by the MINLP ones by "disaggregation"*

Sudoku model file

```
param Kcard integer, >= 1, default 9;
param Hcard integer, >= 1, default 3;
set K := 1..Kcard;
set H := 1..Hcard;
set R{H};
set C{H};
param alpha := card(K) / card(H);
param Instance {K,K} integer, >= 0, default 0;
let {h in H} R[h] := {i in K : i > (h-1) * alpha and i <= h * alpha};
let {h in H} C[h] := {j in K : j > (h-1) * alpha and j <= h * alpha};
var x{K,K,K} binary;

minimize nothing: 0;

subject to assignment {i in K, j in K} : sum{k in K} x[i,j,k] = 1;
subject to rows {i in K, k in K} : sum{j in K} x[i,j,k] = 1;
subject to columns {j in K, k in K} : sum{i in K} x[i,j,k] = 1;
subject to squares {h in H, l in H, k in K} :
  sum{i in R[h], j in C[l]} x[i,j,k] = 1;
```

Sudoku data file

```
param Instance :=
1 1 2          1 9 1          2 2 4          2 3 1          2 4 9
2 6 2          2 7 8          2 8 6          3 1 5          3 2 8
3 8 2          3 9 7          4 4 5          4 5 1          4 6 3
5 5 9          6 4 7          6 5 8          6 6 6          7 1 3
7 2 2          7 3 6          7 8 4          7 9 9          8 2 1
8 3 9          8 4 4          8 6 5          8 7 2          8 8 8
9 1 8          9 9 6 ;
```

Sudoku run file

```
# sudoku
# replace "/dev/null" with "nul" if using Windows
option randseed 0;
option presolve 0;
option solver_msg 0;
model sudoku.mod;
data sudoku-feas.dat;
let {i in K, j in K : Instance[i,j] > 0} x[i,j,Instance[i,j]] := 1;
fix {i in K, j in K : Instance[i,j] > 0} x[i,j,Instance[i,j]];
display Instance;
option solver cplex;
solve > /dev/null;
param Solution {K, K};
if (solve_result = "infeasible") then {
  printf "instance is infeasible\n";
} else {
  let {i in K, j in K} Solution[i,j] := sum{k in K} k * x[i,j,k];
  display Solution;
}
```

Sudoku AMPL output

```
liberti@nox$ cat sudoku.run | ampl
Instance [*,*]
:   1   2   3   4   5   6   7   8   9    :=
1   2   0   0   0   0   0   0   0   1
2   0   4   1   9   0   2   8   6   0
3   5   8   0   0   0   0   0   2   7
4   0   0   0   5   1   3   0   0   0
5   0   0   0   0   9   0   0   0   0
6   0   0   0   7   8   6   0   0   0
7   3   2   6   0   0   0   0   4   9
8   0   1   9   4   0   5   2   8   0
9   8   0   0   0   0   0   0   0   6
;
instance is infeasible
```

Sudoku data file 2

But with a different data file. . .

```
param Instance :=
1 1 2          1 9 1          2 2 4          2 3 1          2 4 9
2 6 2          2 7 8          2 8 6          3 1 5          3 2 8
3 8 2          3 9 7          4 4 5          4 5 1          4 6 3
5 5 9          6 4 7          6 5 8          6 6 6          7 1 3
7 2 2                         7 8 4          7 9 9          8 2 1
8 3 9          8 4 4          8 6 5          8 7 2          8 8 8
9 1 8          9 9 6 ;
```

Sudoku data file 2 grid

. . . corresponding to the grid below. . .

2								1
	4	1	9		2	8	6	
5	8						2	7
			5	1	3			
				9				
			7	8	6			
3	2						4	9
	1	9	4		5	2	8	
8								6

Sudoku AMPL output 2

. . . we find a solution!

```
liberti@nox$ cat sudoku.run | ampl
Solution [*,*]
:   1   2   3   4   5   6   7   8   9    :=
1   2   9   6   8   5   7   4   3   1
2   7   4   1   9   3   2   8   6   5
3   5   8   3   6   4   1   9   2   7
4   4   7   8   5   1   3   6   9   2
5   1   6   5   2   9   4   3   7   8
6   9   3   2   7   8   6   1   5   4
7   3   2   7   1   6   8   5   4   9
8   6   1   9   4   7   5   2   8   3
9   8   5   4   3   2   9   7   1   6
;
```

Kissing Number Problem

KNP: problem class

What is the problem class?

- There is no number in the problem definition:

  How many unit balls with disjoint interior can be placed adjacent to a central unit ball in $\mathbb{R}^d$?

- Hence the KNP is already defined as a problem class
- Instances are given by assigning a positive integer to the only parameter D

Modelling the KNP

- Q: *What are the decisions to be taken?*
- A: How many spheres to place, and where to place them
- **For each sphere, two types of variables**
  1. a logical one: $y_i = 1$ if sphere i is present, and 0 otherwise
  2. a D-vector of continuous ones: $x_i = (x_{i1}, \ldots, x_{iD})$, position of i-th sphere center
- Q: *What is the objective function?*
- A: **Maximize the number of spheres**
- Q: *What are the constraints?*
- A: **Two types of constraints**
  1. the i-th center must be at distance 2 from the central sphere if the i-th sphere is placed (*center constraints*)
  2. for all distinct (and placed) spheres i, j, for their interior to be disjoint their centers must be at distance ≥ 2 (*distance constraints*)

Assumptions

1. **Logical variables** y
   - Since the objective function counts the number of placed spheres, it must be something like $\sum_i y_i$
   - What set N does the index i range over?
   - Denote $k^*(d)$ the optimal solution to the KNP in $\mathbb{R}^d$
   - Since $k^*(d)$ is unknown *a priori*, we cannot know N *a priori*; however, without N, we cannot express the objective function
   - *Assume* we know an **upper bound** $\bar{k}$ to $k^*(d)$; then we can define $N = \{1, \ldots, \bar{k}\}$ (and $D = \{1, \ldots, d\}$)
2. **Continuous variables** x
   - Since any sphere placement is invariant by translation, we *assume* that the central sphere is placed at the origin
   - Thus, each continuous variable x_{ik} ($i \in N, k \in D$) cannot attain values outside $[-2, 2]$ (why?)
   - Limit continuous variables: $-2 \leq x_{ik} \leq 2$

Problem restatement

- The above assumptions lead to a **problem restatement**

> *Given a positive integer $\bar{k}$, what is the maximum number (smaller than $\bar{k}$) of unit spheres with disjoint interior that can be placed adjacent to a unit sphere centered at the origin of $\mathbb{R}^d$?*

- *Each time assumptions are made for the sake of modelling, one must always keep track of the corresponding changes to the problem definition*
- The **Objective function** can now be written as:

$$\max \sum_{i \in N} y_i$$

Constraints

- *Center constraints*:

$$\forall i \in N \quad ||x_i|| = 2y_i$$

(if sphere i is placed then $y_i = 1$ and the constraint requires $||x_i|| = 2$, otherwise $||x_i|| = 0$, which implies $x_i = (0, \ldots, 0)$)

- *Distance constraints*:

$$\forall i \in N, j \in N \ : \ i \neq j \quad ||x_i - x_j|| \geq 2y_i y_j$$

(if spheres i, j are both are placed then $y_i y_j = 1$ and the constraint requires $||x_i - x_j|| \geq 2$, otherwise $||x_i - x_j|| \geq 0$ which is always by the definition of norm)

KNP model

$$
\left.
\begin{array}{rrcl}
\max & \sum_{i \in N} y_i & & \\
\forall i \in N & \sqrt{\sum_{k \in D} x_{ik}^2} & = & 2y_i \\
\forall i \in N, j \in N : i \neq j & \sqrt{\sum_{k \in D} (x_{ik} - x_{jk})^2} & \geq & 2y_i y_j \\
\forall i \in N & y_i & \geq & 0 \\
\forall i \in N & y_i & \leq & 1 \\
\forall i \in N, k \in D & x_{ik} & \geq & -2 \\
\forall i \in N, k \in D & x_{ik} & \leq & 2 \\
\forall i \in N & y_i & \in & \mathbb{Z}
\end{array}
\right\}
$$

For brevity, we shall write $y_i \in \{0, 1\}$ and $x_{ik} \in [-2, 2]$

Reformulation 1

- Solution times for NLP/MINLP solvers often also depends on the number of nonlinear terms
- We square both sides of the nonlinear constraints, and notice that since y_i are binary variables, $y_i^2 = y_i$ for all $i \in N$; we get:

$$\forall i \in N \quad \sum_{k \in D} x_{ik}^2 = 4y_i$$

$$\forall i \neq j \in N \quad \sum_{k \in D} (x_{ik} - x_{jk})^2 \geq 4y_i y_j$$

which has fewer nonlinear terms than the original problem

Reformulation 2

- Distance constraints are called *reverse convex* (because if we replace $\geq$ with $\leq$ the constraints become convex); these constraints often cause solution times to lengthen considerably
- Notice that distance constraints are repeated when i, j are swapped
- Change the quantifier to $i \in N, j \in N : i < j$ reduces the number of reverse convex constraints in the problem; get:

$$\forall i \in N \quad \sum_{k \in D} x_{ik}^2 = 4y_i$$

$$\forall i < j \in N \quad \sum_{k \in D} (x_{ik} - x_{jk})^2 \geq 4y_i y_j$$

KNP model revisited

$$
\left.
\begin{array}{rrcl}
\max & \sum_{i \in N} y_i & & \\
\forall i \in N & \sum_{k \in D} x_{ik}^2 & = & 4y_i \\
\forall i \in N, j \in N : i < j & \sum_{k \in D} (x_{ik} - x_{jk})^2 & \geq & 4y_i y_j \\
\forall i \in N, k \in D & x_{ik} & \in & [-2, 2] \\
\forall i \in N & y_i & \in & \{0, 1\}
\end{array}
\right\}
$$

This formulation is a (nonconvex) MINLP

KNP model file

```
# knp.mod
param d default 2;
param kbar default 7;
set D := 1..d;
set N := 1..kbar;

var y{i in N} binary;
var x{i in N, k in D} >= -2, <= 2;

maximize kstar : sum{i in N} y[i];

subject to center{i in N} : sum{k in D} x[i,k]^2 = 4*y[i];
subject to distance{i in N, j in N : i < j} :
   sum{k in D} (x[i,k] - x[j,k])^2 >= 4*y[i]*y[j];
```

KNP data file

Since the only data are the parameters d and $\bar{k}$ (two scalars), for simplicity we do not use a data file at all, and assign values in the model file instead

KNP run file

```
# knp.run
model knp.mod;
option solver boncouenne;
solve;
display x,y;
display kstar;
```

KNP solution (?)

- We tackle the easiest possible KNP instance ($d = 2$), and give it an upper bound $\bar{k} = 7$
- It is easy to see that $k^*(2) = 6$ (place 6 circles adjacent to another circle in an exagonal lattice)
- Yet, after *several minutes* of CPU time COUENNE has not made any progress from the trivial feasible solution $y = 0, x = 0$
- Likewise, heuristic solvers such as BONMIN and `MINLP_BB` only find the trivial zero solution and exit

What do we do next?

In order to solve the KNP and deal with other difficult MINLPs, we need more advanced techniques

Some useful MP theory

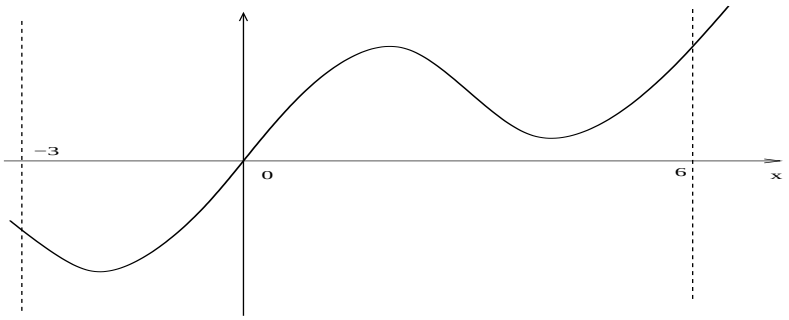
Canonical MP formulation

$$\left.\begin{array}{rrcl} \min_x & f(x) & & \\ \text{s.t.} & l \leq & g(x) & \leq u \\ & x^L \leq & x & \leq x^U \\ \forall i \in Z \subseteq \{1,\ldots,n\} & x_i & \in & \mathbb{Z} \end{array}\right\} [P] \qquad (1)$$

where $x, x^L, x^U \in \mathbb{R}^n$; $l, u \in \mathbb{R}^m$; $f : \mathbb{R}^n \to \mathbb{R}$; $g : \mathbb{R}^n \to \mathbb{R}^m$

- $\mathcal{F}(P) =$ feasible region of P, $\mathcal{L}(P) =$ set of local optima, $\mathcal{G}(P) =$ set of global optima
- Nonconvexity $\Rightarrow \mathcal{G}(P) \subsetneq \mathcal{L}(P)$

$$\min_{x \in [-3,6]} \frac{1}{4}x + \sin(x)$$



Open sets

- In general, MP cannot directly model problems involving sets which are not closed in the usual topology (such as e.g. open intervals)
- The reason is that the minimum/maximum of a non-closed set might not exist
- E.g. what is $\min_{x \in (0,1)} x$? Since $(0,1)$ has no minimum (for each $\delta \in (0,1)$, $\frac{\delta}{2} < \delta$ and is in $(0,1)$), the question has no answer
- *This is why the MP language does not allow writing constraints that involve the $<$, $>$ and $\neq$ relations*
- Sometimes, problems involving open sets can be reformulated exactly to problems involving closed sets

Best fit hyperplane 1

- Consider the following problem:

  Given m points $p_1, \ldots, p_m \in \mathbb{R}^n$, find the hyperplane $w_1 x_1 + \cdots + w_n x_n = w_0$ minimizing the piecewise linear form $f(p, w) = \sum_{i \in P} | \sum_{j \in N} w_j p_{ij} - w_0 |$

- Mathematical programming formulation:
  1. **Sets**: $P = \{1, \ldots, m\}$, $N = \{1, \ldots, n\}$
  2. **Parameters:** $\forall i \in P \; p_i \in \mathbb{R}^n$
  3. **Decision variables**: $\forall j \in N \; w_j \in \mathbb{R}$, $w_0 \in \mathbb{R}$
  4. **Objective**: $\min_w f(p, w)$
  5. **Constraints**: none
- Trouble: $w = 0$ is the obvious, trivial solution of no interest
- We need to enforce a constraint $(w_1, \ldots, w_n, w_0) \neq (0, \ldots, 0)$
- *Bad news:* $\mathbb{R}^{n+1} \setminus \{(0, \ldots, 0)\}$ *is not a closed set*

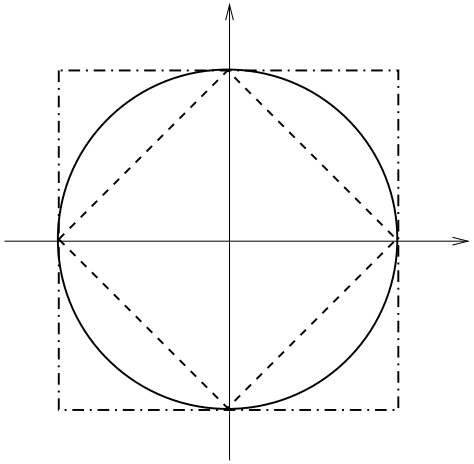
Best fit hyperplane 2

- We can implicitly impose such a constraint by transforming the objective function to $\min_w \frac{f(p,w)}{||w||}$ (for some norm $||\cdot||$)
- This implies that w is nonzero but the feasible region is $\mathbb{R}^{n+1}$, which is both open and closed
- **Obtain fractional objective — difficult to solve**
- Suppose $\mathbf{w}^* = (w^*, w_0^*) \in \mathbb{R}^{n+1}$ is an optimal solution to the above problem
- Then for all $d > 0$, $f(d\mathbf{w}^*, p) = df(\mathbf{w}^*, p)$
- Hence, it suffices to determine the optimal *direction* of $\mathbf{w}^*$, because the actual vector length simply scales the objective function value
- Can impose constraint $||w|| = 1$ and recover original objective
- *Solve reformulated problem*:

$$\min\{f(w,p) \mid ||w|| = 1\}$$

Best fit hyperplane 3

- The constraint $||w|| = 1$ is a *constraint schema*: we must specify the norm
- Some norms can be reformulated to linear constraints, some cannot
- max-norm (l_∞) 2-sphere (square), Euclidean norm (l_2) 2-sphere (circle), abs-norm (l_1) 2-sphere (rhombus)



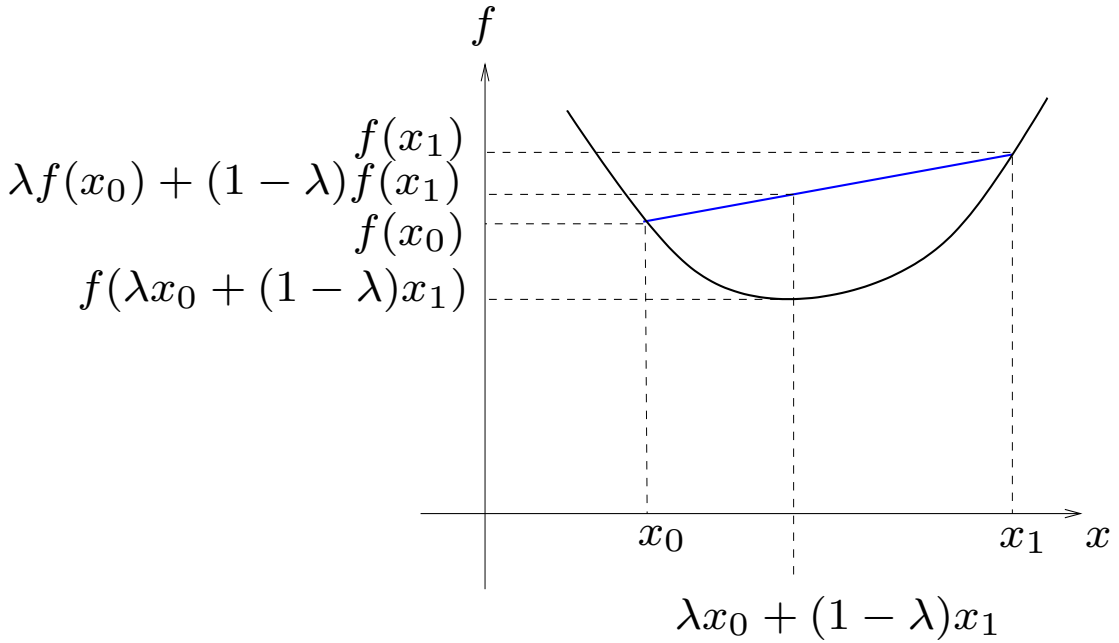
- max- and abs-norms are piecewise linear, they can be linearized exactly by using binary variables (see later)

Convexity

- A function $f(x)$ is convex if the following holds:

$$\forall x_0, x_1 \in \text{dom}(f) \quad \forall \lambda \in [0,1]$$

$$f(\lambda x_0 + (1-\lambda)x_1) \quad \leq \quad \lambda f(x_0) + (1-\lambda) f(x_1)$$



- A set $C \subseteq \mathbb{R}^n$ is convex if $\forall x_0, x_1 \in C, \lambda \in [0,1]\ (\lambda x_0 + (1-\lambda)x_1 \in C)$
- If $g : \mathbb{R}^m \to \mathbb{R}^n$ are convex, then $\{x \mid g(x) \leq 0\}$ is a convex set
- If f, g are convex, then every local optimum of $\min_{g(x) \leq 0} f(x)$ is global
- **A local NLP solver suffices to solve the NLP to optimality**

Convexity in practice

- Recognizing whether an arbitrary function is convex is an undecidable problem
- For some functions, however, this is possible
  - Certain functions are *known* to be convex (such as all affine functions, cx^{2n} for $n \in \mathbb{N}$ and $c \geq 0$, $\exp(x)$, $-\log(x)$)
  - Norms are convex functions
  - The sum of two convex functions is convex
- Application of the above rules repeatedly sometimes works (for more information, see Disciplined Convex Programming [Grant et al. 2006])
- *Warning*: problems involving integer variables are in general not convex; however, if the objective function and constraints are *convex forms*, we talk of *convex MINLPs*

Recognizing convexity 1

Consider the following mathematical program

$$\begin{aligned} \min_{x,y\in[0,10]} \quad & 8x^2 - 17xy + 10y^2 \\ x - y &\geq 1 \\ xy &\geq \frac{1}{x} \end{aligned}$$

- Objective function and constraints contain nonconvex term xy
- Constraint 2 is $\leq \frac{1}{x}$; the function $\frac{1}{x}$ is convex (only in $x \geq 0$) but constraint sense makes it *reverse convex*
- Is this problem convex or not?

Recognizing convexity 2

- The objective function can be written as $(x, y)^\mathsf{T} Q(x, y)$ where $Q = \begin{pmatrix} 8 & -8 \\ -9 & 10 \end{pmatrix}$
- The eigenvalues of Q are $9 \pm \sqrt{73}$ (both positive), hence the Hessian of the objective is positive definite, hence **the objective function is convex**
- The affine constraint $x - y \geq 1$ is **convex by definition**
- $xy \leq \frac{1}{x}$ can be reformulated as follows:
  1. Take logarithms of both sides: $\log xy \leq \log \frac{1}{x}$
  2. Implies $\log x + \log y \geq -\log x \Rightarrow -2 \log x - \log y \leq 0$
  3. $-\log$ is a convex function, sum of convex functions is convex, *convex* $\leq$ *affine* is a convex constraint
- Thus, the constraint **is convex**

Recognizing convexity 3

```
model;
var x <= 10, >= -10;
var y <= 10, >= -10;
minimize f: 8*x^2 -17*x*y + 10*y^2;
subject to c1: x-y >= 1;
subject to c2: x^2*y >= 1;
option solver_msg 0;
printf "solving with sBB (couenne)\n";
option solver boncouenne; solve > /dev/null;
display x,y;
printf "solving with local NLP solver (ipopt)\n";
option solver ipopt; let x := 0; let y := 0;
solve > /dev/null; display x,y;
```

Get same solution $(1.5, 0.5)$ from COUENNE and IPOPT

Total Unimodularity

- A matrix A is Totally Unimodular (TUM) if all invertible square submatrices of A have determinant ± 1

  Thm.

  If A is TUM, then all vertices of the polyhedron

  $$\{x \geq 0 \mid Ax \leq b\}$$

  have integral components

- *Consequence*: if the constraint matrix of a given MILP is TUM, then it suffices to solve the relaxed LP to get a solution for the original MILP

- **An LP solver suffices to solve the MILP to optimality**

TUM in practice 1

- If A is TUM, A^{T} and $(A|I)$ are TUM
- *TUM Sufficient conditions*. An $m \times n$ matrix A is TUM if:
  1. for all $i \leq m$, $j \leq n$ we have $a_{ij} \in \{0, 1, -1\}$;
  2. each column of A contains at most 2 nonzero coefficients;
  3. there is a partition R_1, R_2 of the set of rows such that for each column j, $\sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij} = 0$.
- Example: take $R_1 = \{1, 3, 4\}$, $R_2 = \{2\}$

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 1 \\ -1 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 & -1 & 0 \end{pmatrix}$$

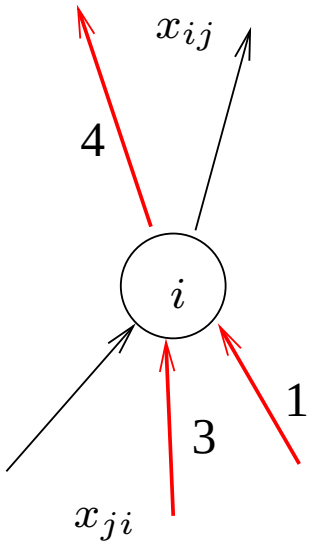
TUM in practice 2

- Consider digraph $G = (V, A)$ with nonnegative variables $x_{ij} \in \mathbb{R}_+$ defined on each arc
- *Flow constraints* $\boxed{\forall i \in V \quad \sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = b_i}$ yield a TUM matrix (partition: $R_1 =$ all rows, $R_2 = \emptyset$ — prove it)
- Maximum flow problems can be solved to integrality by simply solving the continuous relaxation with an LP solver
- *The constraints of the set covering problem do not form a TUM. To prove this, you just need to find a counterexample*

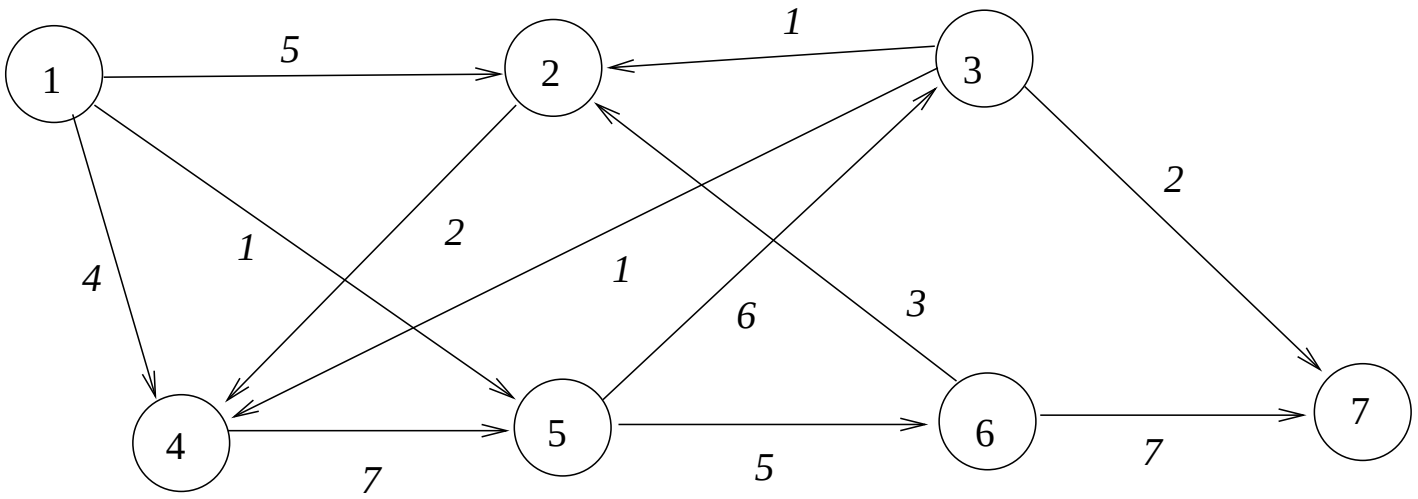
Maximum flow problem

Given a network on a directed graph $G = (V, A)$ with a source node s, a destination node t, and integer capacities u_{ij} on each arc (i, j). We have to determine the maximum *integral* amount of material flow that can circulate on the network from s to t. The variables $x_{ij} \in \mathbb{Z}$, defined for each arc (i, j) in the graph, denote the number of flow units.

$$\left.\begin{array}{rll} & \max_x & \displaystyle\sum_{(s,i)\in A} x_{si} \\ \forall\, i \leq V,\ \begin{array}{l} i \neq s \\ i \neq t \end{array} & \displaystyle\sum_{(i,j)\in A} x_{ij} = \sum_{(j,i)\in A} x_{ji} & \\ \forall (i,j) \in A & 0 \leq x_{ij} \leq u_{ij} & \\ \forall (i,j) \in A & x_{ij} \in \mathbb{Z} & \end{array}\right\}$$



Max Flow Example 1



arc capacities as shown in italics: find the maximum flow between node $s = 1$ and $t = 7$

Max Flow: MILP formulation

- **Sets**: $V = \{1, \ldots, n\}$, $A \subseteq V \times V$
- **Parameters**: $s, t \in V$, $u : A \to \mathbb{R}_+$
- **Variables**: $x : A \to \mathbb{Z}_+$
- **Objective**: $\max \sum_{(s,i) \in A} x_{si}$
- **Constraints**: $\forall i \in V \setminus \{s, t\} \quad \sum_{(i,j) \in A} x_{ij} = \sum_{(j,i) \in A} x_{ji}$

Max Flow: .mod file

```
# maxflow.mod
param n integer, > 0, default 7;
param s integer, > 0, default 1;
param t integer, > 0, default n;
set V := 1..n;
set A within {V,V};
param u{A} >= 0;

var x{(i,j) in A} >= 0, <= u[i,j], integer;

maximize flow : sum{(s,i) in A} x[s,i];

subject to flowcons{i in V diff {s,t}} :
  sum{(i,j) in A} x[i,j] = sum{(j,i) in A} x[j,i];
```

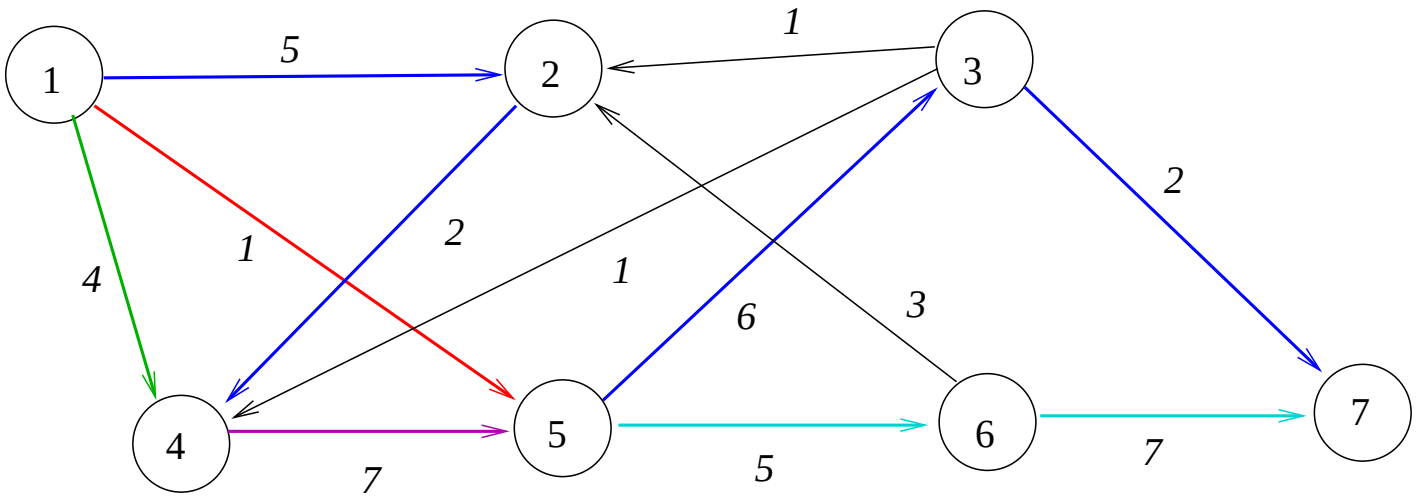
Max Flow: .dat file

```
# maxflow.dat
param : A   : u :=
        1 2   5
        1 4   4
        1 5   1
        2 4   2
        3 2   1
        3 4   1
        3 7   2
        4 5   7
        5 3   6
        5 6   5
        6 2   3
        6 7   7 ;
```

Max Flow: .run file

```
# maxflow.run
model maxflow.mod;
data maxflow.dat;
option solver_msg 0;
option solver cplex;
solve;
for {(i,j) in A : x[i,j] > 0} {
  printf "x[%d,%d] = %g\n", i,j,x[i,j];
}
display flow;
```

Max Flow: MILP solution



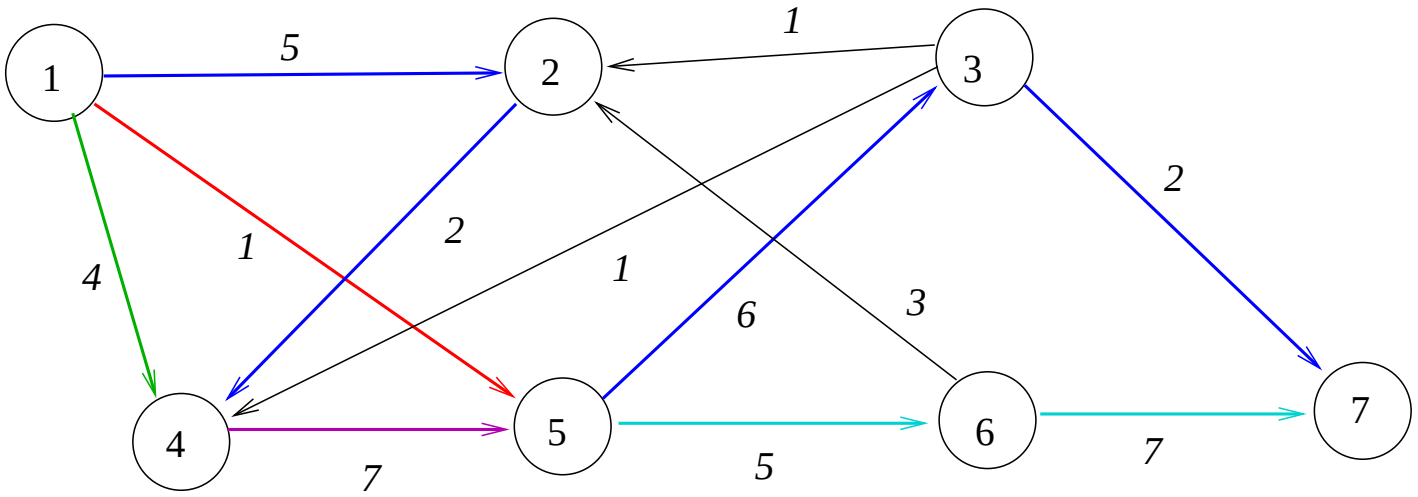
1unit of flow
2 units of flow
4 units of flow
5 units of flow
6 units of flow
maximum flow = 7

```
x[1,2] = 2
x[1,4] = 4
x[1,5] = 1
x[2,4] = 2
x[3,7] = 2
```

```
x[4,5] = 6
x[5,3] = 2
x[5,6] = 5
x[6,7] = 5
flow = 7
```

Max Flow: LP solution

Relax integrality constraints (take away `integer` keyword)



1unit of flow — 5 units of flow

2 units of flow — 6 units of flow

4 units of flow — maximum flow = 7

Get the same solution

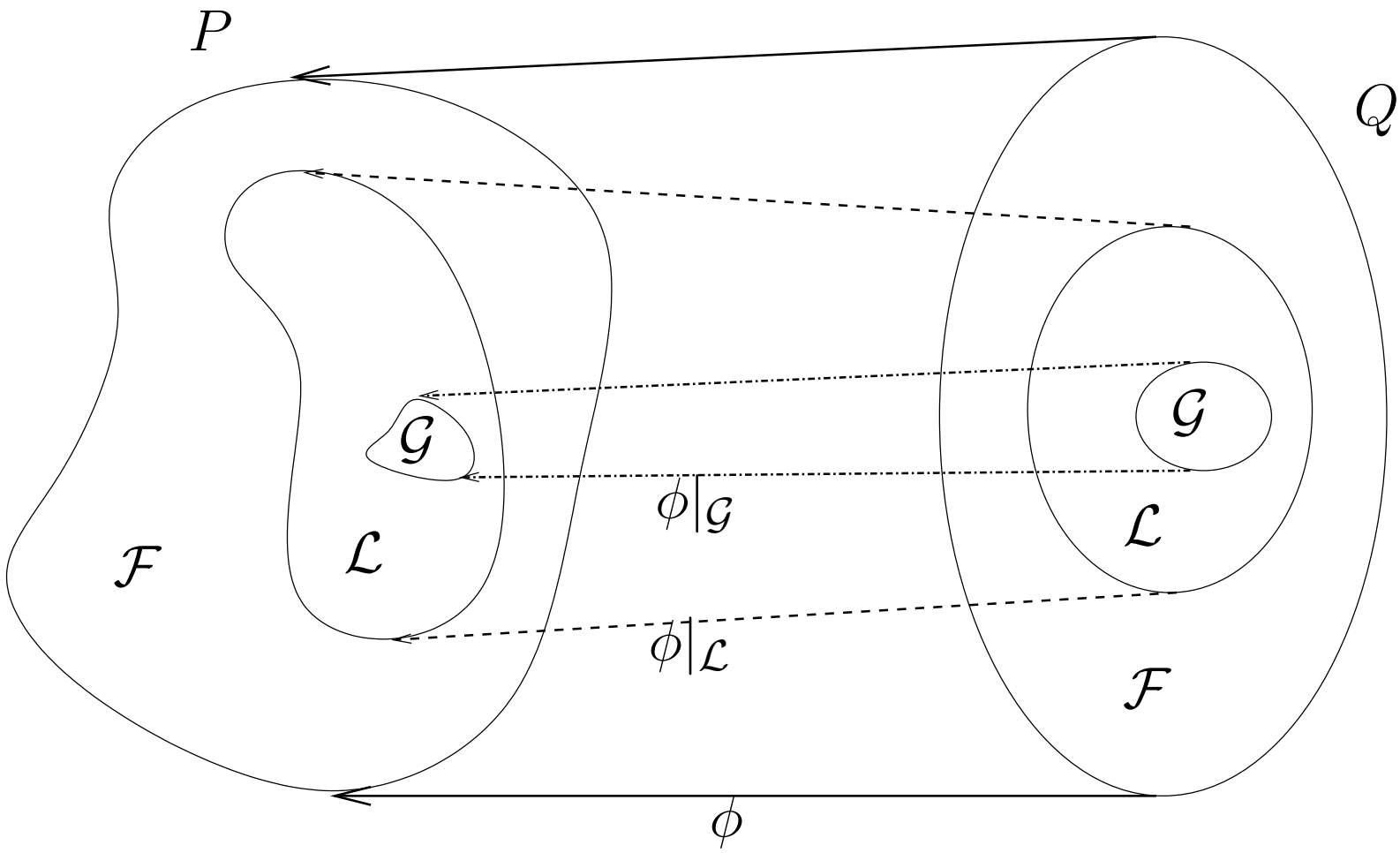
Reformulations

Reformulations

If problems P, Q are related by a computable function f through the relation $f(P, Q) = 0$, Q is an *auxiliary problem* with respect to P.

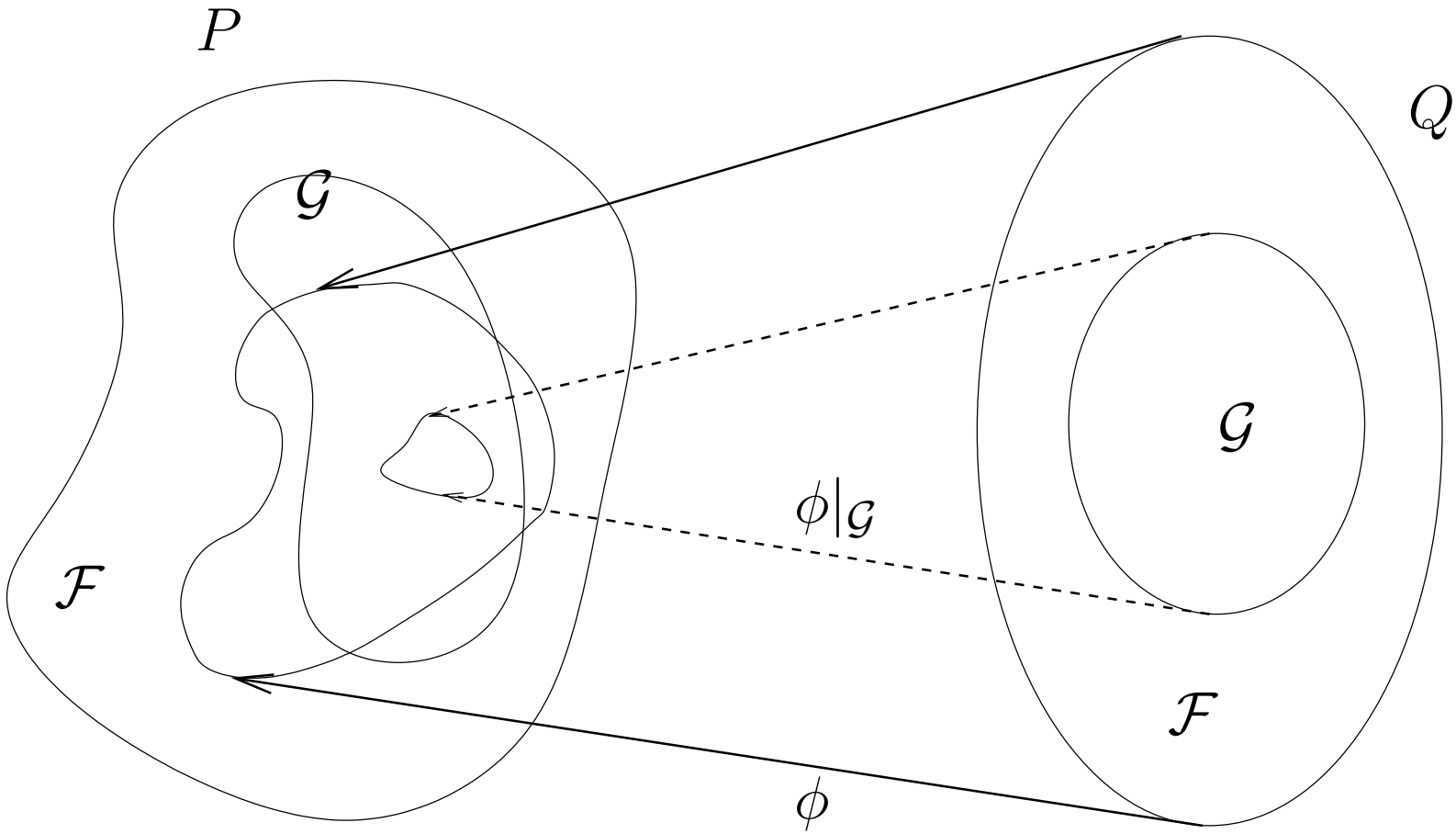
- **Opt-reformulations**: preserve all optimality properties
- **Narrowings**: preserve some optimality properties
- **Relaxations**: provide bounds to the optimal objective function value
- **Approximations**: formulation Q depending on a parameter k such that "$\lim_{k \to \infty} Q(k)$" is an opt-reformulation, narrowing or relaxation

Opt-reformulations



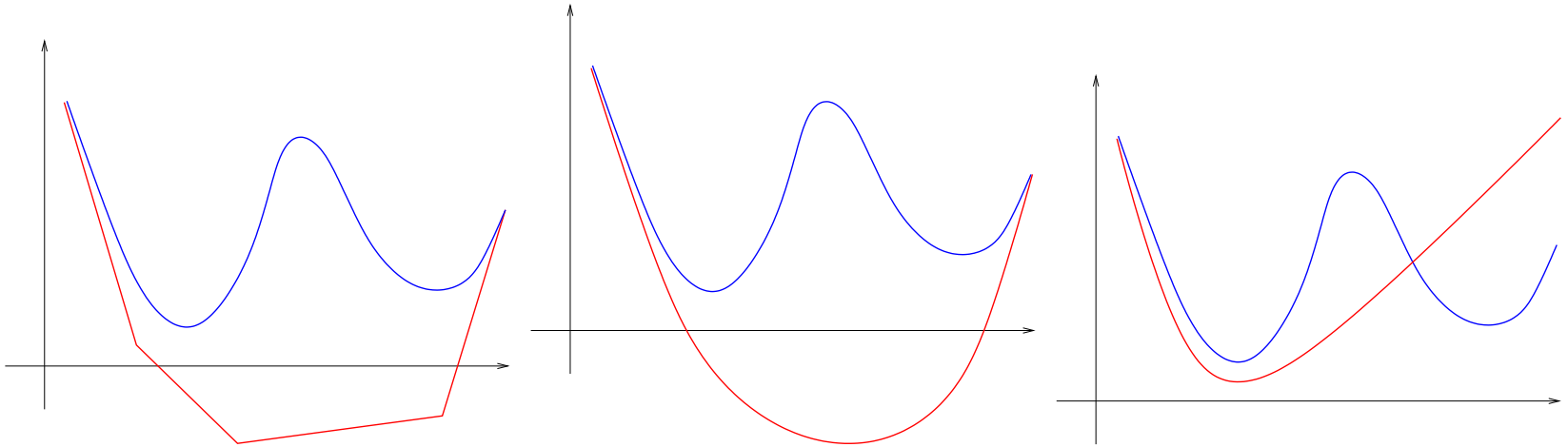
Main idea: if we find an optimum of Q, we can map it back to the same type of optimum of P, and for all optima of P, there is a corresponding optimum in Q. Also known as *exact reformulation*

Narrowings



Main idea: if we find a global optimum of Q, we can map it back to a global optimum of P. There may be optima of P without a corresponding optimum in Q.

Relaxations



A problem Q is a relaxation of P if the globally optimal value of the objective function $\min f_Q$ of Q is a lower bound to that of P.

Approximations

Q is an *approximation* of P if there exist: (a) an auxiliary problem Q^* of P; (b) a sequence $\{Q_k\}$ of problems; (c) an integer $\ell > 0$; such that:

1. $Q = Q_\ell$
2. $\forall$ objective f^* in Q^* there is a sequence of objectives f_k of Q_k converging uniformly to f^*;
3. $\forall$ constraint $l_i^* \leq g_i^*(x) \leq u_i^*$ of Q^* there is a sequence of constraints $l_i^k \leq g_i^k(x) \leq u_i^k$ of Q_k such that g_i^k converges uniformly to g_i^*, l_i^k converges to l_i^* and u_i^k to u_i^*

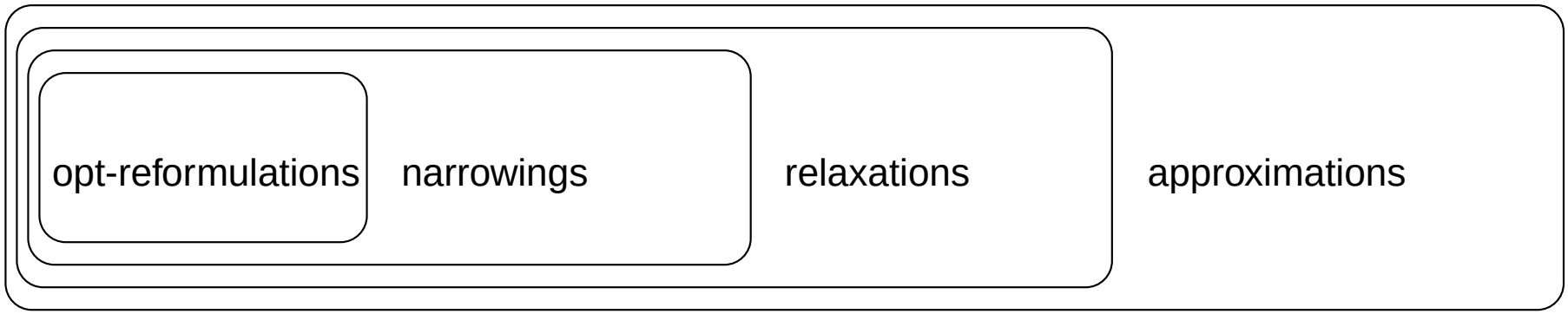
There can be approximations to opt-reformulations, narrowings, relaxations.

$Q_1, Q_2, Q_3, \ldots Q_\ell, \ldots \longrightarrow Q^*$ ⟨auxiliary problem of⟩ P

approximation of P

Fundamental results

- Opt-reformulation, narrowing, relaxation, approximation are all transitive relations
- *An approximation of any type of reformulation is an approximation*
- A reformulation consisting of opt-reformulations, narrowings, relaxations is a relaxation
- *A reformulation consisting of opt-reformulations and narrowings is a narrowing*
- A reformulation consisting of opt-reformulations is an opt-reformulation

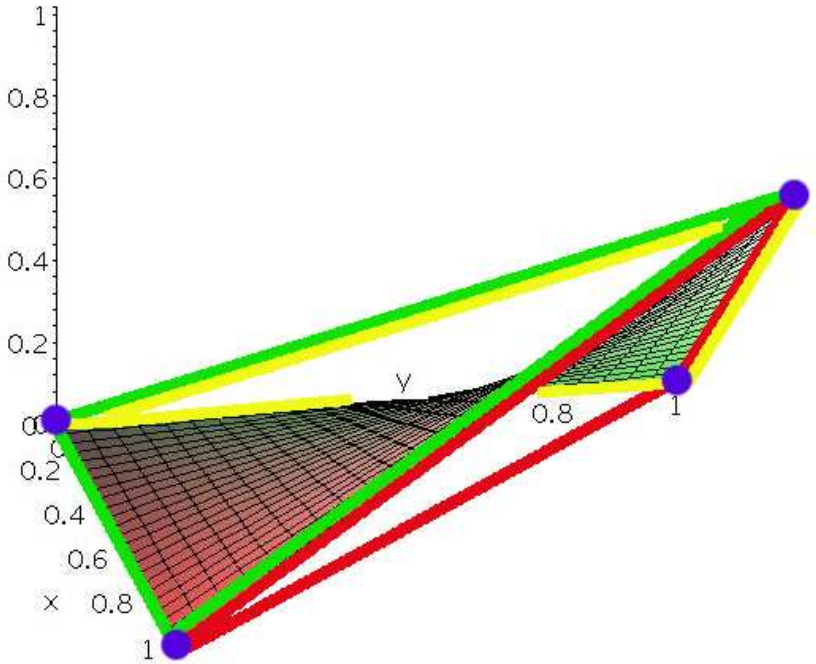


Reformulations in practice

- Reformulations are used to transform problems into equivalent (or related) formulations which are somehow "better"
- **Basic reformulation operations**:
  1. adding / deleting variables / constraints
  2. replacing a term with another term (e.g. a product xy with a new variable w)

Product of binary variables

- Consider binary variables x, y and a cost c to be added to the objective function only of $xy = 1$
- $\Rightarrow$ Add term cxy to objective
- Problem becomes mixed-integer (some variables are binary) and nonlinear
- Reformulate "xy" to MILP form (PRODBIN reform.):



- replace xy by z
- add $z \leq y$, $z \leq x$
  $z \geq 0$, $z \geq x + y - 1$
- $x, y \in \{0, 1\} \Rightarrow$
  $z = xy$

Application to the KNP

- In the RHS of the KNP's distance constraints we have $4y_iy_j$, where y_i, y_j are binary variables
- We apply ProdBin (call the added variable w_{ij}):

$$
\left.
\begin{array}{rrcl}
\min & \sum_{i \in N} y_i & & \\
\forall i \in N & \sum_{k \in D} x_{ik}^2 & = & 4y_i \\
\forall i \in N, j \in N : i < j & \sum_{k \in D} (x_{ik} - x_{jk})^2 & \geq & 4w_{ij} \\
\forall i \in N, j \in N : i < j & w_{ij} & \leq & y_i \\
\forall i \in N, j \in N : i < j & w_{ij} & \leq & y_j \\
\forall i \in N, j \in N : i < j & w_{ij} & \geq & y_i + y_j - 1 \\
\forall i \in N, j \in N : i < j & w_{ij} & \in & [0, 1] \\
\forall i \in N, k \in D & x_{ik} & \in & [-2, 2] \\
\forall i \in N & y_i & \in & \{0, 1\}
\end{array}
\right\}
$$

- Still a MINLP, but fewer nonlinear terms
- Still numerically difficult (2h CPU time to find $k^*(2) \geq 5$)

Product of bin. and cont. vars.

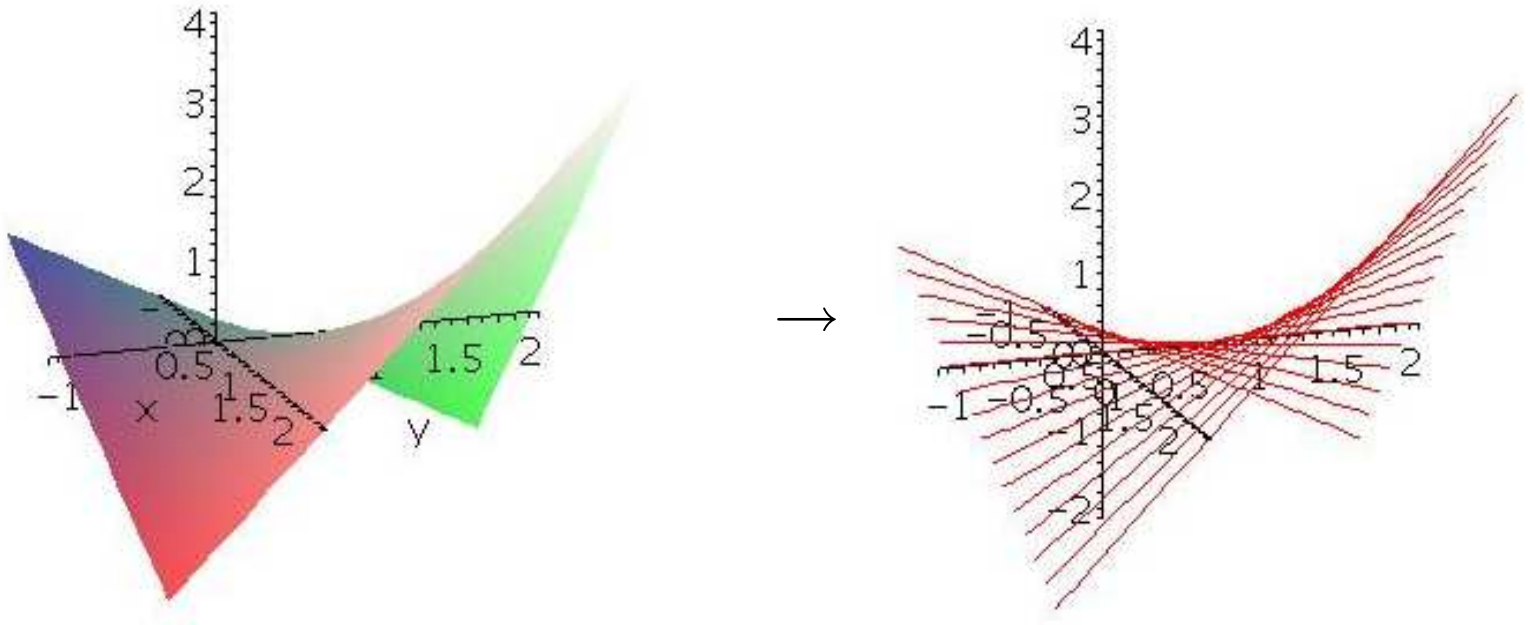
- PRODBINCONT reformulation
- Consider a binary variable x and a continuous variable $y \in [y^L, y^U]$, and assume product xy is in the problem
- Replace xy by an added variable w
- Add constraints:

$$\begin{aligned} w &\leq y^U x \\ w &\geq y^L x \\ w &\leq y + y^L(1-x) \\ w &\geq y - y^U(1-x) \end{aligned}$$

- Exercise 1: show that PRODBINCONT is indeed a reformulation
- Exercise 2: show that if $y \in \{0, 1\}$ then PRODBINCONT is equivalent to PRODBIN

Prod. cont. vars.: approximation

- BILINAPPROX approximation
- Consider $x \in [x^L, x^U], y \in [y^L, y^U]$ and product xy
- Suppose $x^U - x^L \leq y^U - y^L$, consider an integer $d > 0$
- Replace $[x^L, x^U]$ by a finite set
  $D = \{x^L + (i-1)\gamma \mid 1 \leq i \leq d\}$, where $\gamma = \frac{x^U - x^L}{d-1}$



BILINAPPROX

- Replace the product xy by a variable w
- Add binary variables z_i for $i \leq d$
- Add assignment constraint for z_i's

$$\sum_{i \leq d} z_i = 1$$

- Add definition constraint for x:

$$x = \sum_{i \leq d} (x^L + (i-1)\gamma) z_i$$

(x takes exactly one value in D)

- Add definition constraint for w

$$w = \sum_{i \leq d} (x^L + (i-1)\gamma) z_i y \qquad (2)$$

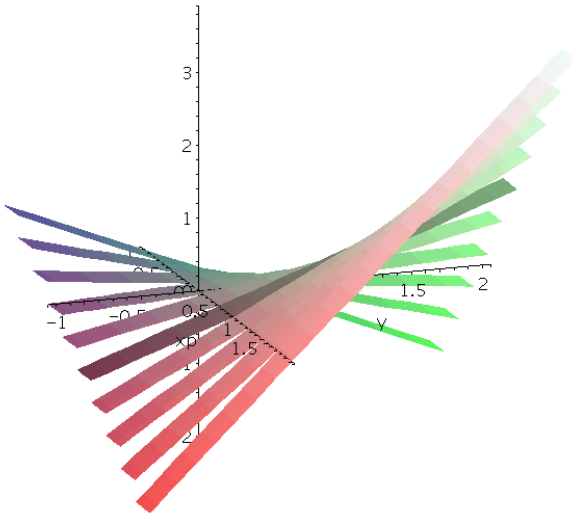
- Reformulate the products $z_i y$ via PRODBINCONT

BILINAPPROX2

BILINAPPROX2: problem P has a term xy where $x \in [x^L, x^U], y \in [y^L, y^U]$ are continuous; assume $x^U - x^L \leq y^U - y^L$

1. choose integer $k > 0$; add $q = \{q_i \mid 0 \leq i \leq k\}$ to $\mathcal{P}$ so that $q_0 = x^L, q_k = x^U$, $q_i < q_{i+1}$ for all i
2. add continuous variable $w \in [w^L, w^U]$ (computed from ranges of x, y by interval arithmetic) and replace term xy by w
3. add binary variables z_i for $1 \leq i \leq k$ and constraint $\sum_{i \leq k} z_i = 1$
4. for all $1 \leq i \leq k$ add constraints:

$$\left.\begin{array}{c} \sum_{j=1}^{k} q_{j-1} z_j \leq x_i \leq \sum_{j=1}^{k} q_j z_j \\ \frac{q_i + q_{i-1}}{2} y - (w^U - w^L)(1 - z_i) \leq w \leq \frac{q_i + q_{i-1}}{2} y + (w^U - w^L)(1 - z_i), \end{array}\right\}$$

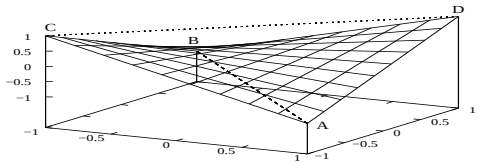


$k \to \infty$: get identity opt-reformulation

Relaxing bilinear terms

- RRLTRELAX: quadratic problem P with terms $x_i x_j$ $(i < j)$ and constrs $Ax = b$ (x can be bin, int, cont); perform opt-reformulation RRLT first:
  1. add continuous variables w_{ij} (let $w_i = (w_{i1}, \ldots, w_{1n})$)
  2. replace product $x_i x_j$ with w_{ij} (for all i, j)
  3. add the *reduced RLT* (RRLT) system $\forall k \; Aw_k - bx_k = 0$
  4. find a partition (B, N) of basic/nonbasic variables of $\forall k \; Aw_k = 0$ such that B corresponds to variables with smallest range
  5. for all $(i, j) \in N$ add constraints $w_{ij} = x_i x_j$ (†)
- then replace nonlinear constraints (†) with McCormick's envelopes

$$
\begin{aligned}
w_{ij} &\geq \max\{x_i^L x_j + x_j^L x_i - x_i^L x_j^L, x_i^U x_j + x_j^U x_i - x_i^U x_j^U\} \\
w_{ij} &\leq \min\{x_i^U x_j + x_j^L x_i - x_i^U x_j^L, x_i^L x_j + x_j^U x_i - x_i^L x_j^U\}
\end{aligned}
$$



- The effect of RRLT is that of using information in $Ax = b$ to eliminate some of the problematic product terms (those with indices in B)

Linearizing the l_∞ norm

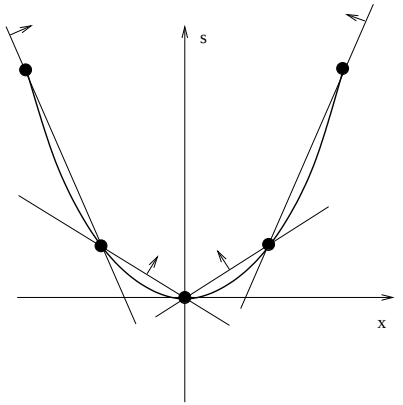
- INFNORM [Coniglio et al., MSc Thesis, 2007]. P has vars $x \in [-1,1]^d$ and constr. $||x||_\infty = 1$, s.t. $x^* \in \mathcal{F}(P) \leftrightarrow -x^* \in \mathcal{F}(P)$ and $f(x^*) = f(-x^*)$.
  1. $\forall k \leq d$ add binary var u_k
  2. delete constraint $||x||_\infty = 1$
  3. add constraints:

$$\begin{aligned} \forall k \leq d \quad x_k &\geq 2u_k - 1 \\ \sum_{k \leq d} u_k &= 1. \end{aligned}$$

- Narrowing INFNORM(P) cuts away all optima having $\max_k |x_k| = 1$ with $x_k < 1$ for all $k \leq d$

Approximating squares

- INNERAPPROXSQ: P has a continuous variable $x \in [x^L, x^U]$ and a term x^2 appearing as a convex term in an objective or constraint
  1. add parameters $n \in \mathbb{N}$, $\varepsilon = \frac{x^U - x^L}{n-1}$, $\bar{x}_i = x^L + (i-1)\varepsilon$ for $i \leq n$
  2. add a continuous variable $w \in [w^L, w^U]$, where $w^L = 0$ if $x^L x^U \leq 0$ or $\min((x^L)^2, (x^U)^2)$ otherwise and $w^U = \max((x^L)^2, (x^U)^2)$
  3. replace all occurrences of term x^2 with w
  4. add constraints
     $$\forall i \leq n \quad w \geq (\bar{x}_i + \bar{x}_{i-1})x - \bar{x}_i \bar{x}_{i-1}.$$



$n \to \infty$: get identity opt-reformulation

- Replace convex term by piecewise linear approximation

Conditional constraints

- Suppose $\exists$ a binary variable y and a constraint $g(x) \leq 0$ in the problem
- We want $g(x) \leq 0$ to be active iff $y = 1$
- Compute maximum value that $g(x)$ can take over all x, call this M
- Write the constraint as:

$$g(x) \leq M(1 - y)$$

- This sometimes called the "big M" modelling technique

Example:

Can replace constraint (2) in BILINAPPROX as follows:

$$\forall i \leq d \quad -M(1 - z_i) \leq w - (x^L + (i-1)\gamma)y \leq M(1 - z_i)$$

where M s.t. $w - (x^L + (i-1)\gamma)y \in [-M, M]$ for all w, x, y

Symmetry

Example

Consider the problem

$$\left.\begin{array}{rrcl} \min & x_1 + x_2 & & \\ & 3x_1 + 2x_2 & \geq & 1 \\ & 2x_1 + 3x_2 & \geq & 1 \\ & x_1, x_2 & \in & \{0,1\} \end{array}\right\}$$

AMPL code:

```
set J := 1..2;
var x{J} binary;
minimize f: sum{j in J} x[j];
subject to c1: 3*x[1] + 2*x[2] >= 1;
subject to c2: 2*x[1] + 3*x[2] >= 1;
option solver cplex;
solve;
display x;
```

The solution (given by CPLEX) is $x_1 = 1$, $x_2 = 0$

If you swap x_1 with x_2, you obtain the same problem, with swapped constraints

Hence, $x_1 = 0$, $x_2 = 1$ is *also* an optimal solution!

Permutations

- We can represent permutations by maps $\mathbb{N} \to \mathbb{N}$
- The permutation of our example is $\begin{pmatrix} 1 & 2 \\ \downarrow & \downarrow \\ 2 & 1 \end{pmatrix}$
- Permutations are usually written as *cycles*: e.g. for a permutation $\begin{pmatrix} 1 & 2 & 3 \\ \downarrow & \downarrow & \downarrow \\ 3 & 1 & 2 \end{pmatrix}$, which sends $1 \to 3$, $3 \to 2$ and $2 \to 1$, we write $(1, 3, 2)$ to mean $1 \to 3 \to 2 (\to 1)$
- The permutation of our example is $(1, 2)$ — a cycle of *length* 2 (also called a *transposition*, or *swap*)

Cycles

- Cycles can be multiplied together, but the multiplication is not commutative: $(1,2,3)(1,2) = (1,3)$ and $(1,2)(1,2,3) = (2,3)$
- The *identity* permutation e fixes all $\mathbb{N}$
- Notice $(1,2)(1,2) = e$ and $(1,2,3)(1,3,2) = e$, so $(1,2) = (1,2)^{-1}$ and $(1,3,2) = (1,2,3)^{-1}$
- Cycles are *disjoint* when they have no common element
- *Thm.* Disjoint cycles commute
- *Thm.* Every permutation can be written uniquely (up to order) as a product of disjoint cycles
- For each permutation π, let $\Gamma(\pi)$ be the set of its disjoint cycles

Groups

- A *group* is a set G together with a multiplication operation, an inverse operation, and an identity element $e \in G$, such that:
  1. $\forall g, h \in G\ (gh \in G)$ (multiplication closure)
  2. $\forall g \in G\ (g^{-1} \in G)$ (inverse closure)
  3. $\forall f, g, h \in G\ ((fg)h = f(gh))$ (associativity)
  4. $\forall g \in G\ (eg = g)$ (identity)
  5. $\forall g \in G\ (g^{-1}g = e)$ (inverse)
- The set $\{e\}$ is a group (denoted by 1) called the *trivial group*
- The set of all permutations over $\{1, \ldots, n\}$ is a group, called the *symmetric group of order* n, and denoted by S_n
- For all $B \subseteq \{1, \ldots, n\}$ define S_B as the symmetric group over the symbols of B

Generators

- Given any subset $T \subseteq S_n$, the smallest group containing the permutations in T is the *group generated by* T, denoted by $\langle T \rangle$
- For example, if $T = \{(1,2),(1,2,3)\}$, then $\langle T \rangle$ is $\{(1),(1,2),(1,3),(2,3),(1,2,3),(1,3,2)\} = S_3$
- For any $n \in \mathbb{N}$, $\langle (1,\ldots,n) \rangle$ is the *cyclic group of order* n, denoted by C_n
- C_n is commutative, whereas S_n is not
- Commutative groups are also called *abelian*
- *Thm.* $\langle (1,2),(1,\ldots,n) \rangle = \langle (i,i+1) \mid 1 \leq i < n \rangle = S_n$

Subgroups and homomorphisms

- A *subgroup* of a group G is a subset H of G which is also a group (denoted by $H \leq G$); e.g. $C_3 = \{e, (1,2,3), (1,3,2)\}$ is a subgroup of S_3
- Given two groups G, H, a map $\phi : G \to H$ such that $\forall f, g \in G \ (\ \phi(fg) = \phi(f)\phi(g)\ )$ is a *homomorphism*
- Ker$\phi = \{g \in G \mid \phi(g) = e\}$ is the *kernel* of ϕ (Ker$\phi \leq G$)
- Im$\phi = \{h \in H \mid \exists g \in G \ (h = \phi(g))\}$ is the *image* of ϕ (Im$\phi \leq H$)
- If ϕ is injective and surjective (i.e. if Ker$\phi = 1$ and Im$\phi = H$), then ϕ is an *isomorphism*, denoted by $G \cong H$
- *Thm.*[Lagrange] For all groups G and $H \leq G$, $|H|$ divides $|G|$
- *Thm.*[Cayley] Every finite group is isomorphic to a subgroup of S_n for some $n \in \mathbb{N}$

Normal subgroups

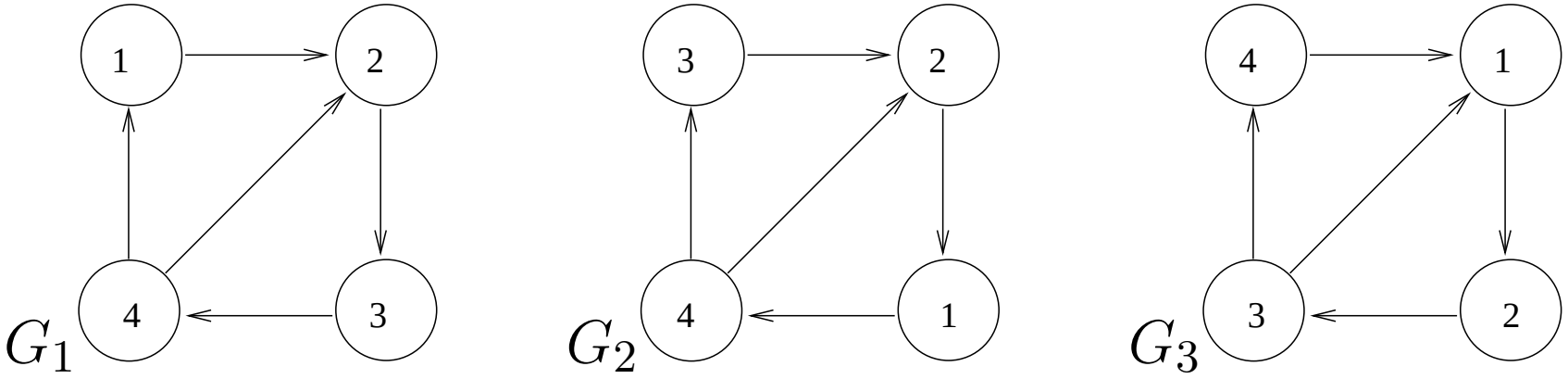
- Let $H \leq G$; for all $g \in G$, $gH = \{gh \mid h \in H\}$ and $Hg = \{hg \mid h \in H\}$ are in general *subsets* (not necessarily subgroups) of G, and in general $gH \neq Hg$
- If $\forall g \in G\ (gH = Hg)$ then H is a *normal subgroup* of G, denoted by $H \lhd G$ (e.g. $C_3 \lhd S_3$)
- If $H \lhd G$, then $\{gH \mid g \in G\}$ is denoted by G/H and has a group structure with multiplication $(fH)(gH) = (fg)H$, inverse $(gH)^{-1} = (g^{-1})H$ and identity $eH = H$
- For every group homomorphism ϕ, Ker$\phi \lhd G$ and $G/\text{Ker}\phi \cong \text{Im}\phi$

Group actions

- Given a group G and a set X, the *action* of G on X is a set of mappings $\alpha_g : X \to X$ for all $g \in G$, such that $\alpha_g(x) = (gx) \in X$ for all $x \in X$
- Essentially, the action of G on X is the definition of what happens to $x \in X$ when g is applied to it
- For example, if $X = \mathbb{R}^n$ and $G = S_n$, a possible action of G on X is given by gx being the vector x with components permuted according to g (e.g. if $x = (0.1, -2, \sqrt{2})$ and $g = (1, 2)$, then $gx = (-2, 0.1, \sqrt{2})$)
- *Convention*: left multiplication if x is a column vector ($\alpha_g(x) = gx$), right if x is a row vector ($\alpha_g(x) = xg$): treat g as a matrix
- For $Y \subseteq X$ and $H \leq G$, $HY = \{hy \mid h \in H, y \in Y\}$ and $YH = \{yh \mid h \in H, y \in Y\}$ are the left and right *orbits* of Y in H (also denoted $\text{orb}(Y, H)$); notice $\text{orb}(Y, H) \subseteq X$
- $\text{stab}(Y, G) = \{g \in G \mid gY \subseteq Y\}$ is the *stabilizer* of Y in G; notice $\text{stab}(Y, G) \leq G$

Groups and graphs

- Given a digraph $G = (V, A)$ with $V = \{v_1, \ldots, v_n\}$, the action of $\pi \in S_n$ on G is the natural action of π on V
- π is a *graph automorphism* if $\forall (i, j) \in A \ (\pi(i), \pi(j)) \in A$
- For example:

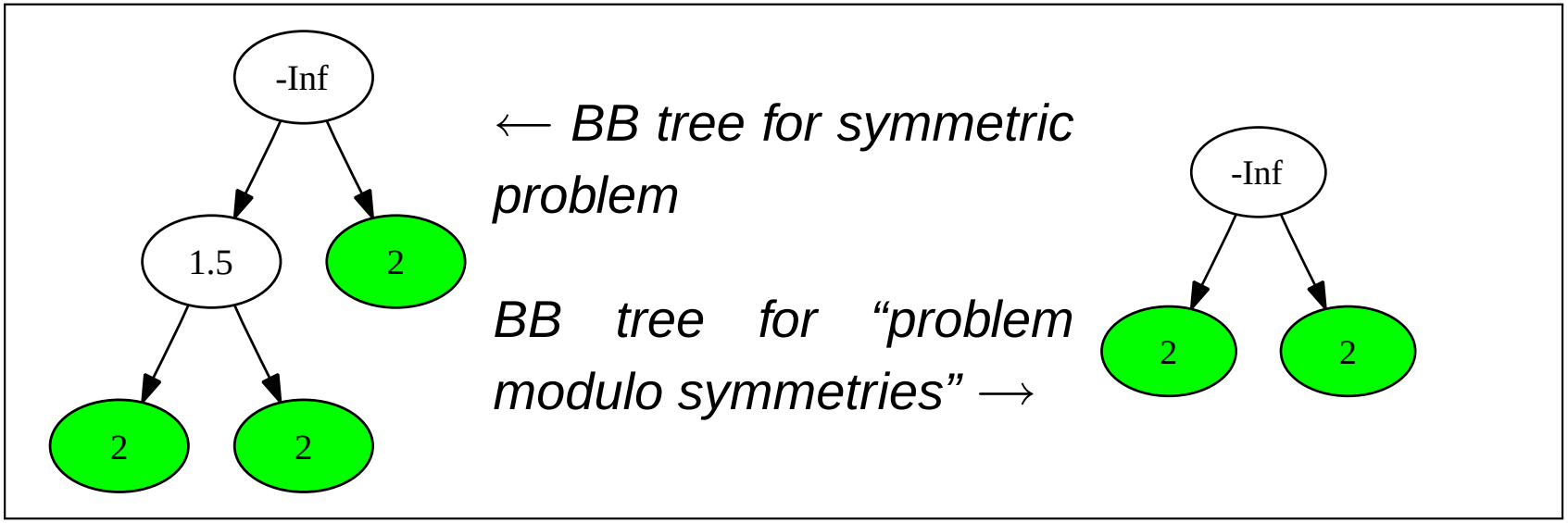


- $G_2 = (1, 3)G_1$ is a graph automorphism of G_1
- $G_3 = (1, 2, 3, 4)G_1$ is not an automorphism of G_1: $(4, 2) \in A$ but $(\pi(4), \pi(2)) = (1, 3) \notin A$
- The *automorphism group* of G_1 is $\langle e, (1, 3) \rangle \cong C_2$ (denoted by Aut(G_1))

Back to MP: Symmetries and BB

- Symmetries are **bad** for Branch-and-Bound techniques: many branches will contain (symmetric) optimal solutions and therefore will not be pruned by bounding

  $\Rightarrow$ *deep and large BB trees*



- How do we write a "mathematical programming formulation modulo symmetries"?

Solution symmetries

- The set of solutions of the following problem:

$$\begin{array}{lccccccccc}
\min & x_{11} & +x_{12} & +x_{13} & +x_{21} & +x_{22} & +x_{23} & & & \\
 & x_{11} & +x_{12} & +x_{13} & & & & \geq & 1 \\
 & & & & x_{21} & +x_{22} & +x_{23} & \geq & 1 \\
 & x_{11} & & & +x_{21} & & & \geq & 1 \\
 & & x_{12} & & & +x_{22} & & \geq & 1 \\
 & & & x_{13} & & & +x_{23} & \geq & 1
\end{array}$$

  is $\mathcal{G}(P) = \{(0,1,1,1,0,0),(1,0,0,0,1,1),(0,0,1,1,1,0),$
  $(1,1,0,0,0,1),(1,0,1,0,1,0),(0,1,0,1,0,1)\}$

- $G^* = \text{stab}(\mathcal{G}(P), S_n)$ is the *solution group* **(variable permutations keeping $\mathcal{G}(P)$ fixed)**
- For the above problem, G^* is
  $\langle (2,3)(5,6),(1,2)(4,5),(1,4)(2,5)(3,6)\rangle \cong D_{12}$
- For all $x^* \in \mathcal{G}(P)$, $G^*x^* = \mathcal{G}(P) \Rightarrow \exists$ only 1 orbit $\Rightarrow \exists$ only *one* solution in $\mathcal{G}(P)$ (modulo symmetries)
- How do we find G^* before solving P?

Formulation symmetries

- The cost vector $c^{\mathsf{T}} = (1, 1, 1, 1, 1, 1)$ is fixed by all (column) permutations in S_6
- The vector $b = (1, 1, 1, 1, 1)$ is fixed by all (row) permutations in S_5
- Consider P's constraint matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

- Let $\pi \in S_6$ be a column permutation such that $\exists$ a row permutation $\sigma \in S_5$ with $\sigma(A\pi) = A$
- Then permuting the variables/columns in P according to π does not change the problem formulation (*the constraint order is not important*)

The formulation group

- For a MILP with binary variables only,

$$G_P = \{\pi \in S_n \mid c\pi = c \wedge \exists \sigma \in S_m \ (\sigma A\pi = A \wedge \sigma b = b)\}$$

  is called the *formulation group* of P

- In the example above, we get $G_P \cong D_{12} \cong G^*$

  Thm.

  $G_P \leq G^*$.

- Result can be extended to all MILPs [Margot 02, 03, 07]

Symmetries in MINLPs

- **Consider a MINLP P**

$$\left.\begin{array}{rrcl} \min & f(x) & & \\ & g(x) & \leq & 0 \\ & x & \in & X. \end{array}\right\} \tag{3}$$

where the set X may contain integrality constraints on x

- For a row permutation $\sigma \in S_m$ and a column permutation $\pi \in S_n$, we define $\sigma P \pi$ as follows:

$$\left.\begin{array}{rrcl} \min & f(x\pi) & & \\ & \sigma g(x\pi) & \leq & 0 \\ & x\pi & \in & X. \end{array}\right\} \tag{4}$$

- Define $G_P = \{\pi \in S_n \mid \exists \sigma \in S_m \ (\sigma P \pi = P)\}$

Representing $g(x\pi)$

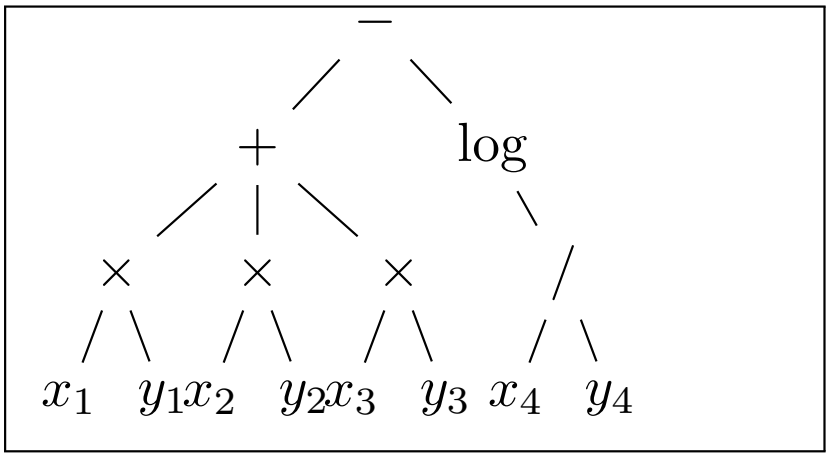
- In the linear case, writing $Ax\pi$ is easy — how do we deal with $g(x\pi)$?

  How do we decide whether $g_i(x) = g_h(x\pi)$ for $i, h \leq m$?

- **Answer**: consider the *expression DAG* (DAG=Directed Acyclic Graph) representation of g

  $$\sum_{i=1}^{3} x_i y_i - \log(x_4/y_4)$$

  List of expressions $\equiv$ expression DAG sharing variable leaf nodes

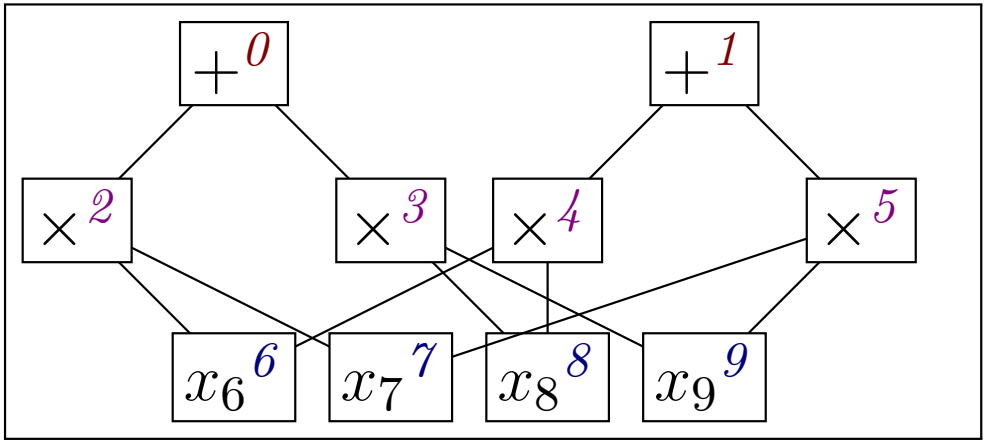


- *Every function $g : \mathbb{R}^n \to \mathbb{R}^m$ is represented by a DAG whose leaf nodes are variables and constants and whose intermediate nodes are mathematical operators*

- Look for relationships between the DAGs representing $g(x)$ and $\sigma g(x\pi)$

Example

$$\begin{aligned} \mathsf{c}_0 : x_6x_7 + x_8x_9 &= 1 \\ \mathsf{c}_1 : x_6x_8 + x_7x_9 &= 1 \end{aligned}$$



- G_{DAG} = **group of automorphisms of expression DAG fixing: (a) root node set having same constr. direction and coeff. (constraint permutations), (b) operators with same label and rank and (c) leaf node set (variable permutations)**

$$G_{\mathsf{DAG}} = \langle (45)(67)(89),\ (23)(68)(79),\ (01)(24)(35)(78) \rangle$$

- G_P is the projection of G_{DAG} to variable indices $\langle (6,7)(8,9), (6,8)(7,9), (7,8) \rangle \cong D_8$

Node colors

- Let $D_P = (\mathcal{V}, \mathcal{A})$ be the union of all objective and constraint DAGs in the MINLP (a.k.a *the DAG of* P)
- *Colors on the DAG nodes* $\mathcal{V}$ *are used to identify those subsets of nodes which can be permuted* (e.g. variable and operator nodes can't be permuted)

1. **Root nodes (i.e. constraints) can be permuted if they have the same RHS**
2. **Operator nodes (including root nodes) can be permuted if they have the same DAG rank and label; if an operator node is non-commutative, then the order of the children node must be maintained**
3. **Constant nodes can be permuted if they have the same DAG rank level and value**
4. **Variable nodes can be permuted if they have the same bounds and integrality constraints**

- The relation $(u \sim v \iff u, v$ have the same color$)$ is an *equivalence relation* on V (reflexive, symmetric, transitive)
- $\sim$ partitions $\mathcal{V}$ into a disjoint union $\mathcal{V}/\sim$ of *equivalence classes* $V_1, \ldots, V_p$

MINLP formulation groups

- Let P be a MINLP and $D = (\mathcal{V}, \mathcal{A})$ be the DAG of P
- Let G_{DAG} be the group of automorphisms of D that fix each color class in $\mathcal{V}/\sim$
- Define $\phi : G_{\text{DAG}} \to S_n$ by $\phi(\pi) =$projection of π on variable indices; then
  Thm.
  ϕ is a group homomorphism and Im$\phi \cong G_P$
- Hence can find G_P by computing Imϕ
- Although the complexity status (**P**/**NP**-complete) of the GRAPH ISOMORPHISM problem is currently unknown, `nauty` is a practically efficient software for computing G_{DAG}
- **So now we have G_P, how do we write "P modulo G_P"?**

Symmetry-breaking reformulation

- Consider our first example P:

$$\left.\begin{array}{rrcl} \min & x_1 + x_2 & & \\ & 3x_1 + 2x_2 & \geq & 1 \\ & 2x_1 + 3x_2 & \geq & 1 \\ & x_1, x_2 & \in & \{0,1\} \end{array}\right\}$$

- P has $\mathcal{G}(P) = \{(0,1),(1,0)\}$, $G^* = \langle (1,2) \rangle \cong C_2$ and $G_P = G^*$
- The orbit $G_P\{(0,1)\}$ is the whole of $\mathcal{G}(P)$
- We look for a reformulation of P where only *one* representative of each orbit is feasible
- Let Q be the reformulation of P consisting of P with the added constraint $x_1 \leq x_2$
- We have $\mathcal{G}(Q) = \{(0,1)\}$ and $G^* = G_Q = 1$

Breaking orbital symmetries 1

- Every group $G \leq S_n$ acting on the variable indices $N = \{1, \ldots, n\}$ partitions N into *disjoint orbits* (all subsets of N)
- *This follows from the equiv. rel.* $i \sim j \Leftrightarrow \exists g \in G\ (g(i) = j)$
- Let Ω be the set of *nontrivial* orbits ($\omega \in \Omega \iff |\omega| > 1$)
- *Thm.* G acts transitively on each of its orbits
- This means that $\forall \omega \in \Omega\ \forall i \neq j \in \omega\ \exists g \in G\ (g(i) = j)$
- **Applied to MP, if i, j are distinct variable indices belonging to the same orbit of G_P acting on N, then there is $\pi \in G_P$ sending x_i to x_j**
- Pick $x \in \mathcal{G}(P)$; if P is bounded, for all $\omega \in \Omega\ \exists i \in \omega$ s.t. x_i is a component having minimum value over all components of x
- By theorem above, $\exists \pi \in G_P$ sending x_i to $x_{\min \omega}$
- Hence $\bar{x}\pi$ is s.t. $\bar{x}_{\min \omega}$ is minimum over all other components of $\bar{x}$, and since $G_P \leq G^*$, $\bar{x} \in \mathcal{G}(P)$

Breaking orbital symmetries 2

- Thus, for all $\omega \in \Omega$ there is at least one optimal solution of P which is feasible w.r.t. the constraints $\forall j \in \omega \; (x_{\min \omega} \leq x_j)$
- Such constraints are called (orbit-based) *symmetry breaking constraints* (SBCs)
- Adding these SBCs to P yields a reformulation Q of P of the narrowing type (prove it!)
- *Thm.* If $g^\omega(x) \leq 0$ are SBCs for each orbit ω with "appropriate properties", then $\forall \omega \in \Lambda \; (g^\omega(x) \leq 0)$ are also SBCs
- Thus we can combine orbit-based SBCs for "appropriate properties"
- **Yields narrowings with fewer symmetric optima**

Breaking the symmetric group

- The above SBCs work with any group G_P, but their extent is limited (they may not break all that many symmetries)
- If we find $\Lambda' \subseteq \Lambda$ such that $\forall \omega \in \Lambda'$ the action of G_P on ω is S_ω, then there are much tighter SBCs
- For all $\omega \in \Lambda'$ let $\omega^- = \omega \setminus \{\max \omega\}$ and for all $j \in \omega^-$ let j^+ be the successor of j in ω
- The following are valid SBCs:

$$\forall \omega \in \Lambda' \; \forall j \in \omega^- \quad x_j \leq x_{j^+}$$

which are likely to break many more symmetries

The final attack on the KNP

Decision KNP

- Recall the binary KNP variables are used to count the number of spheres
- Suggests simply considering whether a *fixed number of spheres* can be placed around a central sphere in a kissing configuration, or not
- This is the *decision version* of the KNP (dKNP):

  Given positive integers n, d, can n unit spheres with disjoint interior be placed adjacent to a unit sphere centered at the origin of $\mathbb{R}^d$?

- Should eliminate binary variables, yielding a (nonconvex) NLP, simpler than the original MINLP
- In order to find the maximum value for n, we proceed by bisection on n and solve the dKNP repeatedly

The dKNP formulation

- Let $N = \{1, \ldots, n\}$; the following formulation P correctly models the dKNP:

$$\left.\begin{array}{rrcl} \max & 0 & & \\ \forall i \in N & \sum_{k \in D} x_{ik}^2 & = & 4 \\ \forall i \in N, j \in N : i < j & \sum_{k \in D} (x_{ik} - x_{jk})^2 & \geq & 4 \\ \forall i \in N, k \in D & x_{ik} & \in & [-2, 2] \end{array}\right\}$$

- If $\mathcal{F}(P) \neq \emptyset$ then the answer to the dKNP is YES, otherwise it is NO
- However, solving nonconvex feasibility NLPs is numerically *extremely difficult*

Feasibility tolerance

- We therefore add a *feasibility tolerance* variable α:

$$\left.\begin{array}{rrcl} \max & \alpha & & \\ \forall i \in N & \sum_{k \in D} x_{ik}^2 & = & 4 \\ \forall i \in N, j \in N : i < j & \sum_{k \in D} (x_{ik} - x_{jk})^2 & \geq & 4\alpha \\ \forall i \in N, k \in D & x_{ik} & \in & [-2, 2] \\ & \alpha & \geq & 0 \end{array}\right\}$$

- The above formulation Q is always feasible (why?)
- Much easier to solve than P, numerically
- Q also solves the dKNP: if the optimal α^* is ≥ 1 then the answer is YES, otherwise it is NO

The KNP group

- The dKNP turns out to have group S_d (i.e. each spatial dimension can be swapped with any other)
- Rewriting the distance constraints as follows:

$$\begin{aligned} ||x_i - x_j||^2 &= \sum_{k \in D} (x_{ik} - x_{jk})^2 \\ &= \sum_{k \in D} (x_{ik}^2 + x_{jk}^2 + 2x_{ik}x_{jk}) \\ &= 2(d + \sum_{k \in D} x_{ik}x_{jk}) \end{aligned}$$

  (for $i < j \leq n$) yields an opt-reformulation Q' of Q (prove it)
- The formulation group $G_{Q'}$ turns out to be $S_d \times S_n$ (pairs of distinct spatial dimensions can be swapped, and same for spheres), much larger than S_d
- Yields more effective SBC narrowings

Results

Instance D	Instance N	Solver	Without SBC Time	Without SBC Nodes	Without SBC Oi	Without SBC Gap	With SBC Time	With SBC Nodes	With SBC Oi	With SBC Gap
2	6	*Couenne*	4920.16	516000 **110150**	1	0.04%	100.19	14672	1	0%
2	6	*BARON*	1200*	45259 **6015**	1	10%	59.63	2785	131	0%
2	7	*Couenne*	7200†	465500 **127220**	1	41.8%	7200†	469780 **38693**	1	17.9%
2	7	*BARON*	10800	259800 **74419**	442	83.2%	16632	693162	208	0%

Oi: *Iteration where optimum was found*

†: *default Couenne CPU time limit*

*: *default BARON CPU time limit*

nodes: *total nodes* **still on tree**

Thus, we finally established by MP that $k^*(2) = 6$

Actually, solutions for $k^*(3)$ and $k^*(4)$ can be found by using MINLP heuristics (VNS)

The end