cetic

Your Connection to ICT Research

Local search with OscaR.cbls explained to my neighbor

OscaR v3.0 (Sept 2015)

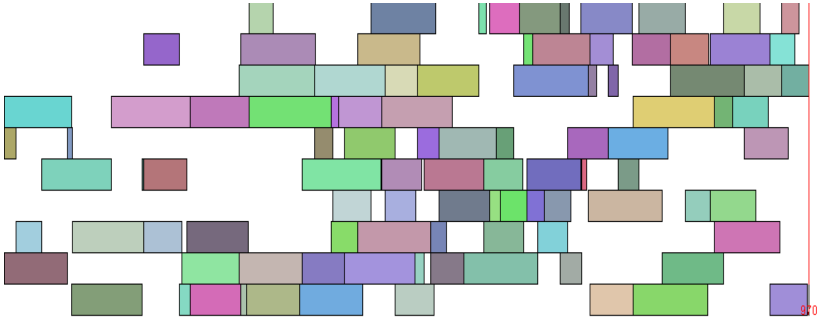
Renaud De Landtsheer, Yoann Guyot,
Christophe Ponsard, Gustavo Ospina

FEDER
UNION EUROPEENNE
Wallonie
LE FONDS EUROPEEN DE DEVELOPPEMENT REGIONAL
ET LA WALLONIE INVESTISSENT DANS VOTRE AVENIR.

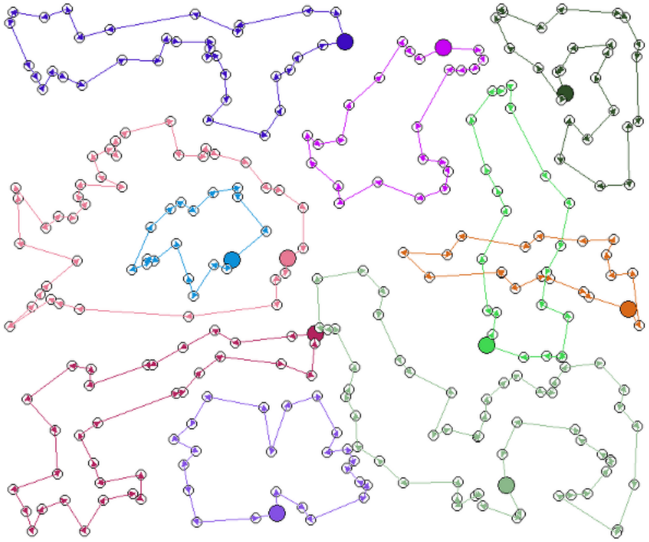
OscaR
OPERATIONAL RESEARCH IN SCALA

What are optimization problems?

- Scheduling
  - Tasks, precedence's
  - Shared resources
  - Deadlines



- Routing
  - Points, vehicles
  - Distance
  - Time windows
  - Minimize overall distance



- In general
  - Find values (possibly “structured values”)
  - Minimizing / optimizing objective (s)
  - Satisfying constraint (s)

OscaR
OPERATIONAL RESEARCH IN SCALA

- 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

Why open sourcing this code?

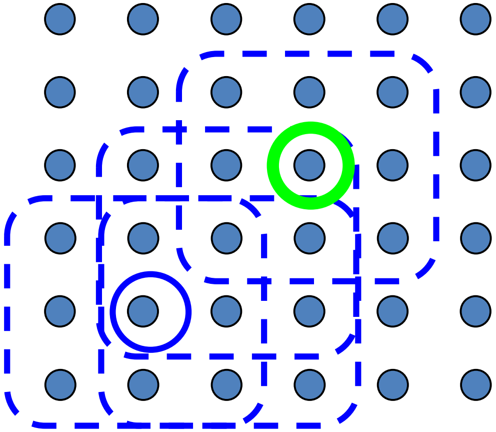
- Higher credibility
  - Since it is very intricate algorithms, customers can look at the quality of the work
  - Being able to look at the commit activity is also a plus for customers
- Easier transfer
- Mutualise extensions between customers
- Attract contributions
  - From external contributors
  - Find internships

Optimization by local search (LS)

- Perform a descend in the solution space; repeatedly move from one solution to a better one
- Next solution identified via neighborhood exploration

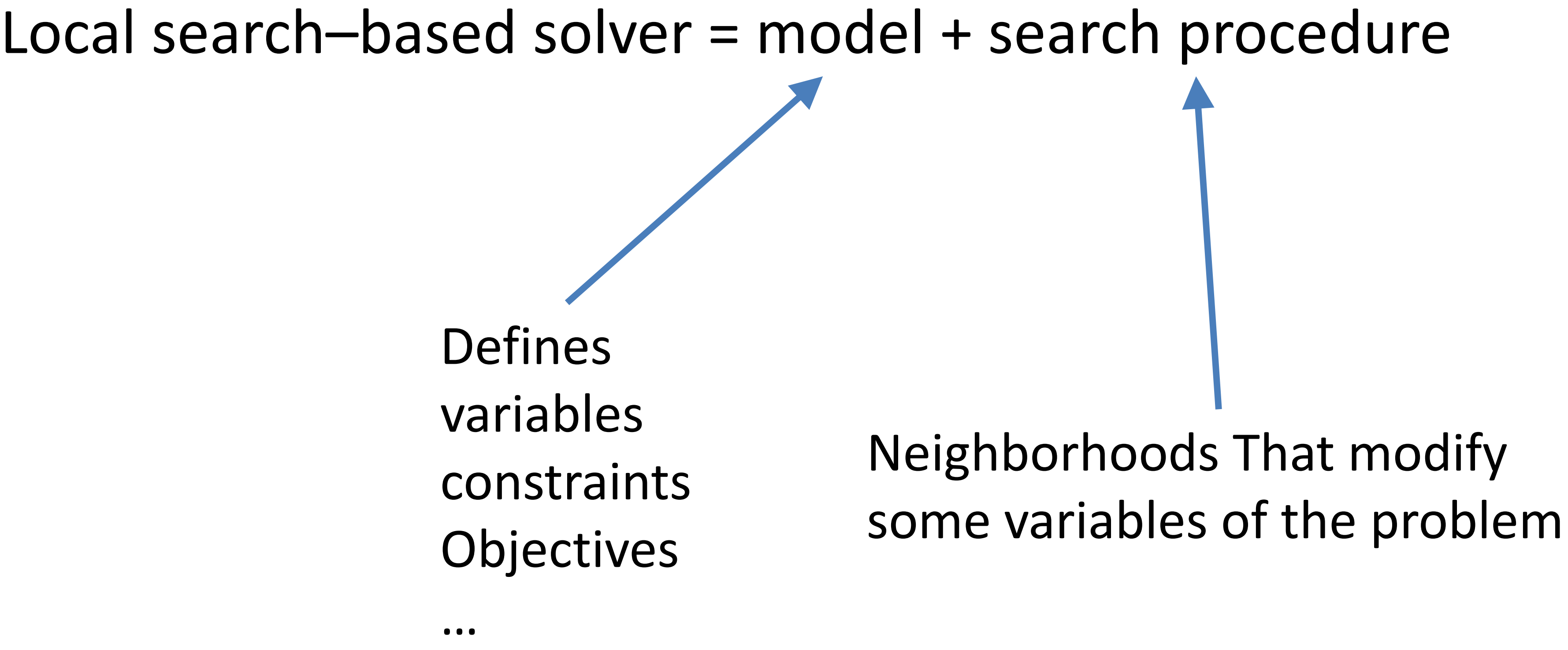
  TSP Example: moving a city to another position in the current circuit

  - Current state: a → b → **c** → d → e → a
  - Moving c gives three neighbors:
    - a → **c** → b → d → e → a
    - a → b → d → **c** → e → a
    - a → b → d → e → **c** → a
  - O(n²) neighbors in total



- Lots of black magic's, to escape from local minima

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

Constraint-based local search

- Goal: make it easy to write optimization engine based on the principle of local search

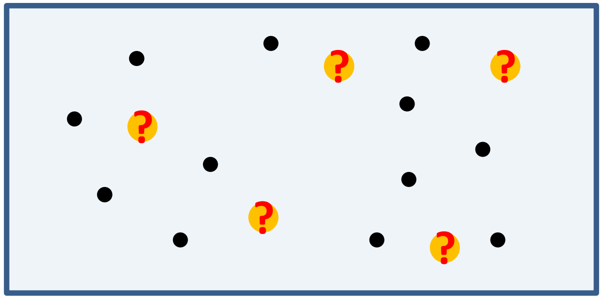
- Approach: Separate the modeling from the search in different component
  - Represent the problem as a large collection of mathematical formulas
  - Evaluate moves on this formula

- Technically:
  - Have an engine to evaluate the formula quickly
  - Based on the fact that very few decision variables are impacted by a move
  - So rely on incremental model updates

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 model of the WLP, written with OscaR.cbls

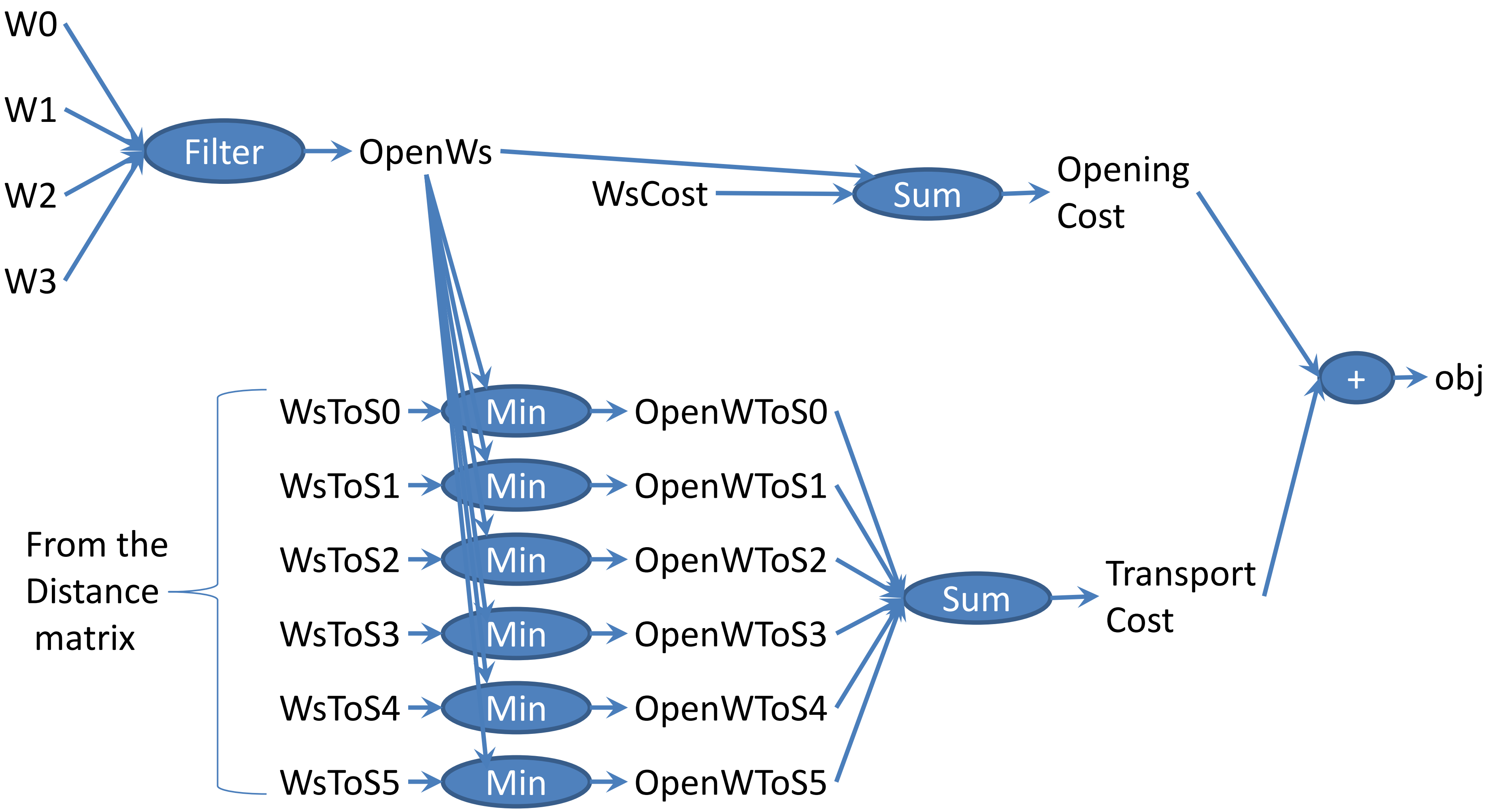
```
val m = new Store()

//An array of Boolean variables representing that the warehouse is open or not
val warehouseOpenArray = Array.tabulate(W)
           (w => CBLSIntVar(m, 0 to 1, 0, "warehouse_" + w + ""))

//The set of open warehouses
val openWarehouses = Filter(warehouseOpenArray)

//for each shop, the distance to the nearest open warehouse
val distanceToNearestOpenWarehouse = Array.tabulate(D)
           (d => Min(distanceCost(d), openWarehouses,
                     defaultCostForNoOpenWarehouse))

//summing up the distances and the warehouse opening costs
val obj = Objective(Sum(distanceToNearestOpenWarehouse)
                     + Sum(costForOpeningWarehouse, openWarehouses))
```

Modeling Support with OscaR

- Two types of variables
  - IntVar and SetVar
- Invariant library (they are functions, actually)
  - Logic, such as:
    - Acces on array of IntVar, SetVar
    - Sort
    - Filter, Cluster (indexes of element whose value is…)
  - MinMax, such as:
    - Min, Max
    - ArgMin, ArgMax
  - Numeric, such as:
    - Sum, Prod, Minus, Div, Abs
  - Set, such as:
    - Inter, Union, Diff, Cardinality

  Summing up to roughly 80 invariants in the library

Propagation graph for the WLP(4,6)



What we can do with a model

- Model has some input variables
  - *warehouseOpenArray*
- We can modify the value of these input variables
- The model is updated through a procedure called *propagation*.
  - *Propagation* is triggered when the value of an output variable is queried, so you always have coherent answers on the model
  - Propagation is very fast, thanks to adequate algorithms and data structures

Let’s play with the model in console

```
>println(openWarehouses)
openWarehouses:={}
>println(obj)
IntVarObjective(Sum2:=1500000)

> warehouseOpenArray(0) := 1
> println(openWarehouses)
openWarehouses:={0}
> println(obj)
IntVarObjective(Sum2:=7849)

> warehouseOpenArray(5) := 1
> println(openWarehouses)
openWarehouses:={0,5}
> println(obj)
IntVarObjective(Sum2:=6024)
```

How the model will help optimizing?

- Model is fit for local search, based on neighborhood exploration
  - Eg: switching one warehouse (open or close it)
- Does a move improve on the objective?
  - Perform the move Eg: switch the warehouse
  - Query the objective value
  - RollBack
  - Methods available in the Objective class perform this

```
//summing up the distances and the warehouse opening costs
val obj = Objective(Sum(distanceToNearestOpenWarehouse)
                           + Sum(costForOpeningWarehouse, openWarehouses))
```

- Neighborhood exploration is fast:
  - Propagation is incremental
  - Propagation is not performed after the rollback
  - Partial propagation: only involves what is needed to evaluate obj

Some Relevant Neighborhoods

- Switching a single warehouse
  - either closing an open warehouse,
    or opening a closed one
  - Size: O(#W)
  - Connected: all solutions are reachable

- Swapping two warehouses
  - close an open warehouse and open a closed one
  - Size: O(#W^2)
  - Not Connected

- Randomization at local minimum
  - Randomize a fraction of the warehouses

How can we assemble these bricks?

Searching the WLP: sample strategy

- Do all switch moves
- Then all the swap moves
- Iterate until no more moves

**We want to try also
the random neighborhood choice**

- Perform some randomization when minimum reached

- Stop criterion: only two randomizations authorized

- Save the best solution at all time,
  and restore it when search is finished

Note: the idea of combining neighborhood is not new (eg. [Glo84], [Ml97], and many papers at MIC)

A WLP solver written with neighborhood combinators

```
val m = new Store()
val warehouseOpenArray = Array.tabulate(W)
              (w => CBLSIntVar(m, 0 to 1, 0, "warehouse_" + w + ""))
val openWarehouses = Filter(warehouseOpenArray)

val distanceToNearestOpenWarehouse = Array.tabulate(D)
              (d => Min(distanceCost(d), openWarehouses,
                            defaultCostForNoOpenWarehouse))

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

m.close()

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

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

The console output

```
WarehouseLocation(W:15, D:150)
SwitchWarehouse(warehouse_0:=0 set to 1; objAfter:7052)            - #
SwitchWarehouse(warehouse_1:=0 set to 1; objAfter:5346)            - #
SwitchWarehouse(warehouse_2:=0 set to 1; objAfter:4961)            - #
SwitchWarehouse(warehouse_3:=0 set to 1; objAfter:4176)            - #
SwitchWarehouse(warehouse_4:=0 set to 1; objAfter:3862)            - #
SwitchWarehouse(warehouse_9:=0 set to 1; objAfter:3750)            - #
SwitchWarehouse(warehouse_12:=0 set to 1; objAfter:3620)           - #
SwitchWarehouse(warehouse_0:=1 set to 0; objAfter:3609)            - #
SwapWarehouses(warehouse_0:=0 and warehouse_4:=1; objAfter:3572)   - #
SwapWarehouses(warehouse_1:=1 and warehouse_6:=0; objAfter:3552)   - #
SwapWarehouses(warehouse_0:=1 and warehouse_1:=0; objAfter:3532)   - #
SwitchWarehouse(warehouse_7:=0 set to 1; objAfter:3528)            - #
RandomizeNeighborhood(warehouse_12:=1 set to 0, warehouse_
SwitchWarehouse(warehouse_7:=0 set to 1; objAfter:3656)            -
SwapWarehouses(warehouse_12:=0 and warehouse_13:=1; objAfter:3528) - °
RandomizeNeighborhood(warehouse_14:=0 set to 1, warehouse_
SwitchWarehouse(warehouse_7:=0 set to 1; objAfter:3907)            -
SwitchWarehouse(warehouse_12:=1 set to 0; objAfter:3882)           -
SwitchWarehouse(warehouse_13:=1 set to 0; objAfter:3862)           -
SwitchWarehouse(warehouse_14:=1 set to 0; objAfter:3658)           -
SwitchWarehouse(warehouse_12:=0 set to 1; objAfter:3528)           - °
MaxMoves: reached 2 moves
openWarehouses:={1,2,3,6,7,9,12}
```

Three shades of Warehouse Location

- The presented one:

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

- Chosing the neighborhood randomly

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

- Learning about neighborhood efficiency

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

Conclusion: Features of Oscar.cbls

- Modeling part: Rich modeling language
  - IntVar, SetVar
  - 80 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 neighborhoods
- Search part
  - Library of standard neighborhoods
  - 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 neighborhoods
- FlatZinc Front End [Bjö15]
- 27kLOC

To some extend, brain cycle is more valuable than CPU cycle (1/2)

- *Why don't you use C with templates, and compile with gcc –o3? You would be 2 times faster!*

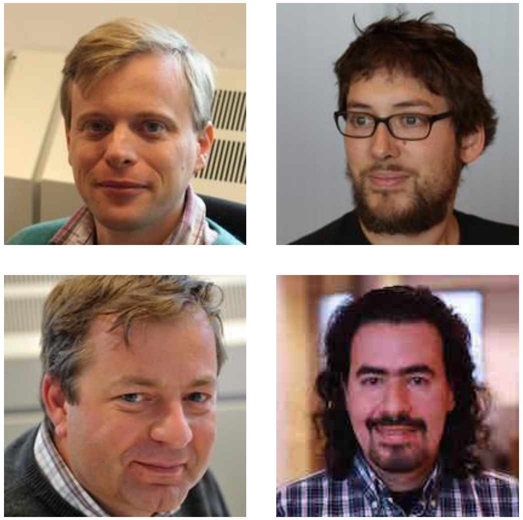
- *Why should I use your stuff? I can program a dedicated solver that will run 2 times faster because it will not need the data structures you need in OscaR*

To some extend, brain cycle is more valuable than CPU cycle (2/2)

- That is true, but
  - – Algorithmic tunings deliver more than 2 to 4!
    - Ex: We lately had a speedup 10 by tuning a search procedure
    - Using symmetry elimination on neighborhoods
    - Restricting your neighborhood to relevant search zones
  - – Our approach cuts down dev cost, so you have time to focus on these high-level tunings.
    - Since budget is always limited
  - – Next step: parallel propagation
    - So you will have the same “basic speed” than a dedicated implem, by using more cores
    - A core is cheaper than a single day of work for an engineer

Who is behind OscaR.cbls?

- CETIC team
  - Renaud De Landtsheer
  - Yoann Guyot
  - Christophe Ponsard
  - Gustavo Ospina
- Contributions from Uppsala
  - Jean-Noël Monette
    - Gustav Björdal



Where is OscaR?

- 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

Thank you
Merci

cetic

Your Connection to ICT Research

Aéropôle de Charleroi-Gosselies
Rue des Frères Wright, 29/3
B-6041 Gosselies
info@cetic.be

www.cetic.be