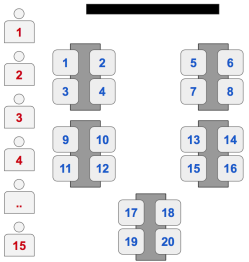
L05: Viewpoints and redundant constraints



Prof. Tias Guns and Dr. Dimos Tsouros

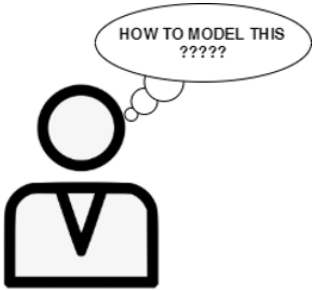
KU LEUVEN

Partly based on slides from Pierre Flener and Barbara Smith.

Quick Recap

1. Modelling: express problem in terms of
   - parameters,
   - decision variables, with their domains
   - constraints, and
   - (optionally) objective.
2. Solving: solve using a state-of-the-art solver.

A Modelling Methodology: Overview



1. Make sure you understand the problem correctly:
   - Find out what is given and what is unknown
     $\rightarrow$ what needs to be decided.
   - Find out what the objective is, if any.
2. Construct or select some small instance(s)
   of your problem to work with.
3. Model incrementally:
   - Add a constraint at a time and verify that it is correct.
   - Only add the objective function, if any, at the end.
4. Model first for correctness and clearness, then for efficiency:
   - Try to use predefined predicates when relevant.
   - Once you have a correct model, consider symmetries and redundant constraints.
   - Only as a last resort, add annotations.

Outline

1. Viewpoints
2. Auxiliary variables
3. Redundant Constraints
4. Combining viewpoints: Redundant Variables & Channelling
5. Modelling guidelines

What is a good model for a constraint problem?

You get the problem definition of a constraint problem.

- You now have to model it correctly (and efficiently)!

How do you choose which of the entities of the problem are the decision variables?

Example (Task allocation)

There are three workers (W1, W2, W3). Our current project has 4 tasks that have to be allocated to a worker: Construct Product (CP) and Quality Check (QC), which will be conducted on Thursday, and Package Product (PP) and Transport Product (TP), which will be conducted on Friday. No worker should handle two tasks on the same day. Worker W3 cannot work this on Thursday. Worker W1 can only perform tasks QC and TP.

Viewpoints

Different models of a problem may result from viewing the problem from different angles or perspectives, i.e. different viewpoints.

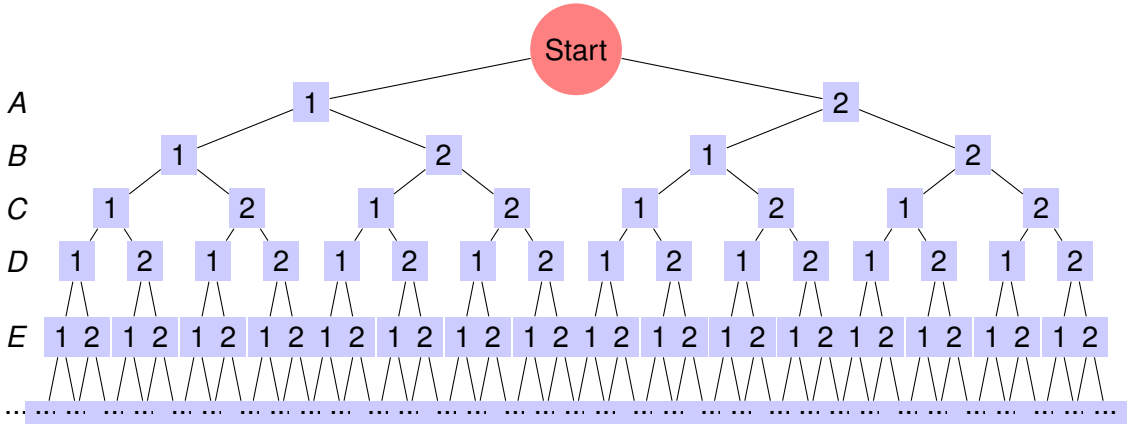
The viewpoint defines the variables and domains of your model.

Definition (Law and Lee, 2002)

A viewpoint is a pair $\langle X, D \rangle$, where $X = \{x_1, \ldots, x_n\}$ is a set of variables, and D is a set of domains. For each $x_i \in X$, the associated domain D_i is the set of possible values for x_i. It must be possible to ascribe a meaning to the variables and values of the CSP in terms of the problem P, and so to say what an assignment in the viewpoint $\langle X, D \rangle$ is intended to represent in terms of P.

Based on the chosen viewpoint, the size of the search space changes

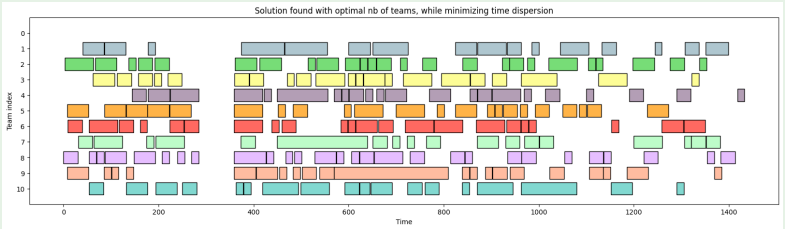
Viewpoints and size of the search space



Search space has breadth d and depth n, with d being the size of the domains, and n the amount of decision variables used: d^n **leaves = complete assignments**

What is a good model for a constraint problem?

Example (Task allocation)



There are three workers (W1, W2, W3). Our current project has 4 tasks that have to be allocated to a worker: Construct Product (CP) and Quality Check (QC), which will be conducted on Thursday, and Package Product (PP) and Transport Product (TP), which will be conducted on Friday. No worker should handle two tasks on the same day. Worker W3 cannot work this Thursday. Worker W1 can only perform tasks QC and TP.

What is a good model for a constraint problem?

Example (Task allocation)

There are three workers (W1, W2, W3). Our current project has 4 tasks that have to be allocated to a worker: Construct Product (CP) and Quality Check (QC), which will be conducted on Thursday, and Package Product (PP) and Transport Product (TP), which will be conducted on Friday. No worker should handle two tasks on the same day. Worker W3 cannot work this Thursday. Worker W1 can only perform tasks QC and TP.

We have 2 types of entities: the workers $\{W1, W2, W3\}$ and the tasks $\{CP, QC, PP, TP\}$. Which are my variables?

1. Viewpoint 1: Workers as variables $\rightarrow$ Which task(s) is assigned to each worker?
2. Viewpoint 2: Tasks as variables $\rightarrow$ Which worker is each task assigned to?

Workers as variables

Decision Variables One for each worker, so we have 3 variables: W1, W2 and W3

Initial Domains The workers are 3 and the tasks are 4. So, each worker has to be assigned one or more shifts, to have all tasks allocated, meaning that there must be (at least) one worker that will have 2 task.

So, the domains will be all tasks, but also all pairs of them:
$\{\{CP\}, \{QC\}, \{PP\}, \{TP\}, \{CP, QC\}, \{CP, PP\}, \{CP, TP\},$
$\{QC, PP\}, \{QC, TP\}, \{PP, TP\}\}$

Size of the search space?
Domains of size 10, and 3 variables: $10^3 = 1000$ candidate solutions.

Constraints All tasks have to be assigned to only one worker. Can we represent it with just pairwise, not equal constraints? No.
Also, too many unary constraints to model each restriction of the workers.

Tasks are variables

Decision Variables One for each shift, so we have 4 variables: $\{CP, QC, PP, TP\}$

Initial Domains Each task has to be assigned to a worker. So, the initial domain of every decision variable is the set of workers: $\{W1, W2, W3\}$.

The problem representation is much easier even before formulating the constraints! The initial domains are much more straightforward

Size of the search space?
Domains of size 3, and 4 variables: $3^4 = 81$ candidate solutions.

Tasks are variables

Constraints Each task has to be assigned to a worker. No constraints are needed to represent that now. Let's model the unary constraints:

1. No worker should be assigned two tasks in the same day:
   - $CP \neq QC$
   - $PP \neq TP$

2. Worker W3 cannot work this Thursday.
   - $CP \neq W3$
   - $QC \neq W3$

3. Worker W1 can only perform tasks QC and TP.
   - $CP \neq W1$
   - $PP \neq W1$

Viewpoints

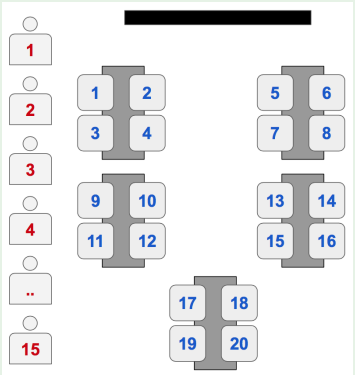
Choosing the right viewpoint can make modeling way easier.

Ease of formulating constraints and (optionally) the objective function: Important on choosing the viewpoint to use.

- Typically leads also to better **readability!**
- This may be very important, depending on the user of the model and their background ...

But is one viewpoint always clearly the right one to use?

Example (Student Seating Problem)



`nStudents` $= 15$
`nPgms` $= 3$
`nChairs` $= 20 \geq$ `nStudents`
`nTables` $= 5$
`Chairs` $=$ [1..4, 5..8, 9..12, 13..16, 17..20]

Given:

- `nStudents` students,
- `nPgms` study programmes
- `nChairs` chairs around `nTables` tables, and
- `Chairs`[`t`] as the set of chairs of table `t`,

Find a seating arrangement such that:

1. each table has students of distinct study programmes;
2. each table has either at least half or none of its chairs occupied;
3. a maximum number of student preferences on being seated at the same table are satisfied.

What are suitable decision variables for this problem?

A viewpoint is a choice of decision variables.

Example (Student Seating Problem)

Viewpoint 1: Which chair does each student sit on?
Variables represent the students, domains represent the chairs they will sit on.

$$\texttt{student}_s \in \{1, \ldots, \texttt{nChairs}\}, \quad \forall s \in \{1, \ldots, \texttt{nStudents}\}$$

$$\textsc{AllDifferent}(\texttt{student})$$

Viewpoint 2: Which student, if any, sits on each chair?
Variables represent the chairs, domains represent the students that will sit on them.

$$\texttt{chair}_c \in \{0, \ldots, \texttt{nStudents}\}, \quad \forall s \in \{1, \ldots, \texttt{nChairs}\}$$

$$\textsc{AllDifferentExceptN}(\texttt{chair}, 0)$$

Notice that we use student 0 to represent that no student is sitting on that chair ...
It does not need to be a 0, but a dummy value. Depending on the problem, we may have several dummy values

A viewpoint is a choice of decision variables.

Example (Student Seating Problem – CPMpy)

Viewpoint 1: Which chair does each student sit on?
Variables represent the students, domains represent the chairs they will sit on.

```
students = cp.intvar(1, n_chairs, shape=n_students)
model.add(cp.AllDifferent(students))
```

Viewpoint 2: Which student, if any, sits on each chair?
Variables represent the chairs, domains represent the students that will sit on them.

```
chairs = cp.intvar(0, n_students, shape=n_chairs)
model.add(cp.AllDifferentExceptN(chairs, 0))
```

Notice that we use student 0 to represent that no student is sitting on that chair ...
It does not need to be a 0, but a dummy value. Depending on the problem, we may have several dummy values

Different viewpoints offer different advantages and disadvantages.

- Ease of formulating the constraints and (optionally) the objective (as discussed).
  - It depends on the unstated other constraints.
  - Ideally, we want a viewpoint that allows global constraints to be used.
- Readability.
  - Who is going to read the model, and what is their background?
- Size of the search space.
  - Viewpoint 1: $\mathcal{O}(\texttt{nChairs}^{\texttt{nStudents}})$
  - Viewpoint 2: $\mathcal{O}((\texttt{nStudents}+1)^{\texttt{nChairs}})$
- Performance:
  - The size is not the only factor for the performance
  - In general, hard to tell: we have to run experiments!

Typically, there is no one correct answer here:
when in doubt: run comparative experiments.

Outline

Auxiliary variables

Definition

Auxiliary variables are variables that are introduced in the model during the modeling process, and are not necessary related to entities of our problem.

Auxiliary variables may be necessary:

- You may not be able to express certain constraints directly using the variables of the chosen *viewpoint*
- Depending on the used *solving technology/language*, auxiliary variables may need to be introduced to allow the specification of higher arity constraints.

Auxiliary variables may be useful:

- Expressing constraints may be *easier* using auxiliary variables.
- Resulting model may be more *readable*.
- Propagation of constraints may be more *efficient*.

Auxiliary variables

Example (Car Sequencing; Dincbas, Simonis and van Hentenryck, 1988)

- 10 cars to be made on a production line, each requires some options
- Stations installing options have lower capacity than rest of line e.g. at most 1 car out of 2 for option 1
- Find a feasible production sequence

Classes	1	2	3	4	5	6	Capacity
Option 1	1	0	0	0	1	1	1/2
Option 2	0	0	1	1	0	1	2/3
Option 3	1	0	0	0	1	0	1/3
Option 4	1	1	0	1	0	0	2/5
Option 5	0	0	1	0	0	0	1/5
No. of cars	1	1	2	2	2	2	

Auxiliary variables - Car Sequencing

- Viewpoint:
  - A variable for each position in the sequence, $s_1, s_2, \ldots, s_{10} \in S$
  - Value of s_i is the class of car in position i
- Constraints:
  - Each class occurs the correct number of times

$$\mathit{CountEq}(s, k, d_k) \quad \forall k \in \texttt{Classes}$$

with d_k being the production demand for class k. Or better:

$$\mathit{GlobalCardinalityCount}(S, \mathit{Classes}, D)$$

with D being the list of production demands for each class.

  - Option capacities are respected - ?

Cannot model the option capacities directly by using variables s_i! (Actually we can, with a long and complex constraint that will create auxiliary variables either way)

Auxiliary variables - Car Sequencing

How can we model the constraints for the option capacities?

- Introduce auxiliary variables o_{ij}:
  - $o_{ij} = 1$ iff the car in the i-th slot in the sequence requires option j
- *Constraints*, e.g. option 1 capacity is one car in every two:
  - $o_{i,1} + o_{i+1,1} \leq 1 \quad \text{for} \quad 1 \leq i < 10$
- Relate the auxiliary variables to the s_i variables:
  - Matrix $\lambda_{jk} = 1$ representing if car class k requires option j
  - $o_{ij} = \lambda_{js_i} \quad 1 \leq i \leq 10, 1 \leq j \leq 5 \rightarrow$ Reminder: This uses in practice the `Element` constraint. A variable (s_i) is used as index in an array.

Auxiliary variables – Car Sequencing – CPMpy

Example (CPMpy model for Car Sequencing)

```
# Sequence of different car types
sequence = cp.intvar(0, n_classes - 1, shape=n_cars, name="sequence")
# Auxiliary variables for options
option = cp.boolvar(shape=(n_cars, n_options), name="option")

# Each car class occurs the correct number of times
model.add(cp.GlobalCardinalityCount(sequence, range(n_classes), demand))

# connect auxiliary variables with decision variables
for s in range(n_cars):
    model.add([option[s, o] == requires[sequence[s], o] for o in range(
n_options)])

# Option capacities are respected
for o in range(n_options):
    for s in range(n_cars - per_slots[o] + 1):
        slotrange = range(s, s + per_slots[o])
        model.add(cp.sum(option[slotrange, o]) <= at_most[o])
```

Outline

Redundant constraints

We have chosen a viewpoint, and formulated the constraints to solve our problem correctly. But, can we do something to make it more efficient?

During the search process, the solver will explore several infeasible parts of the search tree. Can we avoid that?

Definition

Redundant constraints, also called *implied* constraints, are constraints which are entailed by the constraints defining the problem. They do not change the set of solutions, and hence are logically redundant.

Although redundant constraints are logically redundant, and do not change the set of solutions, they can be used to reduce the search effort of the solving process.

Redundant constraints

Definition

Redundant constraints, also called *implied* constraints, are constraints which are entailed by the constraints defining the problem. They do not change the set of solutions, and hence are logically redundant.

Redundant constraints are often added to the model to reduce the search effort, by cutting infeasible parts of the search tree early. So the solver does not need to explore possible assignments in these subproblems ...

- at some point during search, we will have a partial assignment that cannot lead to a solution
- the redundant constraint can detect inconsistency of the partial assignment
  - the redundant constraint forbids it, but **does not change the set of solutions**
- while without the redundant constraint, the search would continue deeper

Redundant constraints – Car Sequencing

- Existing constraints only say that the option capacities cannot be *exceeded*
  - but we can't stay too far below capacity either
- Suppose there are 30 cars, and 12 require option 1
- Reminder: Option 1 capacity is 1 car in 2
- At least one car in slots 1 to 8 of the production sequence must have option 1
  - otherwise 12 of cars 9 to 30 will require option 1, i.e. too many
- Cars 1 to 10 must include at least two option 1 cars, . . . , and cars 1 to 28 must include at least 11 option 1 cars
- These are *implied constraints* that can be used to speed up search in car sequencing

Example (Magic Series of length `n`)

The element at index `i` in `I = 0..(n-1)` is the number of occurrences of `i`.
Solutions: `Magic` $= [1,2,1,0]$ and `Magic` $= [2,0,2,0]$ for `n` $= 4$.

Decision variables:

`Magic` =

0	1	⋯	n-1
∈ I	∈ I	⋯	∈ I

Problem Constraint:

$$\forall i \in I, \quad \text{Magic}[i] = \sum_{j \in I} (\text{Magic}[j] = i)$$

or, logically equivalently but better:

$$\forall i \in I, \quad \text{CountEq}(\text{Magic}, i, \text{Magic}[i])$$

or, logically equivalently and even better:

$$\text{GlobalCardinalityCount}(\text{Magic}, \text{array1d}(I, [i \mid i \in I]), \text{Magic})$$

Example (Magic Series of length n)

The element at index i in I = 0..(n-1) is the number of occurrences of i.
Solutions: Magic = [1, 2, 1, 0] and Magic = [2, 0, 2, 0] for n = 4.

Decision variables:

Magic =

0	1	···	n-1
∈ I	∈ I	···	∈ I

Problem Constraint:

```
[Magic[i] == cp.sum(Magic == i) for i range(len(Magic))]
```

or, logically equivalently but better:

```
[Magic[i] == cp.Count(Magic, i) for i range(len(Magic))]
```

or, logically equivalently and even better:

```
cp.GlobalCardinalityCount(Magic, range(Magic), Magic)
```

Redundant constraints

We have modeled the problem correctly. What redundant constraints can we add?

Redundant Constraints:

$$\sum_{i \in I} \text{Magic} = n$$

$$\sum_{i \in I} (i \cdot \text{Magic}[i]) = n$$

Depending on the formulation above of the problem constraint,
the implied constraints can accelerate a CP solver up to 100 times for $\mathrm{n} = 150$.

Redundant Constraints

Be careful though: not all redundant constraints accelerate the solving.

- The assignments forbidden by an redundant constraint may never actually arise during search
  - This means more propagation time and no benefit
- e.g. in car sequencing,
  - at least one of cars 1 to 8 must require option 1
  - in general, *any* 8 consecutive cars must have one option 1 car
  - but putting redundant constraints for all such cases will not offer more benefits
- If we find a *class* of redundant constraints, maybe only some are useful
  - adding a lot of constraints that don't reduce search will increase the run-time

Outline

Combining viewpoints

Adding auxiliary variables may be important to model your problem (efficiently), while adding redundant constraints can speed up the search process.

Can we do something more to get a more efficient model??

We may have spotted different viewpoints for our problem . . .

And one is not *clearly* better than the other:
Different perspectives often allow different expression of the constraints and different redundant constraints

Why choose one of them? Can use both!

Combining viewpoints

We have chosen our main viewpoint for the problem,

- but we also introduce a second viewpoint with the hope that it will help in propagation during search.

This means we will use redundant variables, that do not capture information not already captured in the model.

Definition

A redundant decision variable denotes information already denoted by other variables: mutual redundancy (same information) vs non-mutual redundancy.

The constraints of the second viewpoint will be used on these variables.

Channelling

Including the variables and constraints of both viewpoints may allow the solver to infer different things from each one, and reduce the search space.

But we have to make sure that our 2 sets of variables solve the same problem!

We need to channel the 2 viewpoints, linking the variables i.e. make sure that assignments (and domain pruning) on one viewpoint are propagated to the other.

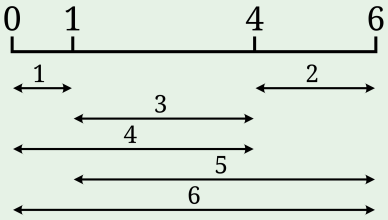
Definition

A channelling constraint fixes the value of 1 set of decision variables when the values of the decision variables they are redundant with from the second set are fixed. Channeling can be (1-way) or (2-way).

Channelling

Example (Golomb Rulers)

- A Golomb ruler with m marks consists of
  - a set of m integers $0 = x_1 < x_2 < \ldots < x_m$
  - the $\frac{m(m-1)}{2}$ differences $x_j - x_i$ are all different
  - Objective: find a ruler with minimum length x_m
- First viewpoint: variables $x_1, x_2, \ldots, x_m$
  - $x_j - x_i \neq x_l - x_k$ for all distinct pairs
  - $x_1 < x_2 < \ldots < x_m$
- Second viewpoint: variables d_{ij}, $1 \leq i < j \leq m$
  - AllDifferent$(d_{12}, d_{13}, \ldots, d_{m-1,m})$
  - $d_{ik} = d_{ij} + d_{jk}$ for $1 \leq i < j < k \leq m$
- Channelling constraints: $d_{ij} = x_j - x_i$



What about the Student Seating problem?

Try it in the exercise session

Outline

From a Problem to a Model

What is a good model for a constraint problem?

- A model that correctly represents the problem
- A model that is easy to understand and maintain
- A model that is solved efficiently, that is:
  - short solving time to find one, all, or best solution(s)
  - good solution within a limited amount of time
  - small search space (under systematic search)

Food for thought: What is correct, easy, short, good, . . . ?

Modelling Issues

Modelling is still more a craft than a science:

- Choice of viewpoint: which are the decision variables and their domains?
- Choice of the constraint formulations
- Choice of solver and solver configuration (later lectures)

Make a model correct before making it efficient!

Choice of viewpoint

As shown in the previous slides, the choice of the viewpoint can have a large impact on the:

- Readability
- Ease of formulating
- Efficiency
- ...

Maybe we even need to combine viewpoints for better inference!

Choice of constraints

Example (The AllDifferent() constraint)

The constraint AllDifferent(X) on an array `X` of size `n` usually leads to faster solving than its definition by a conjunction of $\frac{n \cdot (n-1)}{2}$ disequality constraints:

$$var1 != var2 \qquad \forall var1, var2 \in \texttt{all\_pairs}(X)$$

We typically use *global constraints* such as AllDifferent(X) because:

- they can sometimes do stronger propagation then a decomposition
- they can avoid the introduction of additional variables
- they can use specialised data-structures to handle data

Use redundant constraints to propagate information that will not be propagated otherwise!

→ Typical when no global constraint exists to capture a substructure of the problem ...

Guidelines: Reveal Problem Structure

- Express the problem concisely: global constraints and functions
- Use few decision variables, and declare tight domains
  - Use redundant variables only if constraints on them can offer additional propagation during search.
- Beware of nonlinear and power constraints: $x * y, x^y$
- Beware of division constraints: x/y as well as $x \texttt{ mod } c$

Careful: These guidelines of course have their exceptions!
It is important to test empirically combinations of model, solver, and solving technology and what their impact on runtime is.

Declare the Decision Variables with Tight Domains

Tight domains for decision variables might accelerate the solving.

Think about the bounds of your integer variables, avoid lazily putting a generic large constant.

Example (Think about tight domains)

If you have decision variable $x \in \{1, \ldots, 20\}$ and $y \in \{5, \ldots, 10\}$, then if you want to create an explicit decision variable with $z = \text{MAX}(x, y)$, think it through:

$$z \in \{5, \ldots, 20\}$$
$$z = \text{MAX}(x, y)$$
$$\text{minimize } z$$

If your modeling library allows it, even better is to use nested expressions; the library will decompose and create an auxiliary variable only if the solver needs it:

$$\text{minimize } \text{MAX}(x, y)$$

Beware of Nonlinear and Power Constraints

Constraining the product of two or more decision variables often makes the solving slow. Try and find a linear reformulation if possible.

Example

The model snippet

$$
\begin{aligned}
& X_i \in \{0,1\} && \forall i \\
& Y_i \in \{0,1\} && \forall i \\
& \textsc{Count}(X * Y, 1) == b
\end{aligned}
$$

should be reformulated as:

$$
\begin{aligned}
& X_i \in \{0,1\} && \forall i \\
& Y_i \in \{0,1\} && \forall i \\
& \textsc{Count}(X + Y, 2) == b
\end{aligned}
$$

Beware of Division Constraints

The use of *division* and *modulo* on decision variables often makes the solving slow.
Reformulate where possible.

Example (Avoid unnecessary divisions)

If you want to minimize the (integer) average value of an array of decision variables X:

$$\text{minimize} \sum X / \texttt{len}(X)$$

then realizing that $\texttt{len}(X)$ is a constant, you can reformulate the objective as:

$$\text{minimize} \sum X$$

given that you post-process the objective value by dividing it by $\texttt{len}(X)$.

Avoid floating point constants in pure integer models

Example

$$\sum X \geq (2/3) \cdot b$$

instead bring the denominator to the other side:

$$3 \cdot \sum X \geq 2 \cdot b$$

Summary

- Find the different viewpoints that can be used to model the problem
  - Is one of them clearly better at formulating the constraints?
  - Are they better propagation on a part of the problem?
  - Should you combine viewpoints?
- Express the problem concisely: global constraints and functions
- Possibly use redundant constraints to speed up the search
- Use few decision variables, and declare tight domains
  - Auxiliary variables may be needed to formulate some of the constraints
- Beware of nonlinear and power constraints: $x * y, x^y$
- Beware of division constraints: x/y as well as $x \texttt{ mod } c$

Above all: have fun and try different things!