



Your Connection to **ICT** Research

# Routing Optimisation with Oscala.cblls and some context

Oscala v4.0 - Spring 2017

Renaud De Landtsheer, Yoann Guyot,  
Christophe Ponsard, Gustavo Ospina,  
Fabian Germeau



# Oscala

OPERATIONAL RESEARCH IN SCALA

- Academia:
  - **Given**
    - the distance matrix
  - **Find**
    - The cheapest permutation
- In the real world:
  - You have to compute the distance matrix
    - 100 points leads to ~10k distance
    - 0.1 sec per distance leads to 16 minutes

- Staff: 42
- Budget 2015: 4.8 M€
- Three research department:
  - Software & System Engineering
    - Software engineering, formal methods, code analysis, algorithmic, optimization, requirements engineering
  - Software and service technologies
    - Cloud computing, distributes architectures, data management
  - Embedded and Communication Systems
    - IoT, heterogeneous hardware architectures
- Two economical activities:
  - Research projects:
    - H2020, Cornet, Regional, etc
  - Service to industry:
    - custom, short term, paid by company, IP transferred

- PIPAs project: Job-shop scheduling
  - Lot of budget; develop a CBLs engine with iFlatRelax: Asteroid
- Merging code base with Scampi (Pierre Schaus): Oscala
- Service on routing optimization, delivered with GoogleCP
- SimQRi research project: Cornet
  - Research on how to represent search strategies, notion of combinators
- Service on Routing optimization round2:
  - Generating the distance matrix (a lot of work)
    - with traffic jam
    - Lots of algorithmic there (closed source, NDA)
  - Switching to Oscala.cbls (not so much work)
    - Because GoogleCP could not handle traffic jams
    - Speed improvement,
    - routing neighbourhoods into combinator framework

- Internships: symmetry elimination, parallel propagation, routing, bin packing, PDP, etc.
- Ongoing projects with OscaR.cbls:
  - SAMOBI research project
    - Sequence variable, refreshing the routing engine, PDP
  - H2020 TANGO
    - Flexible job-shop
  - 2 Regional
    - Large capacitated warehouse with additional constraints
    - Routing /scheduling stuff (not clear yet)
  - Cornet
    - Stochastic scheduling
  - (?factory scheduling?, eval ongoing)
- Tutorial ongoing

## — Oscar

- Open source framework for combinatorial optimization
- CP, CBLS, MIP, DFO engines

## — Open source LGPL license

- <https://bitbucket.org/oscarlib/oscar>
- Implemented in Scala

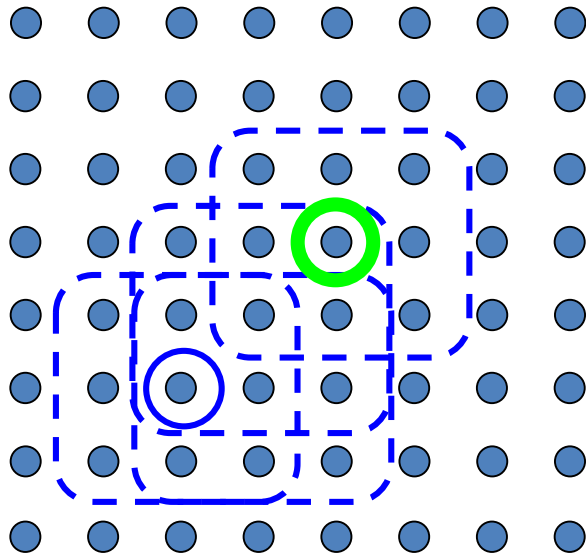
## — Consortium

- CETIC, UCL, N-Side Belgium
- Contributions from UPPSALA Sweden

- Introduction
  - CETIC, OascaR.cbIs
  - OascaR.CbIs
- Using OascaR.cbIs
  - Local Search
  - Warehouse location
- The OascaR.cbIs framework
  - Modelling
  - Searching
- Routing with OascaR.cbIs
  - Modelling
  - Searching
  - A simple example
  - A complex example
- More examples:
  - car sequencing
  - FlowShop
- Conclusion
- Future work
- Who is Who
- Some fun, in case you have questions

# Local search in one slide

*TSP : all the possible tours  
n cities;  $(n-1)!$  tours*



● Point in the search space

Some black magic required  
to escape from local minima

*TSP : random tour?*

Pick an initial solution

**Repeat**

Explore neighborhood

Move to best neighbor

**Until** no better neighbor

*TSP : moving a city*

*to another position in the tour*

*Current state:  $a \rightarrow b \rightarrow \textcolor{red}{c} \rightarrow d \rightarrow e \rightarrow a$*

*Moving city  $\textcolor{red}{c}$  yields three neighbors:*

*$a \rightarrow \textcolor{red}{c} \rightarrow b \rightarrow d \rightarrow e \rightarrow a$*

*$a \rightarrow b \rightarrow d \rightarrow \textcolor{red}{c} \rightarrow e \rightarrow a$*

*$a \rightarrow b \rightarrow d \rightarrow e \rightarrow \textcolor{red}{c} \rightarrow a$*

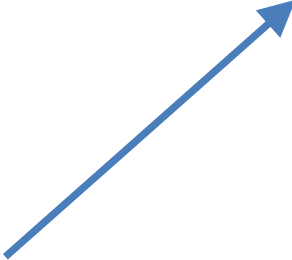
*$O(n^2)$  neighbors when considering all cities*



# *The basic equation of local search*

Local search–based solver = model + search procedure

Defines  
variables  
constraints  
Objectives  
...



Neighborhoods That modify  
some variables of the problem



# The uncapacitated warehouse location problem

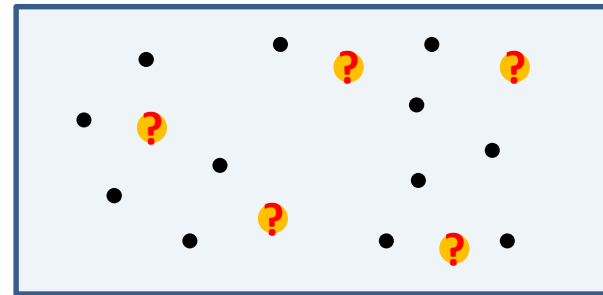
## • Given

- S: set of stores that must be stocked by the warehouses
- W: set of potential warehouses
  - Each warehouse has a fixed cost  $f_w$
  - transportation cost from warehouse  $w$  to store  $s$  is  $c_{ws}$

## • Find

- O: subset of warehouses to open
- Minimizing the sum of the fixed and the transportation cost.

$$\sum_{w \in O} f_w + \sum_{s \in S} \min_{w \in O} (c_{ws})$$



## • Notice

- A store is assigned to its nearest open warehouse

# A WLP solver written with neighbourhood combinators

```
val m = new Store()

val warehouseOpenArray = warehouses.map(
    CBLSIntVar(m, 0 to 1, 0, "warehouse_" + _ + "")).toArray

val openWarehouses = Filter(warehouseOpenArray)

val distanceToNearestOpenWarehouse = stores.map(
    min(distanceCost(_), openWarehouses,
        defaultCostForNoOpenWarehouse)).toArray

val obj = Objective(Sum(distanceToNearestOpenWarehouse)
    + Sum(costForOpeningWarehouse, openWarehouses))

m.close()

val neighborhood = (BestSlopeFirst(List(
    AssignNeighborhood(warehouseOpenArray, "SwitchWarehouse"),
    SwapsNeighborhood(warehouseOpenArray, "SwapWarehouses")),
    onExhaustRestartAfter(
        RandomizeNeighborhood(warehouseOpenArray, W/10), 2, obj))

val it = neighborhood.doAllMoves(obj)
```

# *Local search is (most of the time) black magic!*

- Non exhaustive
  - Seldom proof of optimality, only benchmarking
- Needs tuning:
  - Neighborhood rule
    - What neighborhood? What parameters?
  - Modeling
    - Soft, hard, implicit, automatic constraint?
  - Meta-heuristics
    - When to call neighborhoods? tabu? Restart? Simulated annealing?
- ... But it (can) work
  - 3-opt for TSP <3% to optimum in practice!!
  - iFlatRelax <1% to optimality for cumulative jobShop

→ Need for benchmarking, tuning, etc

OscaR.cbjs is about making this quick, so you can get the most of your algorithm

# *Local search is (most of the time) black magic!*

- Non exhaustive
  - Seldom proof of optimality, only benchmarks
- Needs tuning:
  - Neighborhood

*Local search is black magic*  
*You are wizards*  
*or spell inventors*  
*I am a wand maker*  
*with my team*

→ I

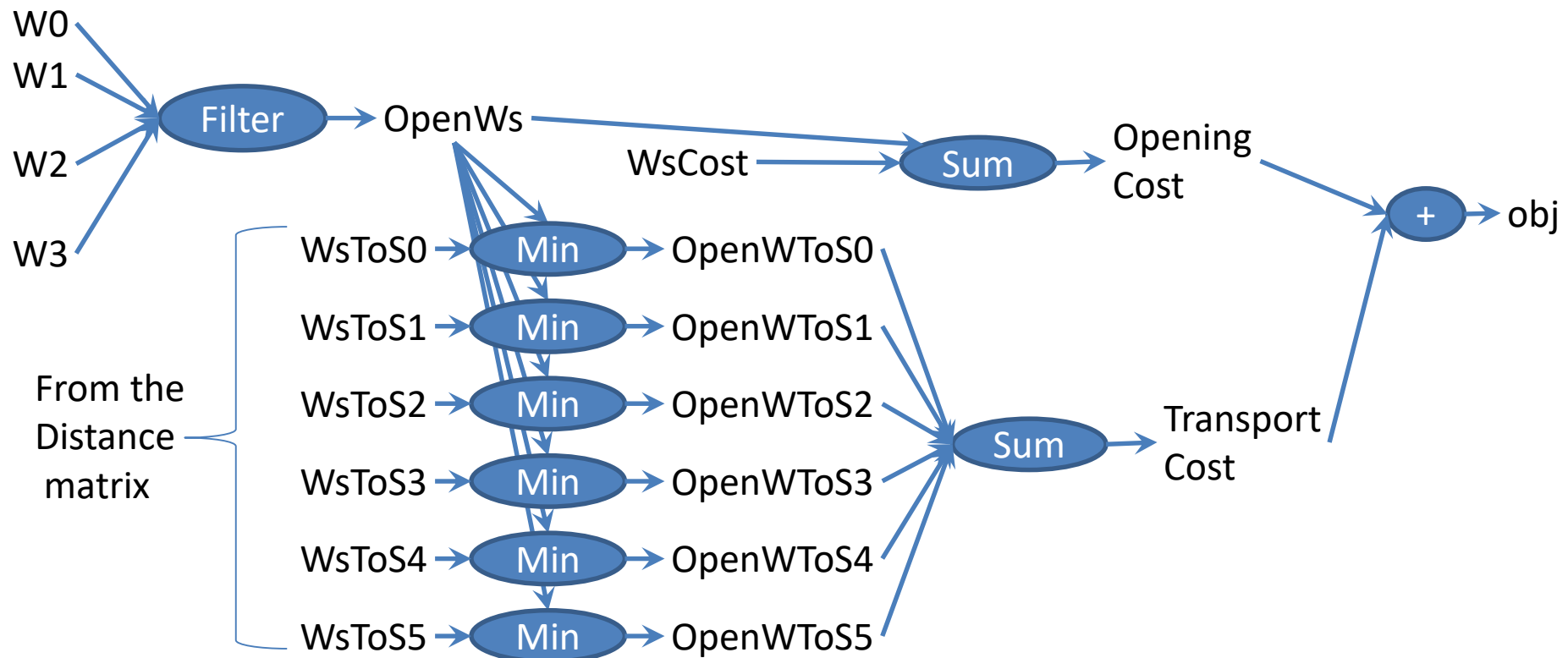
Os

can

so you

- Three types of variables
    - IntVar, SetVar, and SeqVar
  - Invariant library
    - Logic:
      - Access on array of Int/SetVar, Filter, Cluster , etc.
    - MinMax:
      - Min, Max, ArgMin, ArgMax
    - Numeric:
      - Sum, Prod, Minus, Div, Abs , etc.
    - Set:
      - Inter, Union, Diff, Cardinality , etc.
    - Seq:
      - Concatenate, Size, Content , etc.
    - Routig on Seq:
      - Constant Distance, Node-Vehicle restrictions, etc.
- Summing up to roughly 100 invariants in the library

# A quick look under the hood: Propagation graph for the WLP(4,6)



Propagation: update the output(s) to reflect a change on the inputs

- **Single wave**: elements are touched at most once
- **Incremental**: all invariants update their outputs incrementally
- **Selective**: only things that need to be updated wrt. changes are updated
- **Partial**: only things contributing to the needed output are updated

Automatic when using objectives, so mostly you do not have to worry about that

- Three sets of neighbourhoods
  - **Domain-independent:** assign, swap, flip, roll, shift, etc.
  - **Routing:** one point move, 2-opt, 3-opt, insert point, etc.
  - **Scheduling:** flatten, relaxlots of tuning: symmetry elimination, hot restart, best/first, search zone, etc.
- Neighbourhood combinators
  - Selecting neighbourhood
  - Stop criteria
  - Solution management
  - Meta-heuristics: restart, simulated annealing
  - Combined neighbourhood: cross-product “AndThen”, linear aggregation
  - Graphical display of objective function vs. run time
- Can also use customized search procedure based on linear selectors



- All Assigns, all swaps, all assigns, etc

```
val neighborhood = (AssignNeighborhood(warehouseOpenArray, "SwitchWarehouse")
    exhaustBack SwapsNeighborhood(warehouseOpenArray, "SwapWarehouses")
    orElse (RandomizeNeighborhood(warehouseOpenArray, W/5) maxMoves 2)
    saveBestAndRestoreOnExhaust obj)
```

- ... with best move for switch

```
search = (AssignNeighborhood(warehouseOpenArray, "SwitchWarehouse", best=true)
    exhaustBack SwapsNeighborhood(warehouseOpenArray, "SwapWarehouses")
    orElse (RandomizeNeighborhood(warehouseOpenArray, W/5) maxMoves 2)
    saveBestAndRestoreOnExhaust obj)
```

- Tabu search (requires model extension)

```
search = (AssignNeighborhood(warehouseOpenArray, "SwitchWarehouse "
    searchZone = nonTabuWarehouses, best=true)
```

```
acceptAll
```

```
afterMoveOnMove((a:AssignMove) => tabu(a.id) = lt.value + tabulength; lt += 1)
```

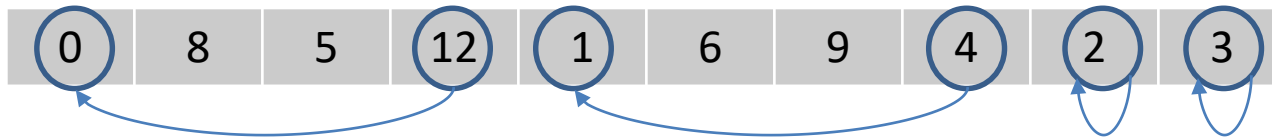
```
maxMoves xx withoutImprovementOver obj)
```

```
saveBestAndRestoreOnExhaust obj)
```

- Modelling
  - The sequence variable
  - Library of invariants
- Searching
  - Library of neighbourhoods
  - Compatible with our combinators
- Example
  - Simple benchmark VRP
  - A complex search strategy for deliveries

- Why? SPEED & GENERICITY !!
  - Efficient representation of moves in the sequence
  - Symbolic information on moves & check pointing
    - Makes it possible to develop efficient constraints and invariants
  - Library of efficient constraint and invariants,
    - Routing: distance matrix, node-vehicle restriction, etc.
    - Standard: size, content, flip, append, etc.
- Dedicated, efficient data-structures

V=4



- All vehicle in the same sequence variable
- Vehicle  $[0..v-1]$  start from nodes  $[0..v-1]$
- Vehicle starts are always in the sequence in that order
- Vehicle implicitly come back to their start point
  - All invariants use this assumption
  - You can “tune” distance matrices if it is not the case
- Vehicle starts cannot be moved
  - But you can of course move all other nodes
- At most one occurrence of every value in the sequence

# Routing: an (optional) VRP class

```
class MySimpleRoutingWithUnroutedPoints (n:Int, v:Int,  
    symmetricDistance:Array[Array[Int]], m:Store, maxPivot:Int)  
extends VRP (n, v, m, maxPivot)  
with ClosestNeighbors{  
    traits that add standard features.  
    You can also add features in the class below  
override def getDistance (from:Int, to:Int) :Int=  
    symmetricDistance (from) (to)  
  
val penaltyForUnrouted = 10000  
  
val routed = Content(routes.createClone(50))  
val unrouted = Diff(CBLSSetConst(SortedSet(nodes:_*)),  
    routed)  
  
val totalDistance = ConstantRoutingDistance(routes, v, false,  
    symmetricDistance, true) (0)  
  
val obj = Objective(totalDistance +  
    (penaltyForUnrouted* (n - Size(routes))))  
  
val closestNeighboursForward = computeClosestNeighborsForward()  
  
def size = routes.value.size  
}
```

- ConstantRoutingDistance
  - **given** a distance matrix,
  - **maintains** the driven distance
  - **options**: isSymmetric? perVehicle? preCompute?
  - $O(1)$  update on classical neighbourhoods (with proper options)
- ForwardCumulativeIntegerDimensionOnVehicle
  - **given** a function  $(\text{node} \times \text{content} \times \text{node}') \Rightarrow \text{content}'$
  - **maintains** an array  $\text{node} \Rightarrow \text{content}$
- ForwardCumulativeConstraintOnVehicle
  - **given**
    - a function  $(\text{node} \times \text{content} \times \text{node}') \Rightarrow \text{content}'$
    - a max capacity
  - **maintains** a violation per vehicle (sum of overshoot per node)
- NodesOfVehicle
  - **given** route
  - **maintains** vehicle  $\Rightarrow$  set of nodes reached by vehicle

- NodeVehicleRestrictions
  - **given** set of couples (node,vehicle)
  - **maintains** number of such couples (n,v) such that vehicle v reaches node n
  - $O(1)$  update on classical neighbourhoods!
- RouteSuccessorAndPredecessors
  - **given** route
  - **maintains** two IntVar arrays: node  $\Rightarrow$  predecessor, node  $\Rightarrow$  successor
  - you can declare virtually anything from these arrays, using element invariant
- VehicleOfNodes
  - **given** route
  - **maintains** a SetVar array: vehicle  $\Rightarrow$  nodes reached by vehicle

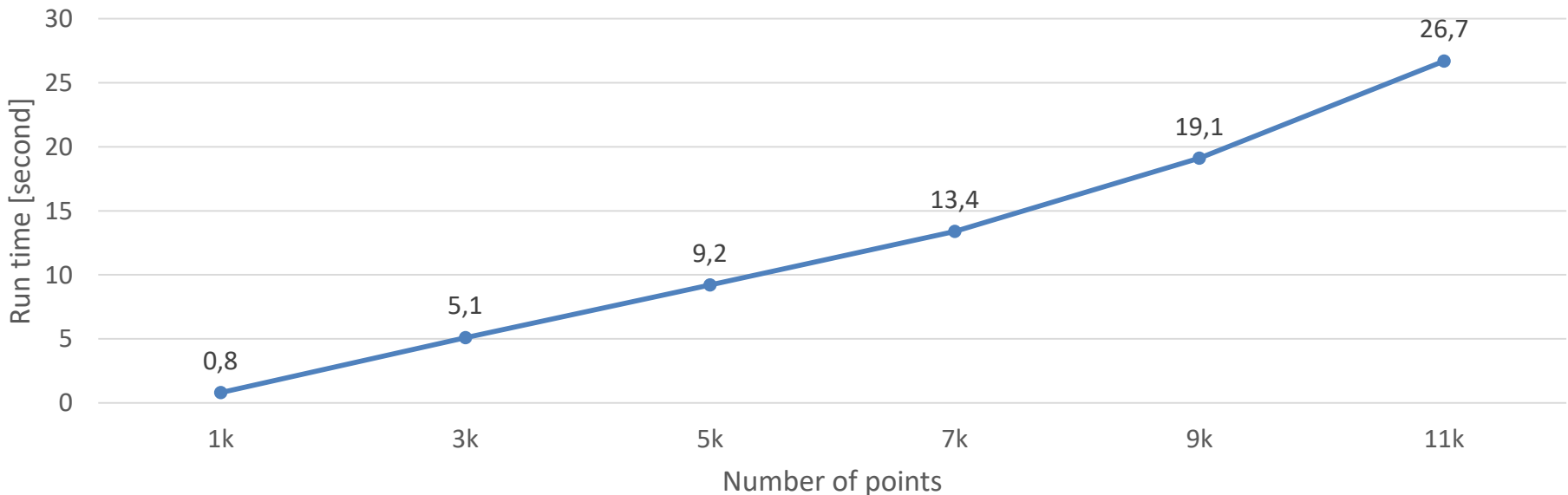
- InsertPoint
  - InsertPointRoutedFirst:  
for( $r \leftarrow$  routed)  
for( $u \leftarrow$  unrouted relevant wrt  $r$ )  
...  
– InsertPointUnroutedFirst  
for( $u \leftarrow$  unrouted)  
for( $r \leftarrow$  routed relevant wrt  $u$ )  
...
- OnePointMove
- RemovePoint
- SegmentExchange
- ThreeOpt
- TwoOpt
  - TwoOpt1
  - TwoOpt2



# Symmetric VRP ( $v = 100$ ) $N$ vs. run time

```
val search = (BestSlopeFirst(List(  
    insertPointUnroutedFirst(k=10),  
    insertPointRoutedFirst(k=10),  
    onePointMove(k=10),  
    twoOpt(k=10),  
    threeOpt(k=10)))  
    exhaust threeOpt(k=20))
```

Median over 10 runs with symmetric distance:  
square map with randomly placed points and straight line distance



- ```
val routingWithDepotSearch =
  insertPoint
  orElse (insertDepot andThen insertPoint)
  exhaustBack new Learning(onePointMove,
                           threeOpt, swapInsert, ...)
```

- Comet
  - First CBLS implem by pascal van Hentenryck
  - Not maintained since 2008?
- Kangaroo
  - One paper @CP2011, status unknown, not available
- LocalSolver
  - Commercial tool, with acad licence
  - Only Booleans and floats, very few invariants
  - Closed search procedure, closed source
- EasyLocal++
  - No support for modelling
- GoogleCP
  - Not a CBLS tool; a CP engine mimicking CBLS, less scalability
- InCell
- Lion

# *Conclusion: Features of Oscar.cblls*

- Modelling part: Rich modelling language
  - IntVar, SetVar, SeqVar
  - ~100 invariants: Logic, numeric, set, min-max, etc.
  - 17 constraints: LE, GE, AllDiff, Sequence, etc.
  - Constraints can attribute a violation degree to any variable
  - Model can include cycles
  - Fast model evaluation mechanism
    - Efficient single wave model update mechanism
    - Partial and lazy model updating, to quickly explore neighbourhoods
- Search part
  - Library of standard neighbourhoods
  - Combinators to define your global strategy in a concise way
  - Handy verbose and statistics feature, to help you tuning your search
- Business packages: Routing, scheduling
  - Model and neighbourhoods
- FlatZinc Front End [Bjö15]
- 49kLOC

# *Who is behind OscaR.cbls?*

- CETIC team
  - Renaud De Landtsheer
  - Yoann Guyot
  - Fabian Germeau
  - Gustavo Ospina
  - Christophe Ponsard
- Contributions from Uppsala
  - Jean-Noël Monette
  - Gustav Björdal
- Internships & MS Theses
  - UMONS: Gaël Thouvenin, Sébastien Drobisz, Florent Ghilain, Jannou Bohée
  - IPL: Fabian Germeau
  - HENALUX: Quentin Wautelet



- Repository / source code
  - <https://bitbucket.org/oscarlib/oscar/wiki/Home>
- Released code and documentation
  - <https://oscarlib.bitbucket.org/>
- Discussion group / mailing list
  - <https://groups.google.com/forum/?fromgroups#!forum/oscar-user>

- *Why don't you use C/C++ with templates, and compile with gcc -o3? You would be 2 times faster!*
- *I can develop a dedicated solver that will run 2 times faster because it will not need the overhead data structures of OscaR.cbls*

*... these remarks are correct, but ...*

- Algorithmic tunings deliver more than 2 to 4!
  - Ex: symmetry elimination on neighbourhoods
  - Ex: Restricting your neighbourhood to relevant search zones
  - Ex: Tuning when your neighbourhoods are actually used
  - We lately had a speedup 10 by tuning a search procedure
- **Our framework cuts down dev cost, so you have time to focus on these high-level tunings!**
- TODO: parallel propagation
  - Goal: same “basic speed” as dedicated implem
  - A core is cheaper than a single day of work for an engineer



In the real world, solving  
optimization problems  
using exact methods is a  
waste of resources