Industrial project and machine scheduling with Constraint Programming

Philippe Laborie
IBM, IBM Data & A.I.
laborie@fr.ibm.com

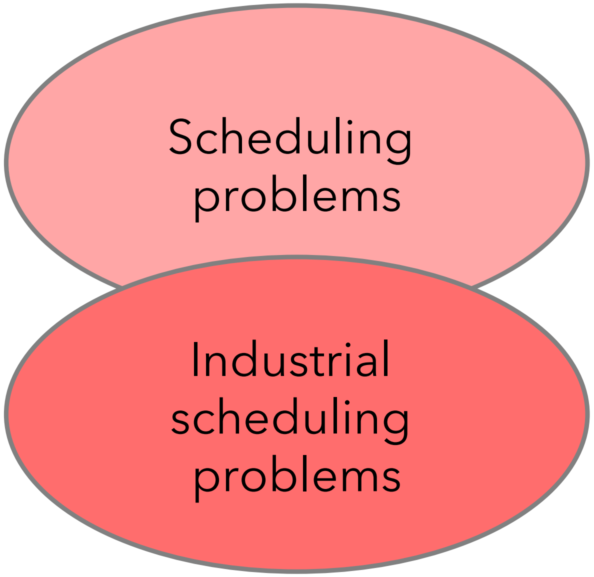
Constraint Programming (CP)

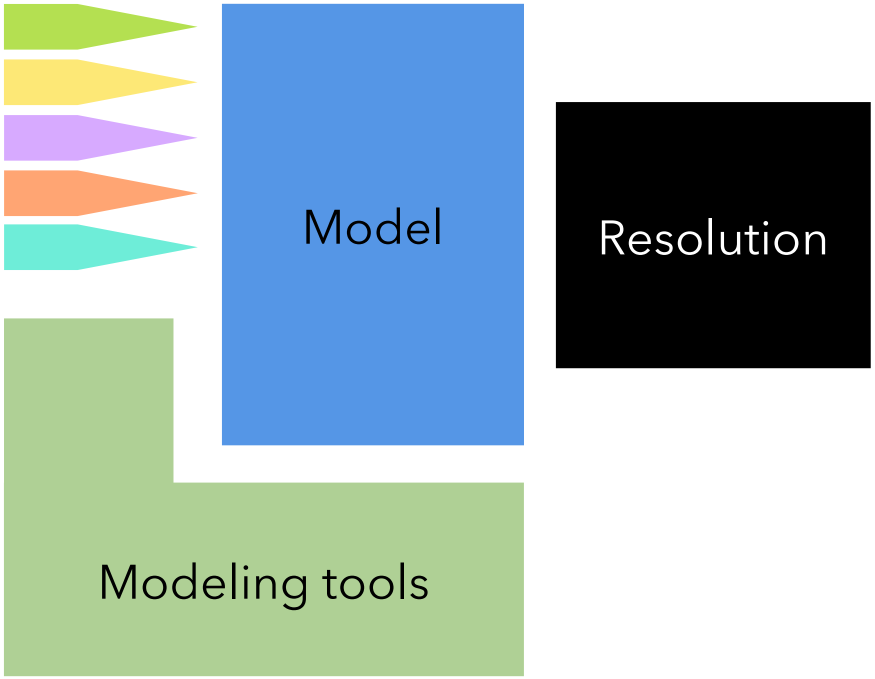
- Exact method to solve combinatorial optimization problems
- Provides a modeling framework much richer than Integer Linear Programming (ILP) with additional types of:
  - Decision variables
  - Constraints (non-linear)
- **You don't need to program anything !**
  - Modern CP Solvers implement powerful automatic search
- State-of-the-art methods for solving many classical scheduling problems and their variants:
  - Job-shop
  - RCPSP
- Many industrial applications

Constraint Programming (CP)

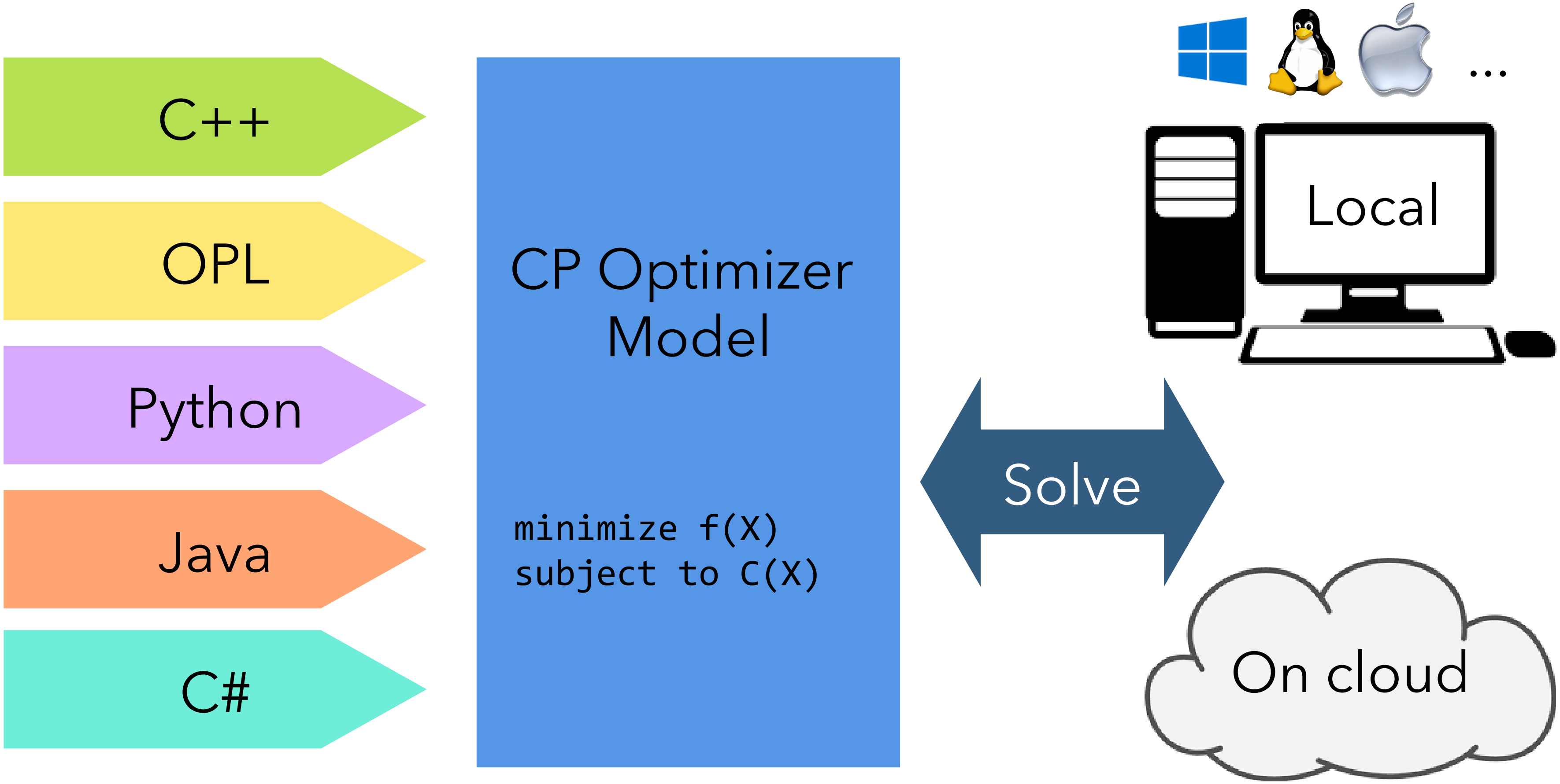
- Several CP engines are available:
  - Choco
  - Gecode
  - Google OR-Tools
  - IBM CP Optimizer
  - ...
- I will use CP Optimizer as illustration because:
  - It has a strong focus on **scheduling** problems
  - Its main targets are **industrial** applications
  - **You can use it without knowing anything about CP**
  - I like it ...

Some topics we will cover





Overview of CP Optimizer

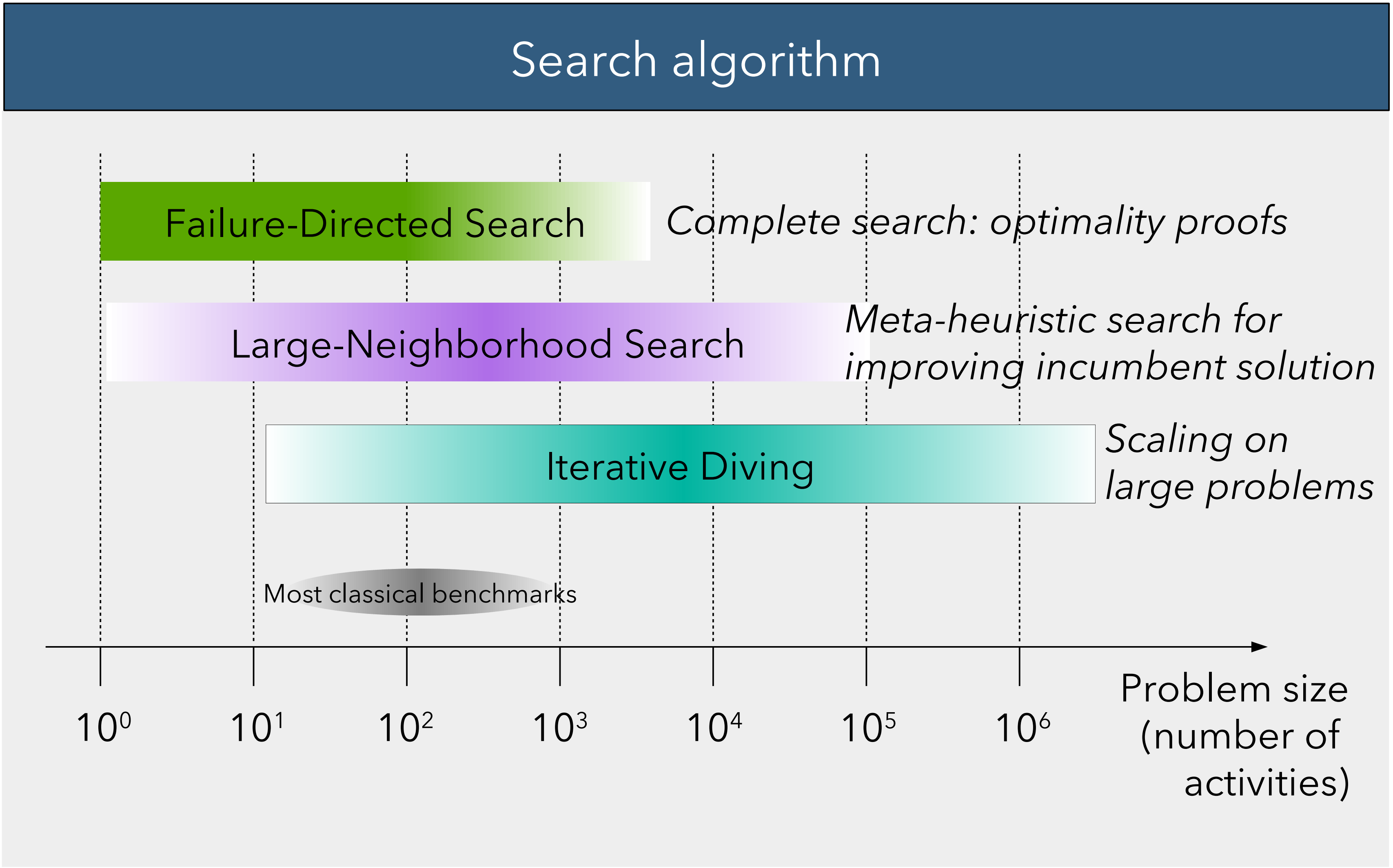


Properties of the automatic search

- Search is **complete** (exact algorithm)
- Search is **anytime** (first solution is produced fast)
- Search is **parallel** (unless stated otherwise)
- Search is **randomized**
  - Internally, some ties are broken using random numbers
  - The seed of the random number generator is a parameter of the search
- Search is **deterministic**
  - Solving twice the same problem on the same machine (even when using multiple parallel workers) with the same seed for the internal random number generator will produce the same result
  - Determinism of the search is essential in an industrial context and for debugging

CP Optimizer automatic search

- Main principle: cooperation between several approaches



CP Optimizer automatic search - Under the hood

Artificial Intelligence **Operations Research**

Constraint propagation

Learning

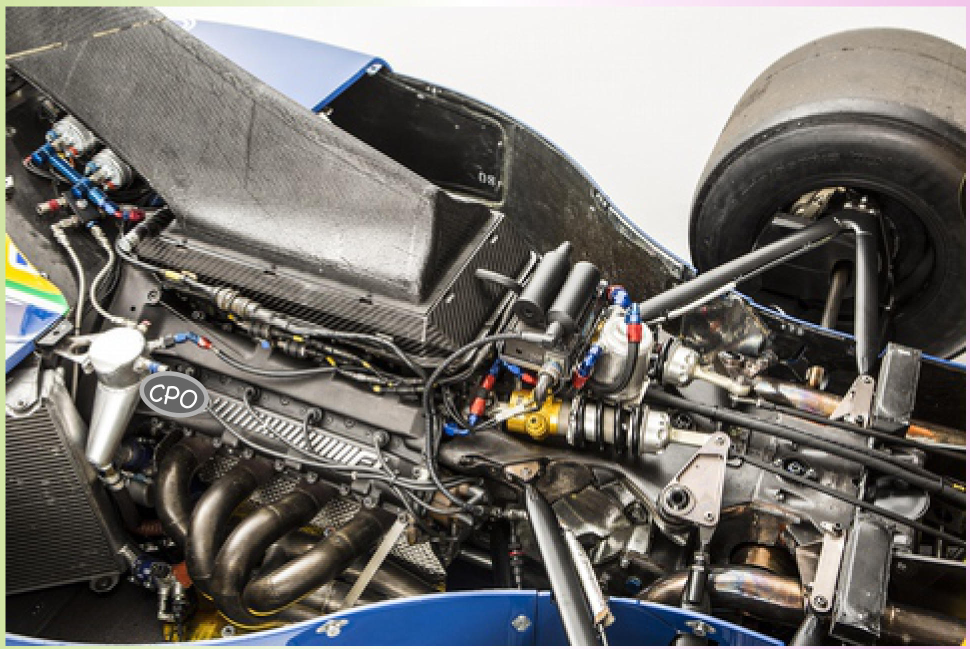
Temporal constraint networks

2-SAT networks

No-goods

Heuristics

Model presolve



Restarts

Tree search

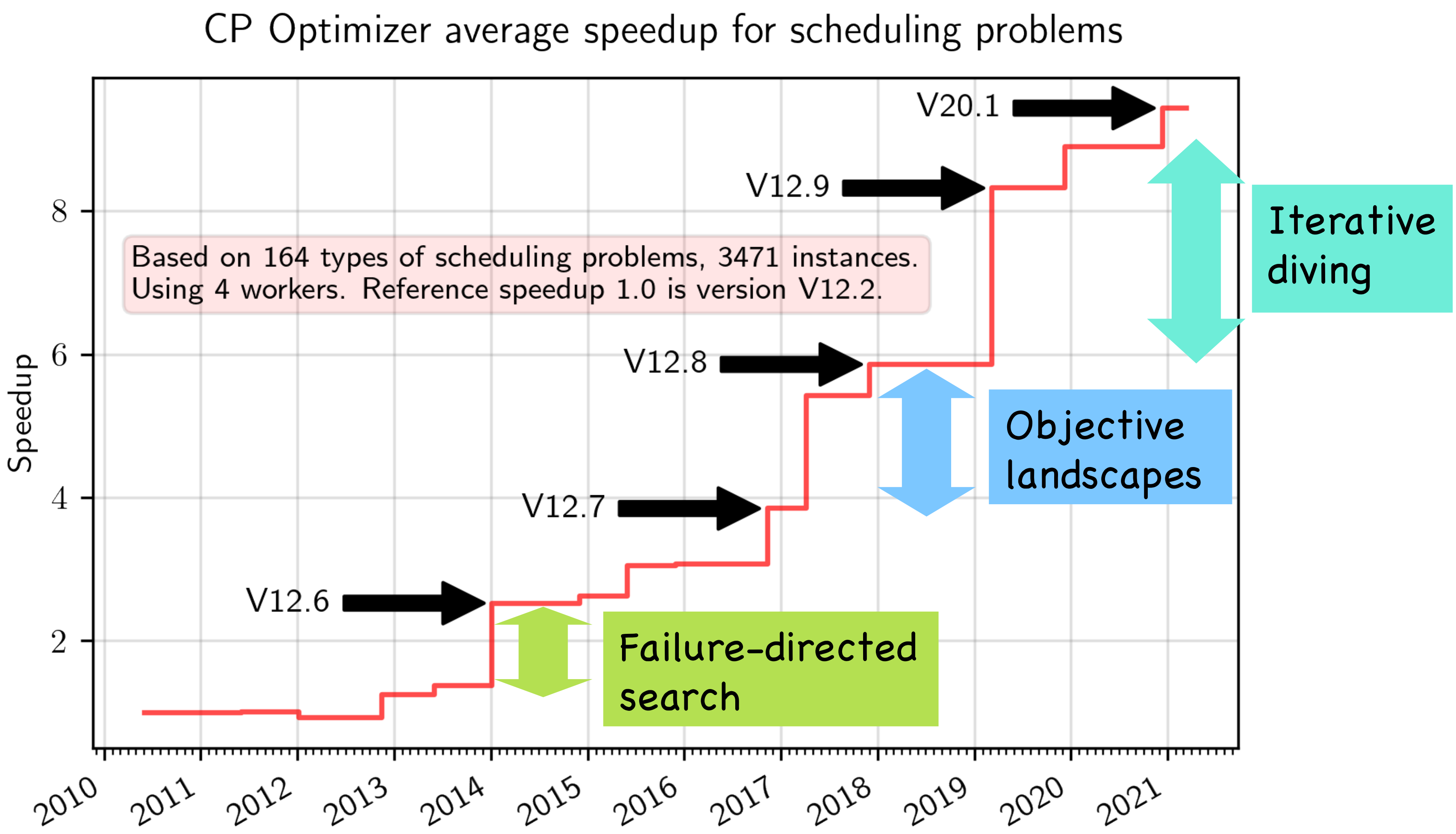
LNS

Randomization

Linear relaxations

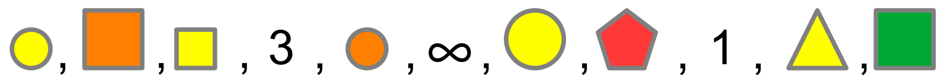
Problem specific scheduling algorithms

Performance evolution

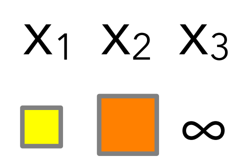


A parenthesis on constrained optimization problems

- **Given X, a set of decision variables, minimize f(X) subject to C(X)**
- **A decision variable x∈X does not need to be a numerical variable** … it can be anything defined as a set of possible values (domain) provided a non-ambiguous semantics is defined for constraints and expressions:

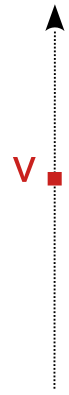
$x_1, x_2, x_3 \in \{$  $\}$


```
maximize ( nbColors([x1, x2, x3]) )
subject to :
  x1 ≠ ∞
  shape(x1)==shape(x2)
  smaller(x1, x2)
  smaller(x2, x3)
  allDifferent([x1, x2, x3])
```



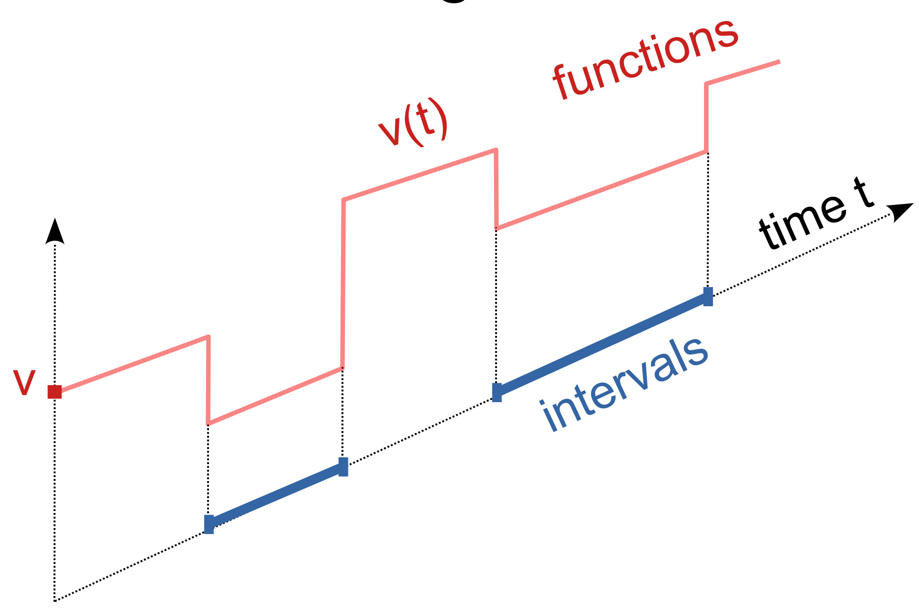
Basic concepts of CP Optimizer

Formulating scheduling problem with numerical variables only (ex: ILP) … is frustrating



Basic concepts of CP Optimizer

- Formulating scheduling problem with numerical variables only (ex: ILP) … is frustrating



- Scheduling is about time …
  - Intervals of time (activities, etc.)
  - Functions of time (resource use, resource state, inventory levels, ...)

Basic concepts of CP Optimizer

- Introduction of a some simple mathematical concepts in the formulation :
  - Integers — integer variables
  - Intervals — interval variables
  - Sequences of intervals — sequence variables
  - Functions — state/cumul functions

Basic concepts of CP Optimizer

- Introduction of a some simple mathematical concepts in the formulation :
  - Integers — integer variables
  - Intervals — interval variables
  - Sequences of intervals — sequence variables
  - Functions — state/cumul functions
- Interval variables
  - The **value** of an interval variable x is an interval of integers [s,e): s is the start, e is the end, (e-s) is the size

Basic concepts of CP Optimizer

- Introduction of a some simple mathematical concepts in the formulation :

• Integers	integer variables
• Intervals	interval variables
• Sequences of intervals	sequence variables
• Functions	state/cumul functions

- Interval variables
  - The **value** of an interval variable x is an interval of integers [s,e): s is the start, e is the end, (e-s) is the size
  - An interval variables can be optional meaning that its value can also be "absent"

Basic concepts of CP Optimizer

- Introduction of a some simple mathematical concepts in the formulation :
  - Integers — integer variables
  - Intervals — interval variables
  - Sequences of intervals — sequence variables
  - Functions — state/cumul functions

- Interval variables
  - The **value** of an interval variable x is an interval of integers [s,e): s is the start, e is the end, (e-s) is the size
  - An interval variables can be optional meaning that its value can also be "absent"
  - Example: `interval x, optional, size=10`
    Some possible values for variable x in a solution:
    `absent`, `[0,10)`, `[1,11)`, `[1000,1010)`, ...

Academic problems: Job-shop scheduling

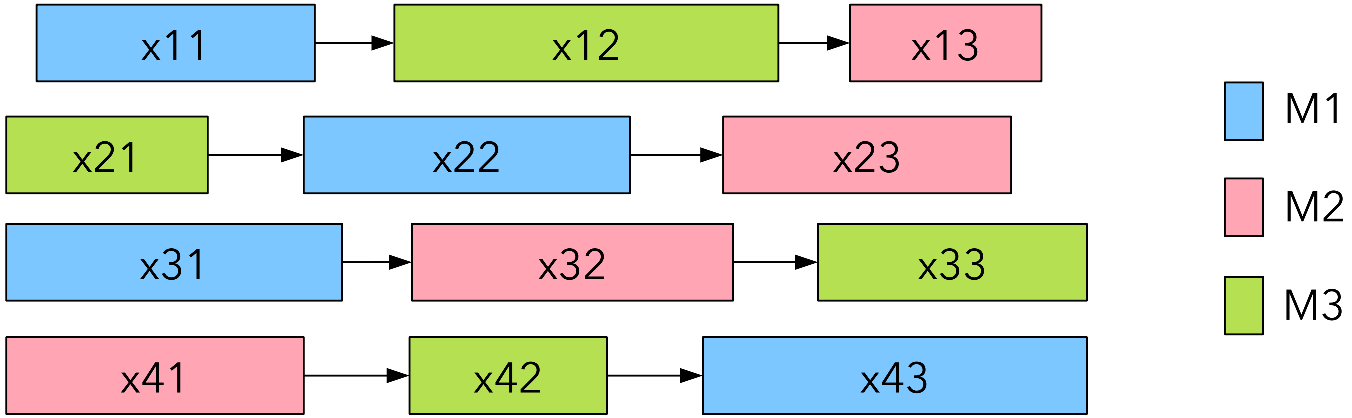
- Job-shop scheduling problem:

$$\min \max_{i \in [1..N]} \text{endOf}(x_{iM}) \quad (1)$$

$$\text{noOverlap}([x_{ij}]_{i,j \in [1..N] \times [1..M] : MC_{ij}=k}) \quad \forall k \in [1..M] \quad (2)$$

$$\text{endBeforeStart}(x_{ij-1}, x_{ij}) \quad \forall i \in [1..N], j \in [2..M] \quad (3)$$

$$\text{interval } x_{ij}, \text{ size} = PT_{ij} \quad \forall i \in [1..N], j \in [1..M] \quad (4)$$



Academic problems: Job-shop scheduling

- Job-shop scheduling problem:

$$\min \max_{i\in[1..N]} \text{endOf}(x_{iM}) \qquad (1)$$

$$\text{noOverlap}([x_{ij}]_{i,j\in[1..N]\times[1..M]:MC_{ij}=k}) \qquad \forall k \in [1..M] \qquad (2)$$

$$\text{endBeforeStart}(x_{ij-1}, x_{ij}) \qquad \forall i \in [1..N], j \in [2..M] \qquad (3)$$

$$\text{interval } x_{ij},\ \text{size} = PT_{ij} \qquad \forall i \in [1..N], j \in [1..M] \qquad (4)$$

- Python formulation:

```
x = { o : interval_var(size=PT[o])            for o in O }             # (4)

model.add(
 [ minimize( max( end_of(x[i,L[i]]) for i in N ) )                     ] +   # (1)
 [ no_overlap( x[o] for o in O if MC[o]==k )    for k in M             ] +   # (2)
 [ end_before_start( x[i,j-1], x[i,j] )         for (i,j) in O if 0<j  ]     # (3)
)

sol = model.solve()
```


Academic problems: Job-shop scheduling

- This formulation with automatic search of CP Optimizer improved 43 bounds on classical instances in 2015

Instance	LB	UB
tail11	**1357**	1357
tail12	**1367**	1367
tail13	**1342**	1342
tail15	**1339**	1339
tail16	**1360**	1360
tail18	**1377**	1396
tail19	**1332**	1332
tail20	**1348**	1348
tail21	**1642**	1642
tail22	**1561**	1600
tail23	**1518**	1557

Instance	LB	UB
tail24	**1644**	1644
tail25	**1558**	1595
tail26	**1591**	1643
tail27	**1652**	1680
tail28	**1603**	1603
tail29	**1573**	1625
tail30	**1519**	1584
tail33	**1788**	1791
tail40	**1651**	1669
tail41	**1906**	2005
tail42	**1884**	1937

Instance	LB	UB
tail44	**1948**	1979
tail46	**1957**	2004
tail47	**1807**	1889
tail49	**1931**	1961
tail50	**1833**	1923
abz07	656	**656**
abz08	**648**	667
abz09	**678**	678
swv03	**1398**	1398
swv04	**1464**	**1464**
swv05	1424	**1424**

Instance	LB	UB
swv06	**1630**	1671
swv07	**1513**	1595
swv08	**1671**	1752
swv09	**1633**	1655
swv10	**1663**	1743
yam1	**854**	884
yam2	**870**	904
yam3	**859**	892
yam4	**929**	968

- J. van Hoorn. "*The Current state of bounds on benchmark instances of the job-shop scheduling problem*." Journal of Scheduling, volume 21, pages 127–128 (2018).

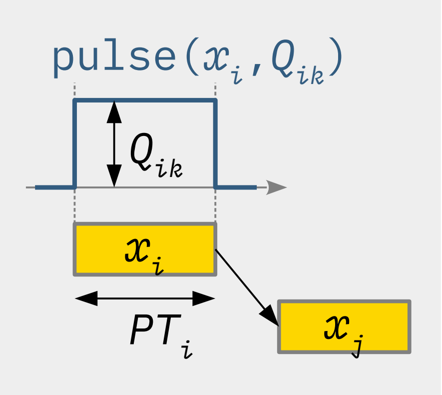
- Resource-Constrained Project Scheduling (RCPSP)

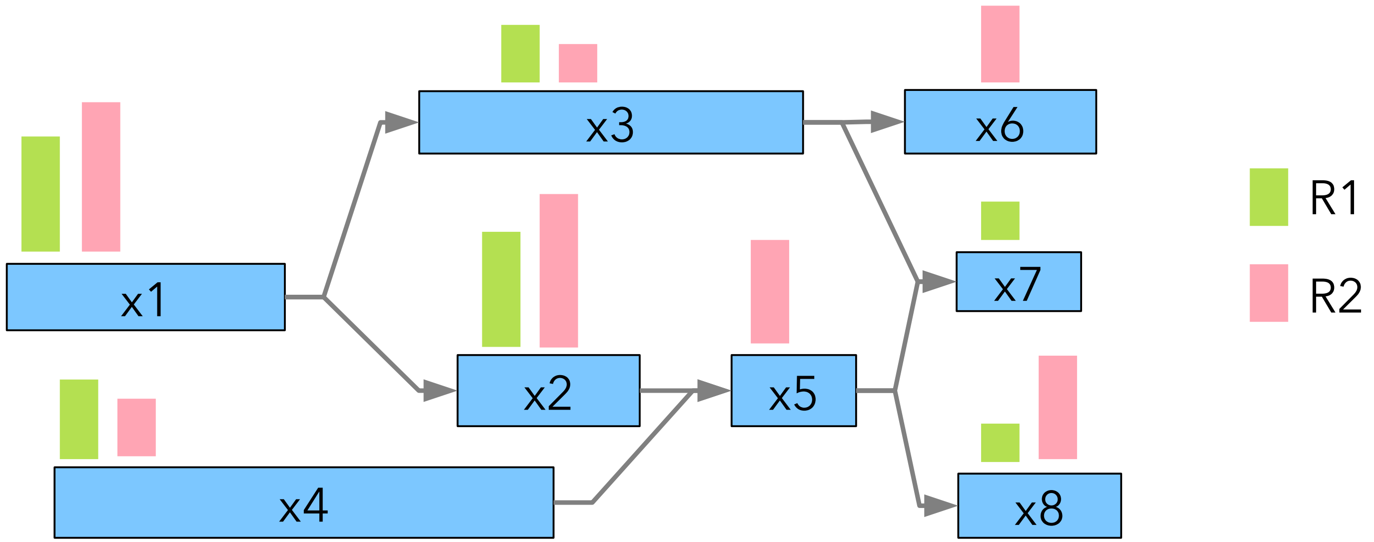
$$\min \max_{i\in[1..N]} \text{endOf}(x_i) \quad (1)$$

$$\sum_{i\in[1..N]} \text{pulse}(x_i, Q_{ik}) \leq C_k \quad \forall k \in [1..M] \quad (2)$$

$$\text{endBeforeStart}(x_i, x_j) \quad \forall (i,j) \in P \quad (3)$$

$$\text{interval } x_i, \text{ size} = PT_i \quad \forall i \in [1..N] \quad (4)$$





Academic problems: RCPSP

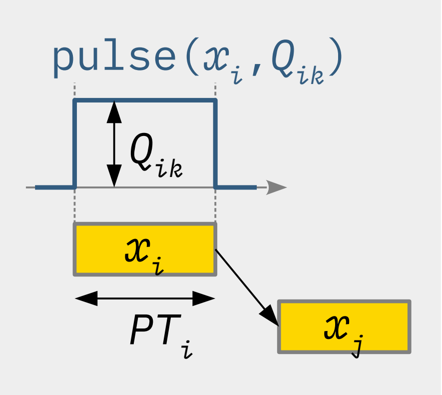
- Resource-Constrained Project Scheduling (RCPSP)

$$\min \max_{i \in [1..N]} \text{endOf}(x_i) \quad (1)$$

$$\sum_{i \in [1..N]} \text{pulse}(x_i, Q_{ik}) \leq C_k \quad \forall k \in [1..M] \quad (2)$$

$$\text{endBeforeStart}(x_i, x_j) \quad \forall (i,j) \in P \quad (3)$$

$$\text{interval } x_i, \text{ size} = PT_i \quad \forall i \in [1..N] \quad (4)$$



- Python formulation:

```
x = [ interval_var(size = PT[i])                for i in N      ]          # (4)

model.add(
  [ minimize( max( end_of(x[i]) for i in N ) )                   ] +       # (1)
  [ sum( pulse(x[i],q) for (i,q) in R[k] ) <= C[k]  for k in M   ] +       # (2)
  [ end_before_start( x[i], x[j] )                  for (i,j) in P ]       # (3)
)

sol = model.solve()
```

Academic problems: RCPSP

- This formulation with automatic search of CP Optimizer improved 53 bounds on classical instances of the PSPLib in 2015

Instance	LB	UB
j60_9_5	**82**	85
j60_9_10	**91**	93
j60_13_5	**93**	97
j60_25_8	**96**	99
j60_29_3	**115**	121
j60_29_6	**145**	154
j60_29_10	**112**	119
j60_45_5	**100**	106
j60_45_6	**133**	144
j60_45_10	**105**	114
j90_5_4	101	**102**
j90_9_1	**100**	104
j90_9_9	**107**	116
j90_13_6	**118**	124

Instance	LB	UB
j90_13_7	**117**	124
j90_25_2	**123**	131
j90_25_3	**115**	123
j90_25_7	**123**	130
j90_25_8	**133**	140
j90_29_2	**123**	126
j90_29_4	**141**	149
j90_41_2	**158**	168
j90_41_4	**144**	154
j90_41_7	**146**	157
j90_41_10	**147**	150
j90_45_5	**164**	174
j120_8_5	**101**	104

Instance	LB	UB
j120_11_3	**190**	203
j120_11_4	**183**	196
j120_11_10	**166**	181
j120_12_3	**133**	136
j120_16_5	**185**	200
j120_16_9	**190**	205
j120_17_4	**118**	120
j120_17_9	**130**	134
j120_26_3	**161**	167
j120_31_6	**184**	192
j120_31_8	**177**	192
j120_31_10	**203**	227
j120_36_10	**199**	216

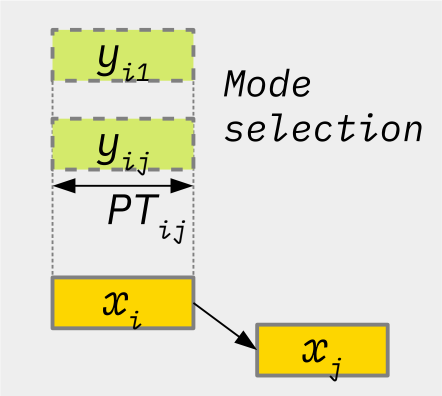
Instance	LB	UB
j120_37_3	**136**	139
j120_37_4	**157**	163
j120_37_7	**152**	161
j120_39_2	**106**	108
j120_46_5	**140**	149
j120_47_8	**127**	133
j120_48_2	**112**	113
j120_48_6	**103**	105
j120_51_1	**187**	206
j120_52_10	**135**	144
j120_57_2	**152**	161
j120_58_1	**134**	141
j120_59_2	**104**	106

- Additional instances were improved in 2019:

  `http://www.om-db.wi.tum.de/psplib/getdata_sm.html`

Academic problems: Multi-Mode RCPSP

- Multi-Mode RCPSP (MMRCPSP)



$$\min \max_{i\in[1..N]} \text{endOf}(x_i)$$

$$\text{alternative}(x_i, [y_{ij}]_{j\in M[i]}) \qquad \forall i \in [1..N] \qquad (2)$$

$$\sum_{i\in[1..N], j\in M[i]} \text{pulse}(y_{ij}, QR_{ijk}) \leq CR_k \qquad \forall k \in [1..R] \qquad (3)$$

$$\sum_{i\in[1..N], j\in M[i]} \text{presenceOf}(y_{ij}) \cdot QS_{ijk} \leq CS_k \qquad \forall k \in [1..S] \qquad (4)$$

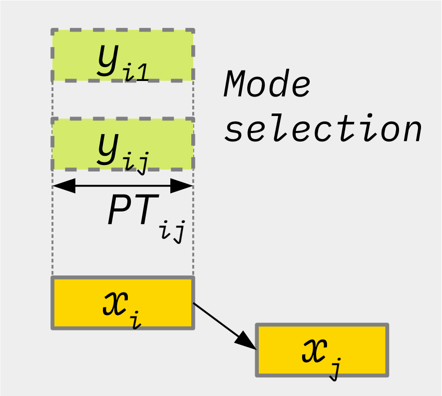
$$\text{endBeforeStart}(x_i, x_j) \qquad \forall (i,j) \in P \qquad (5)$$

$$\text{interval } x_i \qquad \forall i \in [1..N] \qquad (6)$$

$$\text{interval } y_{ij}, \text{ optional, size} = PT_{ij} \qquad \forall i \in [1..N], j \in M[i] \qquad (7)$$

Academic problems: Multi-Mode RCPSP-DC

- Multi-Mode RCPSP with Discounted Cash Flows



$$\max \sum_{i \in [1..N]} CF[i] \cdot e^{-\alpha \cdot \text{endOf}(x_i)}$$

$$\text{alternative}(x_i, [y_{ij}]_{j \in M[i]}) \qquad \forall i \in [1..N] \qquad (2)$$

$$\sum_{i \in [1..N], j \in M[i]} \text{pulse}(y_{ij}, QR_{ijk}) \leq CR_k \qquad \forall k \in [1..R] \qquad (3)$$

$$\sum_{i \in [1..N], j \in M[i]} \text{presenceOf}(y_{ij}) \cdot QS_{ijk} \leq CS_k \qquad \forall k \in [1..S] \qquad (4)$$

$$\text{endBeforeStart}(x_i, x_j) \qquad \forall (i,j) \in P \qquad (5)$$

$$\text{interval } x_i \subset [0, H) \qquad \forall i \in [1..N] \qquad (6)$$

$$\text{interval } y_{ij}, \text{ optional}, \text{ size} = PT_{ij} \qquad \forall i \in [1..N], j \in M[i] \qquad (7)$$

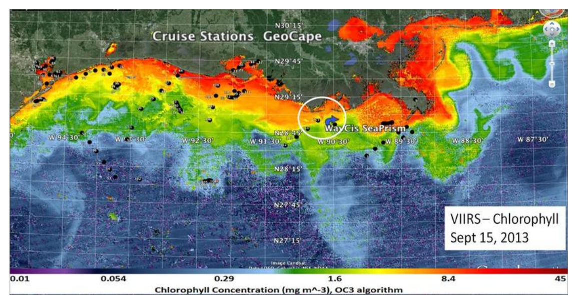
Industrial scheduling applications

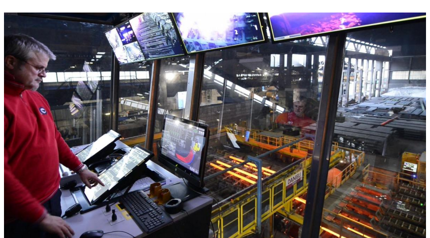
- In the real life, scheduling problems are complex

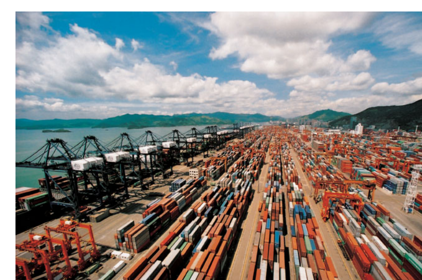






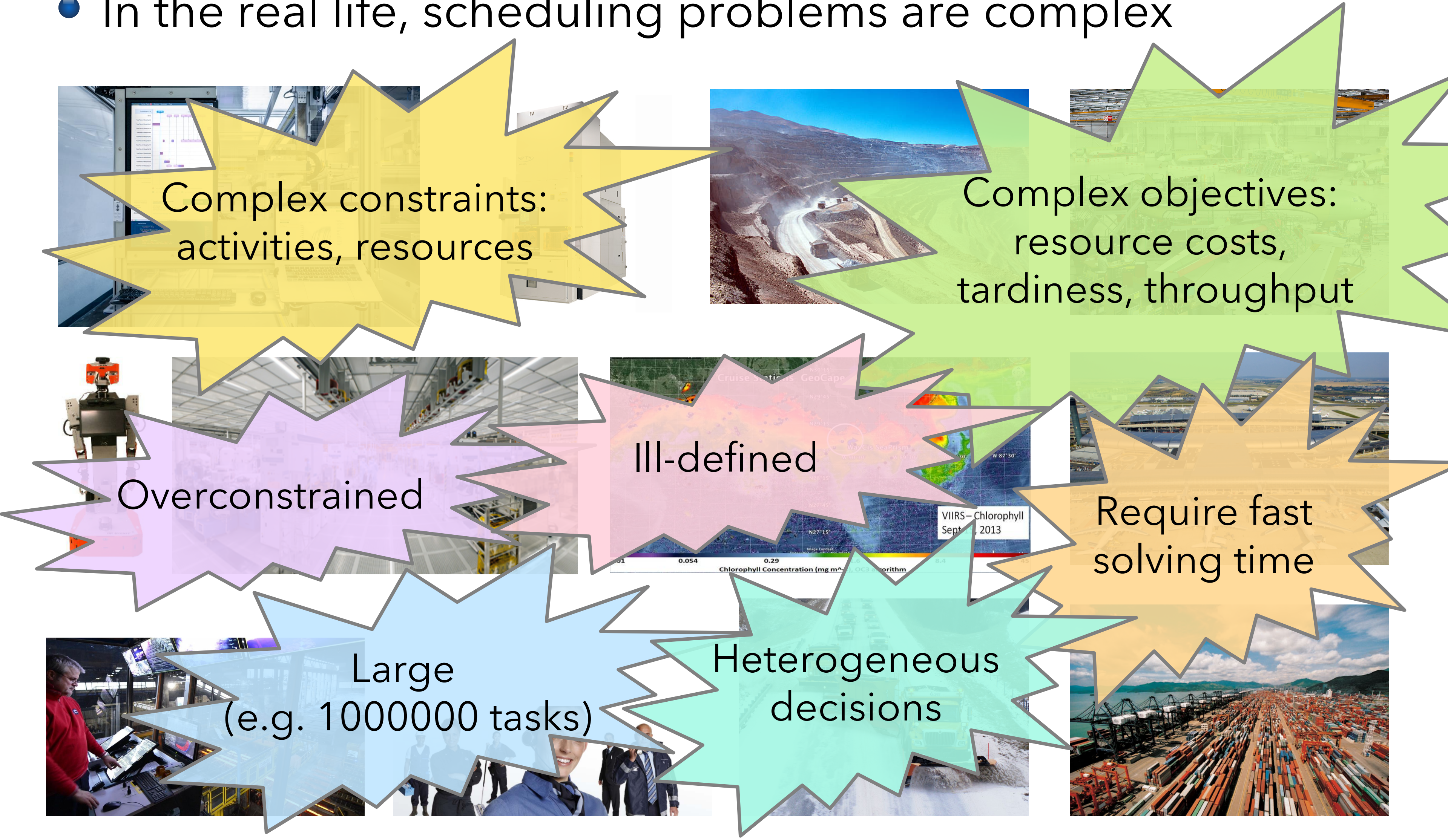


Industrial scheduling applications

- In the real life, scheduling problems are complex



Few but versatile modeling concepts

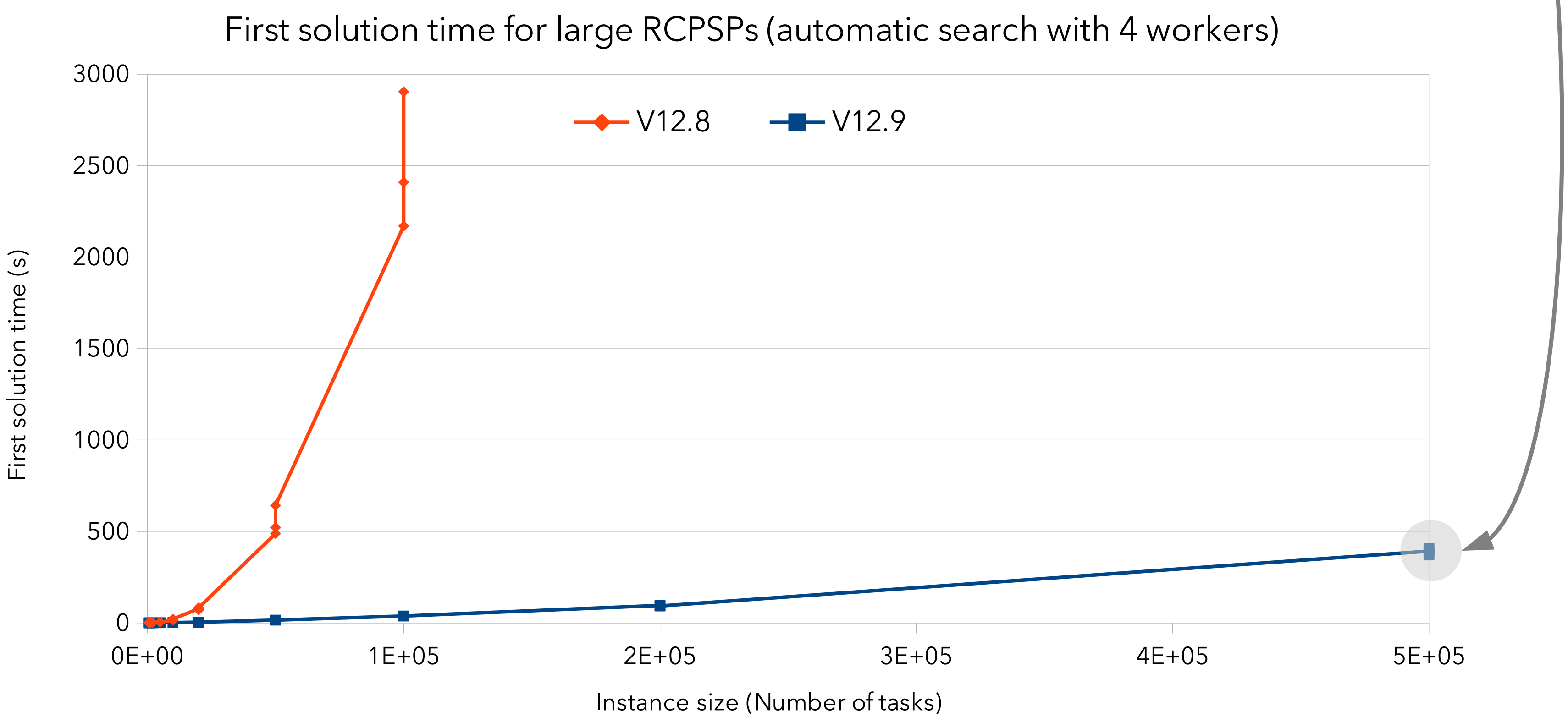
Earliness/tardiness costs	Constant functions	Resource calendars Resource efficiency
Temporal constraints Optional activities Over-constrained problems Alternative resources/modes Work-breakdown structures	Interval variables	
	Sequence variables	Unary resources Setup times/costs Travel times/costs
	Cumul functions	Cumulative resources Inventories, Reservoirs
	State functions	Parallel batches Activity incompatibilities
Aggregation of individual costs (max, weighted sum, Net Present Value)	General arithmetical expressions	

Scaling

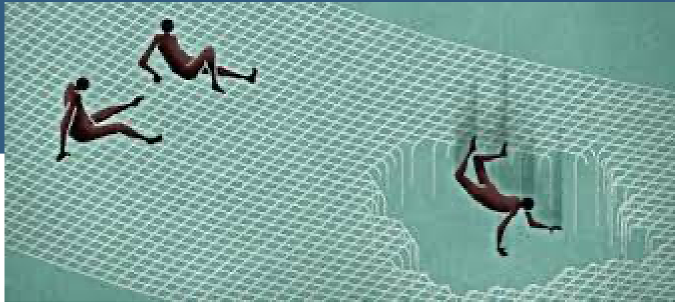
- First question before starting to think of an approach to solve a real (scheduling) problem:
  - What is the **actual** size *n* of the problem ?
  - Start thinking of an approach/formulation to solve problems of size *2n* or *5n* … Not *n/10* or *n/100* !!!
- Example: if *n=1.000.000*, forget about a formulation (number of variables or constraints) that would be in $O(n^2)$ or even worse
- From the start of the project, work with data of realistic size (even if simplified, even if synthetic)
- Size of CP Optimizer formulations for scheduling problems usually scale in $O(n)$

Scaling example on RCPSP

- New benchmark with RCPSP from 500 to 500.000 tasks
  - Largest problem: 500.000 tasks, 79 resources, 4.740.783 precedences, 4.433.550 resource requirements
- Time to first feasible solution (V12.8 v.s. V12.9)



Safety nets



- Starting point solutions (a.k.a. warm start)
- Blackbox expressions (NEW)

Safety nets: starting point solutions

- The search can be specified a **starting point solution** as input. Use cases:
  - Search process has been interrupted; restart from last solution
  - A problem specific heuristic is available to provide a solution to start from
  - Multi-objective lexicographical objective: minimize f1, then minimize f2 with some constraint on f1, …
  - When hard to find a feasible solution: start from a relaxed problem that minimizes constraint violation
  - Solving very similar successive models, for instance in dynamic scheduling, in re-scheduling
- If the starting point is feasible and complete, the search is **guaranteed** to first visit this solution
- Otherwise, the information in the starting point is used as a heuristic guideline for the search

Safety nets: blackbox expressions

- Black-Box function:
  - A function f(X): $R^n \rightarrow R$ for which the analytic form is not known
  - The user provides a function that can be called to compute the value f(X) on fixed parameter values X
- A black-box function can be evaluated to obtain:
  - Value : f(4,6,2) → 4.435
  - Definiteness : f(5,5,2) → undefined
- Example of black-box function:
  - Formulation with predefined expressions would be very costly
  - Legacy code: no access to what is inside a library/executable
  - Numerical code involving differential equations, integrals, …
  - Result of a complex simulation (schedule, policy)
  - Prediction of a machine learning model

Safety nets: blackbox expressions

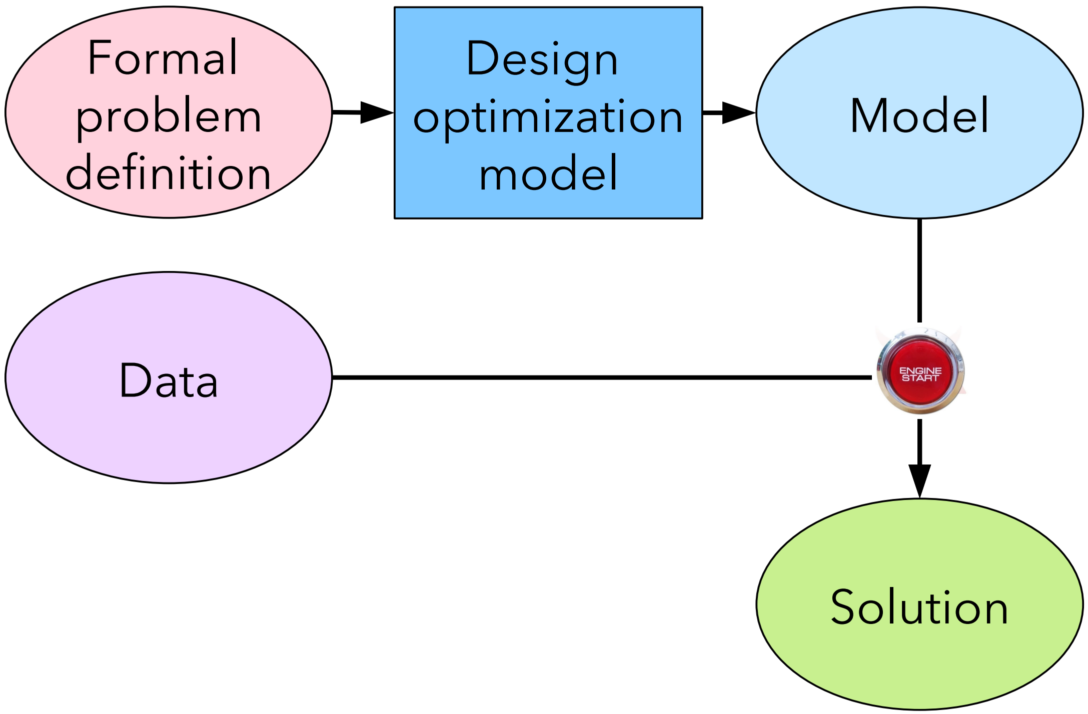
- Since last release **blackbox expressions** permit to extend the predefined set of expressions

```
double f(double a, double b, double c); // Blackbox function

ILOBLACKBOX3(BBF, IloNumExpr, u, IloNumExpr, v, IloNumExpr, w) {
  returnValue( f(getValue(u), getValue(v), getValue(w)) );
}

model.add( … );
IloNumExpr bbf = BBF(env,x,y,z);
model.add( IloMinimize(env, bbf) );
model.add( x+y+z <= bbf );
```

- All types of variables/expressions are supported as arguments (integer, interval, ...), individually or in arrays
- Blackbox expressions can be used as any other expression in the model (no restriction)
- Search remains complete

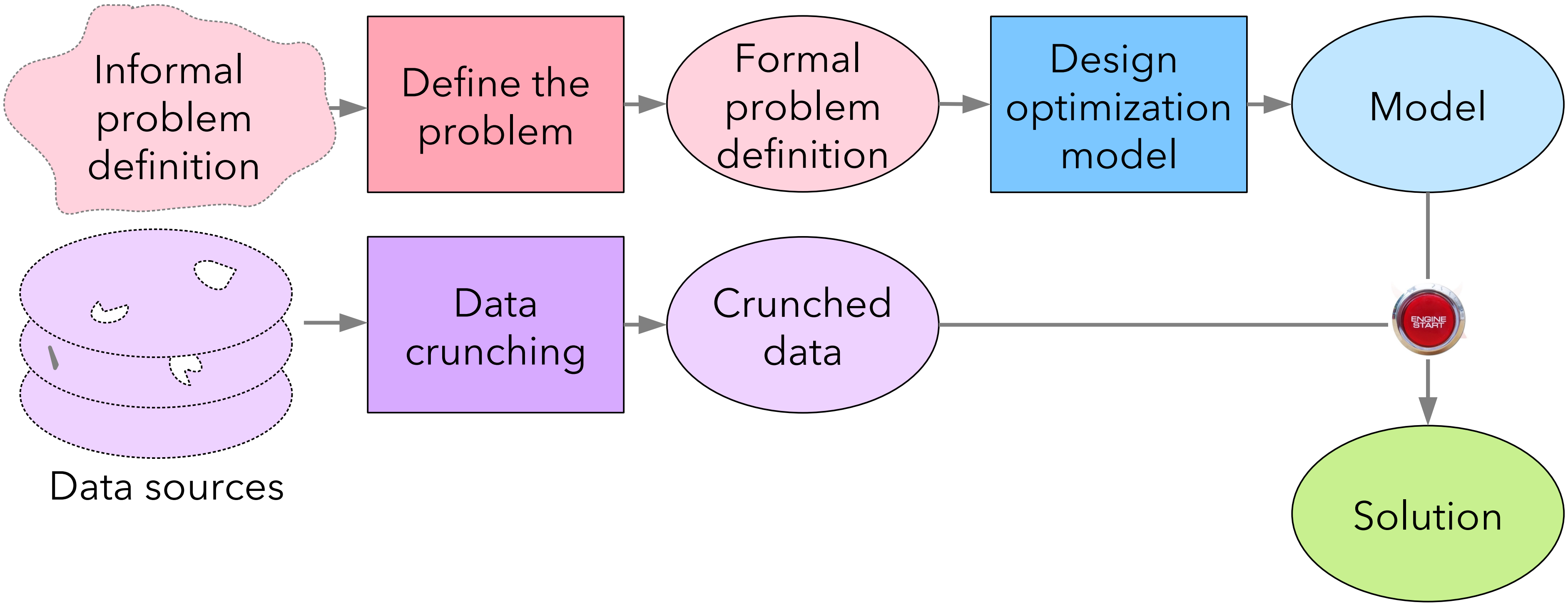
Modeling tools

- Process for building an optimization engine for an application



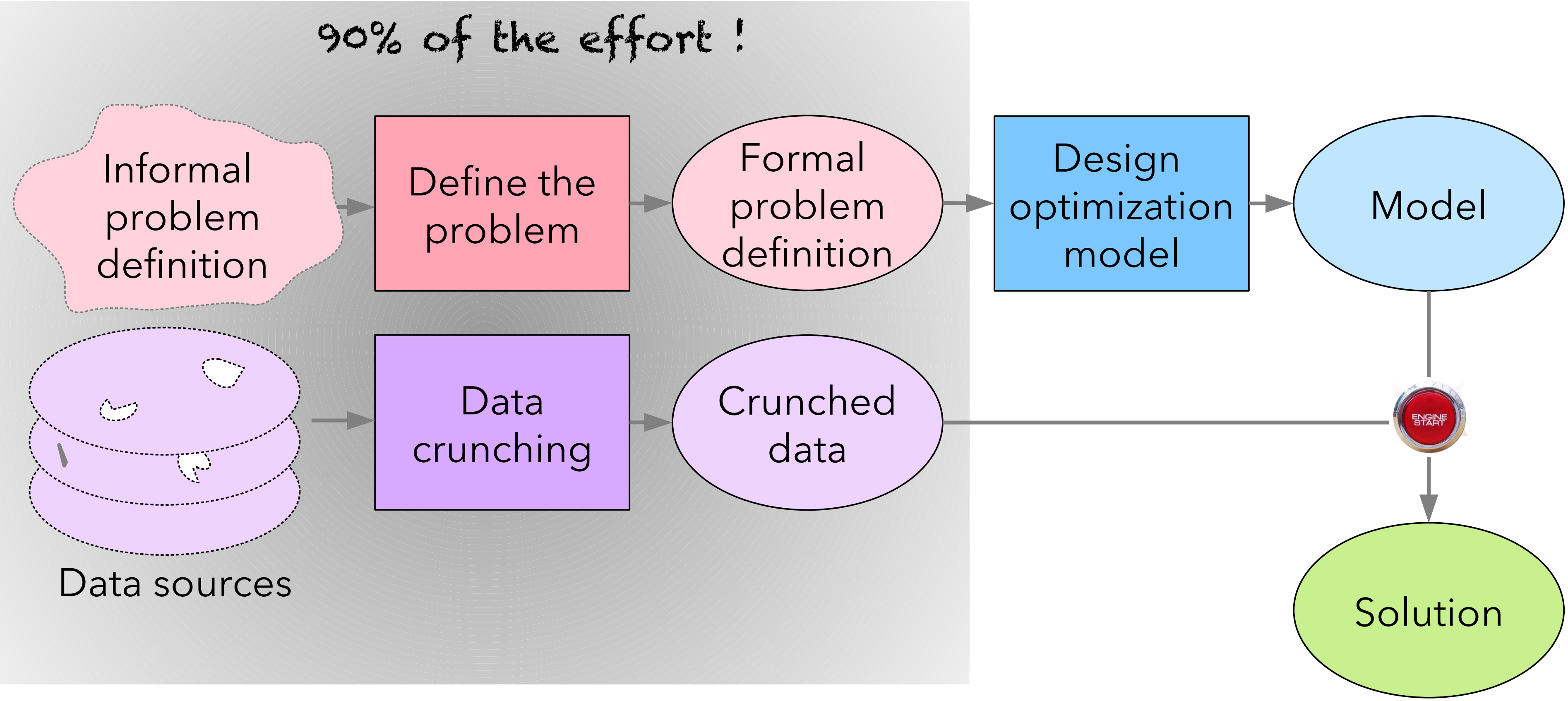
Modeling tools

- Process for building an optimization engine for an application
- Reality of industrial projects is more complex



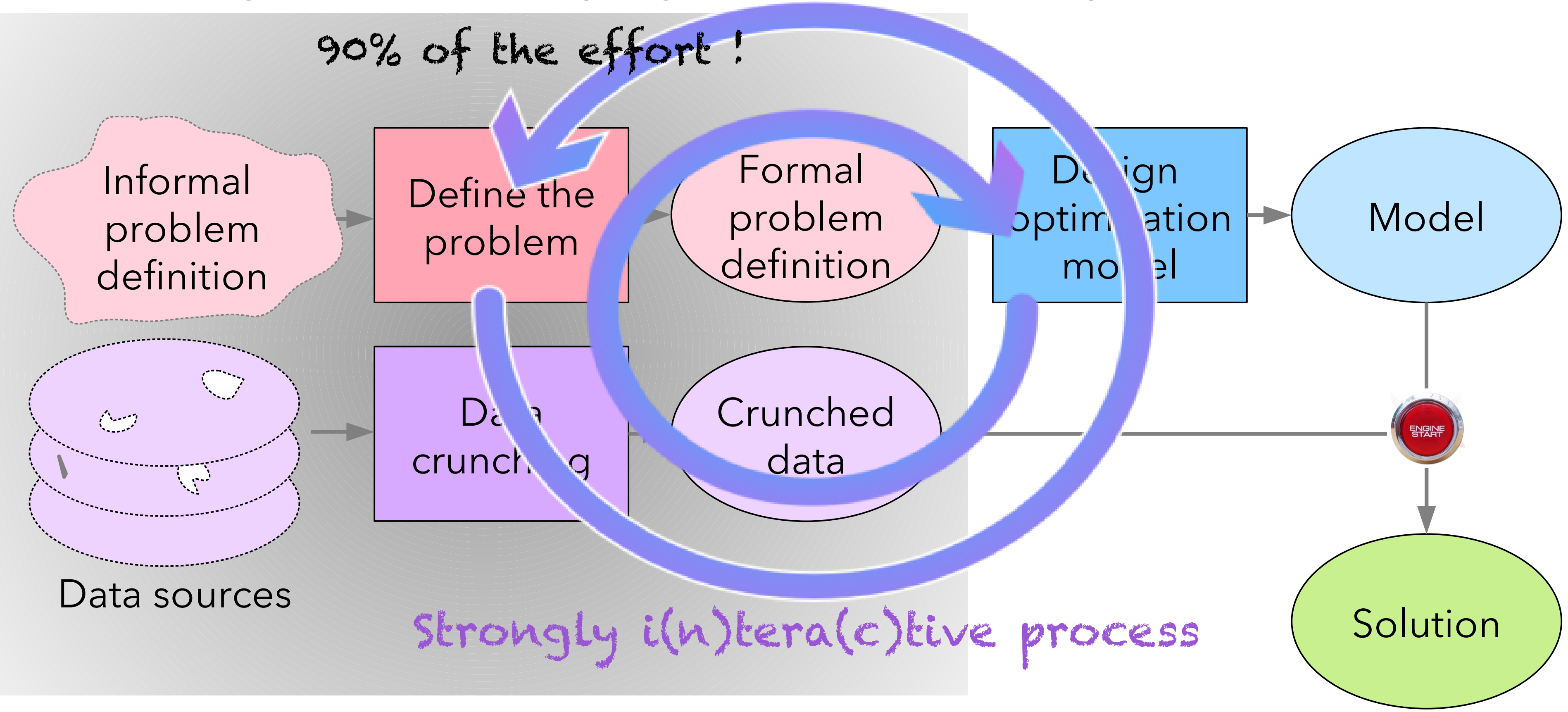
Modeling tools

- Process for building an optimization engine for an application
- Reality of industrial projects is more complex



Modeling tools

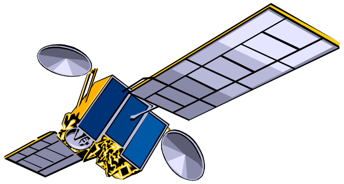
- Process for building an optimization engine for an application
- Reality of industrial projects is more complex



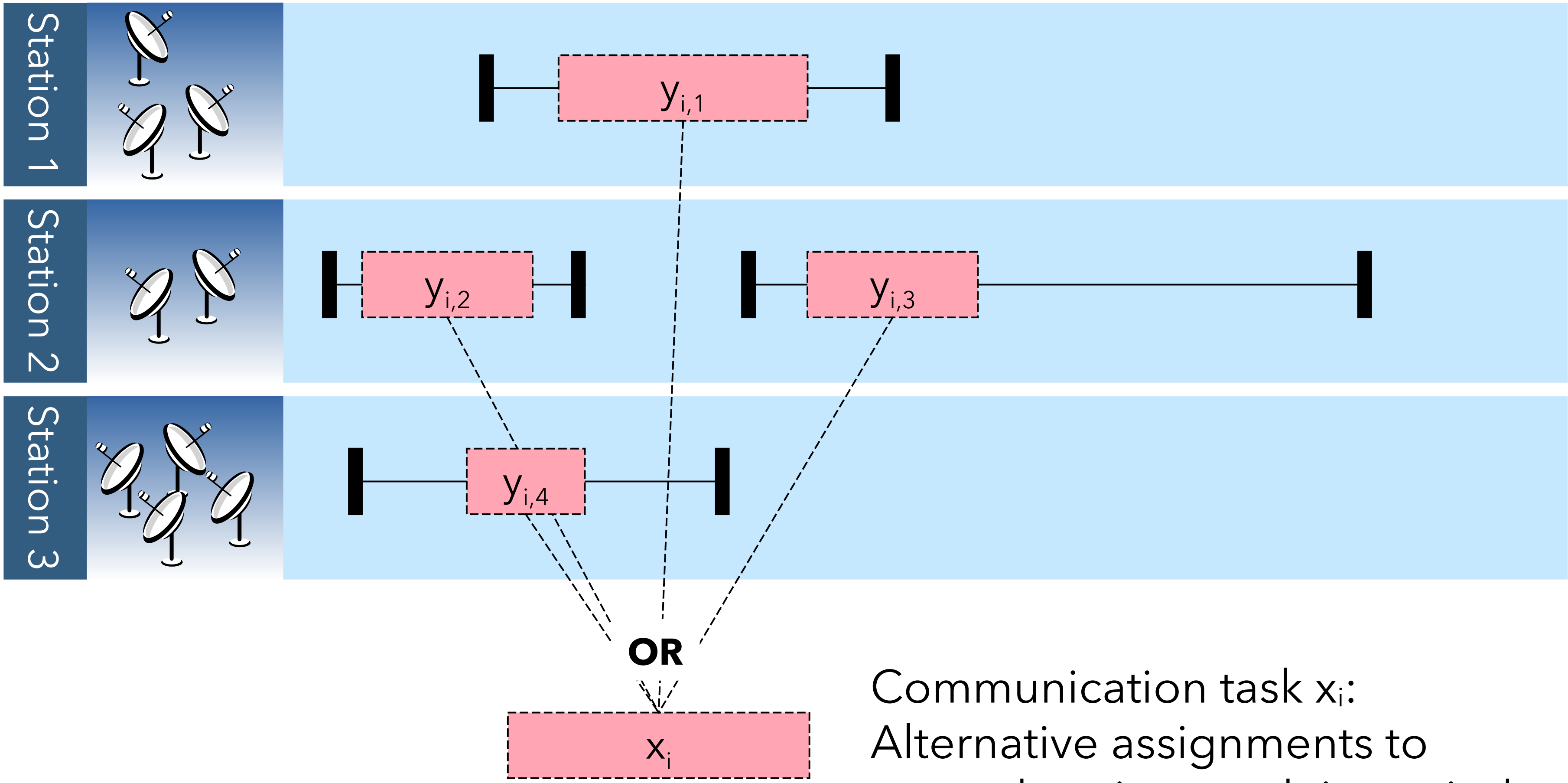
Modeling tools

- Typical questions/issues arising during model design
  - How does my current model look like when instantiated on some data ?
  - Does it contains some weird things I'm not aware of ?
  - Why is it infeasible ?
    - Bug in the model ?
    - Bug in the data ?
  - Why is it difficult to find a feasible solution?
  - Is my model performing better than another variant I tried?

Example: satellite communication scheduling



L. Kramer, L. Barbulescu, S. Smith. "*Understanding Performance Trade-offs in Algorithms for Solving Oversubscribed Scheduling*". In: Proc. AAAI 2007.



Communication task x_i: Alternative assignments to ground stations and time windows

Modeling tools: example

- Example of a satellite scheduling problem

```
x = { i : interval_var(name=i)  for i in T }
y = { o : interval_var(optional=True, size=o[3], start=[o[2],o[4]],
                       end=[o[2],o[4]], name=str(o))        for o in O }

model.add(
 [ maximize( sum( [ presence_of(x[i])   for i in T ]))                ] +
 [ alternative(x[i], [ y[o] for o in O if o[0]==i ] )        for i in T ] +
 [ sum( [ pulse(y[o],1) for o in O if o[1]==s ]) <= S[s][2]  for s in S ]
)

model.export_model("satellite.cpo")
```

- Export/import model instance as a .cpo file

Modeling tools: input/output file format (.cpo)

```
"42"    = intervalVar();
"43"    = intervalVar();
"42A"   = intervalVar();
"207A"  = intervalVar();
...
"('42', 3, 400, 27, 435)"   = intervalVar(optional, start=400..435, end=400..435, size=27);
"('43', 3, 391, 21, 427)"   = intervalVar(optional, start=391..427, end=391..427, size=21);
"('42A', 3, 389, 34, 424)"  = intervalVar(optional, start=389..424, end=389..424, size=34);
"('207A', 2, 223, 21, 313)" = intervalVar(optional, start=223..313, end=223..313, size=21);
"('207A', 3, 223, 21, 313)" = intervalVar(optional, start=223..313, end=223..313, size=21);
...

maximize(sum([presenceOf("42"), ...,]));

alternative("42",   ["('42', 3, 400, 27, 435)"]);
alternative("43",   ["('43', 3, 391, 21, 427)"]);
alternative("42A",  ["('42A', 3, 389, 34, 424)"]);
alternative("207A", ["('207A', 2, 223, 21, 313)", "('207A', 3, 223, 21, 313)"]);
...
pulse("('42', 3, 400, 27, 435)",1)  + pulse("('43', 3, 391, 21, 427)",1) +
pulse("('42A', 3, 389, 34, 424)",1) + ... + pulse("('207A', 3, 223, 21, 313)",1) <= 2;
...
```

Modeling tools: example

- Example of a satellite scheduling problem

```
x = { i : interval_var(name=i)  for i in T }
y = { o : interval_var(optional=True, size=o[3], start=[o[2],o[4]],
                       end=[o[2],o[4]], name=str(o))        for o in O }

model.add(
 [ maximize( sum( [ presence_of(x[i])   for i in T ]))                ] +
 [ alternative(x[i], [ y[o] for o in O if o[0]==i ] )        for i in T ] +
 [ sum( [ pulse(y[o],1) for o in O if o[1]==s ]) <= S[s][2]  for s in S ]
)

model.solve(TimeLimit=20)
```

Modeling tools: search log

```
! ---------------------------------------------------- CP Optimizer 20.1.0.0 --
! Maximization problem - 2980 variables, 851 constraints
! Initial process time : 0.02s (0.02s extraction + 0.00s propagation)
!  . Log search space  : 30213.9 (before), 30213.9 (after)
!  . Memory usage      : 9.6 MB (before), 9.6 MB (after)
! Using parallel search with 8 workers.
! ----------------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
                       0       2980                 -
+ New bound is 838
! ----------------------------------------------------------------------------
! Search completed, model has no solution.
! Best bound             : 838
! ----------------------------------------------------------------------------
! Number of branches     : 0
! Number of fails        : 0
! Total memory usage     : 15.3 MB (13.7 MB CP Optimizer + 1.6 MB Concert)
! Time spent in solve    : 0.02s (0.00s engine + 0.02s extraction)
! Search speed (br. / s) : 0
! ----------------------------------------------------------------------------
```

Modeling tools: example

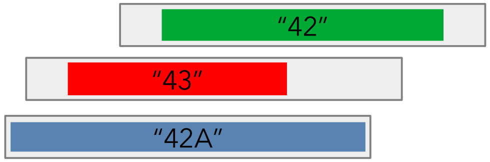
- Example of a satellite scheduling problem

```
x = { i : interval_var(name=i)  for i in T }
y = { o : interval_var(optional=True, size=o[3], start=[o[2],o[4]],
                       end=[o[2],o[4]], name=str(o))        for o in O }

model.add(
 [ maximize( sum( [ presence_of(x[i])   for i in T ]))                  ] +
 [ alternative(x[i], [ y[o] for o in O if o[0]==i ] )        for i in T ] +
 [ sum( [ pulse(y[o],1) for o in O if o[1]==s ]) <= S[s][2]  for s in S ]
)

model.refine_conflict().print_conflict()
```

- Conflict refiner extracts the smallest subset of constraints that explains the infeasibility
- P. Laborie. *An Optimal Iterative Algorithm for Extracting MUCs in a Black-box Constraint Network*. In: Proc. ECAI-2014

Modeling tools: conflict refiner

```
 ! ----------------------------------------------------------------------------
 ! Conflict refining - 851 constraints
 ! ----------------------------------------------------------------------------
 !   Iteration      Number of constraints
 *          1                        851
 *          2                        426
 ...
 *         47                          4
 ! Conflict refining terminated
 ! ----------------------------------------------------------------------------
 ! Conflict status          : Terminated normally, conflict found
 ! Conflict size            : 4 constraints
 ! Number of iterations     : 47
 ! Total memory usage       : 13.7 MB
 ! Conflict computation time : 0.43s
 ! ----------------------------------------------------------------------------

Conflict refiner result:

Member constraints:
   alternative("42",  ["('42',  3, 400, 27, 435)"])
   alternative("43",  ["('43',  3, 391, 21, 427)"])
   alternative("42A", ["('42A', 3, 389, 34, 424)"])
   sum([pulse("('42', 3, 400, 27, 435)",1)  + pulse("('43', 3, 391, 21, 427)",1) +
        pulse("('42A', 3, 389, 34, 424)",1) + ... + pulse("('207A', 3, 223, 21, 313)",1)]) <= 2

"42"  = intervalVar();
"43"  = intervalVar();
"42A" = intervalVar();
```



Modeling tools: model warnings

```
[ maximize( sum( [ presence_of(x[i])   for i in T ]))                ] +
```

```
/Users/laborie/Satellite/satellite.py:21: Warning: Boolean expression 'presenceOf' is
always true because interval variable '42' is declared present.
                                          presenceOf("42")
/Users/laborie/Satellite/satellite.py:21: Warning: Boolean expression 'presenceOf' is
always true because interval variable '43' is declared present.
                                          presenceOf("43")
/Users/laborie/Satellite/satellite.py:21: Warning: Boolean expression 'presenceOf' is
always true because interval variable '42A' is declared present.
                                          presenceOf("42A")
…

Too many warnings of this type. Suppressing further warnings of this type.
```

Modeling tools: example

- Example of a satellite scheduling problem

```
x = { i : interval_var(optional=True, name=i)  for i in T }
y = { o : interval_var(optional=True, size=o[3], start=[o[2],o[4]],
                       end=[o[2],o[4]], name=str(o))        for o in O }

model.add(
 [ maximize( sum( [ presence_of(x[i])   for i in T ]))               ] +
 [ alternative(x[i], [ y[o] for o in O if o[0]==i ] )        for i in T ] +
 [ sum( [ pulse(y[o],1) for o in O if o[1]==s ]) <= S[s][2]  for s in S ]
)

model.solve(TimeLimit=20)
```

Modeling tools: search log

```
! ----------------------------------------------- CP Optimizer 20.1.0.0 --
! Maximization problem - 2980 variables, 851 constraints
! TimeLimit            = 20
! LogPeriod            = 100000
! Initial process time : 0.05s (0.04s extraction + 0.01s propagation)
!  . Log search space  : 4627.3 (before), 4627.3 (after)
!  . Memory usage      : 12.1 MB (before), 12.1 MB (after)
! Using parallel search with 8 workers.
! ----------------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
                       0       2980                    -
+ New bound is 838
! Using iterative diving.
! Using temporal relaxation.
*           785      2142  0.27s        7      (gap is 6.75%)
*           793      9796  0.27s        7      (gap is 5.67%)
*...
            821     52389        271    5   F          -
+ New bound is 837 (gap is 1.95%)
*           822     44536  8.04s        6      (gap is 1.82%)
! Using failure-directed search.
*           823     60147  8.75s        3      (gap is 1.70%)
...
```

Modeling tools: search log

```
...
! Time = 19.37s, Average fail depth = 486, Memory usage = 113.0 MB
! Current bound is 837 (gap is 1.33%)
!          Best Branches  Non-fixed    W       Branch decision
            826     100k          2    4       710  = startOf(('85', 5, 710, 29, 743))
            826     200k          2    2       911  = startOf(('334', 10, 911, 22, 986))
! ----------------------------------------------------------------------------
! Search terminated by limit, 12 solutions found.
! Best objective         : 826 (gap is 1.33%)
! Best bound             : 837
! ----------------------------------------------------------------------------
! Number of branches     : 4399997
! Number of fails        : 210551
! Total memory usage     : 109.2 MB (107.6 MB CP Optimizer + 1.6 MB Concert)
! Time spent in solve    : 20.01s (19.97s engine + 0.04s extraction)
! Search speed (br. / s) : 220330.3
! ----------------------------------------------------------------------------
```

Modeling tools: example

- Example of a satellite scheduling problem

```
x = { i : interval_var(optional=True, name=i)  for i in T }
y = { o : interval_var(optional=True, size=o[3], start=[o[2],o[4]],
                       end=[o[2],o[4]], name=str(o))        for o in O }

model.add(
 [ maximize( sum( [ presence_of(x[i])   for i in T ]))              ] +
 [ alternative(x[i], [ y[o] for o in O if o[0]==i ] )        for i in T ] +
 [ sum( [ pulse(y[o],1) for o in O if o[1]==s ]) <= S[s][2]  for s in S ]
)

model.run_seeds(30, TimeLimit=20)
```

- Run instance *n* times (here *n=30*) with different random seeds (*1,2,…,n*) and perform some statistical analysis on the results to asses stability of the search

Modeling tools: solve stability

```
Benchmarking current problem on 30 runs...
Run        Soln       Proof       Branches     Time (s)        Objective
--------------------------------------------------------------------
  1          1           0        4672277        20.01              826
  2          1           0        4197814        20.06              826
  3          1           0        3040173        20.03              826
  4          1           0        3446413        20.01              826
  5          1           0        3640692        20.12              826
  6          1           0        3532742        20.10              825
  7          1           0        3689278        20.01              826
  8          1           0        3427675        20.02              826
  9          1           0        3457423        20.10              824
...
 29          1           0        3522099        20.01              826
 30          1           0        3696967        20.02              825
--------------------------------------------------------------------
All runs stopped by limit
Mean       1.00        0.00       3578449        20.03       825.833333
Std dev                            424327         0.03         0.461133
Geomean                           3553656        20.03
Min                               2691157        20.01              824
Max                               4672277        20.12              826
```

Conclusion

- Consider using/comparing to **CP** when working on **scheduling problems** (ILP often is not competitive)
- CP Optimizer provides:
  - A **mathematical modeling language** for combinatorial optimization problems that extends ILP (and classical CP) with some algebra on **intervals** and **functions** allowing compact and maintainable formulations for complex scheduling problems
  - A continuously improving **automatic search algorithm** that is complete, anytime, efficient (often competitive with problem-specific algorithms) and **scalable**
- If you are using CPLEX for ILP, then you already have CP Optimizer in the box !

Last-minute slide

- The *two-stage stochastic programming and recoverable robustness* problem described by Marjan this morning

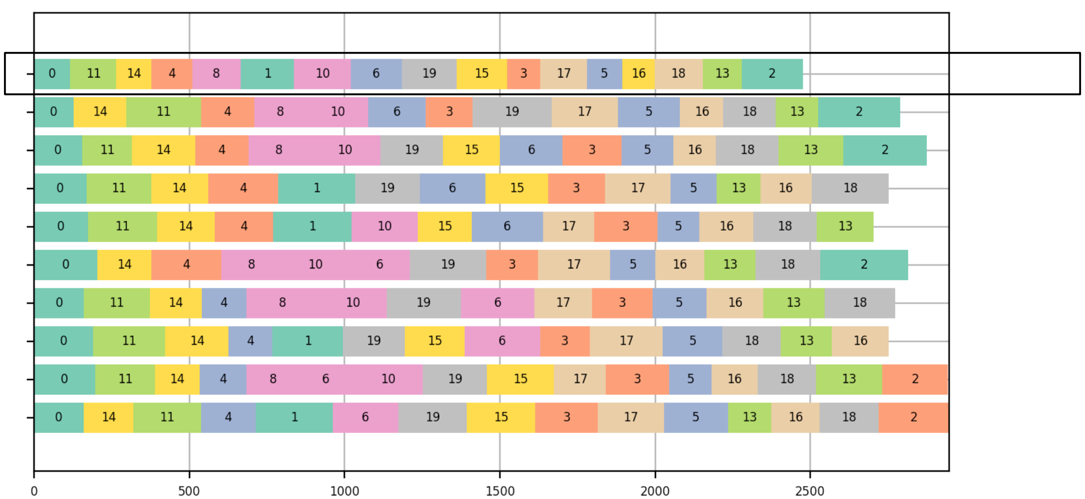
```
x = [ [interval_var(size=P[k][i], optional=True, end=[0,D[i]]) for i in N] for k in S ]

model.add(
 [ maximize(sum(Q[k]*presence_of(x[k][i]) for i in N for k in S ))           ] +
 [ no_overlap(x[k][i] for i in N)                  for k in S                ] +
 [ presence_of(x[k][i]) <= presence_of(x[0][i])  for i in N for k in range(1,s) ]
)
```

Last-minute slide

- The *two-stage stochastic programming and recoverable robustness* problem described by Marjan this morning

```
x = [ [interval_var(size=P[k][i], optional=True, end=[0,D[i]]) for i in N] for k in S ]

model.add(
 [ maximize(sum(Q[k]*presence_of(x[k][i]) for i in N for k in S ))          ] +
 [ no_overlap(x[k][i] for i in N)                  for k in S                ] +
 [ presence_of(x[k][i]) <= presence_of(x[0][i])  for i in N for k in range(1,s) ]
)
```

20 tasks x 10 scenarios



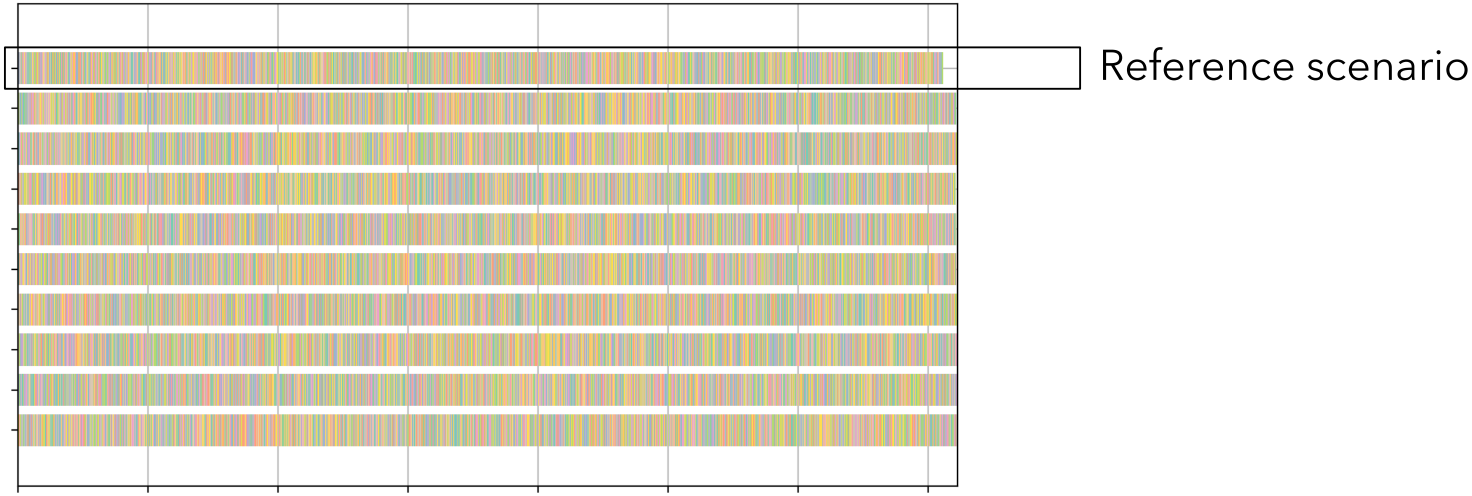
Reference scenario

Last-minute slide

- The *two-stage stochastic programming and recoverable robustness* problem described by Marjan this morning

```
x = [ [interval_var(size=P[k][i], optional=True, end=[0,D[i]]) for i in N] for k in S ]

model.add(
 [ maximize(sum(Q[k]*presence_of(x[k][i]) for i in N for k in S ))             ] +
 [ no_overlap(x[k][i] for i in N)                 for k in S                   ] +
 [ presence_of(x[k][i]) <= presence_of(x[0][i])  for i in N for k in range(1,s) ]
)
```

10.000 tasks x 10 scenarios, solution after 5mn



Some pointers

- Recent review of CP Optimizer (modeling concepts, applications, examples, tools, performance,…) :

  IBM ILOG CP Optimizer for scheduling. Constraints journal (2018) vol. 23, p210–250. **http://ibm.biz/Constraints2018**

- CP Optimizer forum: **http://ibm.biz/COS_Forums** (same as CPLEX)

Some references

- P. Laborie, J. Rogerie. Reasoning with Conditional Time-Intervals. In: Proc. FLAIRS-2008, p555-560.
- P. Laborie, J. Rogerie, P. Shaw, P. Vilím. Reasoning with Conditional Time-Intervals. Part II: An Algebraical Model for Resources. In: Proc. FLAIRS-2009, p201-206.

Modeling concepts

- P. Laborie, D. Godard. Self-Adapting Large Neighborhood Search: Application to Single-Mode Scheduling Problems. In: Proc. MISTA-2007.
- P. Laborie, J. Rogerie. Temporal Linear Relaxation in IBM ILOG CP Optimizer. Journal of Scheduling 19(4), 391–400 (2016).
- P. Vilím. Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources . In: Proc. CPAIOR-2011.
- P. Vilím, P. Laborie, P. Shaw. Failure-directed Search for Constraint-based Scheduling. In: Proc. CPAIOR-2015.

Search algorithm

- P. Laborie, J. Rogerie, P. Shaw, P. Vilím. IBM ILOG CP Optimizer for Scheduling. Constraints Journal (2018).

Overview