Advances in Exact Algorithms for Vehicle Routing

Eduardo Uchoa

Departamento de Engenharia de Produção
Universidade Federal Fluminense, Brazil
INRIA International Chair 2022–2026, Bordeaux
Vehicle Routing Problem (VRP)

One of the most widely studied in Combinatorial Optimization:

- $+6,000$ works published only in 2021 (Google Scholar), mostly heuristics
- Direct application in the real systems that distribute goods and provide services
Reflecting the variety of real transportation systems, VRP literature is spread into hundreds of variants. For example, there are variants that consider:

- Vehicle capacities,
- Time windows,
- Heterogeneous fleets,
- Multiple depots,
- Split delivery, pickup and delivery, backhauling,
- Arc routing (Ex: garbage collection),
- etc, etc.
Traveling Salesman Problem (TSP)
Simplest routing problem: single vehicle, no constraints to the route

Largest solved instance (Concorde): 85,900 points!! Instances with one thousand points often solved in one minute
Capacitated Vehicle Routing Problem (CVRP)

\[ Q = 10 \]

\[ d_i \]

\[ 1 \quad 2 \quad 3 \quad 4 \quad 5 \]

Hausdorff School – Bonn 2022
Advances in Exact Algorithms for Vehicle Routing
Capacitated Vehicle Routing Problem (CVRP)
VRP with Time Windows (VRPTW)

\[ t_{ij} \]

\[ Q=10 \]

Hausdorff School – Bonn 2022

Advances in Exact Algorithms for Vehicle Routing 10 / 109
Heterogeneous Fleet VRP (HFVRP)

\[ Q_1 = 10 \]
\[ Q_2 = 18 \]
Heterogeneous Fleet VRP (HFVRP)

\begin{align*}
Q_1 &= 10 \\
Q_2 &= 18
\end{align*}
Multi-Depot VRP (MDVRP)
Multi-Depot VRP (MDVRP)
Pickup and Delivery VRP (PDVRP)

Q = 10

D1, D2, D3, D4, D5
P1, P2, P3, P4, P5
Pickup and Delivery VRP (PDVRP)

Q = 10

D5 → D2 → P5 → P1 → D3 → D1 → D4 → P2 → P3 → P4

Hausdorff School – Bonn 2022
Advances in Exact Algorithms for Vehicle Routing
Routing of electrical vehicles:

- Limited autonomy. Recharge is only available at a few points and is slow

Dynamic Routing:

- New demands appear during the day, ongoing routes may have to be changed due to unexpected events...
Outline of the presentation

Part I - Advances on Exact CVRP algorithms
- Review of the advances in the last 15 years
- A key advance: cuts with limited memory

Part II - From CVRP to other classic VRP variants

Part III - A Generic Exact VRP Solver
- A generic VRP model
- Computational results
- Downloading and using the code

Conclusion: perspectives on the use of exact algorithms in practice
Part I - Advances on Exact CVRP Algorithms
First (Dantzig and Ramser [1959]) and most basic variant:

**Instance:** Complete graph $G = (V, E)$ with $V = \{0, \ldots, n\}$; 0 is the depot, $N = \{1, \ldots, n\}$ is the set of customers. Each edge $e \in E$ costs $c_e$. Each $i \in N$ demands $d_i$ units. Homogeneous fleet of vehicles with capacity $Q$.

**Solution:** Routes from the depot, respecting the capacities and visiting all customers once; minimizing the total cost.
Variable $x_e$ indicates how many times $e$ is used.

$$\min \sum_{e \in E} c_e x_e$$

(1)

S.t.  $$\sum_{e \in \delta(i)} x_e = 2 \quad \forall \ i \in N,$$

(2)

$$\sum_{e \in \delta(S)} x_e \geq 2 \lceil \sum_{i \in S} d_i / Q \rceil \quad \forall \ S \subseteq N,$$

(3)

$$x_e \in \{0, 1\} \quad \forall \ e \in E \setminus \delta(0),$$

(4)

$$x_e \in \{0, 1, 2\} \quad \forall \ e \in \delta(0).$$

(5)

Constraints (3) are *Rounded Capacity Cuts*. 
Branch-and-Cut (BC) Algorithms for CVRP

Extensive research on families of cuts:
- Framed Capacity, Strengthened Comb, Multistar, Extended Hypotour, etc.

Dominant approach (Naddef and Rinaldi [2002]) until early 2000’s:
- Araque, Kudva, Morin, and Pekny [1994]
- Augerat, Belenguer, Benavent, Corberán, Naddef, and Rinaldi [1995]
- Blasum and Hochstättler [2000]
- Ralphs, Kopman, Pulleyblank, and Trotter Jr. [2003]
- Achuthan, Caccetta, and Hill [2003]
- Ralphs [2003]
- Wenger [2004]
- Lysgaard, Letchford, and Eglese [2004]
Best BC results

<table>
<thead>
<tr>
<th>Class</th>
<th>Size</th>
<th>#Ins</th>
<th>#Unsolved</th>
<th>Root gap</th>
<th>Avg. Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>36-79</td>
<td>22</td>
<td>7</td>
<td>2.06</td>
<td>6638</td>
</tr>
<tr>
<td>B</td>
<td>36-79</td>
<td>20</td>
<td>1</td>
<td>0.61</td>
<td>8178</td>
</tr>
<tr>
<td>E-M</td>
<td>50-199</td>
<td>12</td>
<td>9</td>
<td>2.10</td>
<td>39592</td>
</tr>
<tr>
<td>F</td>
<td>44-134</td>
<td>3</td>
<td>0</td>
<td>0.06</td>
<td>1016</td>
</tr>
<tr>
<td>P</td>
<td>14-100</td>
<td>24</td>
<td>8</td>
<td>2.26</td>
<td>11219</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>81</td>
<td>25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Processor: Intel Celeron 700MHz

Smallest unsolved instance: 49 customers


Why BC is so much better on TSP than on CVRP? One reason may be that the original Dantzig et al. [1954] formulation is much stronger than the original Laporte and Nobert [1983] formulation.
### Best BC results

<table>
<thead>
<tr>
<th>Class</th>
<th>Size</th>
<th>#Ins</th>
<th>#Unsolved</th>
<th>Root gap</th>
<th>Avg. Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>36-79</td>
<td>22</td>
<td>7</td>
<td>2.06</td>
<td>6638</td>
</tr>
<tr>
<td>B</td>
<td>36-79</td>
<td>20</td>
<td>1</td>
<td>0.61</td>
<td>8178</td>
</tr>
<tr>
<td>E-M</td>
<td>50-199</td>
<td>12</td>
<td>9</td>
<td>2.10</td>
<td>39592</td>
</tr>
<tr>
<td>F</td>
<td>44-134</td>
<td>3</td>
<td>0</td>
<td>0.06</td>
<td>1016</td>
</tr>
<tr>
<td>P</td>
<td>14-100</td>
<td>24</td>
<td>8</td>
<td>2.26</td>
<td>11219</td>
</tr>
<tr>
<td>Total</td>
<td>81</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Processor**

Intel Celeron 700MHz

Smallest unsolved instance: 49 customers


Why BC is so much better on TSP than on CVRP? One reason may be that the original Dantzig et al. [1954] formulation is much stronger than the original Laporte and Nobot [1983] formulation.
Set Partitioning Formulation – SPF (Balinski and Quandt [1964])

\( \Omega \) is the set of routes, route \( r \) costs \( c_r \), coefficient \( a_{ir} \) indicates how many times \( r \) visits customer \( i \)

\[
\min \sum_{r \in \Omega} c_r \lambda_r
\]

(6)

S.t.

\[
\sum_{r \in \Omega} a_{ir} \lambda_r = 1 \quad \forall \ i \in \mathcal{N},
\]

(7)

\[
\lambda_r \in \{0, 1\} \quad \forall \ r \in \Omega.
\]

(8)

- Exponential number of variables \( \Rightarrow \) Column generation / Branch-and-Price (BP) algorithms
- Pricing elementary routes is strongly NP-hard \( \Rightarrow \) Relax \( \Omega \) including non-elementary \( q \)-routes (Christofides et al. [1979])
Combining Column Generation and Cut Separation

SPF linear relaxation is very good for VRPTW *with tight windows* (Desrosiers et al. [1984]) but is quite weak for CVRP:

- Typical root gaps >3% (even if only elementary routes are priced!), worse than 2% of BC root gap

Fukasawa et al. [2006] combined both methods. A cut over edge variables

$$\sum_{e \in E} \alpha_e x_e \geq b,$$

is translated to

$$\sum_{r \in \Omega} (\sum_{e \in E} \alpha_e a_{er}) \lambda_r \geq b,$$

where $a_{er}$ is the number of times that $e$ is used in route $r$.

The combination of column generation with robust cuts defined over edges yields root gaps around 1%.
A crucial point in combining column generation with cuts is the effect of the new dual variables in the pricing:

- A cut is **robust** when its dual variable can be translated into costs in the pricing. The subproblem structure does not change.
- **non-robust** cuts change the pricing, each additional cut makes it harder.
Robust BCP results in FLL+06

<table>
<thead>
<tr>
<th>Class</th>
<th>#Ins</th>
<th>NS</th>
<th>Gap</th>
<th>T(s)</th>
<th>NS</th>
<th>Gap</th>
<th>T(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>22</td>
<td>7</td>
<td>2.06</td>
<td>6638</td>
<td>0</td>
<td>0.81</td>
<td>1961</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>1</td>
<td>0.61</td>
<td>8178</td>
<td>0</td>
<td>0.47</td>
<td>4763</td>
</tr>
<tr>
<td>E-M</td>
<td>12</td>
<td>9</td>
<td>2.10</td>
<td>39592</td>
<td>3</td>
<td>1.19</td>
<td>126987</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
<td>0</td>
<td>0.06</td>
<td>1016</td>
<td>0</td>
<td>0.06</td>
<td>2398</td>
</tr>
<tr>
<td>P</td>
<td>24</td>
<td>8</td>
<td>2.26</td>
<td>11219</td>
<td>0</td>
<td>0.76</td>
<td>2892</td>
</tr>
<tr>
<td>Total</td>
<td>81</td>
<td>25</td>
<td></td>
<td>11219</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor</td>
<td>Intel Celeron 700MHz</td>
<td>Pentium 4 2.4GHz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Robust BCP solved all literature instances with up to 134 customers. Three larger instances remained open: M-n151-k12, M-n200-k16 e M-n200-k17.

New exact CVRP methods

After Fukasawa et al. [2006], all the proposed exact CVRP algorithms combine column generation and cuts. Two surveys are:


Uses **non-robust cuts**: Strengthened Capacity and Clique. Root gaps were significantly reduced. Several tricks to keep pricing reasonably tractable.

Important new idea:

- Instead of branching, algorithm finishes by **enumerating** all routes with reduced cost smaller than the gap. The SPF with only those routes was solved by CPLEX. This saves a lot of time in some instances.

---

Introduces *ng-routes*, an effective elementarity relaxation better than *q*-routes:

- For each $i \in N$, $NG(i) \subseteq N$ contains the *ng* closest customers. An *ng*-route can only revisit $i$ if it passes first by a customer $j$ such that $i \notin NG(j)$
- $ng = 8$ does not make pricing too hard and, in practice, eliminates most cycles

Non-robust Subset Row Cuts (Chvátal-Gomory Cuts of Rank 1 over the set partitioning constraints [Jepsen et al., 2008]) replace Cliques, smaller impact on pricing

Back to Robust BCP, but already using $ng$-routes.

- Proposes a sophisticated and aggressive strong branching, reducing a lot the branch-and-bound trees
- Before this work, BCP algorithms only applied SB in a timid way, wrongly believing that aggressive SB would not pay in that context

M-n151-k12 solved in 5 days!

Uses Subset Row Cuts and \( ng \)-routes

Enumeration to a pool with up to several million routes can be performed. After that, pricing is done by inspection in the pool.

- Non-robusts cuts can be freely separated
- As lower bounds improve, fixing by reduced costs reduce pool size
- The problem is finished by a MIP solver only when pool size is much reduced

M-n151-k12 solved in less than 3 hours!

A complex BCP algorithm incorporating elements from all previously mentioned works and presenting some new ideas.

Pecin et al. [2014]: Cuts

- Robust Cuts (separation using code by J.Lysgaard [2004])
  - Rounded Capacity
  - Strengthened Comb
- Non-Robust Cuts
  - Subset Row
- Post-Enumeration Cuts
  - Cliques (separation using code by H.Santos [2012])
Most critical part of the BCP: dynamic programming label setting algorithm, handling:

- $ng$-routes
- The modifications induced by Subset Row Cuts

Features:

- Bi-directional search (Righini and Salani [2006]), using balanced mid-point
- Completion Bounds with under-evaluation (Contardo [2012])
- Fixing by reduced cost considering arc accumulated load (Pessoa et al. [2010])
Even with all care, non-robust cuts are indeed “non-robust”:

- Pricing may be handling hundreds of Subset Row Cuts well. Then, the separation of a few dozen additional cuts makes the pricing 100x or even 1000x slower!
- When such a situation is detected, the algorithm **rolls back**, removing the “bad” cuts.
Hybrid search strategy, combining branching and enumeration:

- **Strong Branching (SB):**
  - Hierarchical, 3 levels
  - Keeps full history to guide future decisions
  - Up to 200 candidates per node.

- **Route enumeration:**
  - Generates pool with up to 20M routes
  - Branching can be done over enumerated node
  - Uses MIP solver (CPLEX) only when pool is reduced to less than 20K routes

Dual stabilization by smoothing (Wentges [1997])
### Results in PPPU14

<table>
<thead>
<tr>
<th>Class</th>
<th>#Ins</th>
<th>BMR11 US</th>
<th>Gap</th>
<th>T(s)</th>
<th>Rop12 US</th>
<th>Gap</th>
<th>T(s)</th>
<th>CM14 US</th>
<th>Gap</th>
<th>T(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>22</td>
<td>0</td>
<td>0.13</td>
<td>30</td>
<td>0</td>
<td>0.57</td>
<td>53</td>
<td>0</td>
<td>0.09</td>
<td>59</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>0</td>
<td>0.06</td>
<td>67</td>
<td>0</td>
<td>0.25</td>
<td>208</td>
<td>0</td>
<td>0.08</td>
<td>34</td>
</tr>
<tr>
<td>E-M</td>
<td>12</td>
<td>3</td>
<td>0.49</td>
<td>303</td>
<td>2</td>
<td>0.96</td>
<td>44295</td>
<td>2</td>
<td>0.27</td>
<td>1548</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
<td>1</td>
<td>0.11</td>
<td>164</td>
<td>0</td>
<td>0.25</td>
<td>2163</td>
<td>0</td>
<td>0.03</td>
<td>27772</td>
</tr>
<tr>
<td>P</td>
<td>24</td>
<td>0</td>
<td>0.23</td>
<td>85</td>
<td>0</td>
<td>0.69</td>
<td>280</td>
<td>0</td>
<td>0.18</td>
<td>240</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>81</td>
<td>4</td>
<td>0.03</td>
<td>5.6</td>
<td>2</td>
<td>0.19</td>
<td>3669</td>
<td>2</td>
<td>0.07</td>
<td>33</td>
</tr>
</tbody>
</table>

**Processor**
- Xeon X7350 2.93GHz
- Core i7-2620M 2.7GHz
- Xeon E5462 2.8GHz

### PPPU14

<table>
<thead>
<tr>
<th>Class</th>
<th>#Ins</th>
<th>UnSolved</th>
<th>Gap</th>
<th>T(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>22</td>
<td>0</td>
<td>0.03</td>
<td>5.6</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>0</td>
<td>0.04</td>
<td>6.2</td>
</tr>
<tr>
<td>E-M</td>
<td>12</td>
<td>0</td>
<td>0.19</td>
<td>3669</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
<td>0</td>
<td>0.00</td>
<td>3679</td>
</tr>
<tr>
<td>P</td>
<td>24</td>
<td>0</td>
<td>0.07</td>
<td>33</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>81</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Processor**
- Core i7-3770 3.4GHz
## Detailed Results: M-n200-k16

<table>
<thead>
<tr>
<th>Algo</th>
<th>Root LB</th>
<th>Final LB</th>
<th>Total Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMR11</td>
<td>1256.6</td>
<td>1256.6</td>
<td>319</td>
</tr>
<tr>
<td>Ropke12</td>
<td>1253.0</td>
<td>1258.2</td>
<td>7200</td>
</tr>
<tr>
<td>CM14</td>
<td>1266.9</td>
<td>1266.9</td>
<td>432000</td>
</tr>
<tr>
<td>PPPU14</td>
<td>1266.5</td>
<td><strong>1274</strong></td>
<td>39869</td>
</tr>
</tbody>
</table>

Previous best known solution: 1278.
Optimal solution M-n200-k16, cost: 1274
Golden, Wasil, Kelly and Chao [1998] proposed 12 CVRP instances, having from 240 to 483 customers.

- Frequent in the heuristic literature
- Considered “out of reach” of exact algorithms

6 instances could be solved, with 240, 252, 300, 320, 360 and 420 customers.
Optimal solution Golden\_20 (420 customers), 7 days CPU time, cost 1817.59; best heuristic 1817.86
The concept of limited memory cuts was pivotal for those improvements.

In order to understand the concept, it is necessary to see how non-robust cuts interfere with the pricing.
The pricing is done by a labeling dynamic programming algorithm.

- $B(i, q), i \in V, q \in \{d_i, \ldots, Q\}$ are buckets
- A label $L(P)$ represents a partial path $P$, with cost $\bar{c}(P)$. All labels corresponding to paths ending in $i$ with load $q$ are kept in bucket $B(i, q)$.
- An initial label representing a null path is put in $B(0, 0)$.
- Labels are expanded producing other labels.
- Dominated labels should be removed along the algorithm.
The Labeling Algorithm

<table>
<thead>
<tr>
<th>q = 0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i = 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Initial label

Buckets
The Labeling Algorithm
The Labeling Algorithm

q = 0  1  2  3  4  5

5
4
3
2
1

i = 0

label expansion
The Labeling Algorithm

$q = 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5$

5
4
3
2
1

$i = 0$

Do both labels need to be kept in bucket (4,4)?
The labels represent partial paths 0-1-4 and 0-3-4
Do both labels need to be kept in bucket (4,4)? Depends on the definition of \( \Omega \)
A label $L(P_1)$ dominates another label $L(P_2)$ in the same bucket if $ar{c}(P_1) \leq \bar{c}(P_2)$ and every valid completion of $P_2$ is also a valid completion for $P_1$.

- $\Omega$ allows cycles: at most one non-dominated label per bucket. Labeling runs in $O(n^2 \cdot Q)$ time (pseudo-polynomial).
- Only elementary routes in $\Omega$: In the worst case, exponential number of non-dominated labels per bucket.
- $ng$-routes: Maximum of $2^{ng-1}$ labels per bucket. $ng = 8$ is a safe choice to avoid combinatorial explosion.
Let $C \subseteq N$ be a set of 3 customers. At most one route that passes by 2 or more customers in $C$ can be used. So, the following 3-SRC is valid:

$$\sum_{r \in \Omega : |r \cap C| \geq 2} \lambda_r \leq 1$$

3-SRCs (and other SRCs) can reduce gaps a lot. However, they can also make the pricing intractable.

Suppose that $m$ 3-SRCs are added, $\sigma_s < 0$ is the dual variable of 3-SRC $s$, $C(s)$ its base set.

Each label needs $m$ additional binary dimensions, $S(P)[s]$ is the parity of the number of times that $P$ visited customers in $C(s)$.

When a label $L(P)$ with $S(P)[s] = 1$ is expanded to a vertex in $C(s)$, new label $L(P')$ has its reduced cost penalized by $\sigma_s$. 
Dominance Rule with 3-SRCs

A label $L(P_1)$ dominates another label $L(P_2)$ in the same bucket if

$$\bar{c}(P_1) \leq \bar{c}(P_2) + \sum_{1 \leq s \leq m : S(P_1)[s] > S(P_2)[s]} \sigma_s$$

and every valid completion of $P_2$ is also a valid completion for $P_1$.

- More 3-SRCs, weaker dominance
- In many instances, 100 3-SRCs are enough for a combinatorial explosion in the number of non-dominated labels
The 3-SRC with base-set $C \subseteq N$ is assigned to a memory-set $M$, $C \subseteq M \subseteq N$. The lm-3-SRC is:

$$\sum_{r \in \Omega} \alpha(C, M, r)\lambda_r \leq 1,$$

(9)

where coefficient of route $r$ is given by:

1: function $\alpha(C, M, r)$
2: $\text{coeff} \leftarrow 0, \text{state} \leftarrow 0$
3: for every vertex $i \in r$ (in order) do
4: \hspace{1em} if $i \notin M$ then
5: \hspace{2em} $\text{state} \leftarrow 0$
6: \hspace{1em} else if $i \in C$ then
7: \hspace{2em} $\text{state} \leftarrow \text{state} + 1/2$
8: \hspace{1em} if $\text{state} \geq 1$ then
9: \hspace{2em} $\text{coeff} \leftarrow \text{coeff} + 1, \text{state} \leftarrow \text{state} - 1$
10: return $\text{coeff}$
Limited Memory 3-Subset Row Cuts (lm-3-SRCs)

1: function $\alpha(C, M, r)$
2: $\text{coeff} \leftarrow 0$, $\text{state} \leftarrow 0$
3: for every vertex $i \in r$ (in order) do
4:   if $i \notin M$ then
5:     $\text{state} \leftarrow 0$
6:   else if $i \in C$ then
7:     $\text{state} \leftarrow \text{state} + 1/2$
8:   if $\text{state} \geq 1$ then
9:     $\text{coeff} \leftarrow \text{coeff} + 1$, $\text{state} \leftarrow \text{state} - 1$
10: return $\text{coeff}$

- If memory-set $M$ is equal to $N$, the lm-3-SRC is equivalent to an 3-SRC. Otherwise, it is weaker.
- However, by adjusting dynamically the memory-sets $M$, it is possible to obtain exactly the same bounds obtainable by regular 3-SRCs.
- If the final $M$ is small, as usually happens, the gains in pricing time are large.
Why limited memory reduces impact in the pricing?

Dominance Rule with lm-3-SRCs

A label $L(P_1)$ dominates another label $L(P_2)$ in the same bucket if

$$\bar{c}(P_1) \leq \bar{c}(P_2) + \sum_{1 \leq s \leq m : S(P_1)[s] > S(P_2)[s]} \sigma_s$$

and every valid completion of $P_2$ is also a valid completion for $P_1$.

Suppose a lm-3-SRC $s$ with memory set $M(s)$ and dual variable $\sigma_s$. The dimension $S(P)[s]$ is set to zero whenever $P$ visits a vertex not in $M(s)$ (so, the previous visits to $C(s)$ are “forgotten”).

- Many more SRCs can be separated before pricing becomes intractable
\(\lambda_{r1}\) has coefficient 1 in the 3-SRC with \(C = \{1, 2, 3\}\)
$\lambda_{r1}$ still has coefficient 1 in the Im 3-SRC
\( \lambda_{r_2} \) has coefficient 1 in the 3-SRC with \( C = \{1, 2, 3\} \).
\( \lambda_{r2} \) still has coefficient 1 in the Im 3-SRC
$\lambda_{r3}$ has coefficient 1 in the 3-SRC with $C = \{1, 2, 3\}$
$\lambda_{r3}$ still has coefficient 1 in the Im 3-SRC
Separation of Im-SRCs

Final memory set

0 2 1 3

Final memory set
The next pricing iteration may generate routes that dodge the memory!
Separation of Im-SRCs

Possibly included in the memory set of C in the next cut round

No problem, memory will be adjusted in the next cut round!
Why it is good to reduce the set $M$ as much as possible?

Solid path may only dominate the dashed path because the Im 3-SRC $\{1, 2, 3\}$ is already forgotten at $i$. 
The advanced state-of-the-art BCPAs are those where the non-robust cuts and the pricing algorithm are jointly and symbiotically designed, in such a way that the pricing can handle a large number of very tailored non-robust cuts without becoming too inefficient.

- The presented **limited memory technique is good for the labeling algorithm**. If the pricing was being solved by another method (for example, by MIP), they would actually make the pricing harder!
MPC Best Paper in 2017

The editorial board of MPC has chosen

**Improved Branch-Cut-and-Price for Capacitated Vehicle Routing**

by Diego Pecin, Artur Pessoa, Marcus Poggi
and Eduardo Uchoa

MPC, volume 9, pp. 61-100, March 2017

This certificate is awarded at the 23rd International Symposium on Mathematical Programming, Bordeaux, France, July 2018
Part II - From CVRP to other classic variants
Facets of the Set Partitioning polyhedron with Chvátal-Gomory rank 1.

**Generalize and strengthen Subset Row Cuts.**


Presents the concept of **memory cut over arcs**

Results on the classic Solomon instances (100 customers)
- All solved, 55/56 at the root node

Results on Gehring-Homberger instances (200 customers)
- 51/60 solved, 27 for the first time

Heterogeneous Fleet VRP (HFVRP)

Presents the concept of memory cut depending on subproblem

- Solves most instances with up to 200 clientes, two times more than previous methods

Capacitated Arc Routing (CARP)

The classic multi-vehicle arc routing

Solves instances two times larger than previous methods.
- 23/24 Eglese instances, 11 for the first time
- 134/135 other instances from the literature

Generic Exact VRPSolver
The difficulty of creating state-of-the-art algorithms for new VRP variants

Those BCP algorithms are very complex:

- Each of the previously mentioned algorithms took a lot of time to be coded, even when they adapt an already existing code for another variant
- There are intricate conceptual issues for adapting some techniques for more complex variants

One would like to have a generic algorithm that could be easily customized to many variants
The SPF is quite generic, by changing the definition of $\Omega$ several classic variants can be modeled. Desaulniers et al. [1998] proposed a framework where many VRPs could be solved by the same BP algorithm. The sets $\Omega$ were associated to the solutions of Resource Constrained Shortest Path (RCSP) Problems.

\[
\min \sum_{r \in \Omega} c_r \lambda_r \quad \quad \quad (10)
\]

\[
\text{S.t.} \quad \sum_{r \in \Omega} a_{ir} \lambda_r = 1, \quad \forall \, i \in N, \quad (11)
\]

\[
\lambda_r \in \{0, 1\} \quad \forall \, r \in \Omega. \quad (12)
\]

A BCP solver for a generic model that encompasses a wide class of VRPs and even some other kinds of problems

Incorporates almost all advanced elements found in the best recent VRP algorithms,

Computational Experiments

- VRPSolver algorithms coded in C++ over BaPCod package (Vanderbeck et al. [2018])
- IBM CPLEX 12.8 used as LP solver
- Models are defined using a Julia–JuMP (Dunning et al. [2017]) based interface.

Tests over 13 problems: CVRP, VRPTW, HFVRP, Multi-Depot VRP (MDVRP), ( Capacitated) Team Orienteering Problem (CTOP/TOP), Capacitated Profitable Tour Problem (CPTP), VRP with Service Level constraints (VRPSL), GAP, Vector Packing Problem (VPP), Bin Packing Problem (BPP) and CARP.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Data set</th>
<th>#</th>
<th>T.L.</th>
<th>VRPSolver</th>
<th>Best Published</th>
<th>2nd Best Published</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVRP</td>
<td>E-M</td>
<td>12</td>
<td>10h</td>
<td>12 (61s)</td>
<td><strong>12 (49s)</strong> Pecin et al. [2017c]</td>
<td>10 (432s) Contardo et al. [2014]</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>58</td>
<td>60h</td>
<td><strong>36 (147m)</strong></td>
<td>34 (209m) Uchoa et al. [2017]</td>
<td></td>
</tr>
<tr>
<td>VRPTW</td>
<td>Sol Hard</td>
<td>14</td>
<td>1h</td>
<td>14 (5m)</td>
<td><strong>13 (17m)</strong> Pecin et al. [2017a]</td>
<td>9 (39m) Baldacci et al. [2011a]</td>
</tr>
<tr>
<td></td>
<td>Hom 200</td>
<td>60</td>
<td>30h</td>
<td><strong>56 (21m)</strong></td>
<td>50 (70m) Pecin et al. [2017a]</td>
<td>7 (-) Kallehauge et al. [2006]</td>
</tr>
<tr>
<td>HFVRP</td>
<td>Golden</td>
<td>40</td>
<td>1h</td>
<td>40 (144s)</td>
<td><strong>39 (287s)</strong> Pessoa et al. [2018]</td>
<td>34 (855s) Baldacci et al. [2009]</td>
</tr>
<tr>
<td>MDVRP</td>
<td>Cordeau</td>
<td>11</td>
<td>1h</td>
<td>11 (6m)</td>
<td><strong>11 (7m)</strong> Pessoa et al. [2018]</td>
<td>9 (25m) Contardo et al. [2014]</td>
</tr>
<tr>
<td>PDPTW</td>
<td>RC</td>
<td>40</td>
<td>1h</td>
<td>40 (5m)</td>
<td><strong>33 (17m)</strong> Gschwind et al. [2018]</td>
<td>32 (14m) Baldacci et al. [2011b]</td>
</tr>
<tr>
<td></td>
<td>LiLim</td>
<td>30</td>
<td>1h</td>
<td>3 (56m)</td>
<td><strong>23 (20m)</strong> Baldacci et al. [2011b]</td>
<td>18 (27m) Gschwind et al. [2018]</td>
</tr>
<tr>
<td>TOP</td>
<td>Chao 4</td>
<td>60</td>
<td>1h</td>
<td><strong>55 (8m)</strong></td>
<td>39 (15m) Bianchessi et al. [2018]</td>
<td>30 (-) El-Hajj et al. [2016]</td>
</tr>
<tr>
<td>CTOP</td>
<td>Archetti</td>
<td>14</td>
<td>1h</td>
<td><strong>13 (7m)</strong></td>
<td>7 (34m) Archetti et al. [2013]</td>
<td>6 (35m) Archetti et al. [2009]</td>
</tr>
<tr>
<td>CPTP</td>
<td>Archetti</td>
<td>28</td>
<td>1h</td>
<td><strong>24 (9m)</strong></td>
<td>0 (1h) Bulhotes et al. [2018b]</td>
<td>0 (1h) Archetti et al. [2013]</td>
</tr>
<tr>
<td>VRPSL</td>
<td>Bulhotes</td>
<td>180</td>
<td>2h</td>
<td><strong>159 (16m)</strong></td>
<td>49 (90m) Bulhotes et al. [2018b]</td>
<td></td>
</tr>
<tr>
<td>GAP</td>
<td>OR-Lib D</td>
<td>6</td>
<td>2h</td>
<td>5 (40m)</td>
<td><strong>5 (30m)</strong> Posta et al. [2012]</td>
<td>5 (46m) Avella et al. [2010]</td>
</tr>
<tr>
<td></td>
<td>Nauss</td>
<td>30</td>
<td>1h</td>
<td><strong>25 (23m)</strong></td>
<td>1 (58m) Gurobi [2017]</td>
<td>0 (1h) Nauss [2003]</td>
</tr>
<tr>
<td>VPP</td>
<td>1,4,5,9</td>
<td>40</td>
<td>1h</td>
<td><strong>38 (8m)</strong></td>
<td>13 (50m) Heßler et al. [2018]</td>
<td>10 (53m) Brandão et al. [2016]</td>
</tr>
<tr>
<td>BPP</td>
<td>Falk T</td>
<td>80</td>
<td>10m</td>
<td>80 (16s)</td>
<td><strong>80 (1s)</strong> Brandão et al. [2016]</td>
<td>80 (24s) Belov et al. [2006,16]</td>
</tr>
<tr>
<td></td>
<td>Hard28</td>
<td>28</td>
<td>10m</td>
<td>28 (17s)</td>
<td><strong>28 (7s)</strong> Belov et al. [2006,16]</td>
<td>26 (14s) Brandão et al. [2016]</td>
</tr>
<tr>
<td>AI</td>
<td>250</td>
<td>1h</td>
<td></td>
<td><strong>160 (25m)</strong></td>
<td>116 (35m) Belov et al. [2006,16]</td>
<td>100 (40m) Brandão et al. [2016]</td>
</tr>
<tr>
<td>ANI</td>
<td>250</td>
<td>1h</td>
<td></td>
<td><strong>103 (35m)</strong></td>
<td>97 (40m) Wei et al. [2019]</td>
<td>67 (45m) Belov et al. [2006,16]</td>
</tr>
<tr>
<td>CARP</td>
<td>Eglese</td>
<td>24</td>
<td>30h</td>
<td><strong>22 (36m)</strong></td>
<td>22 (43m) Pecin et al. [2019]</td>
<td>10 (237m) Bartolini et al. [2013]</td>
</tr>
</tbody>
</table>

**Table:** VRPSolver vs best specific solvers on 13 problems.
VRPSolver on CVRP

Class X with 100 instances, ranging between 100 and 1000 customers:

- Designed to mimic a wide diversity of characteristics found in real applications
- Available at CVRPLIB
  (http://vrp.atd-lab.inf.puc-rio.br/index.php/en/)

53 out of 100 instances could be solved, sometimes with very special parameterization and very long runs (up to one month):

- $100 \leq n < 200 : 22/22$ (100%)
- $200 \leq n < 300: 19/21$ (90%)
- $300 \leq n < 500: 8/25$ (32%)
- $500 \leq n \leq 1000: 4/32$ (12%)

Smallest unsolved: X-n280-k17

Largest solved: X-n856-k95
Optimal solution X-856-k95 (unitary demands), 10 days of CPU time, cost 88,965
A new benchmark of 10,000 instances with 100 customers

- CVRPLIB did not have any small/medium instances with > 25 customers/route. Realizing that limitation (after Amazon Last Mile Routing Challenge!), this recent benchmark included instances with “ultra-long” routes.

- VRPSolver could find all optimal solutions with a median time of 1 minute. Yet, 10.1% of instances could not be solved in 30 minutes using default settings. Solving them required special strategies and/or long runs (up to 80 hours).

- But we also reimplemented the best BC over CPLEX 20

BC for CVRP is not dead! However, BCs are hard to generalize. For example, there is no reasonable BC for VRPTW.
Downloading and using the VRP Solver
The VRP solver is available for academic use (vrpsolver.math.u-bordeaux.fr):

- Algorithms are bundled in a single pre-compiled docker (runs in every OS)
- There is a no-docker version for Linux
- Julia–JuMP user interface for modeling, including several demos

1. Modeling a typical VRP variant requires around 100 lines of Julia code (not counting input/output). An experienced user can build a working solver for a new variant in 1 day
2. Many computer experiments and parameter tuning may be needed for an improved performance
3. In some cases, separation routines for problem-specific (robust) cuts are needed for a better performance
function build_model(data::DataCVRP)
    E = edges(data)
    n = nb_customers(data)
    V = [i for i in 0:n]
    V' = [i for i in 1:n]
    Q = veh_capacity(data)
cvrp = VrpModel()
@variable(cvrp.formulation, x[e in E], Int)
@objective(cvrp.formulation, Min, sum(c(data,e) * x[e] for e in E))
@constraint(cvrp.formulation, deg[i in V'], sum(x[e] for e in δ(data, i)) == 2.0)
function build_graph()
    v_source = v_sink = 0
    G = VrpGraph(cvrp, V, v_source, v_sink, (0, n))
    cap_res_id = add_resource(G, main = true)
    for i in V
        set_resource_bounds(G, i, cap_res_id, 0, Q)
    end
    for (i,j) in E
        arc_id = add_arc(G, i, j, [x[(i,j)]])
        set_arc_consumption(G, arc_id, cap_res_id, d(data, j))
        arc_id = add_arc(G, j, i, [x[(i,j)]])
        set_arc_consumption(G, arc_id, cap_res_id, d(data, i))
    end
    return G
end
G = build_graph()
add_graph(cvrp, G)
set_vertex_packing_sets(cvrp, [[(G,i)] for i in V'])
define_packing_sets_distance_matrix(cvrp, [[dist(data, (i, j)) for j in V'] for i in V'])
add_capacity_cut_separator(cvrp, [((G,i)], d(data, i)) for i in V'), Q)
set_branching_priority(cvrp, "x", 1)
return (cvrp, x)
end
We believe users may find original ways (transformations) of fitting new problems in the proposed model

- Not only VRP variants, possibly also problems from scheduling, network design, etc.

Since VRP solving technology is quite advanced, there is a chance of obtaining better-than-existing-methods performance
Some works that used VRPSolver with creative modeling


Some works that used VRPSolver with creative modeling


VRPSolver is being maintained by a tiny group of people working on spare time

- Documentation is poor, it is quite difficult to understand how to change parameters
- Docker version is stuck in old Julia and JuMP versions

We would like to have a community of users
Conclusions and Perspectives
Impact of Exact VRP algorithms in practice

Historically, exact solvers were rarely used in practical routing

1. Existing algorithms could not solve realistic-sized instances in reasonable times
   - Now many instances of the most classic VRPs with up to 200 customers can be solved
   - More importantly, instances with up to 100 customers can often be solved in a few minutes

2. The real problems seldom correspond exactly to one of the classic variants. Creating a good exact code for a new variant is a hard task
   - Highly customizable codes with state-of-the-art performance are now available

We expect that exact algorithms will be much more used by practitioners, at least for benchmarking their heuristics
Thank you


