EWO Seminar - November 17, 2009

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

Carnegie Mellon Tepper SCHOOL OF BUSINESS

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

71,009 cities

http://www.tsp.gatech.edu

Milestones

Carnegie Mellon Tepper SCHOOL OF BUSINESS

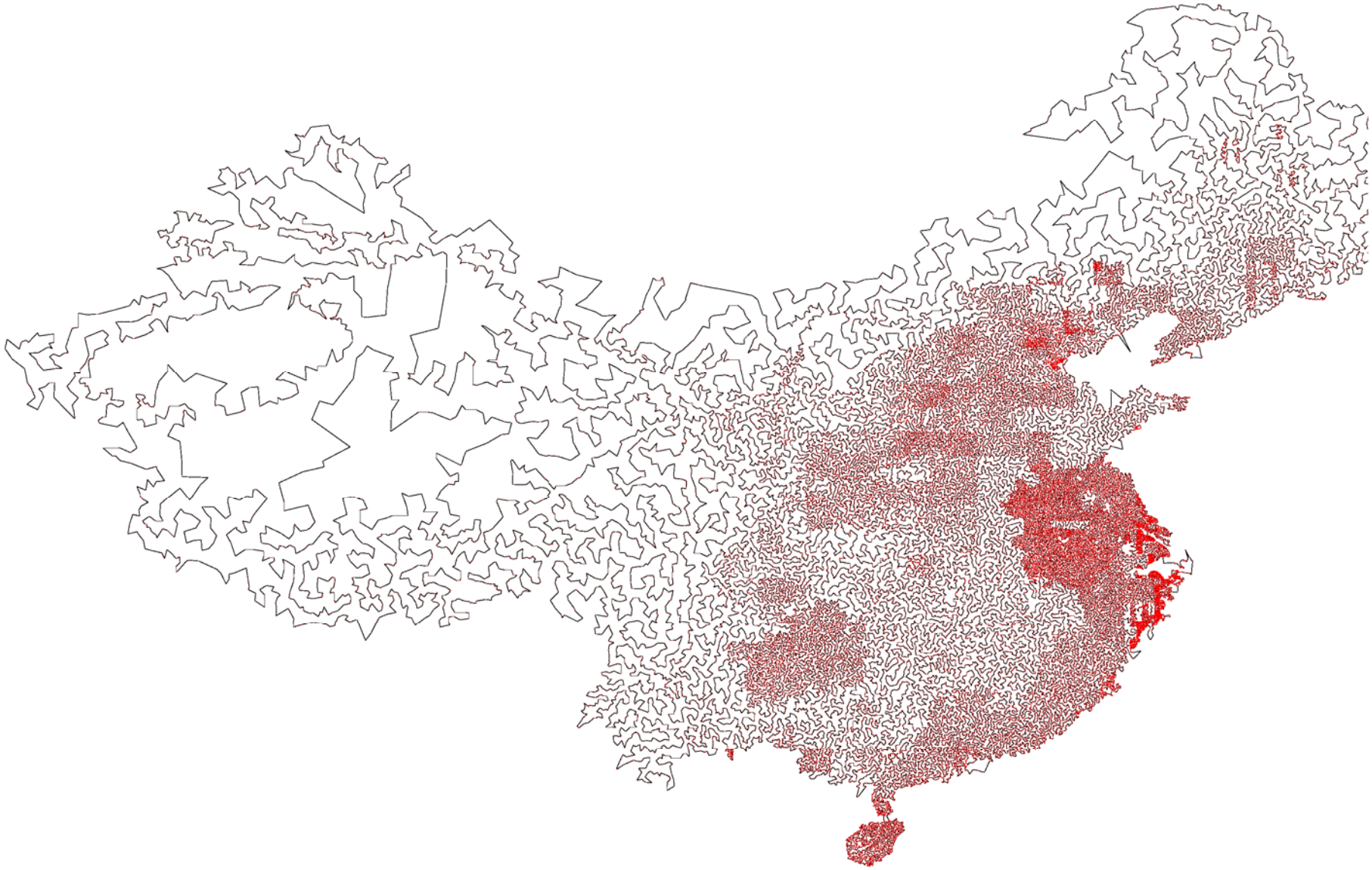
Year	Research Team	Size of Instance
1954	G. Dantzig, R. Fulkerson, and S. Johnson	49 cities
1971	M. Held and R.M. Karp	64 cities
1975	P.M. Camerini, L. Fratta, and F. Maffioli	67 cities
1977	M. Grötschel	120 cities
1980	H. Crowder and M.W. Padberg	318 cities
1987	M. Padberg and G. Rinaldi	532 cities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

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]

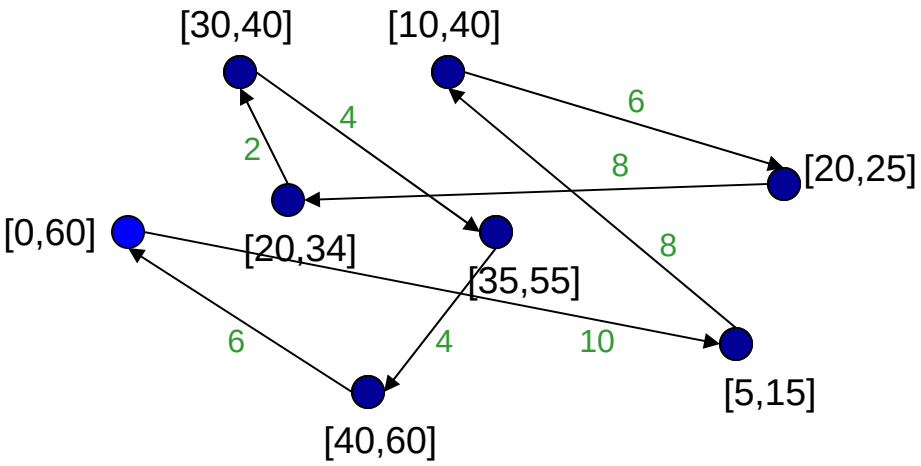
China TSP revisited



Tour within 0.024% of optimal [Hung Dinh Nguyen]

TSP with Time Windows

- Each city must be served within its associated time window

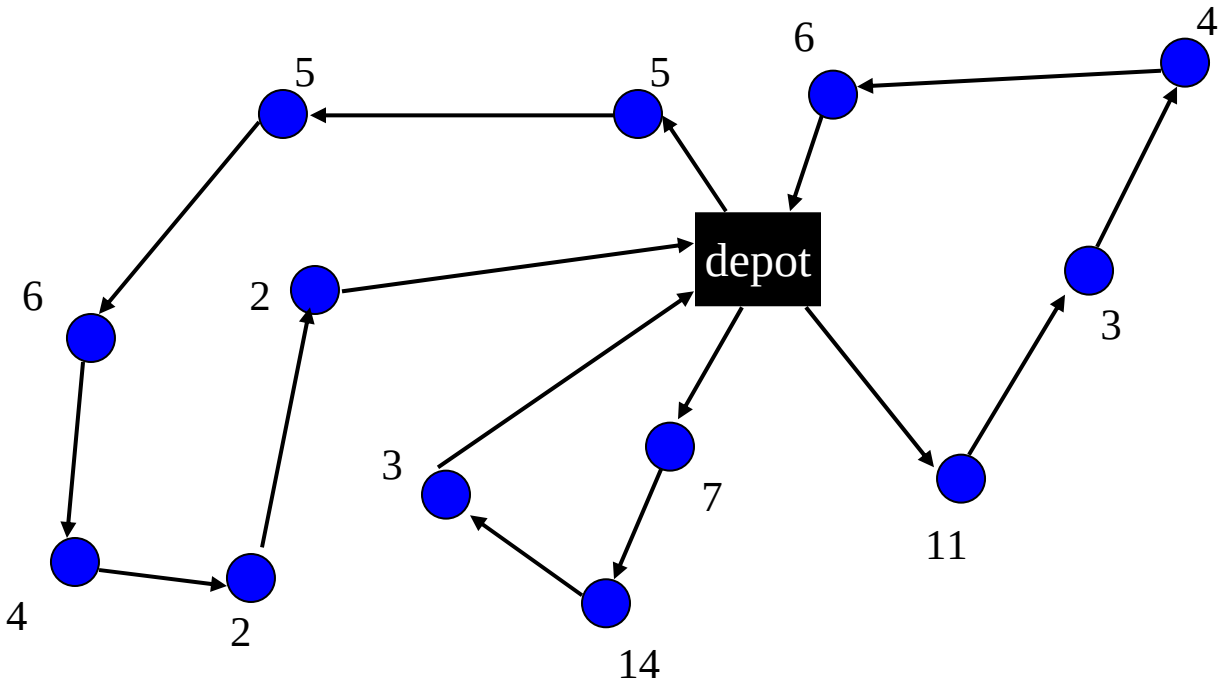


Adding time windows makes it *much* harder than pure TSP

- 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, ...)

Solving Vehicle Problems

Carnegie Mellon Tepper SCHOOL OF BUSINESS

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, ...)

Constraint Programming

Constraint Programming Overview

Carnegie Mellon Tepper SCHOOL OF BUSINESS

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

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

Modeling examples

- variables range over finite or continuous domain:

  $v \in \{a,b,c,d\}$, $start \in \{0,1,2,3,4,5\}$, $z \in [2.18, 4.33]$, $S \in [\ \{b,c\}, \{a,b,c,d,e\}\ ]$

- algebraic expressions:

  $x^3(y^2 - z) \geq 25 + x^2 \bullet \max(x,y,z)$

- variables as subscripts:

  y = 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):

  alldifferent$(x_1,x_2, ...,x_n)$

  UnaryResource$(\ [start_1,..., start_n], [duration_1,...,duration_n]\ )$

Example:

variables/domains	$x_1 \in \{1,2\}$, $x_2 \in \{0,1,2,3\}$, $x_3 \in \{2,3\}$
constraints	$x_1 > x_2$
	$x_1 + x_2 = x_3$
	alldifferent(x_1,x_2,x_3)

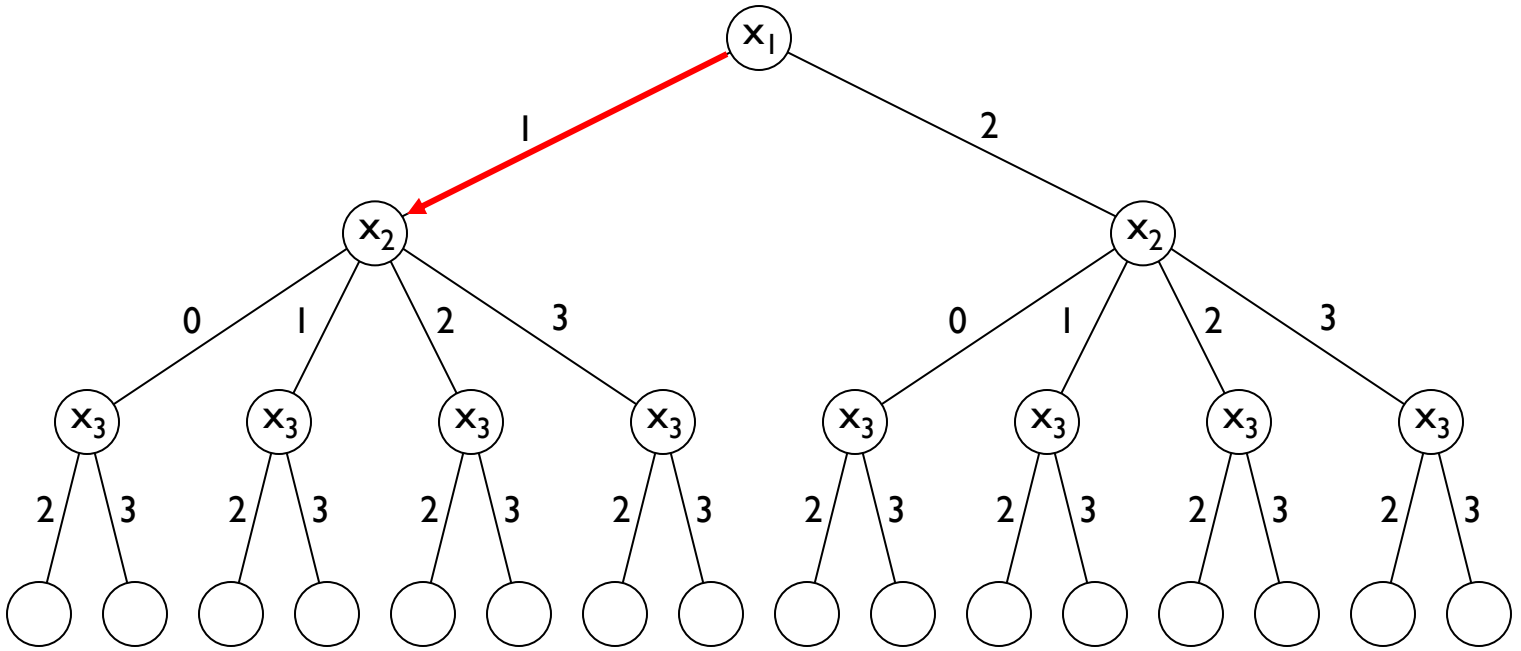
Example:

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

constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1,x_2,x_3)



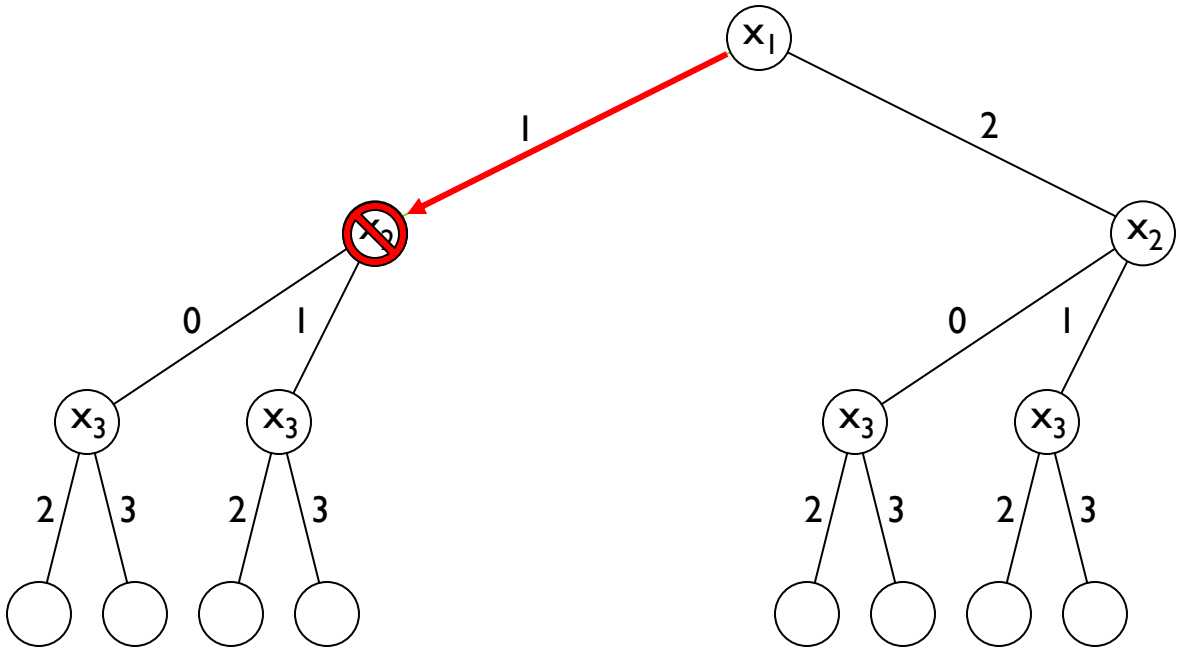
Example:

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

constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1,x_2,x_3)



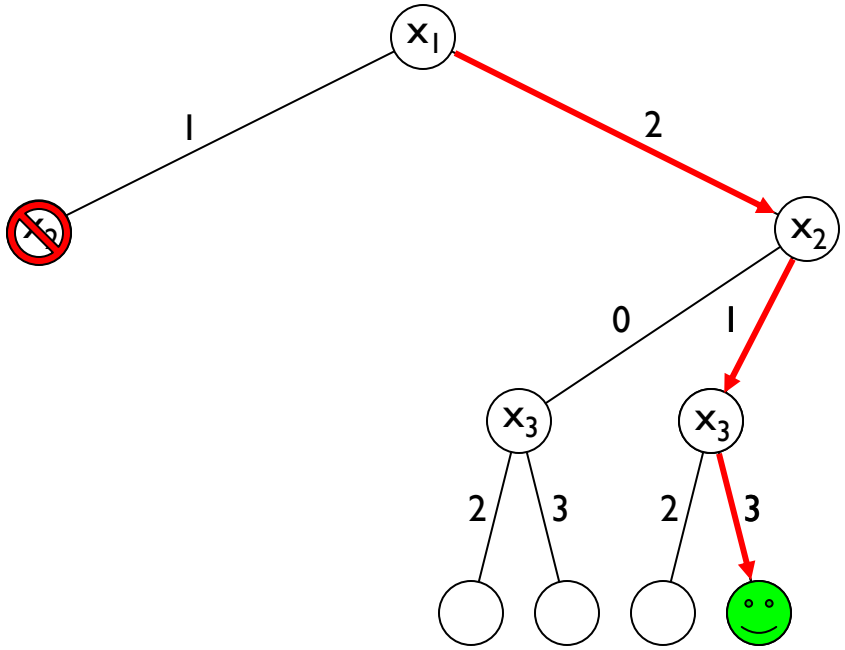
Example:

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

constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

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:

$next_i$ represents the city to visit after city i (i=0,1,...,n)

with domain {1,...,n+1}

Constraint:

Path($next_0$,...,$next_{n+1}$)

additional redundant constraint: alldifferent($next_0$,...,$next_n$)

[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}

$cost_i$ represents travel time from city i to $next_i$

Constraints:

$z = \sum_i cost_i$

$(next_i = j) \Rightarrow (cost_i = T_{ij})$

Alternative: embed cost structure in *Path* constraint (see later)

TSP with Time Windows

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 dur_i

Variables:

$start_i$ represents time at which service starts in city i

$cost_i$ represents travel time from city i to $next_i$

Constraints:

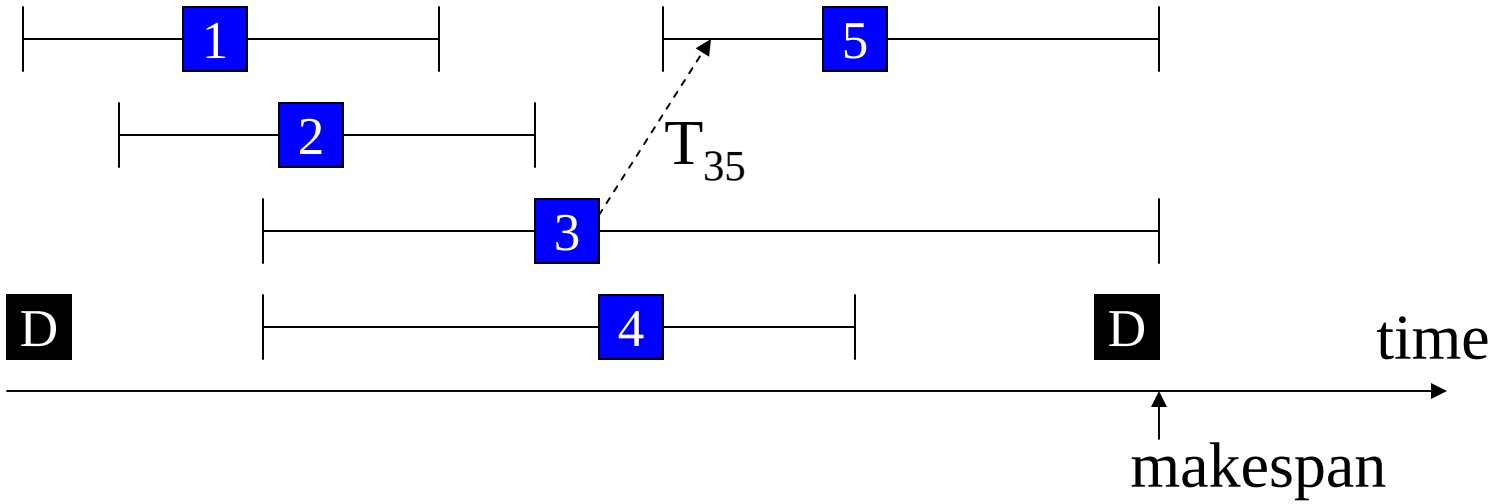
$(next_i = j) \Rightarrow (start_i + dur_i + cost_i \leq start_j)$

$a_i \leq start_j \leq b_i$

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’



- 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)

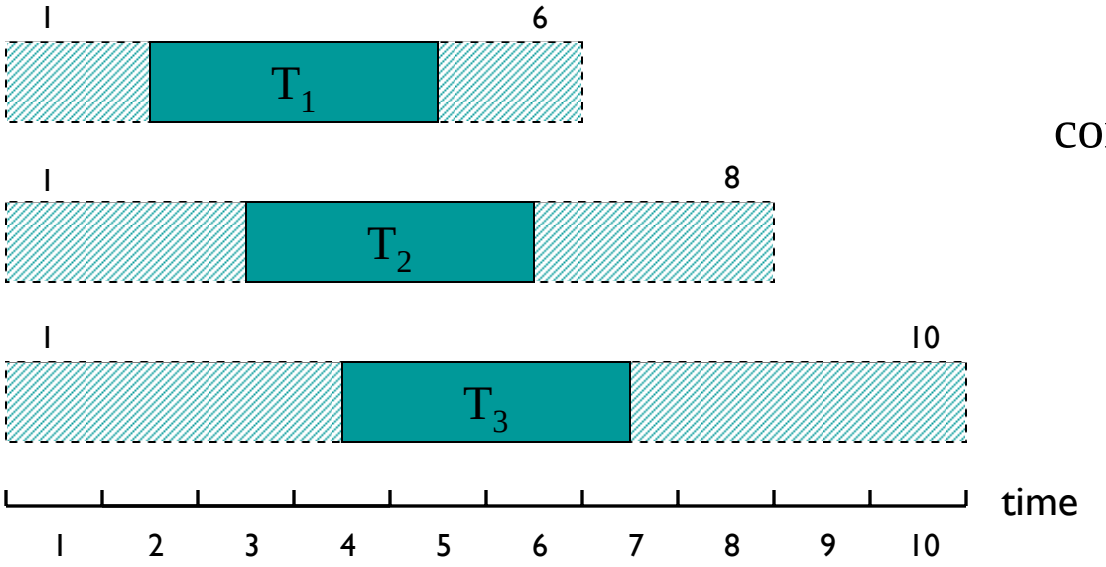
disjunctions versus *UnaryResource* constraint

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

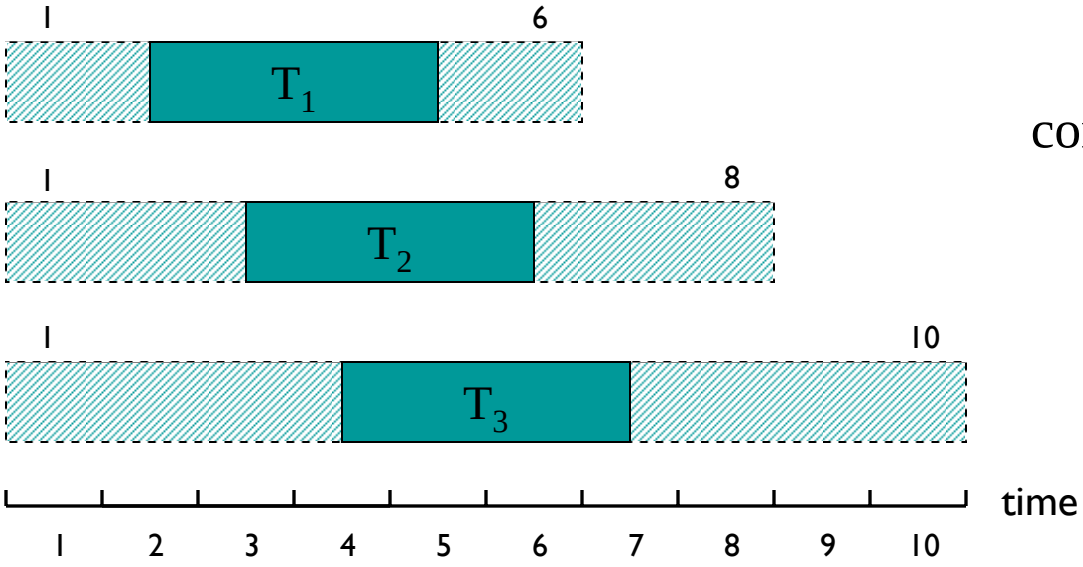
disjunctions versus *UnaryResource* constraint

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



UnaryResource:

compare tasks simultaneously

filtering:

T_3 must start after time 6

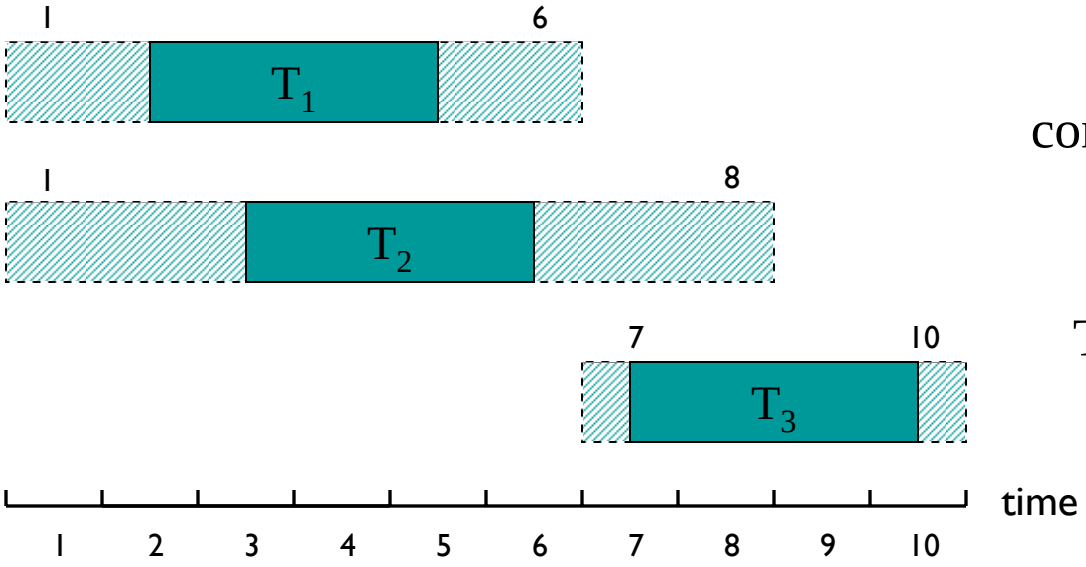
disjunctions versus *UnaryResource* constraint

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

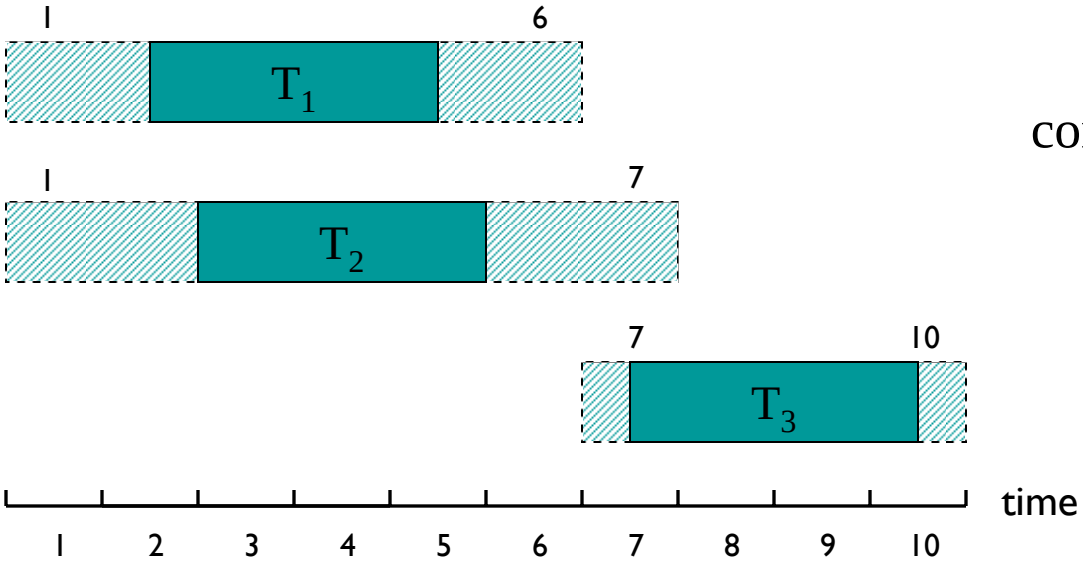
disjunctions versus *UnaryResource* constraint

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



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'

WeightedPath(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]

Assignment Problem as LP relaxation

Mapping between CP and LP model

$\text{next}_i = j \quad \Leftrightarrow \quad y_{ij} = 1$

$\text{next}_i \neq j \quad \Leftrightarrow \quad y_{ij} = 0$

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 'Lagrangian' way to strenghten relaxation

$$\min z = \sum_{j \in V} \sum_{i \in V} c_{ij} y_{ij}$$

$$\text{s.t.} \quad \sum_{i \in V} y_{ij} = 1, \forall j \in V$$

$$\sum_{j \in V} y_{ij} = 1, \forall i \in V$$

$$0 \leq y_{ij} \leq 1, \forall i, j \in V$$

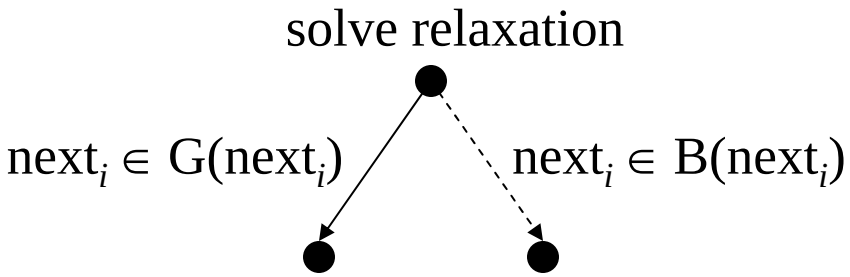
Guide search by reduced costs

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

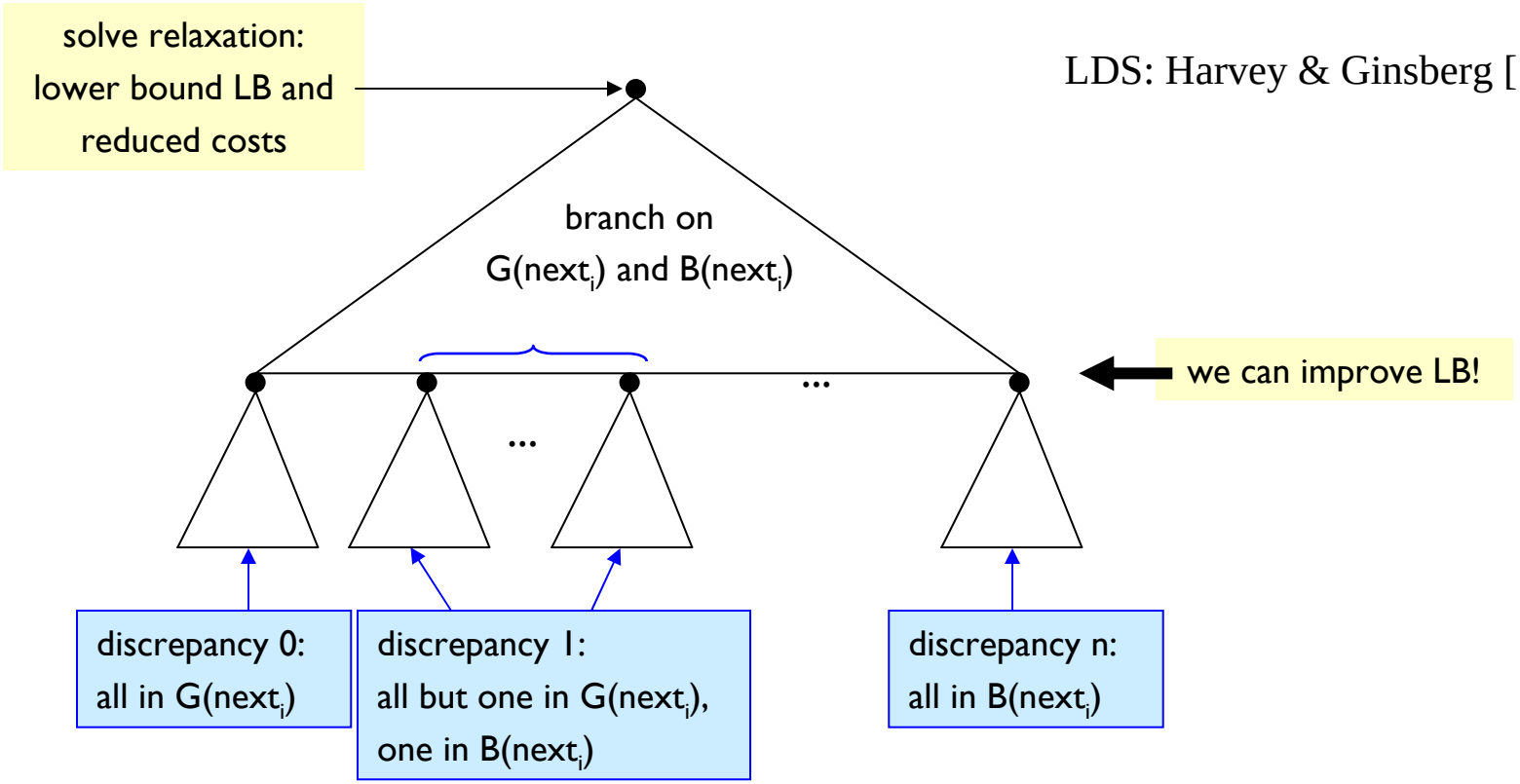
good domain $G(\text{next}_i) = \{ j \mid y_{ij}$ has reduced cost $\leq U \}$

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



[Milano & v.H., 2002]

Apply Limited Discrepancy Search



LDS: Harvey & Ginsberg [1995]

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

- order all minimum reduced costs corresponding to bad domains: r_1, r_2, r_3, ...
- 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)

- 'plain' search: hybrid optimization by [Focacci, Lodi, & Milano, 1999, 2002]
- reduced cost-based search: [Milano & v.H., 2002]

	plain search		**reduced cost-based search**	
instance	backtracks	time (s)	backtracks	time (s)
hk48	15k	12.6	300	1.9
gr48	25k	21.1	22k	19.1
brazil58	n/a	n/a	156k	81.1
rbg034a	13k	55.2	36	0.9
rbg035a.2	5k	36.8	4k	8.2
rbg042a	19k	149.8	24k	70.7
rbg050a	19k	180.4	1k	4.2

Computational results

		Dyn.Prog.	Branch&Cut	CP+LP			
instance		**BS2000**	**AFG2001**	**FLM2002**		**our method**	
name	n	time	time	time	fails	time	fails
rbg021.2	21	9.00	0.22	0.2	44	0.08	15
rbg021.3	21	9.60	27.15	0.4	107	0.14	80
rbg021.4	21	11.52	5.82	0.3	121	0.09	32
rbg021.5	21	127.97	6.63	0.2	55	0.12	60
rbg021.6	21	161.66	1.38	0.7	318	0.16	50
rbg021.7	21	N.A.	4.30	0.6	237	0.21	43
rbg021.8	21	N.A.	17.40	0.6	222	0.10	27
rbg021.9	21	N.A.	26.12	0.8	310	0.11	28
rbg034a	36	18.03	0.98	55.2	13k	0.93	36
rbg035a.2	37	N.A.	64.80	36.8	5k	8.18	4k
rbg035a	37	7.67	1.83	3.5	841	0.83	56
rbg038a	40	8.64	4232.23	0.2	49	0.36	3
rgb040a	42	20.08	751.82	738.1	136k	1200.62	387k
rbg041a	43	24.57	N.A.	N.A.		N.A.	
rbg042a	44	47.38	N.A.	149.8	19k	70.71	24k
rbg050a	52	N.A.	18.62	180.4	19k	4.21	1.5k
rbg067a	69	29.14	5.95	4.0	493	25.69	128
rbg152	152	37.90	N.A.	N.A.		N.A.	

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

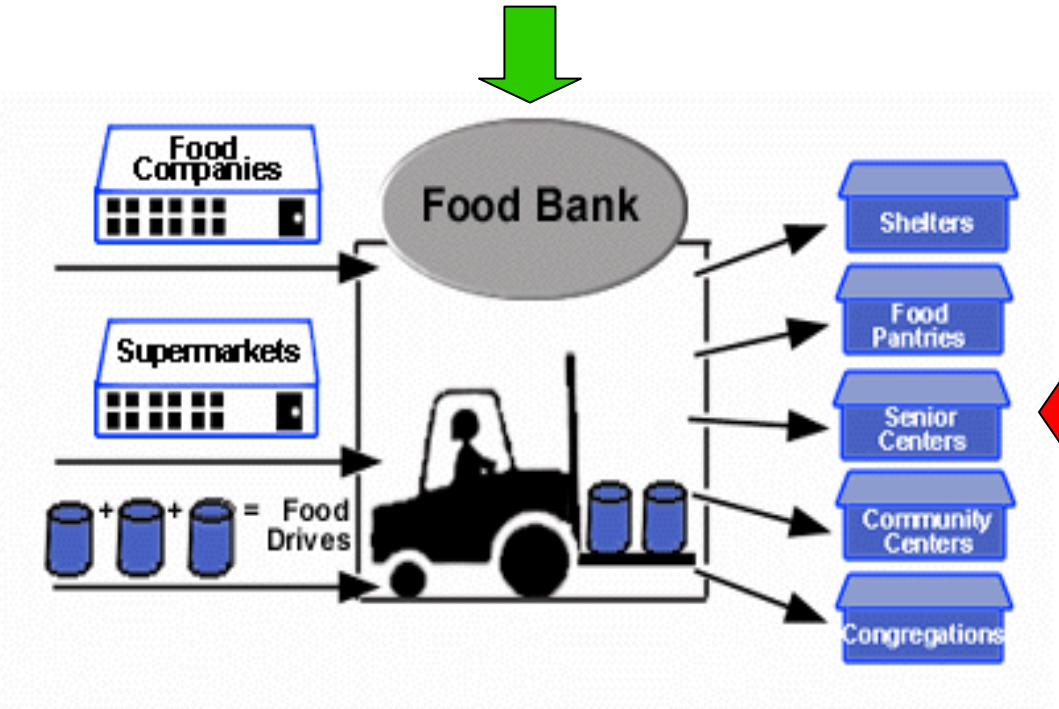
Food Banks

- 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.

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

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

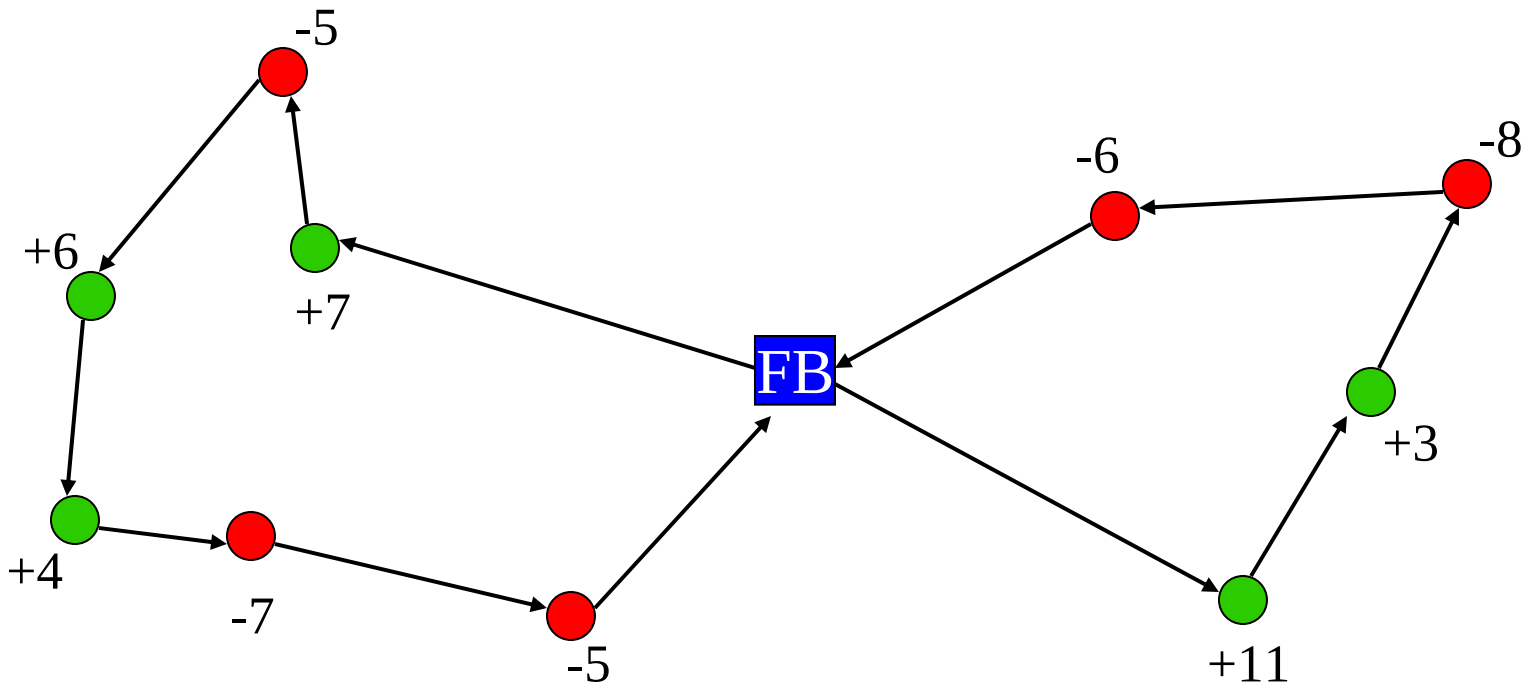
Pittsburgh Food Bank warehouse



Greater Pittsburgh Community Food Bank

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

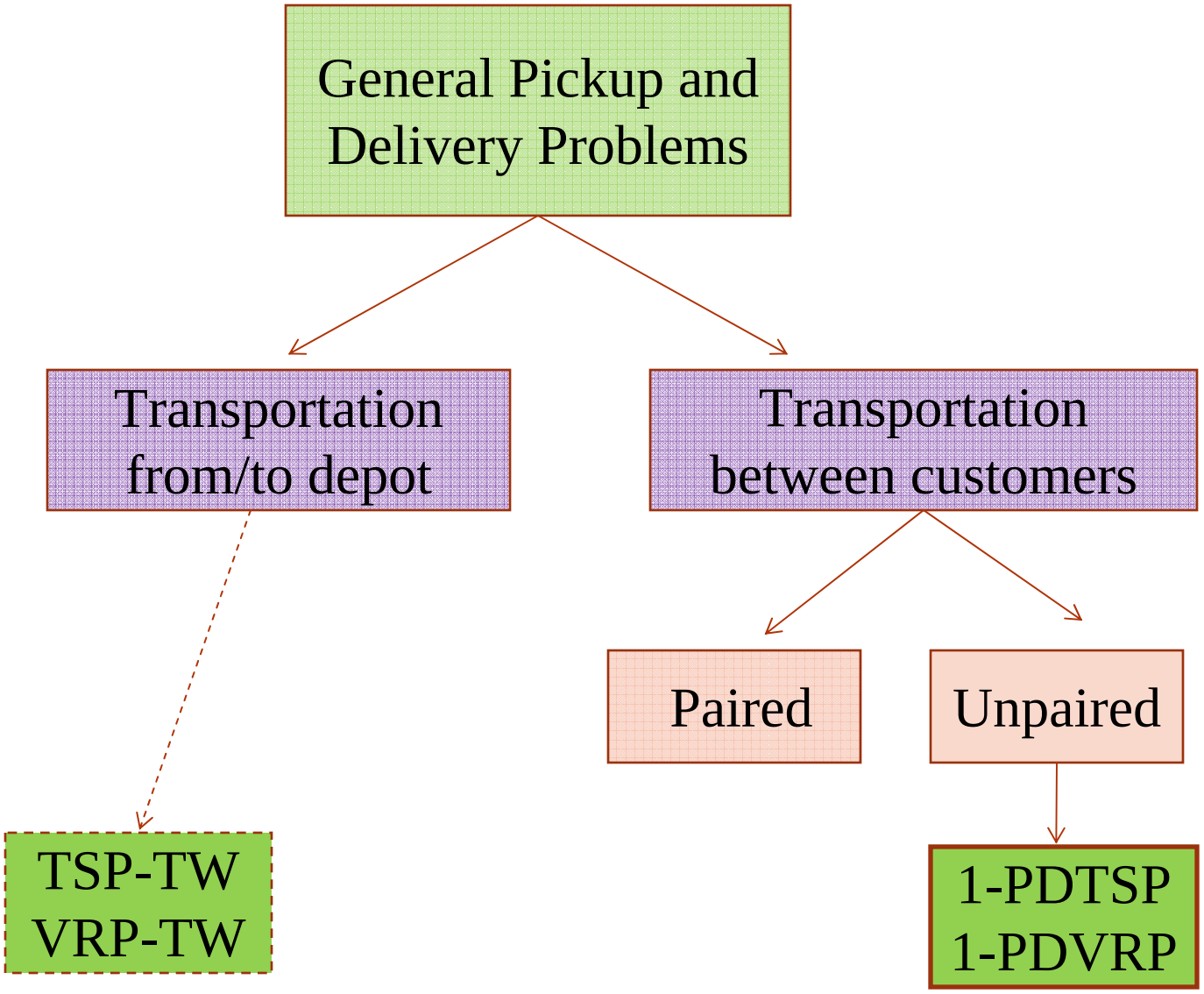
- Partition the locations into subsets to be served by the trucks
- For each partition, solve an optimal TSP
- For each partition, locations must be ordered such that truck capacity is not exceeded nor 'deceeded'

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



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]

Constraint Programming Model

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

Constraint Programming Model (cont'd)

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)

CP Model: Some more details (single truck)

```
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.

Dispatcher Model: Some more details

```
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

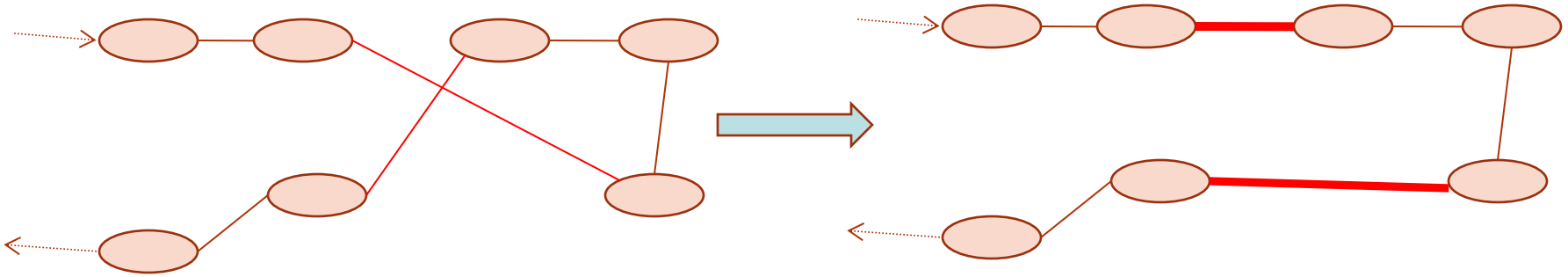
- First phase: Generate a feasible solution, using either one of
  - Savings heuristic
  - Sweep heuristic
  - Nearest-to-depot heuristic
  - Nearest addition heuristic
  - Insertion heuristic
  - Enumeration heuristic

Of course, we can also start from current schedule

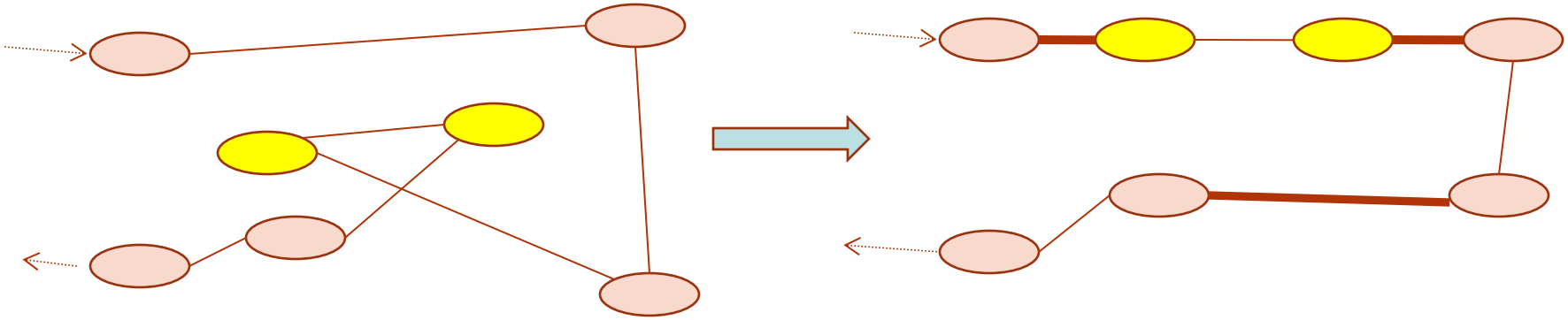
- Second phase: Improve the first solution using local search methods
  - IloTwoOpt, IloOrOpt, IloRelocate, IloCross and IloExchange neighborhoods
  - We apply all these local search methods in sequence and within each local search method we take the first legal cost-decreasing move encountered

Local Search Methods - IntraRoute Neighborhoods

IloTwoOpt: two arcs in a route are cut and reconnected

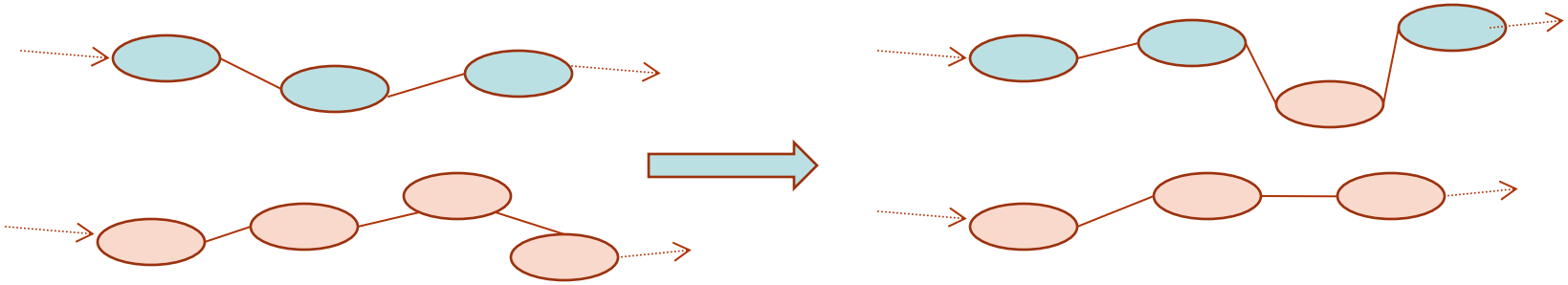


IloOrOpt: segments of visits in the same route are relocated

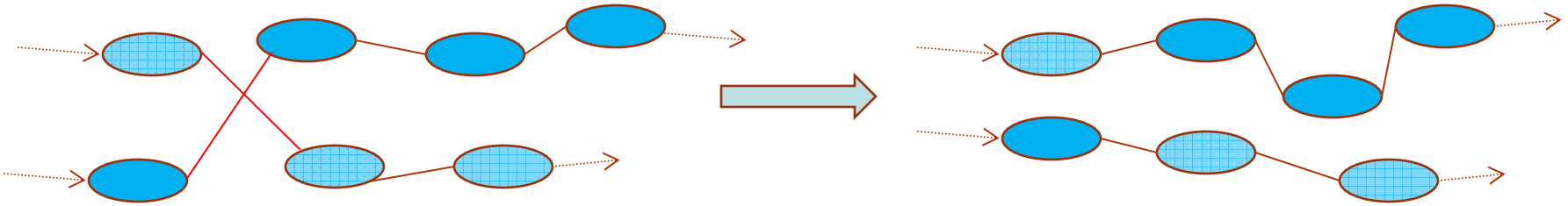


Local Search Methods - InterRoute Neighborhoods

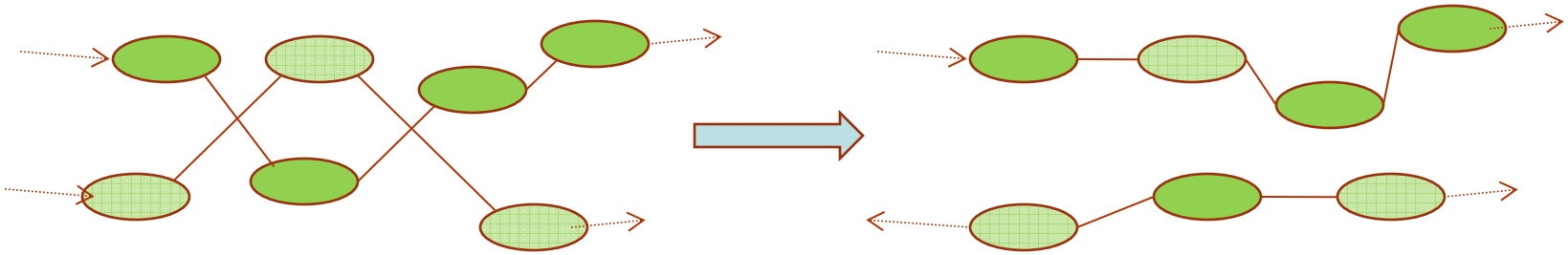
IloRelocate: a visit is inserted in another route



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

Number of locations	Exact CP (Scheduler)	Heuristic CBLS (Dispatcher)
13	12%	12%
14	15%	14%
15	7%	6%
16	5%	3%
18	16%	15%

- 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

Number of locations	Number of trucks	Exact CP (Scheduler)	Heuristic CBLS (Dispatcher)
30	2	-	4%
60	4	-	8%
130	9	-	10%

- 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