Your Connection to ICT Research

Local Search with OscaR.cbls explained to my neighbour

OscaR v4.0 – Spring2018

Renaud De Landtsheer, Thomas Fayolle,
Fabian Germeau, Gustavo Ospina,
Christophe Ponsard

OPERATIONAL RESEARCH IN SCALA

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

OscaR
OPERATIONAL RESEARCH IN SCALA

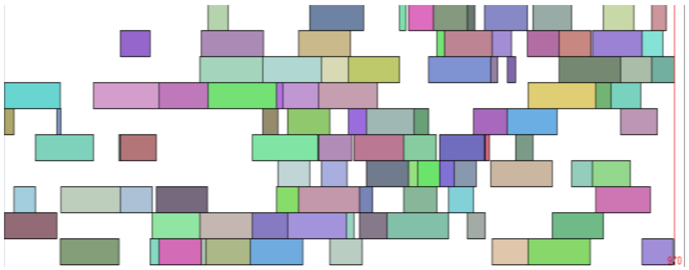
- Oscar
  - Open source framework for combinatorial optimization
  - CP, CBLS
  - Started in 2011

- Open source LGPL license
  - https://bitbucket.org/oscarlib/oscar
  - Implemented in Scala

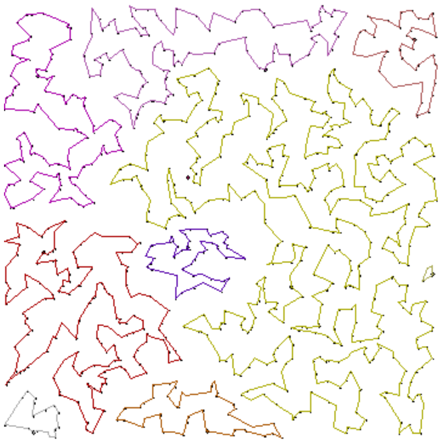
- Consortium
  - CETIC, UCL, N-Side Belgium
  - Contributions from Uppsala Sweden

Combinatorial optimization problems

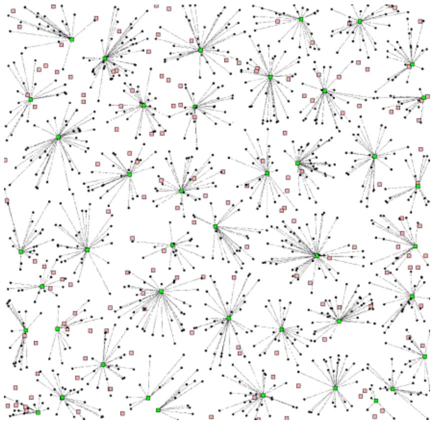
- Ex: Scheduling
  - Tasks, precedence's
  - Shared resources
  - Deadlines
  - Minimize time span



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

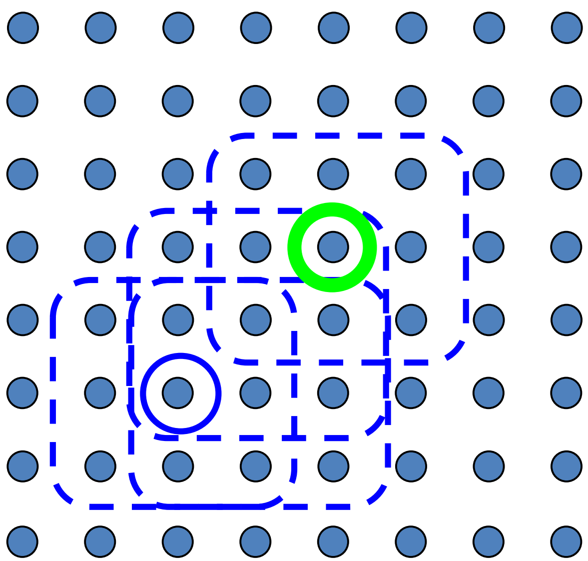


- Ex: Warehouse location
  - Shops to supply
  - Where to build warehouses?
  - Minimize operation + construction costs



Local search in one slide

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



Point in the search space

TSP : random tour?

Pick an initial solution

Repeat

Explore neighbourhood

Move to best neighbour

Until no better neighbour

TSP : moving a city
to another position in the tour

*Current state: a → b → **c** → d → e → a*

*Moving city **c** yields three neighbours:*

*a → **c** → b → d → e → a*

*a → b → d → **c** → e → a*

*a → b → d → e → **c** → a*

$O(n^2)$ neighbours when considering all cities

Some black magic required
to escape from local minima

Local search is black magic

- Non exhaustive
- Needs tuning, benchmarking
- But it works!

Local search practitioners, like you, are magicians

I am a wand maker,

and I will show you

why OscaR.cbls is a good wand

OscaR

OPERATIONAL RESEARCH IN SCALA

Content

- Introduction
  - Goal of OscaR.cbls
  - NQueens
- Warehouse Location Problem
  - Problem statement
  - Solution
  - About modelling
  - About searching
- Under the hood of OscaR models
  - Propagation
  - Architecture
- Routing with OscaR.cbls
  - Routing convention
  - Model support
  - Search support
- Cross product of neighbourhoods
- More examples
  - Flow shop scheduling
  - Car sequencing
- Conclusion

Your Connection to ICT Research

Why are we developing OscaR.cbls?

- OscaR.cbls is developed primarily at CETIC
- CETIC is a research centre in Belgium
  - Focus on technology transfer in IT
  - No fundamental research
  - As such, OscaR.cbls is our research topic:
    - How to make it faster-better-cheaper for users?
      - Cheaper means « faster to develop a solution » since your time is money
    - How to make it faster-better-cheaper for researchers?
    - …
- To fight against shelf research:

  Make the research, write a report, put it in a shelf, do something else

Your Connection to ICT Research

My goal: cut down the cost of having a local search solution

- Why?
  - Expert, like you, are expensive
  - Non-expert can be empowered with smart algorithms in the hands
  - Applications tend to change their requirements
    - Agile approaches
    - Evolving market needs
  - Human brain is limited (at least mine)
    - With OscaR.cbls, you can focus on the black magic part where a lot of gain can be achieved

- How?
  - Declarative approaches
    - CBLS Modelling language for defining your problem
    - Declarative language for defining search procedures
  - Cost of license
    - This is LGPL (free, non-contaminating)
  - Integration
    - This is Scala, compiles to Java bytecode

Who might be interested?

- Developers targeting new applications
  - Obviously
- Researchers
  - Develop their innovative algorithms within OscaR.cbls (constraints, meta-heuristic, neighbourhood, etc.)
  - Don't waste their time on everting else
  - Make their research result be used, add to OscaR.cbls
- Benchmark makers
  - Comparing different algorithms if often a tricky job:
    - not the same programming language,
    - not the same base algorithms,
    - not the same implementation quality, etc.
  - OscaR.cbls can be used as a reference platform for sound comparison of algorithms

The basic equation of local search

Local search - based solver = model + search procedure

variables
constraints
objectives
…

neighbourhoods
metaheuristics
…

Basic Nqueens, the OscaR way

```
val nQueens = 20000 // Number of queens
val queensRange= 0 to nQueens -1
val init = Random.shuffle(0 until nQueens)

// Variables
val queens = Array.tabulate(nQueens)(q =>
                CBLSIntVar(0 to nQueens -1,init(q),"queen" + q))

// Constraints
 val c = new ConstraintSystem(m)
 c.add(allDifferent(queensRange.map(q => queens(q) + q)))
 c.add(allDifferent(queensRange.map(q => q - queens(q))))

close()

// Swapping two queens to decrease overall violation
swapNeighborhood(queens)
      .doAllMoves(_ >= nQueens || c.violation.value == 0, c)

println(queens.mkString(","))
```

Model (variables and constraints)

Search procedure (swapping and doAllMoves)

Advanced Nqueens, the OscaR way

```
val nQueens = 20000 // Number of queens
val queensRange= 0 to nQueens -1
val init = Random.shuffle(0 until nQueens)

// Variables
val queens = Array.tabulate(nQueens)(q =>
               CBLSIntVar(0 to nQueens -1,init(q),"queen" + q))

// Constraints
 val c = new ConstraintSystem(m)
 c.add(allDifferent(queensRange.map(q => queens(q) + q)))
 c.add(allDifferent(queensRange.map(q => q - queens(q))))

 val mostViolatedQueens = argMax(c.violations(queens))
close()

// Swapping a queen with one of the most violated ones
swapNeighborhood(queens, searchZone = mostViolatedQueens,
                        symmetryCanBeBrokenOnIndices = false)
    .doAllMoves(_ >= nQueens || c.violation.value == 0, c)
```

Model

Search procedure

A simple Nqueens with old fashion search procedure

```
val init = Random.shuffle(0 until nQueens)
val queens = Array.tabulate(N)(q => CBLSIntVar(init(q), range, "queen" + q))

c.add(allDifferent(Array.tabulate(N)(q => queens(q) + q)))
c.add(allDifferent(Array.tabulate(N)(q => q - queens(q))))

close()

var it = 0
while(c.violation.value > 0){
 selectMin(range,range)(
  (p,q) => c.violation.swapVal(queens(p),queens(q)),
  (p,q) => p < q)
 match{
  case (q1,q2) =>
   queens(q1) :=: queens(q2)
  }
 it += 1
}
```

Model

Selecting the pair of queens with the best swap

Swapping the values

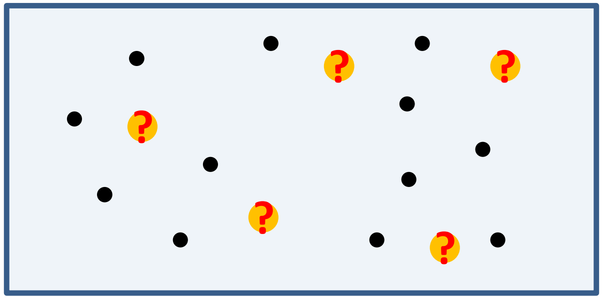
Search procedure

If you really want to do that…

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:

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



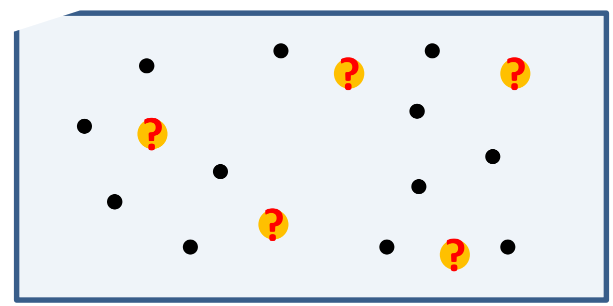
- **Notice**
  - A store is assigned to its nearest open warehouse

The uncapacitated warehouse location problem

- **Given**
  - S: set of stores that must be stocked by the war… es
  - W: set of potential warehouses
    - Each warehouse has a fixed cost f…
    - transportation cost from …
- **Find**
  - O: su…

  …ansportation

I will show you a working solution on a single slide



- **N…ce**
  - A store is assigned to its nearest open warehouse

A WLP solver on a single slide

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

val openWarehouses = Filter(warehouseOpenArray)

val distanceToNearestOpenWarehouse = stores.map((store:Int) =>
        min(distanceCost(store), openWarehouses,
                         defaultCostForNoOpenWarehouse)).toArray

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

m.close()

val neighborhood = (AssignNeighborhood(warehouseOpenArray, "SwitchWarehouse")
         exhaustBack SwapsNeighborhood(warehouseOpenArray, "SwapWarehouses")
         onExhaustRestartAfter(RandomizeNeighborhood(warehouseOpenArray, W/5),
                           maxConsecutiveRestartWithoutImprovement=2, obj)

neighborhood.doAllMoves(obj)
```

The console output

The search can display info roughout the search:

- 0: no verbosities
- 1: every 10th of a second, summarise all performed moves, by neighbourhoods
- 2: print every move
- 3: print every search
- 4: print every explored neighbour

neighborhood.verbose = 2

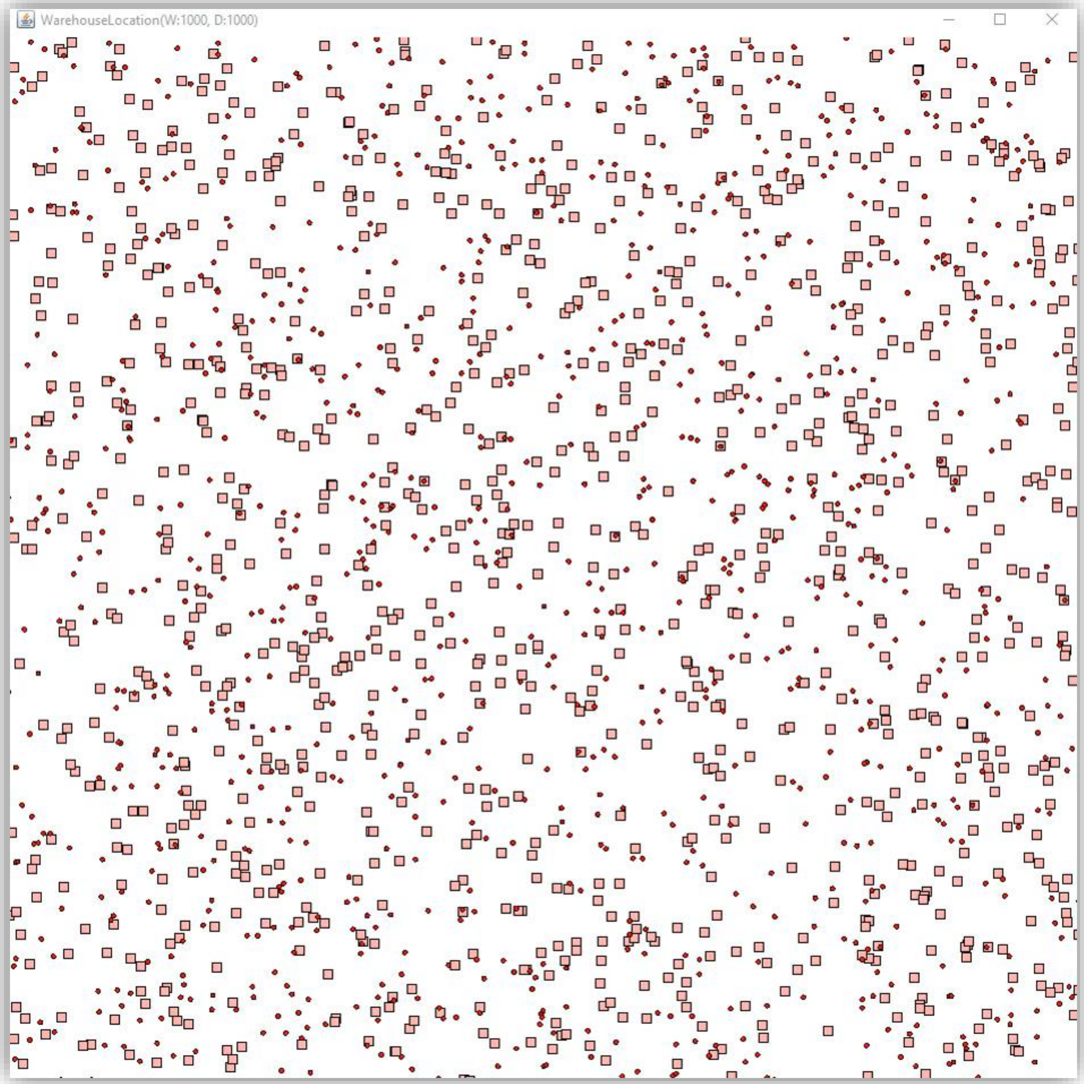
```
WarehouseLocation(W:15, D:150)
SwitchWarehouse(warehouse_0:=0 set to 1; objAfter:7052)             - #
SwitchWarehouse(warehouse_1:=0 set to 1; objAfter:5346)             - #
SwitchWarehouse(warehouse_2:=0 set to 1; objAfter:4961)             - #
SwitchWarehouse(warehouse_3:=0 set to 1; objAfter:4176)             - #
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}
```

- Means:
obj decreases
after this move

Means:
we found a
solution with a
new best
objective

° Means:
we found an
solution with obj
equal to the
best so far

WareHouseLocationVisu



W = 1000
S = 1000

Complex search strategy:

- Switch
- Swap with kNearest
- Swap
- Restarts
- Mu(switch)

Modelling Support with OscaR

- 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

Search support with OscaR

- 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, relax

  lots 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 build your own search procedure based on linear selectors

Best or first improving neighbour?

- Neighbourhoods can search for
  - best neighbour (it must be accepted by the acceptation function)
  - First improving neighbour
- This setting is decided at the level of the basic search neighbourhoods
  - AssignNeighbourhood
  - 2-opt
  - …
- A basic search neighbourhood is a bunch of nested loops, and most of our neighborhoods input a parameter for deciding best/first for each level of their loop
  - Common pattern: `Select…Behavior`
  - Expecting a types: `LoopBehavior`
  - There are two types (with additional parameters): `First()` `Best()`
- Check Scaladoc of your neighbourhoods

Three shades of Warehouse Location

- The presented one, with best Switch:

search = (*AssignNeighborhood*(*warehouseOpenArray*, **"SwitchWarehouse "**,
selectIndiceBehavior = Best())
exhaustBack *SwapsNeighborhood*(*warehouseOpenArray*, **"SwapWarehouses"**)
onExhaustRestartAfter(*RandomizeNeighborhood*(*warehouseOpenArray*, *W*/5),
maxConsecutiveRestartWithoutImprovement=2, *obj*)

- Tabu search (requires model extension)

search = (*AssignNeighborhood*(*warehouseOpenArray*, **"SwitchWarehouse "**
searchZone = *nonTabuWarehouses* , selectIndiceBehavior = Best())
acceptAll
afterMoveOnMove((a:AssignMove) => tabu(a.id) = it + *tabulength*; it += 1)
maxMoves *someIterationBound* **withoutImprovementOver** *obj*)
saveBestAndRestoreOnExhaust *obj*)

- Using the most efficient neighbourhood anytime

search = (**BestSlopeFirst**(*AssignNeighborhood*(*warehouseOpenArray*, **"SwitchWarehouse"**)
SwapsNeighborhood(*warehouseOpenArray*, **"SwapWarehouses"**))
onExhaustRestartAfter(*RandomizeNeighborhood*(*warehouseOpenArray*, *W*/5),
maxConsecutiveRestartWithoutImprovement=*2*, *obj*)

Cetic — Your Connection to ICT Research

Profiling your search

- You need to know how each neighbourhood performed
  - What neighbourhood takes a lot of time?
  - What neighbourhood does never find a move?
  - Etc.
- How to collect profiling statistic
  - Use the Profile combinator where you want to measure

```
val neighborhood =
        (BestSlopeFirst(Profile(AssignNeighborhood(warehouseOpenArray, "Switch"))
                        Profile(SwapsNeighborhood(warehouseOpenArray, "Swap")))
          onExhaustRestartAfter(RandomizeNeighborhood(warehouseOpenArray, W/5),
                        maxConsecutiveRestartWithoutImprovement=2, obj)
```

  - Run the search as usual

```
neighborhood.doAllMoves(obj)
```

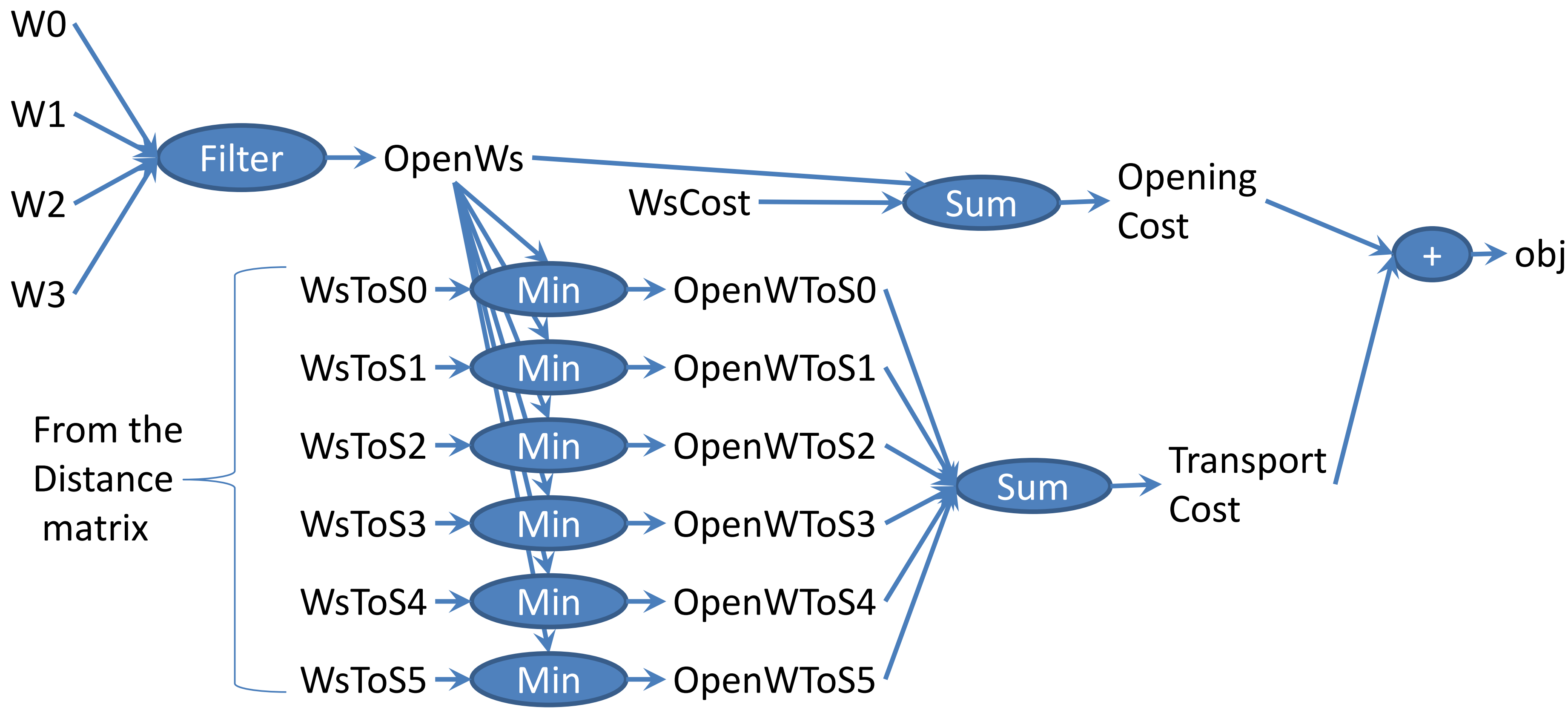
  - Print the profiling statistics

```
println(neighborhood.profilingStatistics)
```

  - You get a ton of info (not all on the slide) Time measures are in ms

Neighborhood	calls	found	sumGain	sumTime	avgGain	avgTime	slope
Switch	631	625	74905	1006	118	1	74458
Swap	17	12	79	21467	4	1262	3

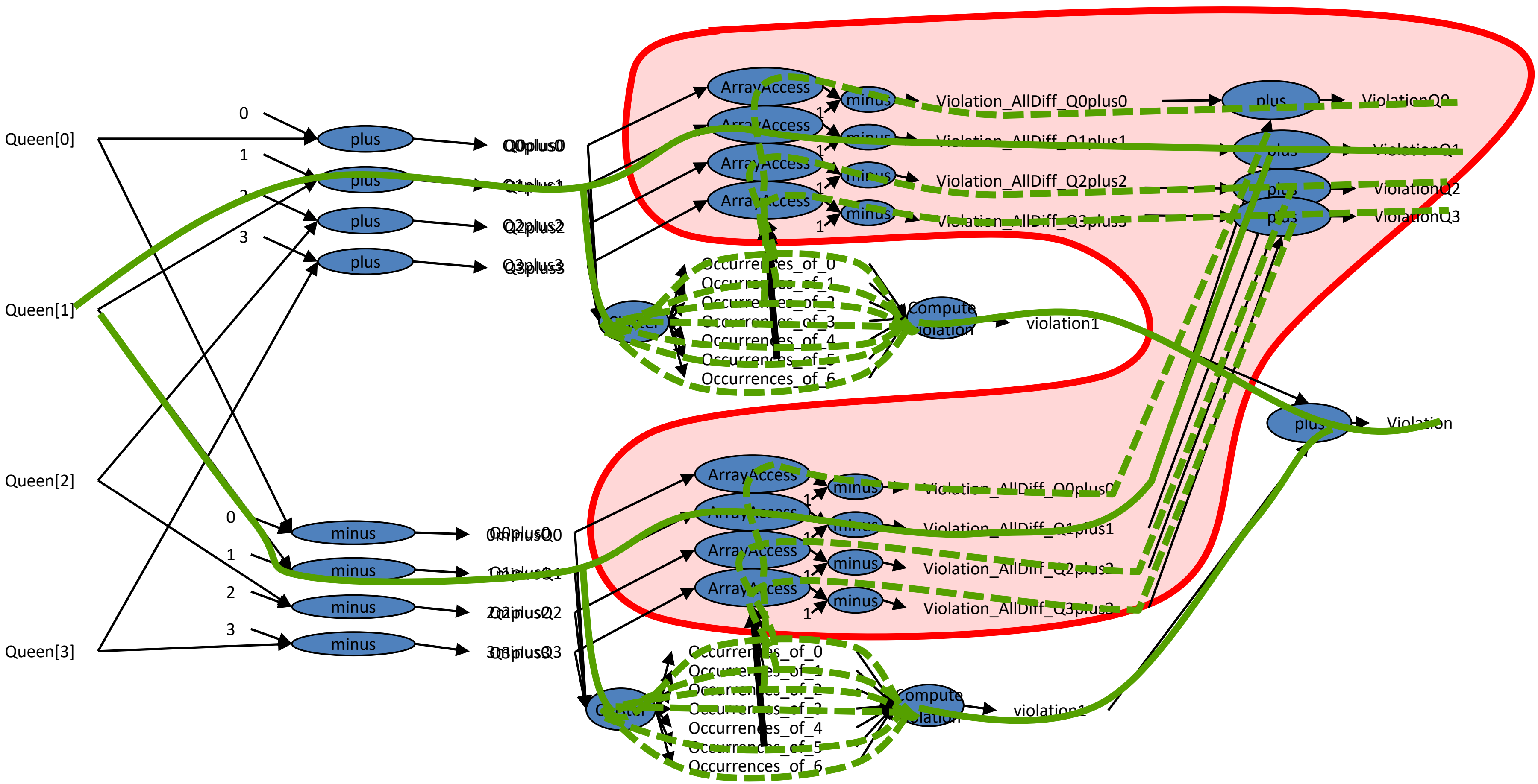
A quick look under the hood: Propagation graph for the WLP(4,6)



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

A quick look under the hood: Selective + partial propagation

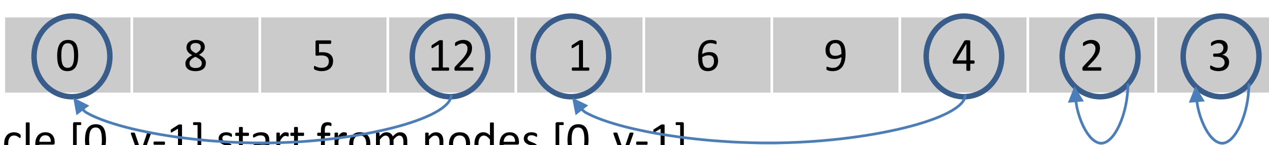


Routing with OscaR.cbls

- Modelling
  - Sequence variable (very efficient to perform classical routing moves)
  - Library of global routing constraints
    - Route length(sequence, distance matrix)
    - Node vehicle restrictions
    - …
- Searching
  - Insert point
  - One point move
  - 2-opt
  - …

- Routing convention: all vehicles in the same sequence variable

V=4



  - 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
  - Vehicle starts cannot be moved by neighbourhoods
  - At most one occurrence of every value in the sequence

A VRP class around the sequence variable

```
val myVRP = new VRP(model,n,v)

val routeLength = constantRoutingDistance(
                      myVRP.routes,n,v,
                      symmetricDistanceMatrix)(0)

val penaltyForUnrouted  = 10000

val obj = Objective(routeLength
                    + penaltyForUnrouted*n
                    - penaltyForUnrouted*length(myVRP.routes))

model.close()
```

Main routing invariants (1/2)

- ConstantRoutingDistance
  - **given** a distance matrix,
  - **maintains** the driven distance
  - **options**: isSymmetric? perVehicle? preCompute?
  - O(log(v)) update on classical neighbourhoods (with proper options)
- ForwardCumulativeIntegerDimensionOnVehicle
  - **given** a function (node × content × node') =>content'
  - **maintains** an array node=>content
- ForwardCumulativeConstraintOnVehicle
  - **given**
    - a function (node × content × node') =>content'
    - a max capacity
  - **maintains** a violation per vehicle (sum of overshoot per node)
- NodesOfVehicle
  - **given** route
  - **maintains** vehicle => set of nodes reached by vehicle

Main routing invariants (2/2)

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

Routing neighbourhoods

- InsertPoint
  - InsertPointRoutedFirst:

```
for(r <- routed)
  for(u <- unrouted relevant wrt r)
     …
```

  - InsertPointUnroutedFirst

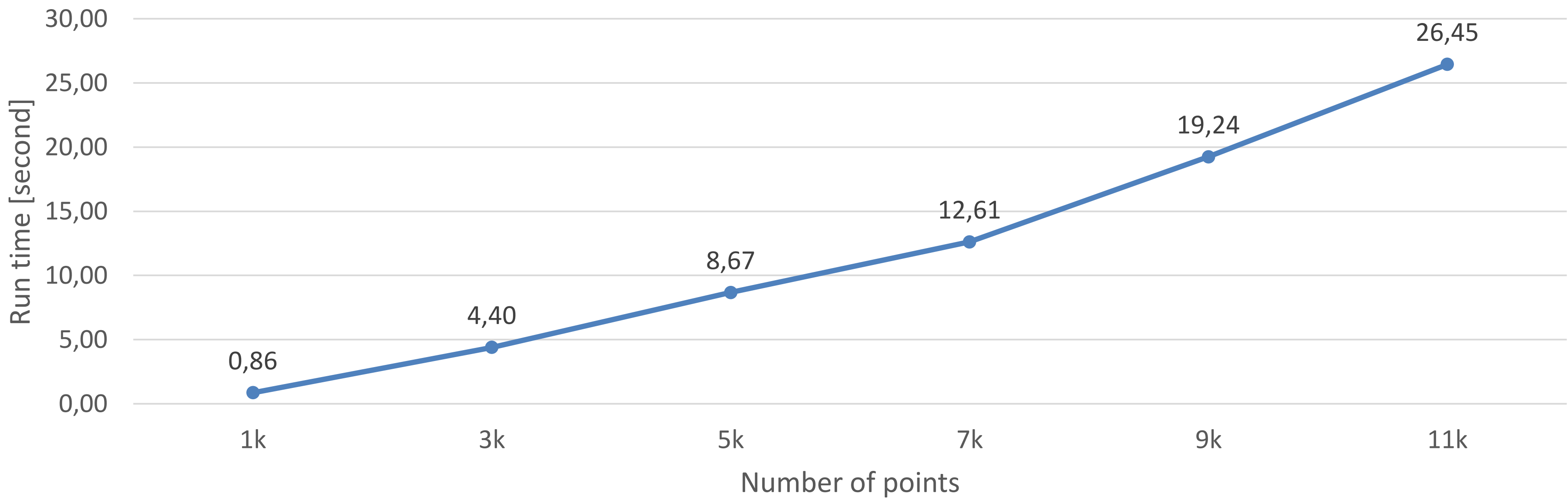
```
for(u <- unrouted)
    for(r <- routed relevant wrt u)
      …
```

- OnePointMove
- RemovePoint
- SegmentExchange
- ThreeOpt
- TwoOpt
  - TwoOpt1
  - TwoOpt2

Symetric 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



Local search … smart neighbourhoods

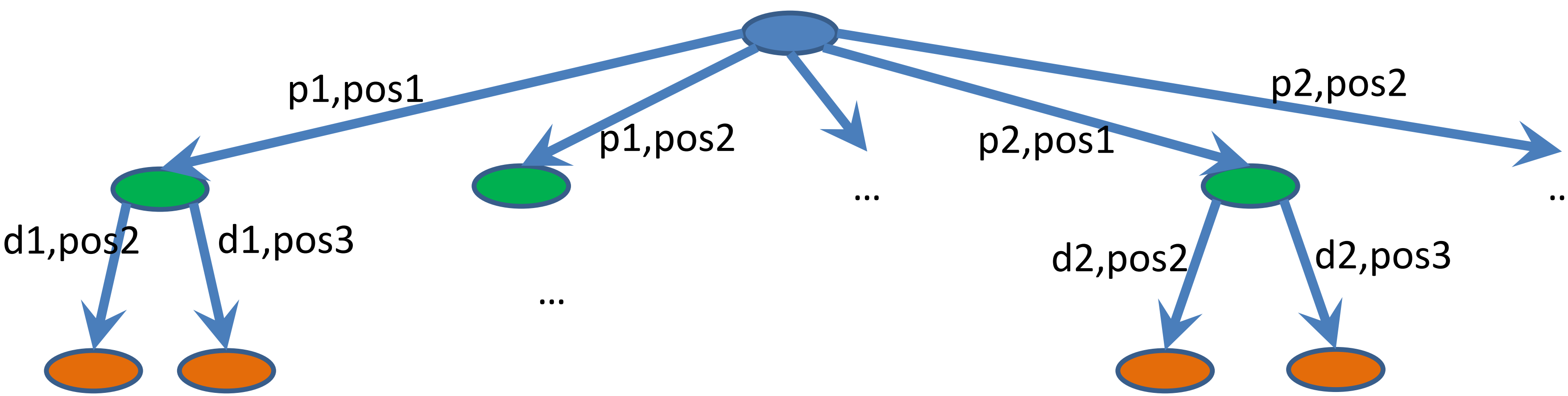
- Additional constraint call for specific neighbourhoods
  - Pick-up & delivery (PDP)
    - Two point insert
    - Two point move
      - Only try moving deliveries after their pick-up, and on the same vehicle
  - …
- Complex neighbourhoods
  - Lin-Kernighan: A succession of two-opts

An important combinator: the cross-product of neighbourhoods

```
val insertPickupAndRelatedDelivery = (
      insertPointUnroutedFirst(nonRoutedPickupPoints, … )
      dynAndThen (insertMove: InsertPointMove) =>
         insertPointUnroutedFirst(
               pickUpToDelivery(insertMove.insertedPoint), …))
```

What is explored: a search tree with two non-root levels

- Objective function is evaluated only at the actual neighbours that form the bottom of the tree
- dynAndThen returns a *compositeMove*, in this case this move includes two instances of *insertpointMove*



Your Connection to ICT Research

Pruning the exploration tree

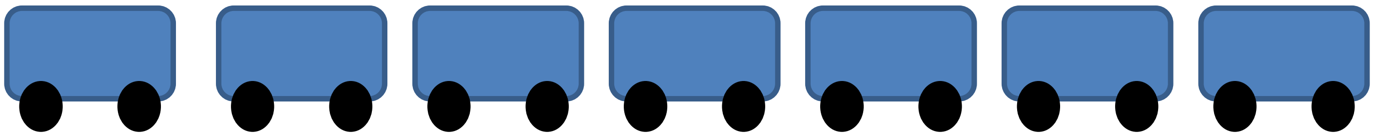
- Once the pick-up node is inserted, some constraints might already be violated, and search tree can be pruned
  - *Deadline constraints*
- Not all constraints can be checked:
  - *"pick-up before delivery"* will be violated anyway
  - *"vehicle content < max capacity" will be inconclusive*

```
val pickupAndDeliveryInsertTW = (
  insertPointUnroutedFirst(nonRoutedPickupPoints, …)
  dynAndThen(
    (insertMove: InsertPointMove) =>
      if(timingConstraints.violation.value == 0) {
            insertPointUnroutedFirst(
               pickUpToDelivery(insertMove.insertedPoint), …))
      }else NoMoveNeighborhood
    ))
```

(triangular inequality holds)

Car Sequencing problem

- Sequencing of cars in assembly lines:



  - Maximum *k* cars of any *n* consecutive cars in the sequence can have option *o in {abs,airCo,esp}*
  - For all option *o*, each having specific *(k, n)*
- They can only build the ordered cars

- Problem statement:
  - Given
    - Order book (set of cars to build, specified by their equipment)
  - Find
    - Ordering for these cars
  - Such that
    - All sequence constraints are enforced

A Car Sequencer: Model

```
val m = new Store()
val c = new ConstraintSystem(m)

//initializing the sequence with a random permutation of the ordered cars
val carSequence = Array.tabulate(nbCars)(CBLSIntVar(…….,carTypes,"carClassAtPosition" + _))

//airCo: class(0, 2, 4) max 2 out of 3
c.post(sequence(carSequence,3,2, makeBoolArray(0,2,4)))
c.post(sequence(carSequence,5,3, makeBoolArray(0,1,4,5)))
c.post(sequence(carSequence,5,3, makeBoolArray(0,1,2)))
c.post(sequence(carSequence,3,2, makeBoolArray(3,4,5)))

val carViolation = c.violations(carSequence)
val violatedCars = filter(carViolation)
val mostViolatedCars = argMax(carViolation)

c.close
val obj:Objective = c.violation
s.close()
```

A Car Sequencer: a Search Strategy Two Neighbourhoods And Three Restarts

```
val search =
 (swapsNeighborhood(carSequence,"mostViolatedSwap",
                    searchZone2 = mostViolatedCars,
                    symmetryCanBeBrokenOnIndices = false)

  exhaust wideningFlipNeighborhood(carSequence,"flipSubSequence")

  onExhaustRestartAfter(
        shuffleNeighborhood(carSequence, mostViolatedCars,
                            name = "shuffleMostViolatedCars")
        guard(() => mostViolatedCars.value.size > 2), 2, obj)

  onExhaustRestartAfter(
         shuffleNeighborhood(carSequence, violatedCars,
                             name = "shuffleSomeViolatedCars",
             numberOfShuffledPositions = () => 5 max (violatedCars.value.size/2)), 2, obj)

  orElse (shuffleNeighborhood(carSequence, name = "shuffleAllCars") maxMoves 4)

  saveBestAndRestoreOnExhaust obj)
```

A car sequencer problem, and a solution

```
val orderedCarsByType = (0 -> 110, 1 -> 60, 2 -> 110 ,
                         3 -> 120, 4 -> 40, 5 -> 30)
```

totalNumberOfCars:470
Proposed car sequence:
0,2,1,3,3,0,1,2,4,3,1,2,2,5,3,0,2,1,4,3,2,1,2,4,3,0,1,2,3,4,0,1,2,3,4,1,0,2,3
,4,2,1,2,4,5,2,1,2,5,3,1,2,0,3,4,1,2,0,3,3,1,0,2,3,4,0,1,2,3,4,1,0,2,3,4,2,1,2,
4,5,2,0,3,2,3,0,1,3,0,3,0,1,3,2,5,0,2,3,0,3,0,0,3,0,3,0,2,3,0,5,2,0,3,0,5,2,0,
3,5,1,2,0,3,4,2,1,2,5,3,0,1,2,4,3,1,2,0,3,5,2,0,1,3,3,0,4,1,2,3,0,3,0,0,3,2,5,
0,0,3,2,5,0,0,3,2,4,1,2,3,0,5,2,2,5,0,5,2,3,1,1,3,0,3,0,0,3,0,3,2,1,4,0,3,2,1,
4,2,5,2,0,3,2,3,0,5,2,2,3,4,1,2,0,3,4,2,1,2,4,3,2,0,3,0,5,2,2,5,2,3,0,0,3,2,3,
0,5,0,2,3,0,3,2,4,1,2,3,0,3,0,3,2,0,3,0,3,2,0,3,0,3,0,2,5,0,3,2,0,5,2,3,0,0,5,
2,3,2,1,4,0,3,2,0,3,2,4,1,2,3,0,4,1,2,3,4,1,2,0,3,4,1,2,2,5,3,0,1,2,4,3,2,1,0,
4,3,2,1,0,3,3,2,0,1,4,3,2,0,5,0,3,2,0,3,2,3,2,0,3,2,3,0,1,3,0,3,1,0,3,2,4,1,0,
3,2,5,0,0,3,2,3,0,0,3,0,3,0,0,3,3,0,1,0,3,3,1,0,0,3,3,2,1,2,4,5,2,1,2,4,3,0,2,1,
5,3,0,2,1,3,4,2,1,0,3,3,2,1,2,4,3,0,0,3,0,3,0,2,3,0,4,1,2,3,0,3,2,2,3,0,3,0,0,3,
0,3,1,2,4,1,3,1,2,4,1,3,2,2,3,2,5,2,0,5,2,3,2,0,3,4,1,2,2,3,4,1,2,0,3,3,1,0,0

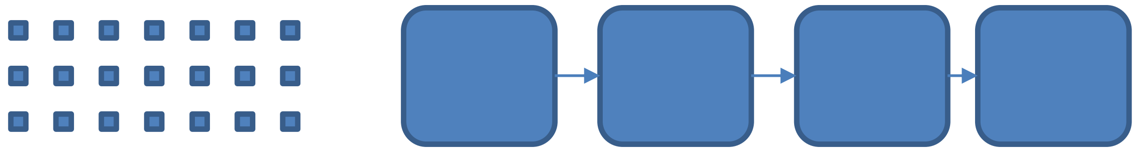
Solving time: 2.8s

Flow Shop Scheduling

- Factory scheduling
  - A number of pieces must be machined
  - They follow the same path on machines
    - Step1 on machine1, step2 on machine2, etc.
  - Each part takes a different amount of time on each machine
  - Parts are ordered at the start, and never between machines
  - A machine must wait if the next part is not ready
  - A part must wait if the next machine is not ready
  - Minimize the total machining time by properly sequencing the parts

Flow shop scheduling

- Parts must pass through a machine line



  - Each part takes a different duration on each machine
  - Parts are sequenced at the start
  - machines must wait if the next part is not ready from previous machine
  - A part must wait if the next machine is not ready

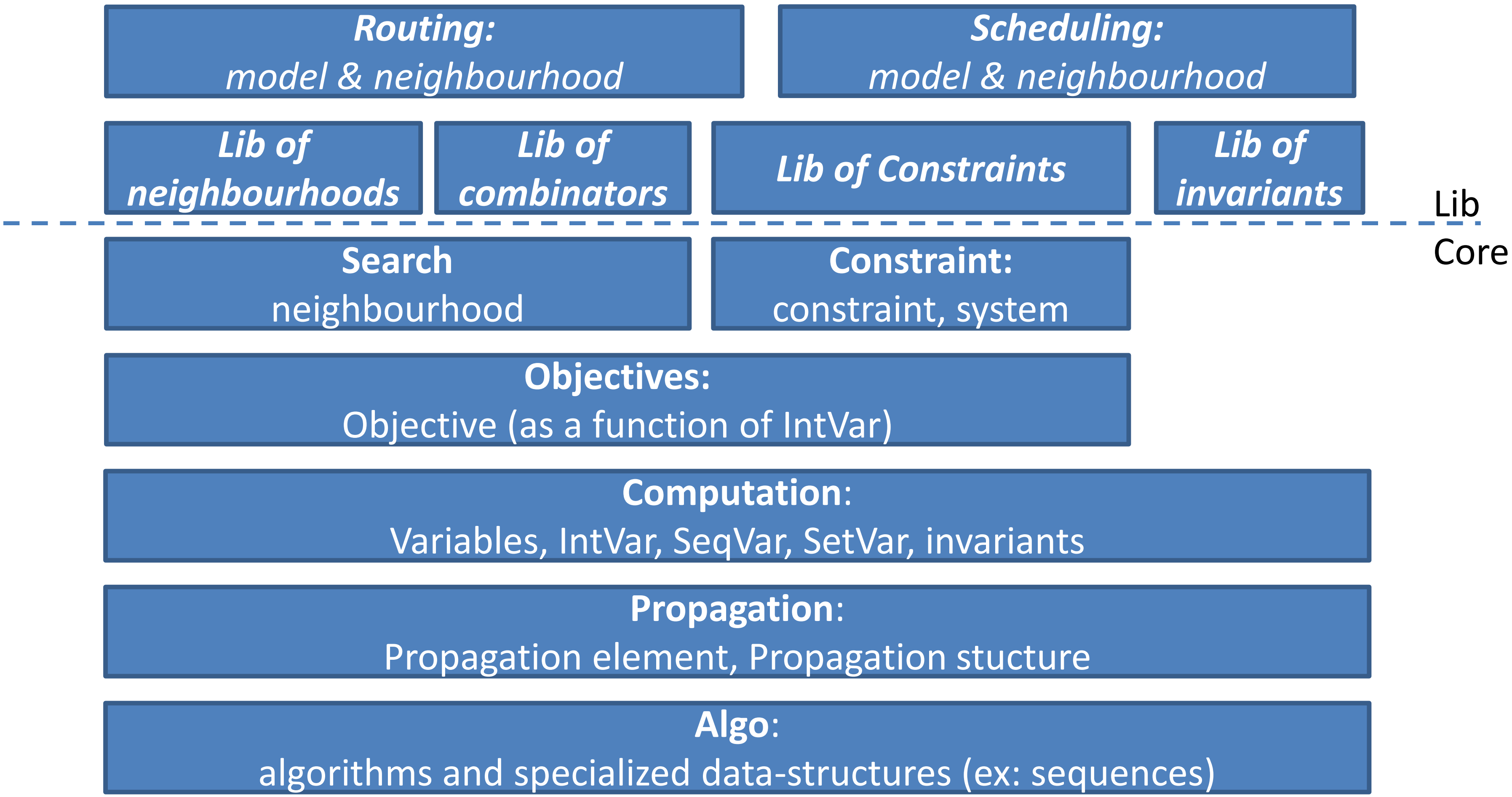
- Problem statement
  - Given
    - Machines, set of parts and duration of each part on each machine
  - Find
    - Proper sequence of the parts
  - Such that
    - total machining time is minimized

A Flow shop scheduling problem and its solution

```
val machineToJobToDuration:Array[Array[Int]] =
  Array(
    Array(1,2,1,7,2,5,5,6,7),
    Array(4,5,3,1,8,3,7,8,4),
    Array(6,8,2,5,3,1,2,2,8),
    Array(4,1,7,2,5,5,6,4,5))
```

no more improvement found after 77 it, 1150 ms
job sequence:0,2,6,8,4,5,3,1,7

«Use» Architecture of OscaR.cbls



Code structure

- algo
- core
  - computation
  - propagation
  - constraint
  - objective
  - search
- lib
  - constraint
  - invariant
  - search
    - neighbourhoods
    - combinators
    - linear selectors
- modelling
- business
  - routing
  - scheduling (deprecated)
- benchmarks
- visual

To write simple models,
modelling package
provides factories to *core* and *lib*

```
import oscar.cbls._
import oscar.cbls.modeling._

object MyStuff extends CBLSModel{…}
```

Business package provides
model and neighbourhoods for

- *routing*
- *scheduling* (deprecated)

```
import oscar.cbls.business.routing._
```

Conclusion: Features of Oscar.cbls

- 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]
- 50kLOC
- Open source LGPL
  - Code using OscaR is not contaminated
  - Extensions and corrections to OscaR are expected to be pushed back to OscaR

Your Connection to ICT Research

Further readings

1. Renaud De Landtsheer, Christophe Ponsard, OscaR.cbls : an open source framework for constraint-based local search, 27th ORBEL Annual Meeting, Kortrijk, Belgium, February 7-8 2013.
2. Renaud De Landtsheer, Yoann Guyot, Gustavo Ospina, Christophe Ponsard, Local Search with OscaR.CBLS, Workshop Design and Analysis of Meta-heuristics, Antwerp, 17-18 March 2016.
3. Renaud De Landtsheer, Yoann Guyot, Gustavo Ospina, Christophe Ponsard, Towards the Complexity of Differentiation Through Lazy Updates in Local Search Engines, 30th ORBEL Annual Meeting, Louvain-La-Neuve, Belgium, January 28-29 2016
4. Renaud De Landtsheer, Yoann Guyot, Gustavo Ospina, Christophe Ponsard, Adding a Sequence Variable to the OscaR.CBLS Engine, 31th ORBEL Annual Meeting, Brussels, Belgium, February 2-3, 2017
5. Renaud De Landtsheer, Gustavo Ospina, Yoann Guyot, Fabian Germeau, Christophe Ponsard, Supporting Efficient Global Moves on Sequences in Constraint-based Local Search Engines, Proceedings of the 6th International Conference on Operations Research and Enterprise Systems, 171-180, 2017, Porto, Portugal
6. Renaud De Landtsheer, Yoann Guyot, Gustavo Ospina, and Christophe Ponsard. Recent developments of metaheuristics, chapter Combining Neighborhoods into Local Search Strategies, pages 43–57. Springer, 2018.
7. Generic Support for Global Routing Constraint in Constraint-Based Local Search Frameworks, Quentin Meurisse, Renaud De Landtsheer, 32th ORBEL Annual Meeting, Liege, Belgium, February 1-2 2018
8. Renaud De Landtsheer, Fabian Germeau, Yoann Guyot, Gustavo Ospina, Christophe Ponsard, Easily Building Complex Neighbourhoods With the Cross-Product Combinator, 32th ORBEL Annual Meeting, Liege, Belgium, February 1-2 2018
9. Renaud De Landtsheer, Yoann Guyot, Gustavo Ospina, Fabian Germeau, and Christophe Ponsard, Reasoning on Sequences in Constraint-Based Local Search Frameworks, accepted at CPAIOR2018, 15th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research June 26-29, 2018, Delft, The Netherlands

Who is behind OscaR.cbls?

- CETIC team
  - Renaud De Landtsheer
  - Thomas Fayolle
  - Fabian Germeau
  - Gustavo Ospina
  - Christophe Ponsard
  - Yoann Guyot (until 2017)
- Contributions from Uppsala
  - Jean-Noël Monette
  - Gustav Björdal
- Internships & MS Theses
  - UMONS: Gaël Thouvenin, Sébastien Drobisz, Florent Ghilain, Jannou Bohée, Quentin Meurisse
  - IPL: Fabian Germeau
  - HENALUX: Quentin Wautelet

Cetic
Your Connection to ICT Research

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

Other CBLS tools

- Comet
  - First CBLS implementation by Pascal van Hentenryck and Laurent Michel
  - Not maintained since 2008
- Kangaroo
  - One paper @CP2011, status unknown, not available
- LocalSolver
  - Commercial tool, with academic licence
  - Booleans, floats, integers, lists with 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
  - CBLS engine, Toulouse, Cedric Pralet
- Yacc
  - ??

Two typical remarks on OscaR.cbls

- *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 …

Brain cycle is more valuable than CPU cycle

- 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 implementation
  - A core is cheaper than a single day of work for an engineer

Sörensen’s conjecture (Prof UAntwerp)

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