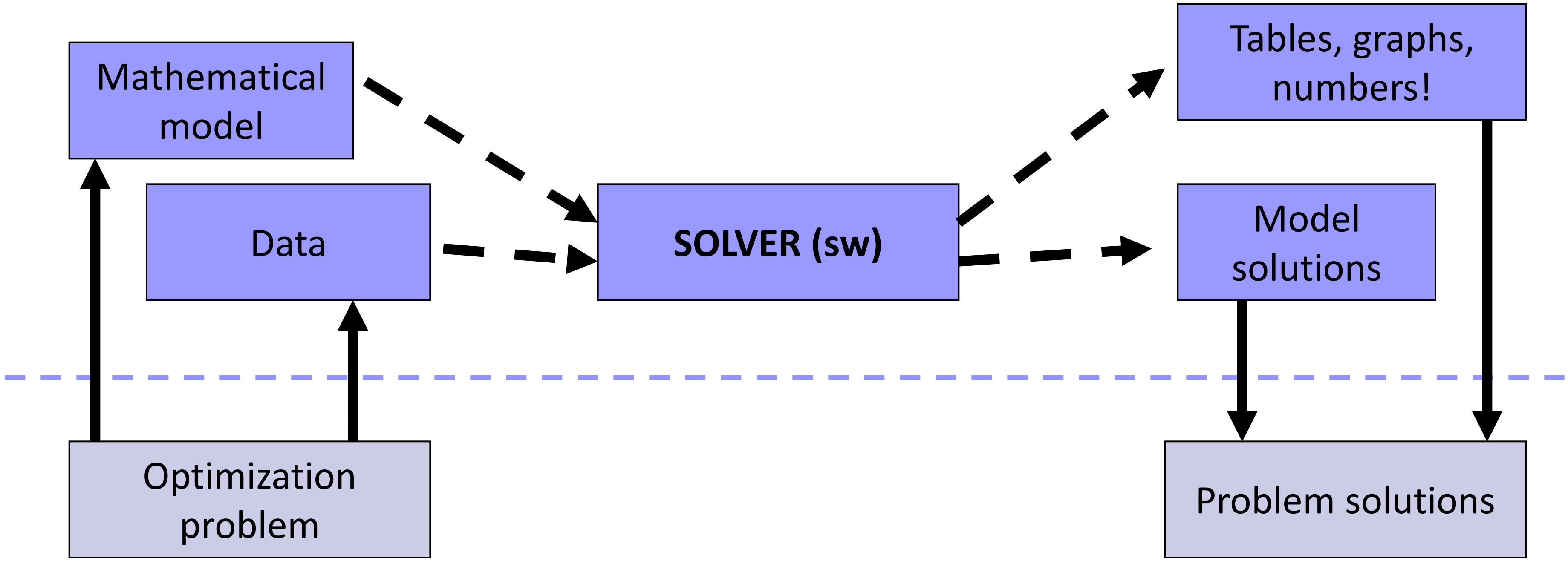
Solvers for Mathematical Programming

Solvers (optimizing engines)

A **solver** is a software application that takes the description of an optimization problem as input and provides the solution of the model (and related information) as output.

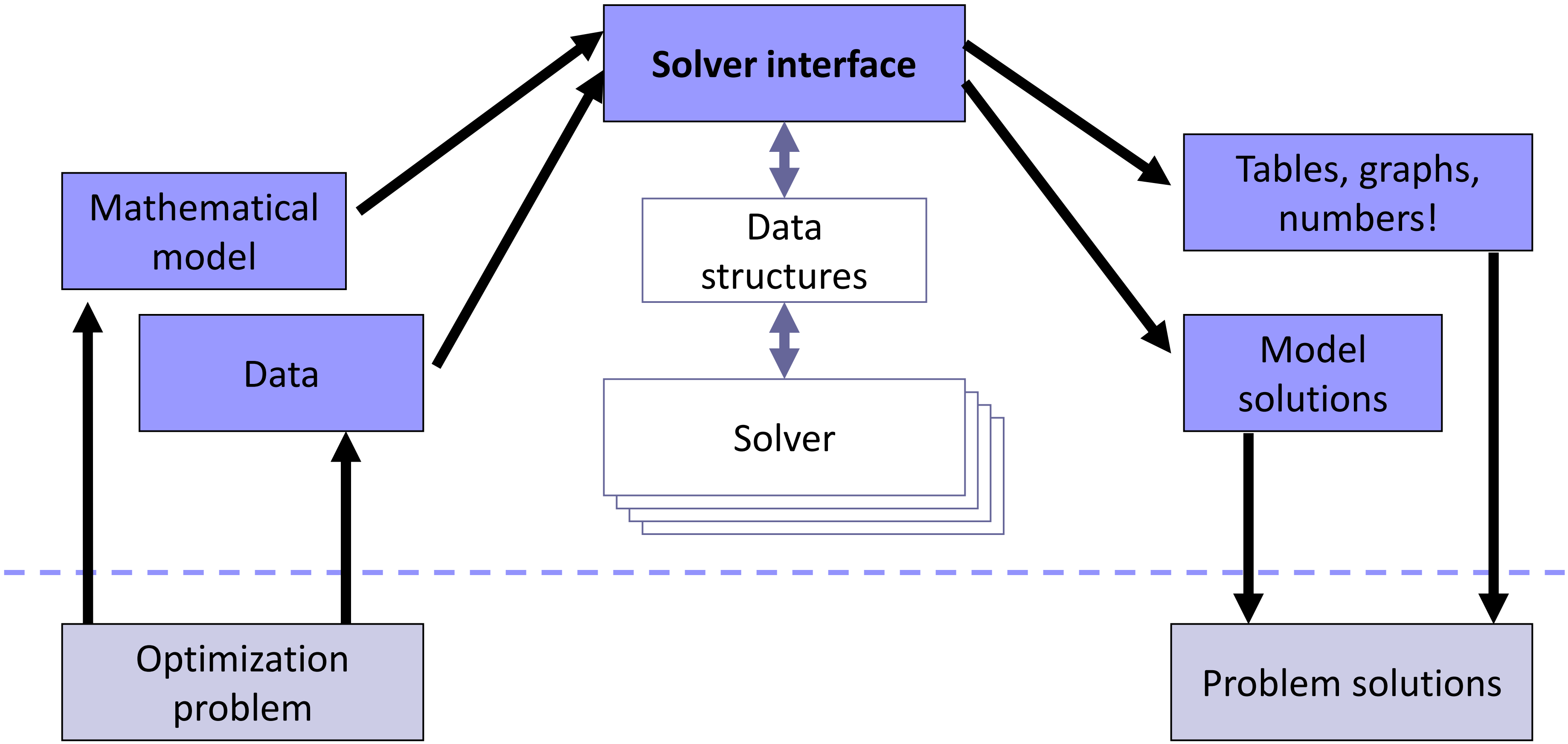


MILP solvers

- **M**ixed **I**nteger **Linear** **P**rogramming solvers most used in practice:
  - very efficient
  - numerical stability
  - easy to use or embed
- more than 1 000 000 000 speed-up in the last 20 years
  - hardware speed-up: x 1000
  - simplex improvements: x 1000
  - branch-and-cut improvement: x 1000
- Cplex, Gurobi, Xpress, Scip, Lindo, GLPK, Google OR Tools **etc.**

Solver interfaces

A solver can be accessed via **modelling languages** or **general-purpose-language libraries**



IBM Ilog Cplex

- One of the first MILP solvers
- Includes **state-of-the-art** technology
- One of the best solvers available (Gurobi, Xpress)
- Possible interfaces
  - Interactive optimizer
  - OPL / AMPL / ZIMPL … algebraic modelling language
  - **C – API libraries (Callable libraries)**
  - C++ libraries (Concert technologies)
  - Python APIs
  - **Python (with *DOcplex*)** / Java / .Net wrapper libraries
  - Matlab / Excel plugins

Accessing / Getting IBM Ilog Cplex

- Installed at LabTA/LabP140 and virtual *Lab24hr*

- From home
  - Getting your own free academic license (!)
  - Virtual *Lab24hr*
  - Accessing via ssh / X-windows (or similar)
  - Accessing Cplex via ssh

- See Getting access to Lab resources: instructions for details!

DOcplex – a Python interface to Cplex

- *IBM **D**ecision **O**ptimization **CPLEX** Modeling for Python*
- Built upon the Cplex Python APIs
- Exploits Python syntax to provide "easy" and flexible encoding of the mathematical model notation, e.g.:
  - Dictionaries for sets of variables
  - **`for`**…**`in`**…**`if`**… to encode "forall" quantifiers or sum indices
- Ideal for prototyping and integration into "modern" applications
- Documentation: docplex landing pages

  `https://pypi.org/project/docplex/`
  `https://ibmdecisionoptimization.github.io/docplex-doc/`
  - *Getting started with DOcplex*
  - *Mathematical Programming Modeling for Python using docplex.mp*
- Installation, e.g.

  `> pip install docplex` or
  `> conda install -c ibmdecisionoptimization docplex`

Basic commands

- **To enable Cplex Studio at Lab**: use Linux, run

  `> . cplex_env` (notice "dot blank")

- Use **DOcplex** with your favourite development environment for python. At Lab, we have

  visual studio code IDE
  jupyter notebook
  gedit + terminal

  … or use any other developing tool you like

- Importing `docplex` mathematical programming library

```
from docplex.mp.model import Model
```

- Full reference: `https://ibmdecisionoptimization.github.io/docplex-doc/`**`mp/docplex.mp.model.html`**

DOcplex basic functions: model definition

- Creating an "empty" model

```
m = Model(name="model name")
```

- Defining a variable

```
decvar1 = m.continuous_var()
decvar2 = m.integer_var()
decvar3 = m.binary_var()
```

  optional arguments

```
name="var name", lb=<lower bound>, ub=<upper bound>
```

  default values: `name = x1, x2` etc. ; `lb = 0 ; ub = +infinity`

- **Expressions** (functions of decision and/or usual variables)

```
expr = 6*decvar1 + Coeff*decvar2 - pow(3,2)*decvar3
```

- Creating a constraint

```
m.add_constraint(expr1 <= expr2)
```

  [or `>=` or `==`]

- Creating the objective function

```
m.minimize(expr)
```

  [or `maximize`]

DOcplex basic functions: model use

- Solving the model

  ```
  m.solve()
  ```
  optional arguments
  ```
  log_output = True|False , cplex_parameters = … , etc
  ```
- Checking status of the solution (*optimal, infeasible, unbounded …*):
  ```
  if m.solution  == None:
    print(“Problems! Status: ", m.get_solve_status())
  ```
- Printing the solution:
  ```
  if m.solution  != None:
     sol = m.solution
     m.print_solution()    #standard info (status, o.f. and vars’ value)
     print(sol[decvar])    #value of one variable
     print(sol.get_objective_value())#value of the objective function
  ```

DOcplex basic functions: export and debug

- Exporting the model in a text file (e.g., LP format)

```
m.export_as_lp(basename='filename', #default None (use model name)
                             path='path', hide_user_names=False)
```

- Exporting the solution in json format:

```
m.solution.export('filename')
```

- Exporting the solution in a string:

```
print(m.solution.to_string())
```

Resources: `example_farmer.py`

Exercise: implement the «diet», the «perfumes» etc. models [`example_*.py`]

Generalizing the model: data

- **Sets**: use, e.g., list, range …

```
Products  = ["tomato","potato"]
Origins = range(0,num_origins)
```

- **Parameters**: use, e.g., (multidim) list, dictionary …

```
unit_revenue = {"tomato": 6000, "potato": 7000}
orig_capacity = [50,70,30]
cost_matrix = [[6,8,3,2],[4,2,1,3],[4,2,6,5]]
cost_dict = {(i,j): random() for i in Orig for j in Dest}
```

Generalizing the model: decision variables

- Variables may be indexed over one or more sets
- Use **for … in … if …** to encode “forall” quantifiers

- Use, e.g., a dictionary having a tuple from the interested sets as index and decision variables as elements

```
x = {i: m.continuous_var(name='x({0})'.format(i), lb = 0,
                                       ub = None) for i in Products}
y = {(i,j): m.integer_var(name='y_{0}_{1}'.format(i,j))
         for i in I for j in J}
y1= {(i,j): m.integer_var(name='y_{0}_{1}'.format(i,j))
         for i in I for j in J if cost[i,j] <= cost_threshold}
```

- Use, e.g., a list (accessed by position, starts from 0)

```
xlist=[i: m.continuous_var(name='x') for i in Products]
```

Generalizing: expressions and constraints

- **Expressions** may contain **indexed sum**
- Use **`m.sum`** and `for` … `in` … `if` … to encode sum indices

```
m.minimize(m.sum(C[i]*xlist[i] for i in range(0,len(Products))))
cost=m.sum(c[i,j]*y1[i,j] for i in I for j in J if cost[i,j]<T)
```

- **Indexed constraints**: use **loops** to encode “forall” quantifiers

```
for i in I: #forall i in I such that i is capacitaded
  if capacitated[i]:
    m.add_constraint(m.sum(x[(i,j)] for j in J) <= capacity[i])
```

Resources: **`prodmix.py`**

Exercise: implement the «perfumes» and the «young money maker» models using the general prodmix.py model [`prodmix.py`]

Reporting the example to be implemented

One possible modeling schema: optimal production mix

- set I: resources $I = \{rose, lily, violet\}$
- set J: products $J = \{one, two\}$
- parameters D_i: availability of resource $i \in I$ e.g. $D_{rose} = 27$
- parameters P_j: unit profit for product $j \in J$ e.g. $P_{one} = 130$
- parameters Q_{ij}: amount of resource $i \in I$ required for each unit of product $j \in J$ e.g. $Q_{rose\ one} = 1.5$, $Q_{lily\ two} = 1$
- variables x_j: amount of product $j \in J$ x_{one}, x_{two}

$$
\begin{array}{llll}
\max & \sum_{j \in J} P_j x_j & & \\
s.t. & \sum_{j \in J} Q_{ij} x_j \leq D_i & & \forall \ i \in I \\
& x_j \in \mathbb{R}_+ \quad [\ \mathbb{Z}_+ \quad | \quad \{0,1\}\ ] & & \forall \ j \in J
\end{array}
$$

Reporting the example to be implemented

Example

A perfume firm produces two new items by mixing three essences: rose, lily and violet. For each decaliter of perfume *one*, it is necessary to use 1.5 liters of rose, 1 liter of lily and 0.3 liters of violet. For each decaliter of perfume *two*, it is necessary to use 1 liter of rose, 1 liter of lily and 0.5 liters of violet. 27, 21 and 9 liters of rose, lily and violet (respectively) are available in stock. The company makes a profit of 130 euros for each decaliter of perfume *one* sold, and a profit of 100 euros for each decaliter of perfume *two* sold. The problem is to determine the optimal amount of the two perfumes that should be produced.

$$\begin{array}{lrcrcrl}
\max & 130\,x_{one} & + & 100\,x_{two} & & & \text{objective function} \\
s.t. & 1.5\,x_{one} & + & x_{two} & \leq & 27 & \text{availability of rose} \\
 & x_{one} & + & x_{two} & \leq & 21 & \text{availability of lily} \\
 & 0.3\,x_{one} & + & 0.5\,x_{two} & \leq & 9 & \text{availability of violet} \\
 & x_{one} & , & x_{two} & \geq & 0 & \text{domains of the variables}
\end{array}$$

Generalizing the model: DOcplex shortcuts

- Indexed **variables**:

```
x=m.continuous_var_dict(keys=Products,lb=0,ub=None,name='x’)
xlist=m.continuous_var_list(keys=Products,name='x’)
y=m.integer_var_matrix(keys1=I, keys2=J, lb=0,ub=None,name='y’)
y=m.binary_var_cube(keys1=Set1,keys2=Set2,keys3=Set3,name='z’)
```

- Indexed **constraints**:

```
m.add_constraints( [x[0]>=1.1,x[1]>=0.7,x[2]>=0.5] ,
                   names=['minVeget','minMeat','minFruit'] )
m.add_constraints(
    m.sum(x[(i,j)] for j in J) <= capacity[i]   for i in I )
```

Resources: `mincostcover.py`

Exercise: solve the «emergency location» problem using `mincostcover.py`

Reporting the example to be implemented

One possible modeling schema: minimum cost covering

- set I: resources $I = \{V, M, F\}$
- set J: requests $J = \{proteins, iron, calcium\}$
- parameters C_i: unit cost of resource $i \in I$
- parameters R_j: requested amount of $j \in J$
- parameters A_{ij}: amount of request $j \in J$ satisfied by one unit of resource $i \in I$
- variables x_i: amount of resource $i \in I$

$$
\begin{aligned}
\min \quad & \sum_{i \in I} C_i x_i \\
s.t. \quad & \\
& \sum_{i \in I} A_{ij} x_i \geq R_j & \forall\, j \in J \\
& x_i \in \mathbb{R}_+ \; [\, \mathbb{Z}_+ \mid \{0,1\} \,] & \forall\, i \in I
\end{aligned}
$$

Reporting the example to be implemented

The diet problem

$$
\begin{array}{llllllllll}
\min & 4\,x_V & + & 10\,x_M & + & 7\,x_F & & & & \text{cost} \\
s.t. & 5\,x_V & + & 15\,x_M & + & 4\,x_F & \geq & 20 & & \text{proteins} \\
 & 6\,x_V & + & 10\,x_M & + & 5\,x_F & \geq & 30 & & \text{iron} \\
 & 5\,x_V & + & 3\,x_M & + & 12\,x_F & \geq & 10 & & \text{calcium} \\
 & x_V & , & x_M & , & x_F & \geq & 0 & & \text{domains of the variables}
\end{array}
$$

Emergency location: MILP model from covering schema

$$
\begin{array}{llllllllllllll}
\min & x_1 & + & x_2 & + & x_3 & + & x_4 & + & x_5 & + & x_6 & & \\
s.t. & & & & & & & & & & & & & \\
 & x_1 & + & x_2 & & & & & & & & & \geq 1 & \text{(cover zone 1)} \\
 & x_1 & + & x_2 & & & & & & & + & x_6 & \geq 1 & \text{(cover zone 2)} \\
 & & & & & x_3 & + & x_4 & & & & & \geq 1 & \text{(cover zone 3)} \\
 & & & & & x_3 & + & x_4 & + & x_5 & & & \geq 1 & \text{(cover zone 4)} \\
 & & & & & & & x_4 & + & x_5 & + & x_6 & \geq 1 & \text{(cover zone 5)} \\
 & & & x_2 & & & & & + & x_5 & + & x_6 & \geq 1 & \text{(cover zone 6)} \\
 & & & & & & & & & & & & & \\
 & x_1 & , & x_2 & , & x_3 & , & x_4 & , & x_5 & , & x_6 & \in \{0,1\} & \text{(domain)}
\end{array}
$$

Generalizing: separating model and data file

- **Data** (sets and parameters) can be read from an external source: a plain text file, a json file, a database, a spreadsheet etc.
- Model and data can live in separate domains (e.g., with differentiated access policy)
- Take advantage from available Python libraries (**`json`**, **`pandas`** etc. etc.)

Resources: `prodmix.ext.py` (read data from json file)
`farmer.json`

Exercise: solve the «perfumes» problem by only modifying the json file

Exercise

Solve the following problem:

We produce bottles of three types of wines (wine1, wine2 and wine3) using four types of grapes (grape1, grape2, grape3 and grape4). The unit profit per bottle of wine of the three types is respectively 21, 15 and 10 euros. The availability of grapes is respectively 100, 200, 50 and 150 units. A bottle of wine1 requires 1.5, 0.8, 1.0 and 0.3 units of grapes 1, 2, 3 and 4 respectively. A bottle of wine2 requires 1.0, 2.0, 0.5 and 1.1 units of grapes 1, 2, 3 and 4 respectively. A bottle of wine3 requires 1.7, 2.4 and 1.6 units of grapes 1, 2 and 4 respectively (and no grape3). Determine the production mix to maximize the profit.

[use `wines.json` with `pridmix.ext.py`]

Applications

- Transportation model
  - Basic model [transport_basic.py , transport_basic.json]
  - Remove expensive (over a parametrized threshold) links
  - Additional constraint 1: if the cost of link from *i* to *j* is at most *LowCost*, then the flow on this link should be at least *LowCostMinOnLink*
  - Additional constraint 2: destination *SpecialDestination* should receive at least *MinToSpecialDest* units from each origin, but for origin *SpecialOrigin*
  - Additional constraint 3: at least a *SignificantNumber* of origins significantly (no less than a *SignificantFraction* of the destination demand) supply each destination [transport_dict.py , transport_plus.json]

- Facility location with fixed costs
  - **Preprocess data** to define data-dependent big-M constants [facility_loc_basic.py]
  - Additional constraint: at most/least max/min number of open locations [facility_loc_plus.py]

Reporting the example to be implemented

One possible modeling schema: transportation

- set I: origins factories $I = \{A, B, C\}$
- set J: destinations stores $J = \{1, 2, 3, 4\}$
- parameters O_i: capacity of origin $i \in I$ factory production
- parameters D_j: request of destination $j \in J$ store request
- parameters C_{ij}: unit transp. cost from origin $i \in I$ to destination $j \in J$
- variables x_{ij}: amount to be transported from $i \in I$ to $j \in J$

$$
\begin{aligned}
\min \quad & \sum_{i \in I} \sum_{j \in J} C_{ij} x_{ij} \\
s.t. \quad & \\
& \sum_{i \in I} x_{ij} \geq D_j && \forall\, j \in J \\
& \sum_{j \in J} x_{ij} \leq O_i && \forall\, i \in I \\
& x_{ij} \in \mathbb{R}_+ \; [\; \mathbb{Z}_+ \mid \{0, 1\} \;] && \forall\, i \in I\; j \in J
\end{aligned}
$$

Reporting the example to be implemented

Modeling fixed costs: binary/boolean variables (linear)

- set I: potential locations
- parameters W, F_i, C_i, R_i, "large-enough" M (e.g. $M = \arg\max_{i \in I}\{W/C_i\}$)
- variables x_i: size (in 100 m^2) of the store in $i \in I$
- variables y_i: taking value 1 if a store is opened in $i \in I$ ($x_i > 0$), 0 otherwise

$$\max \sum_{i \in I} R_i x_i$$

$s.t.$

$$\sum_{i \in I} C_i x_i + F_i y_i \leq W \qquad \text{budget}$$

$$x_i \leq M y_i \quad \forall i \in I \qquad \text{BigM constraint / relate } x_i \text{ to } y_i$$

$$\sum_{i \in I} y_i \leq K \qquad \text{max number of stores}$$

$$x_i \in \mathbb{R}_+, \; y_i \in \{0,1\} \quad \forall i \in I$$

Lab organization: DOcplex, Cplex C APIs or what?

The course unit presents DOcplex and the Cplex C APIs (*Callable Libraries*) as tools for Lab Exercise Part I (implementation of a mathematical programming model)

Other tools may be used (Cplex Concert Technologies or OPL or Matlab connector or AMPL or Gurobi APIs etc.), to be **discussed and agreed** with the teacher

Follow the table to determine your tool! Next Lab classes will concern Cplex APIs or DOcplex or (assisted self-)learning agreed alternative tools (see *proposed exercises*)

Master Degree	Can&want C or C++	Can&want python	priority 1	priority 2	priority 3
Computer Science	Yes	Yes/No	Cplex C APIs		
	No	Yes	Cplex C APIs	DOcplex (with lists)	
	No	No	Cplex C APIs	DOcplex (with lists)	**agreed***
Others	Yes	Yes	C APIs or DOcplex (your choice)		**agreed**
	Yes	No	Cplex C APIs	**agreed**	
	No	Yes	DOcplex	**agreed**	
	No	No	**agreed**		
	using C APIs is appreciated!			***after convincing the teacher!**	

Cplex Callable Libraries

- C API towards *LP/QP/MIP/MIQP* algorithms
- Basic objects: **`Environment`** and **`Problem`**
- **`Environment`**: license, optimization parameters …
- **`Problem`**: contains problem information: variables, constraints …)
- (at least one) environment and problem must be created

```
CPXENVptr CPXopenCPLEX / CPXcloseCPLEX

CPXLPptr  CPXcreateprob / CPXfreeprob
```

Cplex API functions

- The two objects can be accessed (e.g. to add variables or constraints, or to solve a problem) via the functions provided by the API
- (Almost) all the API functions can be called as

```
int CPXfuncName (environment[,problem],...);
```

Error code (0=ok) **CPXgeterrorstring** returns a description of the error

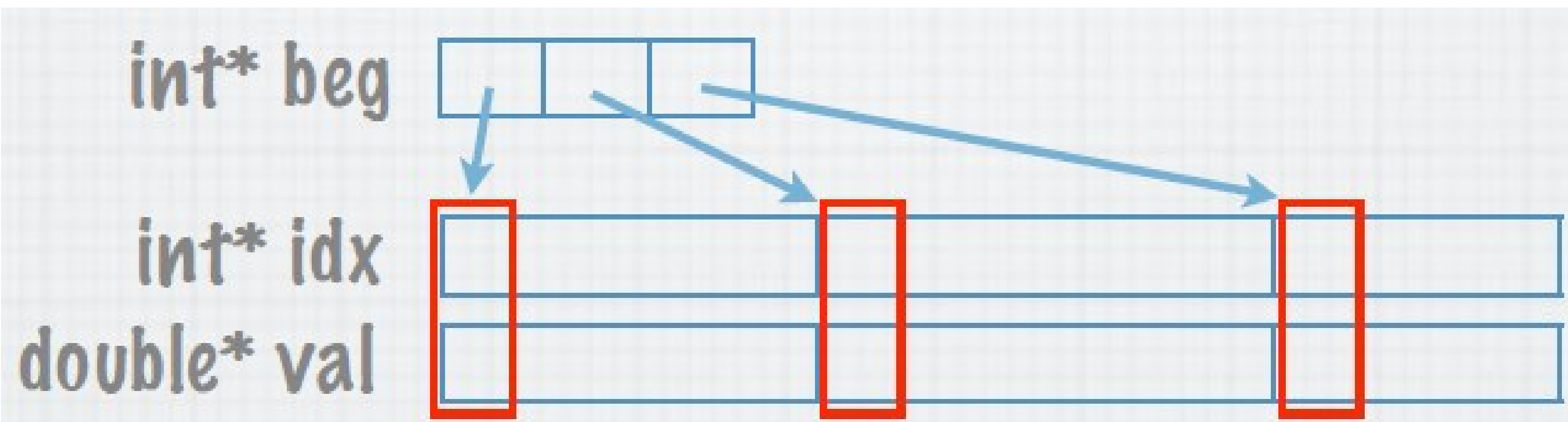
Basic objects

Parameters

Resources: **`cpxmacro.h`**

Sparse matrix representation

- Sparse matrix: many zero entries
- Compact representation:
  - Explicit representation of “nonzeroes”
  - Linearization into indexes (**idx**) and values (**val**) vectors
  - A third vector to indicate where rows begins (**beg**) `cpp]`



Resources: `addrow.xls, first.cpp`

Proposed exercises

- Implement the mathematical models for
  - the “Moving scaffolds between yards” problem

    [Resources: moving_scaffolds.cpp moving_scaffolds.py]
  - the “Four Italian friends” problem

    [Resources: italianFriendsJSP.cpp italianFriendsJSPwithVarMaps.cpp italianFriendsJSP.py]
  - The “TLC-antennas location” problem

    [Resources: antennas.cpp antennas.py]

Reporting the example to be implemented

Moving scaffolds between construction yards: MILP model

[Suggestion: compose transportation and fixed cost schemas]

$$\min \sum_{i\in I, j\in J} C_{ij}\, x_{ij} + F \sum_{i\in I, j\in J} y_{ij} + (L - F)\, z$$

$$
\begin{array}{lrcll}
\text{s.t.} & \sum_{i\in I} x_{ij} & \geq & R_j & \forall \; j \in J \\
& \sum_{j\in J} x_{ij} & \leq & D_i & \forall \; i \in I \\
& x_{ij} & \leq & K\, y_{ij} & \forall \; i \in I, j \in J \\
& \sum_{i\in I, j\in J} y_{ij} & \leq & N + z & \\
& y_{A2} + y_{B2} & \leq & 1 & \\
& x_{ij} & \in & \mathbb{Z}_+ & \forall \; i \in I, j \in J \\
& y_{ij} & \in & \{0,1\} & \forall \; i \in I, j \in J \\
& z & \in & \{0,1\} &
\end{array}
$$