Constraint-based solution methods for vehicle routing problems

Willem-Jan van Hoeve
Tepper School of Business, Carnegie Mellon University

Based on joint work with Michela Milano [2002], and Canan Gunes [2009]
Outline

• Introduction and motivation
  ▪ Vehicle routing
  ▪ Constraint Programming

• CP model for TSP with Time Windows
  ▪ Basic model
  ▪ Hybrid CP/LP approach
  ▪ Experimental results

• CP models for vehicle routing
  ▪ Application: Greater Pittsburgh Community Food Bank
  ▪ Exact CP model
  ▪ Constraint-based local search
  ▪ Experimental results

• Conclusions
Vehicle Routing Problems
Basic problem: Traveling Salesman Problem

Find the shortest closed tour that visits each city exactly once.

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.

71,009 cities

http://www.tsp.gatech.edu
## Milestones

<table>
<thead>
<tr>
<th>Year</th>
<th>Research Team</th>
<th>Size of Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1954</td>
<td>G. Dantzig, R. Fulkerson, and S. Johnson</td>
<td>49 cities</td>
</tr>
<tr>
<td>1971</td>
<td>M. Held and R.M. Karp</td>
<td>64 cities</td>
</tr>
<tr>
<td>1975</td>
<td>P.M. Camerini, L. Fratta, and F. Maffioli</td>
<td>67 cities</td>
</tr>
<tr>
<td>1977</td>
<td>M. Grötschel</td>
<td>120 cities</td>
</tr>
<tr>
<td>1980</td>
<td>H. Crowder and M.W. Padberg</td>
<td>318 cities</td>
</tr>
<tr>
<td>1987</td>
<td>M. Padberg and G. Rinaldi</td>
<td>532 cities</td>
</tr>
<tr>
<td>1987</td>
<td>M. Grötschel and O. Holland</td>
<td>666 cities</td>
</tr>
<tr>
<td>1987</td>
<td>M. Padberg and G. Rinaldi</td>
<td>2,392 cities</td>
</tr>
<tr>
<td>1994</td>
<td>D. Applegate, R. Bixby, V. Chvátal, and W. Cook</td>
<td>7,397 cities</td>
</tr>
<tr>
<td>1998</td>
<td>D. Applegate, R. Bixby, V. Chvátal, and W. Cook</td>
<td>13,509 cities</td>
</tr>
<tr>
<td>2001</td>
<td>D. Applegate, R. Bixby, V. Chvátal, and W. Cook</td>
<td>15,112 cities</td>
</tr>
<tr>
<td>2004</td>
<td>D. Applegate, R. Bixby, V. Chvátal, W. Cook, and K. Helsgaun</td>
<td>24,978 cities</td>
</tr>
<tr>
<td>2005</td>
<td>Applegate et al.</td>
<td>85,900 cities</td>
</tr>
</tbody>
</table>

Chip design application for AT&T/Bell Labs, solved to optimality in 136 CPU years (on a 250-node cluster this took around one year)

Current best approach is based on MIP, using specialized Branch & Cut

Applegate, Bixby, Chvátal & Cook
[2007]
China TSP revisited

Tour within 0.024% of optimal [Hung Dinh Nguyen]
TSP with Time Windows

- Each city must be served within its associated time window

Adding time windows makes it much harder than pure TSP
- State of the art can handle ~100 cities optimally, sometimes even more, depending on instance
Vehicle Routing

- Find minimum cost tours from single origin (depot) to multiple destinations, using multiple (capacitated) trucks.

Generally even harder than TSP-TW
- We need to partition set of cities, and solve TSP for each subset
- Many variations (split/unsplit demand, pick-up & delivery, ...)

![Diagram of vehicle routing network with multiple destinations from a single origin (depot).]
Solving Vehicle Problems

Typical characteristics
• Large scale (hundreds to thousands of locations)
• Time windows, precedence constraints, ...
• Capacity constraints, stacking restrictions, ...

Potential benefits of Constraint Programming
• Natural problem representation
• Specific algorithms to handle combinatorial restrictions (resource capacities, time windows, ...)

9
Constraint Programming
Constraint Programming Overview

Constraint Programming is a way of modeling and solving combinatorial optimization problems

- CP combines techniques from artificial intelligence, logic programming, and operations research
- There exist several industrial solvers (e.g., ILOG CP Solver, Eclipse, Xpress-Kalis, Comet), and academic solvers (e.g., Gecode, Choco, Minion)
- Many industrial applications, e.g.,
  - Gate allocation at the Hong Kong airport
  - Container scheduling at Port of Singapore
  - Timetabling of Dutch Railways (INFORMS Edelman-award)
## Comparison with Integer Programming

<table>
<thead>
<tr>
<th>Integer Linear Programming</th>
<th>Constraint Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>(branch-and-bound/branch-and-cut)</td>
<td>very suitable for highly combinatorial problems, e.g., scheduling, timetabling</td>
</tr>
<tr>
<td>systematic search</td>
<td>systematic search</td>
</tr>
<tr>
<td>at each search state, solve continuous relaxation of problem (expensive)</td>
<td>at each search state, reason on individual constraints (cheap)</td>
</tr>
<tr>
<td>add cuts to reduce search space</td>
<td>filter variable domains to reduce search space</td>
</tr>
<tr>
<td>domains are intervals</td>
<td>domains contain holes</td>
</tr>
</tbody>
</table>

very suitable for optimization problems
Modeling examples

• variables range over finite or continuous domain:
  \( v \in \{a,b,c,d\}, \ \text{start} \in \{0,1,2,3,4,5\}, \ z \in [2.18, 4.33], \ S \in [\{b,c\},\{a,b,c,d,e\}] \)

• algebraic expressions:
  \( x^3(y^2 - z) \geq 25 + x^2 \cdot \max(x,y,z) \)

• variables as subscripts:
  \( y = \text{cost}[x] \) (here \( y \) and \( x \) are variables, ‘cost’ is an array of parameters)

• logical relations in which constraints can be mixed:
  \( ((x < y) \text{ OR } (y < z)) \Rightarrow (c = \min(x,y)) \)

• ‘global’ constraints (a.k.a. symbolic constraints):
  \( \text{alldifferent}(x_1,x_2, \ldots, x_n) \)
  \( \text{UnaryResource}( [\text{start}_1,\ldots, \text{start}_n], [\text{duration}_1,\ldots,\text{duration}_n] ) \)
Example:

variables/domains  \( x_1 \in \{1,2\}, \ x_2 \in \{0,1,2,3\}, \ x_3 \in \{2,3\} \)

constraints  
\( x_1 > x_2 \)
\( x_1 + x_2 = x_3 \)
\( \text{alldifferent}(x_1,x_2,x_3) \)
Example:

variables/domains  \( x_1 \in \{1,2\}, x_2 \in \{0,1,2,3\}, x_3 \in \{2,3\}\)

constraints

\( x_1 > x_2 \)

\( x_1 + x_2 = x_3 \)

\text{alldifferent}(x_1,x_2,x_3)\)
Example:

variables/domains  \[ x_1 \in \{1\}, \ x_2 \in \{0,1\}, \ x_3 \in \{2,3\} \]

constraints  
\[ x_1 > x_2 \]
\[ x_1 + x_2 = x_3 \]
\[ \text{alldifferent}(x_1,x_2,x_3) \]
Example:

variables/domains
\[ x_1 \in \{2\}, \ x_2 \in \{0,1\}, \ x_3 \in \{2,3\} \]

constraints
\[ x_1 > x_2 \]
\[ x_1 + x_2 = x_3 \]
\[ \text{alldifferent}(x_1,x_2,x_3) \]
CP Model for TSP-TW
TSP: basic structure

Most CP models use a ‘path’ representation of the TSP:
- Split the depot into two nodes: node 0 and n+1
- Let nodes 1 up to n represent the cities we have to visit
- Task: find Hamiltonian path (from 0 to n+1)

Variables:

\[ \text{next}_i \text{ represents the city to visit after city } i \ (i=0,1,\ldots,n) \]
with domain \{1,\ldots,n+1\}

Constraint:

\[ \text{Path}(\text{next}_0,\ldots,\text{next}_{n+1}) \]

additional redundant constraint: alldifferent(\text{next}_0,\ldots,\text{next}_{n})

[Caseau & Laburthe, 1997], [Pesant et al., 1998], [Focacci et al., 1999, 2002]
**TSP: distances**

Distances are represented by a ‘transition’ function

\[ T_{ij} \] represents the distance between each pair of cities \( i, j \)

**Variables:**

\[ z \] represents total length of the path, with domain \( \{0, UB\} \)

\[ \text{cost}_i \] represents travel time from city \( i \) to next\( _i \)

**Constraints:**

\[ z = \sum_i \text{cost}_i \]

\[ (\text{next}_i = j) \implies (\text{cost}_i = T_{ij}) \]

*Alternative:* embed cost structure in Path constraint (see later)
Each city $i$ has associated time window $[a_i, b_i]$ in which the service must be started
In addition, we assume that each city $i$ has service time $\text{dur}_i$

Variables:
- $\text{start}_i$ represents time at which service starts in city $i$
- $\text{cost}_i$ represents travel time from city $i$ to next$_i$

Constraints:
- $(\text{next}_i = j) \implies (\text{start}_i + \text{dur}_i + \text{cost}_i \leq \text{start}_j)$
- $a_i \leq \text{start}_j \leq b_i$

Note: The non-overlapping constraints can be grouped together in a UnaryResource constraint
From TSP to machine scheduling

- Vehicle corresponds to ‘machine’
- Visiting a city corresponds to ‘activity’

![Diagram showing sequence-dependent set-up times and makespan]

- Sequence-dependent set-up times
  - Executing task j after task i induces set-up time $T_{ij}$ (distance)
- Minimize ‘makespan’
- Activities cannot overlap (*UnaryResource* constraint)
  - Powerful filtering algorithms (e.g., Edge-finding)
Resource constraints

disjunctions versus UnaryResource constraint

Example:

machine must execute three tasks $T_1$, $T_2$, $T_3$
duration of each task is 3 time units

Filtering task: find earliest start time and latest end time for each task

Disjunctions:

compare two tasks at a time

$T_i$ before $T_j$
or
$T_j$ before $T_i$
disjunctions versus *UnaryResource* constraint

Example:

machine must execute three tasks $T_1$, $T_2$, $T_3$
duration of each task is 3 time units

*Filtering task*: find earliest start time and latest end time for each task

![Diagram](image)

*UnaryResource*: compare tasks simultaneously

**filtering**: $T_3$ must start after time 6
Resource constraints

disjunctions versus UnaryResource constraint

Example:

machine must execute three tasks $T_1$, $T_2$, $T_3$
duration of each task is 3 time units

Filtering task: find earliest start time and latest end time for each task

UnaryResource: compare tasks simultaneously

filtering:
$T_2$ must end before time 8
disjunctions versus *UnaryResource* constraint

Example:

machine must execute three tasks $T_1$, $T_2$, $T_3$
duration of each task is 3 time units

*Filtering task:* find earliest start time and latest end time for each task

---

![Task Diagram]

**UnaryResource:**
compare tasks simultaneously

*edge-finder* algorithm computes these bounds in $O(n \log n)$ time for $n$ tasks

[Carlier & Pinson, 1994]

[Vilim, 2004]

Algorithms for sequence-dependent setup times are more involved
A hybrid approach

Replace the Path constraint by an ‘optimization constraint’

\[ \text{WeightedPath}(\text{next, } T, z) \]

This constraint encapsulates a linear programming relaxation, and performs domain filtering based on optimization criteria (e.g., reduced-cost based filtering)

[Focacci et al., 1999, 2002]
Mapping between CP and LP model

\[
\text{next}_i = j \iff y_{ij} = 1 \\
\text{next}_i \neq j \iff y_{ij} = 0
\]

For the TSP, we apply the Assignment Problem relaxation

- Specialized $O(n^3)$ algorithm
- $O(n^2)$ incremental algorithm
- Reduced costs come for free
- Subtour-elimination constraints are added to objective in ‘Lagrangean’ way to strengthen relaxation

\[
\begin{align*}
\min z &= \sum_{j \in V} \sum_{i \in V} c_{ij} y_{ij} \\
\text{s.t.} \\
\sum_{i \in V} y_{ij} &= 1, \forall j \in V \\
\sum_{j \in V} y_{ij} &= 1, \forall i \in V \\
0 \leq y_{ij} \leq 1, &\forall i, j \in V
\end{align*}
\]
Guide search by reduced costs

Idea: apply reduced costs to guide the search and improve bound

- reduced cost represents *marginal cost increase* if variable becomes part of solution
- variable with low reduced cost is ‘more likely’ to be part of optimal solution
- group together promising values and branch on subdomain

**good domain** \( G(\text{next}_i) = \{ j \mid y_{ij} \text{ has reduced cost } \leq U \} \)

**bad domain** \( B(\text{next}_i) = \{ j \mid y_{ij} \text{ has reduced cost } > U \} \)

solve relaxation

\[ \text{next}_i \in G(\text{next}_i) \quad \text{next}_i \in B(\text{next}_i) \]

[Milano & v.H., 2002]
Bound improvement [Milano & v.H., 2002]:

- order all minimum reduced costs corresponding to bad domains: \( r_1, r_2, r_3, \ldots \)
- for all subproblems with discrepancy \( k \), \( LB + \sum_{i=1}^{k} r_i \) is a valid lower bound
- comes ‘for free’ (just order once)
## Computational results

### Traveling Salesman Problem (with Time Windows)

- reduced cost-based search: [Milano & v.H., 2002]

<table>
<thead>
<tr>
<th>instance</th>
<th>plain search</th>
<th>reduced cost-based search</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>backtracks</td>
<td>time (s)</td>
</tr>
<tr>
<td>hk48</td>
<td>15k</td>
<td>12.6</td>
</tr>
<tr>
<td>gr48</td>
<td>25k</td>
<td>21.1</td>
</tr>
<tr>
<td>brazil58</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>rbg034a</td>
<td>13k</td>
<td>55.2</td>
</tr>
<tr>
<td>rbg035a.2</td>
<td>5k</td>
<td>36.8</td>
</tr>
<tr>
<td>rbg042a</td>
<td>19k</td>
<td>149.8</td>
</tr>
<tr>
<td>rbg050a</td>
<td>19k</td>
<td>180.4</td>
</tr>
</tbody>
</table>
### Computational results

<table>
<thead>
<tr>
<th>instance</th>
<th>name</th>
<th>$n$</th>
<th>BS2000 time</th>
<th>AFG2001 time</th>
<th>FLM2002 time</th>
<th>Fails</th>
<th>our method time</th>
<th>Fails</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbgo21.2</td>
<td>21</td>
<td></td>
<td>9.00</td>
<td>0.22</td>
<td>0.2</td>
<td>44</td>
<td>0.08</td>
<td>15</td>
</tr>
<tr>
<td>rbgo21.3</td>
<td>21</td>
<td></td>
<td>9.60</td>
<td>27.15</td>
<td>0.4</td>
<td>107</td>
<td>0.14</td>
<td>80</td>
</tr>
<tr>
<td>rbgo21.4</td>
<td>21</td>
<td></td>
<td>11.52</td>
<td>5.82</td>
<td>0.3</td>
<td>121</td>
<td>0.09</td>
<td>32</td>
</tr>
<tr>
<td>rbgo21.5</td>
<td>21</td>
<td></td>
<td>127.97</td>
<td>6.63</td>
<td>0.2</td>
<td>55</td>
<td>0.12</td>
<td>60</td>
</tr>
<tr>
<td>rbgo21.6</td>
<td>21</td>
<td></td>
<td>161.66</td>
<td>1.38</td>
<td>0.7</td>
<td>318</td>
<td>0.16</td>
<td>50</td>
</tr>
<tr>
<td>rbgo21.7</td>
<td>21</td>
<td></td>
<td>N.A.</td>
<td>4.30</td>
<td>0.6</td>
<td>237</td>
<td>0.21</td>
<td>43</td>
</tr>
<tr>
<td>rbgo21.8</td>
<td>21</td>
<td></td>
<td>N.A.</td>
<td>17.40</td>
<td>0.6</td>
<td>222</td>
<td>0.10</td>
<td>27</td>
</tr>
<tr>
<td>rbgo21.9</td>
<td>21</td>
<td></td>
<td>N.A.</td>
<td>26.12</td>
<td>0.8</td>
<td>310</td>
<td>0.11</td>
<td>28</td>
</tr>
<tr>
<td>rbgo34a</td>
<td>36</td>
<td></td>
<td>18.03</td>
<td>0.98</td>
<td>55.2</td>
<td>13k</td>
<td>0.93</td>
<td>36</td>
</tr>
<tr>
<td>rbgo35a.2</td>
<td>37</td>
<td></td>
<td>N.A.</td>
<td>64.80</td>
<td>36.8</td>
<td>5k</td>
<td>8.18</td>
<td>4k</td>
</tr>
<tr>
<td>rbgo35a</td>
<td>37</td>
<td></td>
<td>7.67</td>
<td>1.83</td>
<td>3.5</td>
<td>841</td>
<td>0.83</td>
<td>56</td>
</tr>
<tr>
<td>rbgo38a</td>
<td>40</td>
<td></td>
<td>8.64</td>
<td>4232.23</td>
<td>0.2</td>
<td>49</td>
<td>0.36</td>
<td>3</td>
</tr>
<tr>
<td>rbgo40a</td>
<td>42</td>
<td></td>
<td>20.08</td>
<td>751.82</td>
<td>738.1</td>
<td>136k</td>
<td>1200.62</td>
<td>387k</td>
</tr>
<tr>
<td>rbgo41a</td>
<td>43</td>
<td></td>
<td>24.57</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td></td>
</tr>
<tr>
<td>rbgo42a</td>
<td>44</td>
<td></td>
<td>47.38</td>
<td>N.A.</td>
<td>149.8</td>
<td>19k</td>
<td>70.71</td>
<td>24k</td>
</tr>
<tr>
<td>rbgo50a</td>
<td>52</td>
<td></td>
<td>N.A.</td>
<td>18.62</td>
<td>180.4</td>
<td>19k</td>
<td>4.21</td>
<td>1.5k</td>
</tr>
<tr>
<td>rbgo67a</td>
<td>69</td>
<td></td>
<td>29.14</td>
<td>5.95</td>
<td>4.0</td>
<td>493</td>
<td>25.69</td>
<td>128</td>
</tr>
<tr>
<td>rbgo152</td>
<td>152</td>
<td></td>
<td>37.90</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td></td>
</tr>
</tbody>
</table>
Summary on TSP-TW

Benefits of CP model

- Natural problem representation
- When time windows (and other side constraints) are present, CP can be very effective
  - e.g., powerful scheduling algorithms for UnaryResource constraint
- Apply ‘optimization constraint’ to capture and exploit LP relaxation
  - reduced cost-based filtering
  - guide the search, improve bound using LDS

Comparison to other exact approaches

- No clear winner for TSP-TW; specific to problem instance
- But CP is certainly among state of the art
Application: Pittsburgh Food Bank
Food Banks

- A food bank is a non-profit organization that collects and distributes food to needy people through agencies
- More than 200 Food Banks in the U.S.
Our focus: Three Rivers Table Program

- Collect excess food from restaurants, supermarkets,...
- Distribute to agencies (e.g., soup kitchens, shelters), for same-day consumption

Goal: minimize total route length
VRP with side constraints

Food Bank problem combines three sub-problems
• Partition the locations into subsets to be served by the trucks
• For each partition, solve an optimal TSP
• For each partition, locations must be ordered such that truck capacity is not exceeded nor ‘deceeded’

Other aspects
• Some locations must be served within time window (few)
• Trucks can operate maximum 8 hours per day
• Three trucks available per day
• Demand and supply is not splittable
• Problem size: 130 locations

We wish to find optimal weekly schedule
Literature review

General Pickup and Delivery Problems

Transportation from/to depot
- TSP-TW
- VRP-TW

Transportation between customers
- Paired
- Unpaired
  - 1-PDTSP
  - 1-PDVRP
Related work

1-PDTSP (one commodity)

- Hernandez-Perez & Salazar-Gonzalez [2004]
  - No time windows
  - Branch-and-cut algorithm to solve instances with 40 customers
  - Two heuristic approaches that can handle instances up to 500 customers

- Hernandez-Perez & Salazar-Gonzalez [2007]
  - Branch-and-cut algorithm improved with new inequalities that can solve instances up to 100 customers

- Hernandez-Perez et al. [2008]
  - Hybrid algorithm that combines GRASP and VND metaheuristics

1-PDVRP (one commodity)

- Dror, Fortin, & Roucairol [1998]
  - Different approaches (MIP, CP, LS) are applied to 9 locations, with splittable supply and demand
Our approach

Exact methods

- Apply MIP and CP solvers
- What is the maximum problem size that can be solved optimally?

For MIP, we implemented a flow-based model, and a delayed column-generation procedure. The MIP approach was only able to find solutions to very small problem instances. Therefore we omit the MIP results in this talk.

Heuristic methods

- Apply Constraint-Based Local Search
- How close to optimality can we get?
- Can we improve the current schedule?

[Gunes & v.H., 2009]
Model depends on CP Solver that is applied

- Most CP solvers (e.g., ILOG Solver 6.6, Comet, Gecode) have special semantics for scheduling problems, such as activities and resources
- ILOG CP Optimizer (replaces ILOG Solver 6.6) no longer contains these semantics; instead ‘interval variables’ are used

In our work, we applied both ILOG Solver 6.6 and CP Optimizer, but we present here the ‘classical’ CP model
Model is similar to TSP-TW

- Vehicles are alternative resources
  - Type 1: UnaryResource to model time constraints (i.e., non-overlap)
  - Type 2: Reservoir to model capacity w.r.t. pickup and delivery
- Visiting a location is an activity
  - Each activity has start variable, end variable, and fixed duration
  - Each activity can deplete or replenish a reservoir
- Distances are modeled as ‘transition times’ between activities

In this way, the problem can be viewed as a scheduling problem on multiple machines with sequence-dependent setup times (where we want to minimize the makespan)
IloReservoir truckReservoir(ReservoirCapacity, 0);
truckReservoir.setLevelMax(0, TimeHorizon, ReservoirCapacity);

IloUnaryResource truckTime();
IloTransitionTime T(truckTime, Distances);

vector<IloActivity> visit;
visit = vector<IloActivity>(N);

for (int i=0; i<N; i++) {
    visit[i].requires(truckTime);
    if (supply[i] > 0)
        visit[i].produces(truckReservoir, supply[i]);
    else
        visit[i].consumes(truckReservoir, -1*supply[i]);
}
Constraint-Based Local Search

- Use ‘constraint programming’ model to formulate the problem
- Apply built-in neighborhoods and penalty functions to define Local Search algorithm
  - typically based on variable and constraint semantics
  - library is extendible to define own neighborhoods/functions
- In principle, model could be solved either by CP, or LS
  - in practice, this is not always feasible, because different variable/constraint types may be used for CP and LS

ILOG Dispatcher (part of ILOG Solver 6.6) is a library that applies constraint-based local search specifically to vehicle routing problems
**CP model in Dispatcher**

- **Nodes**
  - coordinates of the locations

- **Vehicles**
  - dimensions: time, distance, and weight (load)
  - *UnaryResource* constraint w.r.t. time (automatically defined)
  - ‘Capacity’ constraint w.r.t. load (automatically defined)

- **Visits**
  - location
  - quantity picked up (+) or delivered (-)
  - time window
  - other (problem-specific) constraints

Note: Dispatcher uses Euclidean distances (computed from coordinates). We convert the solutions back to our exact distances when comparing to CP.
class RoutingModel {

    ... 
    IloDimension2 _time;
    IloDimension2 _distance;
    IloDimension1 _weight;
    ...

}

IloNode node(<read coordinates from file>);

IloVisit visit(node);
visit.getTransitVar(_weight) == Supply);
minTime <= visit.getCumulVar(_time) <= maxTime;
visit.getCumulVar(_weight) >= 0);

IloVehicle vehicle(firstNode, lastNode);
vehicle.setCapacity(_weight, Capacity);
vehicle.setCost(_distance);
Two-Phase solution approach

• First phase: Generate a feasible solution, using either one of
  ▪ Savings heuristic
  ▪ Sweep heuristic
  ▪ Nearest-to-depot heuristic
  ▪ Nearest addition heuristic
  ▪ Insertion heuristic
  ▪ Enumeration heuristic

• Second phase: Improve the first solution using local search methods
  ▪ IloTwoOpt, IloOrOpt, IloRelocate, IloCross and IloExchange neighborhoods
  ▪ We apply all these local search methods in sequence and within each local search method we take the first legal cost-decreasing move encountered

Of course, we can also start from current schedule
IloTwoOpt: two arcs in a route are cut and reconnected

IloOrOpt: segments of visits in the same route are relocated
IloRelocate: a visit is inserted in another route

IloCross: the ends of two routes are exchanged

IloExchange: two visits of two different routes swap places
Experimental Results

- Supply data: we have detailed information for each location
- Demand data: precise amount is unknown
  - We approximate the demand based on the population served (known), scaled by the total supply
- Distances: ‘exact’ (Google Maps / MS Mappoint)
  - We assume 15 minutes processing time per location

Small instances

- Subset of food bank problem, e.g., one day of current schedule
- Number of trucks depends on the number of locations. Typically, we can serve up to 15~20 locations per truck.

Larger instances

- Consider multiple days simultaneously (entire week contains 130 locations)
Small instances - Exact versus heuristic

- Small instances, *single* vehicle
- Reported are cost savings with respect to current schedule

<table>
<thead>
<tr>
<th>Number of locations</th>
<th>Exact CP (Scheduler)</th>
<th>Heuristic CBLS (Dispatcher)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>12%</td>
<td>12%</td>
</tr>
<tr>
<td>14</td>
<td>15%</td>
<td>14%</td>
</tr>
<tr>
<td>15</td>
<td>7%</td>
<td>6%</td>
</tr>
<tr>
<td>16</td>
<td>5%</td>
<td>3%</td>
</tr>
<tr>
<td>18</td>
<td>16%</td>
<td>15%</td>
</tr>
</tbody>
</table>

- ILOG Scheduler can solve these instances optimally, within several minutes.
- ILOG Dispatcher finds solutions close to optimality within one second
Larger instances

- Multiple trucks, several days (up to entire week)
- Reported are cost savings with respect to current schedule

<table>
<thead>
<tr>
<th>Number of locations</th>
<th>Number of trucks</th>
<th>Exact CP (Scheduler)</th>
<th>Heuristic CBLS (Dispatcher)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>2</td>
<td>-</td>
<td>4%</td>
</tr>
<tr>
<td>60</td>
<td>4</td>
<td>-</td>
<td>8%</td>
</tr>
<tr>
<td>130</td>
<td>9</td>
<td>-</td>
<td>10%</td>
</tr>
</tbody>
</table>

- Scheduler is not able to find even a feasible solution to problems with more than 20 locations, and 2 trucks
- Dispatcher finds a solution with 10% cost savings for the entire week within one second
- Recent experiments indicate that CP Optimizer (using advanced search) can find good solutions to large problems. For an instance on 62 locations and 4 trucks it found a solution with 16% cost savings.
Summary for VRP/Food Bank

Benefit of CP model

- Natural problem representation, comes with built-in objects for these problem types
- For Local Search: Can add other constraints without changing the search procedures

Computational comparison

- Constraint Programming can be applied to optimally solve small to medium-sized 1-PDVRPs of this kind
  - potential improvements: more advanced search strategies; hybrid MIP/CP approaches
- (Constraint-Based) Local Search provides solutions of good quality very quickly for large-scale problems
Conclusion

• For pure TSP, state of the art can handle thousands of locations optimally
• When side constraints are added (such as time windows), state of the art can only handle up to 100 locations optimally
• VRPs (with side constraints) can be even harder

Benefits of CP models for TSP-TW, VRP, and variants
• Natural problem description
• Powerful algorithms for combinatorial constraints
• Competitive approach (state of the art in some cases)
  ▪ Yet, method of choice highly depends on problem characteristics!
  ▪ Mixed optimization/scheduling problem