A comparative study of branch-and-price algorithms for a vehicle routing problem with time windows and waiting time costs

Stefano Michelini[1], Yasemin Arda[1], Hande Küçükaydın[2]

[1]HEC Liège, Université de Liège

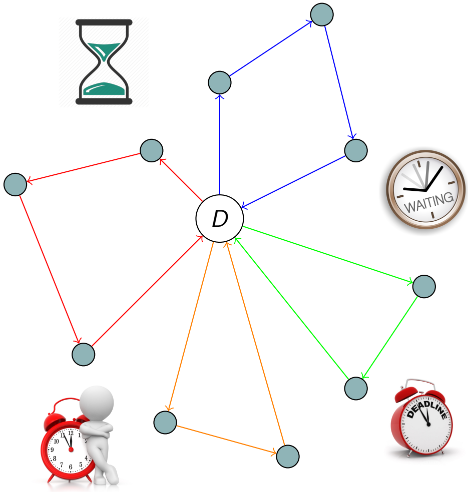
[2]MEF International School, Istanbul

February 1, 2018

Table of Contents

Table of Contents

VRP with Time Windows — variant



- Objective is the total route duration.
- Route start time is a decision variable.
- Maximum duration defined for each route.

Subject of study

- Branch-and-Price (BP): Branch-and-bound + Column Generation (CG)
- CG: Restricted Master Problem $\leftrightarrows$ Pricing Problem
- Master Problem: Set Partitioning

Set Partitioning Model

$$
\begin{aligned}
\min \sum_{r \in \Omega} c_r y_r & & \\
\text{s.t.} \sum_{r \in \Omega} a_{ir} y_r = 1 & & \forall i \in N, \\
y_r \in \{0,1\} & & \forall r \in \Omega.
\end{aligned}
$$

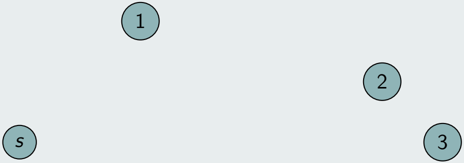
- Pricing Problem: Elementary Shortest Path Problem with Resource Constraints (ESPPRC)
- ESPPRC is NP-hard![1]

[1] Dror 1994.

Labeling algorithm[2]

- Each label L_i represents a partial path ending at node i.
- L_i keeps track of the consumption of resources along the partial path.

Label extension



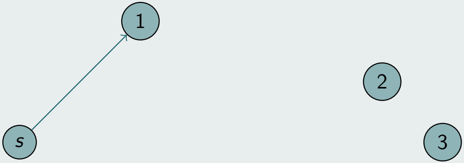
$$L_s = (C_s = 0, \mathbf{R}_s = \mathbf{0}, \mathbf{E}_s = \mathbf{0})$$

[2] Feillet et al. 2004.

Labeling algorithm[2]

- Each label L_i represents a partial path ending at node i.
- L_i keeps track of the consumption of resources along the partial path.

Label extension



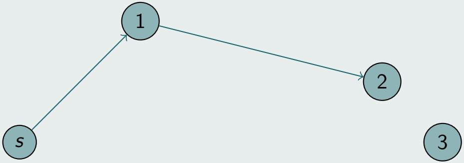
$L_1 = (C_1, \mathbf{R}_1, \mathbf{E}_1)$

[2]Feillet et al. 2004.

Labeling algorithm[2]

- Each label L_i represents a partial path ending at node i.
- L_i keeps track of the consumption of resources along the partial path.

Label extension



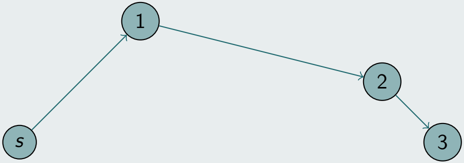
$$L_2 = (C_2, \mathbf{R}_2, \mathbf{E}_2)$$

[2] Feillet et al. 2004.

Labeling algorithm[2]

- Each label L_i represents a partial path ending at node i.
- L_i keeps track of the consumption of resources along the partial path.

Label extension



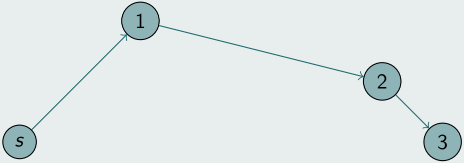
$$L_3 = (C_3, \mathbf{R}_3, \mathbf{E}_3)$$

[2]Feillet et al. 2004.

Labeling algorithm[2]

- Each label L_i represents a partial path ending at node i.
- L_i keeps track of the consumption of resources along the partial path.

Label extension



$$L_3 = (C_3, \mathbf{R}_3, e_3^1, e_3^2, \ldots, e_3^n)$$

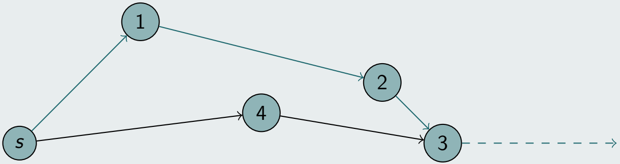
- Keeping binary resources for every node ensures elementarity.

[2] Feillet et al. 2004.

Labeling algorithm[2]

- Each label L_i represents a partial path ending at node i.
- L_i keeps track of the consumption of resources along the partial path.

Label extension



$$L_3 = (C_3, \mathbf{R}_3, e_3^1, e_3^2, \ldots, e_3^n)$$
$$\wedge|$$
$$L'_3 = (C'_3, \mathbf{R}'_3, e_3'^1, e_3'^2, \ldots, e_3'^n)$$

- Keeping binary resources for every node ensures elementarity.
- This makes label domination more difficult.

[2] Feillet et al. 2004.

Adaptation to our variant

New label structure

$$\overline{L}_i = (\delta_i, l_i, \alpha_i, A_i, D_i, \{e_i^k\}_{k \in N})$$

- δ_i = sum of dual prices
- l_i = latest feasible start time from the depot
- α_i = earliest feasible service time at i
- A_i = minimum possible duration of the partial path
- D_i = accumulated vehicle load

Table of Contents

Decremental State Space Relaxation (DSSR)[3]

- Maintain a set Θ of *critical* nodes on which elementarity is enforced.

DSSR

Algorithm 1 DSSR

1: Initialize Θ
2: **repeat**
3: $\quad P = \text{LabelExtension}(\Theta)$
4: $\quad \Phi = \text{MultipleVisits}(P)$
5: $\quad \Theta = \Theta \cup \Phi$
6: **until** P is elementary

- We manipulate the state space in a *global* way.

[3] Righini and Salani 2008.

Parameters — DSSR

- Critical set initialization:
  - `dssr_init_s`: *none*, *tca*, *hca*, *wtca*, *whca*, *mixed*
  - `dssr_init_n`: $]0,1[$
- Critical set update rules:
  - `dssr_path_s`: *all-paths*, *one-path*, *in-between*
  - `dssr_path_n`: $]0,1[$
  - `dssr_node_s`: *all-nodes*, *one-node*, *in-between*
  - `dssr_node_n`: $]0,1[$

ng-route Relaxation (NGRR)[4]

- Define neighbourhoods $N_i, \ \forall i$.

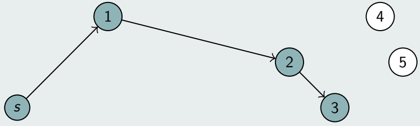
NGRR extension

[4] Baldacci et al. 2010.

ng-route Relaxation (NGRR)[4]

- Define neighbourhoods $N_i, \ \forall i$.
- E_i = set of unreachable nodes for path ending at i.

NGRR extension



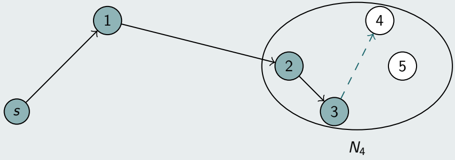
- $\mathbf{E}_3 = \{s, 1, 2, 3\}$

[4]Baldacci et al. 2010.

ng-route Relaxation (NGRR)[4]

- Define neighbourhoods $N_i, \ \forall i$.
- $E_i =$ set of unreachable nodes for path ending at i.
- When extending to j: $\forall k \in E_i$, insert k in E_j only if $k \in N_j$.

NGRR extension



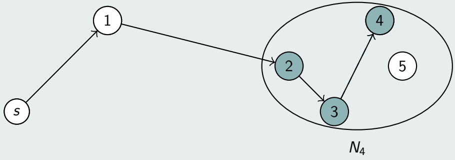
- $\mathbf{E}_3 = \{s, 1, 2, 3\}$
- $j = 4, \ N_4 = \{2, 3, 4, 5\}$

[4]Baldacci et al. 2010.

ng-route Relaxation (NGRR)[4]

- Define neighbourhoods $N_i, \ \forall i$.
- E_i = set of unreachable nodes for path ending at i.
- When extending to j: $\forall k \in E_i$, insert k in E_j only if $k \in N_j$.

NGRR extension



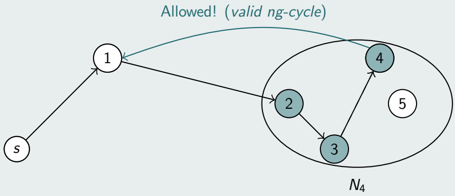
- $\mathbf{E}_3 = \{s, 1, 2, 3\}$
- $j = 4, \ N_4 = \{2, 3, 4, 5\}$
- $\mathbf{E}_4 = (\mathbf{E}_3 \cap N_4) \cup \{4\} = \{2, 3, 4\}$

[4] Baldacci et al. 2010.

ng-route Relaxation (NGRR)[4]

- Define neighbourhoods $N_i,\ \forall i$.
- E_i = set of unreachable nodes for path ending at i.
- When extending to j: $\forall k \in E_i$, insert k in E_j only if $k \in N_j$.

NGRR extension



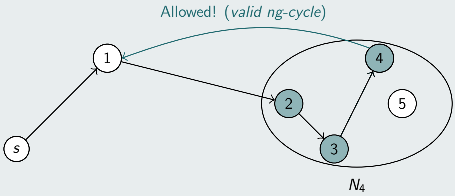
- $\mathbf{E}_3 = \{s, 1, 2, 3\}$
- $j = 4,\ N_4 = \{2, 3, 4, 5\}$
- $\mathbf{E}_4 = (\mathbf{E}_3 \cap N_4) \cup \{4\} = \{2, 3, 4\}$

[4]Baldacci et al. 2010.

ng-route Relaxation (NGRR)[4]

- Define neighbourhoods $N_i, \ \forall i$.
- E_i = set of unreachable nodes for path ending at i.
- When extending to j: $\forall k \in E_i$, insert k in E_j only if $k \in N_j$.
- Resulting path may **not** be elementary.

NGRR extension



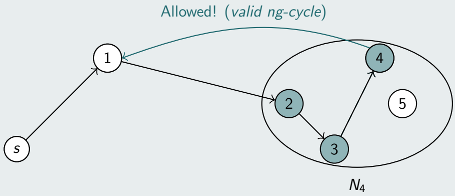
- $\mathbf{E}_3 = \{s, 1, 2, 3\}$
- $j = 4, \ N_4 = \{2, 3, 4, 5\}$
- $\mathbf{E}_4 = (\mathbf{E}_3 \cap N_4) \cup \{4\} = \{2, 3, 4\}$

[4] Baldacci et al. 2010.

ng-route Relaxation (NGRR)[4]

- Define neighbourhoods $N_i, \ \forall i$.
- E_i = set of unreachable nodes for path ending at i.
- When extending to j: $\forall k \in E_i$, insert k in E_j only if $k \in N_j$.
- Resulting path may **not** be elementary.
- We manipulate the state space in a *local* way.

NGRR extension



- $\mathbf{E}_3 = \{s, 1, 2, 3\}$
- $j = 4, \ \ N_4 = \{2, 3, 4, 5\}$
- $\mathbf{E}_4 = (\mathbf{E}_3 \cap N_4) \cup \{4\} = \{2, 3, 4\}$

[4]Baldacci et al. 2010.

Parameters — NGRR

- Neighborhood size `ng_size`: $]0, 1[$
- Node metric `ng_type`: *travel-time, cycle-risk, mixed*
- Coefficient for the mixed metric `ng_mix`: $]0, 1[$

NGRR and DSSR

- We can combine both algorithms in several ways.
- The straightforward combination uses both neighbourhoods and the set of critical nodes.

Global NG-DSSR

Algorithm 2 NG-DSSR-G

1: Initialize $\Theta,\ N_i\ \forall i$
2: **repeat**
3: $\quad P = \text{LabelExtension}(\Theta,\ N_i)$
4: $\quad \Phi = \text{MultipleVisits}(P,\ \{N_i\}_i)$ // Only takes nodes in invalid ng-cycles
5: $\quad \Theta = \Theta \cup \Phi$
6: **until** P is ng-route

NGRR and DSSR

- Let us define *applied* neighbourhoods $\hat{N}_i \subseteq N_i \ \forall \ i$ to use throughout label extension[5].
- When best path has invalid cycle C, update applied neighbourhoods of nodes in C.

Local NG-DSSR

Algorithm 3 NG-DSSR-L

1: Initialize $N_i, \ \hat{N}_i, \ \forall i$
2: **repeat**
3: $\quad P =$ LabelExtension$(\{\hat{N}_i\}_i)$
4: $\quad C =$ InvalidCycle$(P, \ \{N_i\}_i)$
5: $\quad$ Update $(\{\hat{N}_i\}_i, \ C)$
6: **until** P is ng-route

[5] Dayarian et al. 2015.

NGRR and DSSR

- We can derive a different version of DSSR using this local approach[6].
- Let us define local critical sets $\hat{\Theta}_i, \ \forall\ i$.
- When best path has cycle C, update critical sets of nodes in C.

Local DSSR

Algorithm 4 DSSR-L

Initialize $\hat{\Theta}_i, \ \forall i$
repeat
    $P =$ LabelExtension$(\{\hat{\Theta}_i\}_i)$
    $C =$ Cycle(P)
    Update $(\{\hat{\Theta}_i\}_i, C)$
until P is elementary

[6] Martinelli, Pecin, and Poggi 2014.

NGRR and DSSR

- Global and Local ng-DSSR output ng-routes.
- We can continue execution with DSSR (DSSR-L) to obtain elementary routes.

Elementary Global (Local) NG-DSSR

Algorithm 5 NG-DSSR-G (L)-E

```
P = ng-DSSR-local()
if P is not elementary then
    P = DSSR-local(P)
end if
```

Branch-and-Price framework

- After solving the linear relaxation at the root node, solve the integer program with the available columns to obtain an upper bound.
- Branch on the most fractional arc.
- Parameters:
  - Maximum number of generated paths `concat_stop`: $]0, 10000]$
  - Maximum number of paths inserted in the master problem `num_col`: $]0, 1[$
  - Tree navigation strategy `tree_nav`:
    - Depth-first search
    - Breadth-first search
    - Best-first search

Overview

- NGRR
- NG-DSSR-G
- NG-DSSR-L

ng-routes

- DSSR
- DSSR-L
- NG-DSSR-G-E
- NG-DSSR-L-E

Elementary routes

Parameters

Parameter	DSSR	NGRR	NG-D.-G	NG-D.-L	DSSR-L	NG-D.-G-E	NG-D.-L-E	Range
tree_trav	×	×	×	×	×	×	×	{breadth, depth, best}
n_conc	×	×	×	×	×	×	×	$]0, 10000]$
n_col	×	×	×	×	×	×	×	$]0, 1]$
dssr_init_s	×		×		×	×		{hca, tca, whca, wtca, mix}
dssr_init_n	×		×		×	×		$]0, 1[$
dssr_path_s	×		×	×	×	×	×	{1_path, in_btw, all_paths}
dssr_path_n	×		×	×	×	×	×	$]0, 1[$
dssr_node_s	×		×			×		{1_node, in_btw, all_nodes}
dssr_node_n	×		×			×		$]0, 1[$
ng_type		×	×	×		×	×	{tt, ccr, mix}
ng_size		×	×	×		×	×	$]0, 1[$
ng_mix		×	×	×		×	×	$]0, 1[$

Table: Parameters choices

Table of Contents

Methodology

- Tune the parameters with the `irace` package[7]
- The tuning set is derived from the Gehring & Homberger instances for the VRPTW.
- Set of 6 instances, one per type (clustered, random, random-clustered) and per size (25 and 50 customers).
- Tuning budget of 5000 experiments for each algorithm, time limit $TL = 3$ hours per experiment.
- Measure of performance for configuration θ on instance i:

$$t'(i,\theta) = \begin{cases} t(i,\theta) & \text{if } t(i,\theta) < TL \text{ seconds (3 hours),} \\ TL + \mathrm{UB}(i,\theta) & \text{if root node solved and TL reached,} \\ \mathrm{M} & \text{otherwise} \end{cases}$$

- Run the best configuration of each algorithm on the Solomon instances.
- Perform Friedman test + post-hoc Wilcoxon signed rank test on the results.

[7] López-Ibáñez et al. 2016.

Statistical tests on Solomon instances

- The Friedman test reports a p-value $p < 2.2 \times 10^{-16}$, denoting statistically significant difference among the data
- Post-hoc test reports that DSSR-L outperforms DSSR, NGRR, NG-DSSR-G, and NG-DSSR-G-E with a p-value $p < 1.0 \times 10^{-7}$
- DSSR-L, NG-DSSR-L and NG-DSSR-L-E are statistically equivalent

Solved instances

	DSSR	DSSR-L	NG	NG-D-G	NG-D-L	NG-D-G-E	NG-D-L-E
C1–25	(5, 1, 3)	(6, 3, 0)	(5, 4, 0)	(5, 2, 2)	(5, 4, 0)	(5, 1, 3)	(5, 3, 1)
R1–25	(12, 0, 0)	(12, 0, 0)	(12, 0, 0)	(12, 0, 0)	(12, 0, 0)	(12, 0, 0)	(12, 0, 0)
RC1–25	(8, 0, 0)	(8, 0, 0)	(8, 0, 0)	(8, 0, 0)	(8, 0, 0)	(8, 0, 0)	(8, 0, 0)
C1–50	(4, 0, 5)	(6, 0, 3)	(6, 0, 3)	(4, 0, 5)	(5, 1, 3)	(5, 0, 4)	(6, 1, 2)
R1–50	(5, 6, 1)	(9, 3, 0)	(7, 5, 0)	(6, 6, 0)	(9, 3, 0)	(5, 7, 0)	(10, 2, 0)
RC1–50	(0, 6, 2)	(1, 7, 0)	(1, 7, 0)	(1, 7, 0)	(1, 7, 0)	(1, 7, 0)	(1, 7, 0)

Table: Solved Solomon instances, 3 hour time limit

$$(n_s, n_p, n_u)$$

- $n_s = \#$ of solved instances
- $n_p = \#$ of partially solved instances (at least the root node)
- $n_u = \#$ of unsolved instances

Additional tests

- We run the best four algorithms on the Solomon instances for 6 hours, on 25, 50, 75 and 100 customers
- The Friedman test reports a p-value $p = 5.093 \times 10^{-5}$
- Post-hoc reports that NGRR is rejected with p-value $p < 0.01$ by the other algorithms

Solved instances — 2

	DSSR-L	NG-DSSR-L-E	NG	NG-DSSR-L
C1–25	(7, 2, 0)	(6, 3, 0)	(6, 3, 0)	(7, 2, 0)
R1–25	(12, 0, 0)	(12, 0, 0)	(12, 0, 0)	(12, 0, 0)
RC1–25	(8, 0, 0)	(8, 0, 0)	(8, 0, 0)	(8, 0, 0)
C1–50	(6, 1, 2)	(6, 1, 2)	(6, 0, 3)	(6, 1, 2)
R1–50	(9, 3, 0)	(10, 2, 0)	(8, 4, 0)	(9, 3, 0)
RC1–50	(2, 6, 0)	(1, 7, 0)	(1, 7, 0)	(1, 7, 0)
C1–75	(5, 1, 3)	(5, 1, 3)	(5, 0, 4)	(5, 0, 4)
R1–75	(3, 9, 0)	(4, 8, 0)	(4, 8, 0)	(4, 8, 0)
RC1–75	(0, 8, 0)	(0, 8, 0)	(0, 8, 0)	(1, 7, 0)
C1–100	(5, 0, 4)	(5, 0, 4)	(4, 0, 5)	(5, 0, 4)
R1–100	(2, 2, 8)	(3, 1, 8)	(2, 2, 8)	(3, 1, 8)
RC1–100	(0, 2, 6)	(0, 2, 6)	(0, 3, 5)	(0, 5, 3)

Table: Solved Solomon instances, 6 hour time limit

Solved instances — 2

	DSSR-L				NG-DSSR-L-E			
	gap	root gap	time	nodes	gap	root gap	time	nodes
C1-25	0.17%	0.17%	7547.5	51.0	0.13%	0.13%	9236.3	54.8
C1-50	0.01%	0.01%	8333.8	2.1	0.01%	0.01%	7604.8	2.0
C1-75	0.07%	0.07%	11974.3	1.6	0.10%	0.10%	10592.8	1.2
C1-100	0.0%	0.0%	10653.5	0.6	0.00%	0.00%	10130.8	0.6
R1-25	0.42%	0.56%	12.1	7.8	0.42%	0.56%	23.1	14.5
R1-50	0.84%	0.94%	6546.3	167.8	0.76%	1.03%	6600.6	539.2
R1-75	1.02%	1.03%	16213.6	77.2	0.92%	1.05%	16041.1	257.2
R1-100	0.42%	0.50%	19520.2	93.4	0.39%	0.48%	18881.6	93.8
RC1-25	1.15%	1.15%	11.2	25.5	1.15%	1.15%	24.3	13.5
RC1-50	5.67%	5.70%	18819.6	8677.4	5.60%	5.71%	19051.6	8616.4
RC1-75	3.12%	3.14%	21600	1587.3	3.03%	3.10%	21600	2717.1
RC1-100	1.27%	1.78%	21600	261.6	1.72%	1.72%	21600	366.1

Table: Performance on Solomon instances, 6 hour time limit — 1

Solved instances — 2

	NG				NG-DSSR-L			
	gap	root gap	time	nodes	gap	root gap	time	nodes
C1-25	0.17%	0.17%	9125.1	189.3	0.43%	0.47%	8845.6	112.8
C1-50	0.01%	0.01%	7820.6	1.6	0.01%	0.01%	7817.9	1.9
C1-75	0.08%	0.08%	11607.5	1.4	0.08%	0.09%	11262.3	1.4
C1-100	0.00%	0.00%	12608.4	0.4	0.00%	0.00%	10545.3	0.8
R1-25	0.60%	0.78%	33.3	33.0	0.48%	0.69%	13.1	9.2
R1-50	1.00%	1.21%	8317.2	978.5	0.84%	1.04%	6823.3	333.6
R1-75	0.98%	1.08%	15062.0	196.5	1.01%	1.03%	15375.1	176.8
R1-100	0.38%	0.44%	19211.7	59.6	0.39%	0.56%	17823.0	120.5
RC1-25	1.97%	3.77%	1968.1	786.5	1.19%	1.19%	96.8	17.5
RC1-50	6.33%	6.72%	19907.2	10364.8	5.87%	6.14%	18992.2	10211.4
RC1-75	3.50%	3.65%	21600	3591.0	3.26%	3.27%	21101.7	3030.5
RC1-100	1.97%	1.97%	21600	470.8	2.05%	2.38%	21600	302.9

Table: Performance on Solomon instances, 6 hour time limit — 2

Parameter configurations

	n_col_root	conc_stop_root	n_col	conc_stop	tree_nav
DSSR-L					
Best	82%	2820	31%	9247	best-first
Avg	42%	3998	35%	8902	best-first (5)
Range	[10%,81%]	[2764,8481]	[22%,63%]	[7308,9867]	
NG-DSSR-L-E					
Best	27%	7910	42%	9845	depth-first
Avg	45%	5949	37%	8688	depth-first (6)
Range	[26%,98%]	[1003,8320]	[36%,42%]	[5234,9923]	
NG-DSSR-L					
Best	4%	8752	14%	3079	depth-first
Avg	10%	6900	24%	3540	depth-first (6)
Range	[1%,35%]	[5476,8752]	[11%,63%]	[1100,4519]	

Table: BP parameters

Parameter configurations

	init_str_root	init_n_root	ins_path_str_root	ins_path_n_root
DSSR-L				
Best	none	n/a	in-between	25%
Avg	none (6)		in-between (6)	38%
Range				[24%,47%]
NG-DSSR-L-E				
Best	n/a	n/a	one-path	n/a
Avg			one-path (6)	
Range				
NG-DSSR-L				
Best	n/a	n/a	all-paths	n/a
Avg			all-paths (5)	
Range				

Table: DSSR parameters, root

Parameter configurations

	init_str	init_n	ins_path_str	ins_path_n
DSSR-L				
Best	none	n/a	in-between	39%
Avg	none (6)		in-between (6)	57%
Range				[38%,80%]
NG-DSSR-L-E				
Best	n/a	n/a	in-between	98%
Avg			in-between (6)	81%
Range				[72%,97%]
NG-DSSR-L				
Best	n/a	n/a	in-between	35%
Avg			in-between (6)	31%
Range				[26%,34%]

Table: DSSR parameters

Parameter configurations

	ng_type_root	ng_size_root	ng_type	ng_size
NG-DSSR-L-E				
Best	travel-time	6%	travel-time	23%
Avg	travel-time (5)	9%	travel-time (6)	24%
Range		[4%,21%]		[16%,45%]
NG-DSSR-L				
Best	travel-time	17%	travel-time	88%
Avg	travel-time (6)	17%	travel-time (5)	81%
Range		[14%,24%]		[50%,97%]

Table: NG parameters

Table of Contents

Conclusions

- These results are largely the same on the classic VRPTW
- We performed analysis on tuning sets of instances with different sizes
  - There is some evidence that including large instances gives the best results
- "Local" approach for treating elementarity resources seems the best approach
- It makes sense to parametrize differently the algorithm in the root node
- Best parameter values provided by tuning might be different than the ones suggested in the literature
- Future work: adapt to multi-trip extension of the problem

Thanks!

Thanks for your attention.