

# Column Generation for the Pickup and Delivery Problem with Time Windows

Kombinatorisk optimering

1. april 2005

Stefan Røpke

Joint work with Jean-François Cordeau

Work in progress!

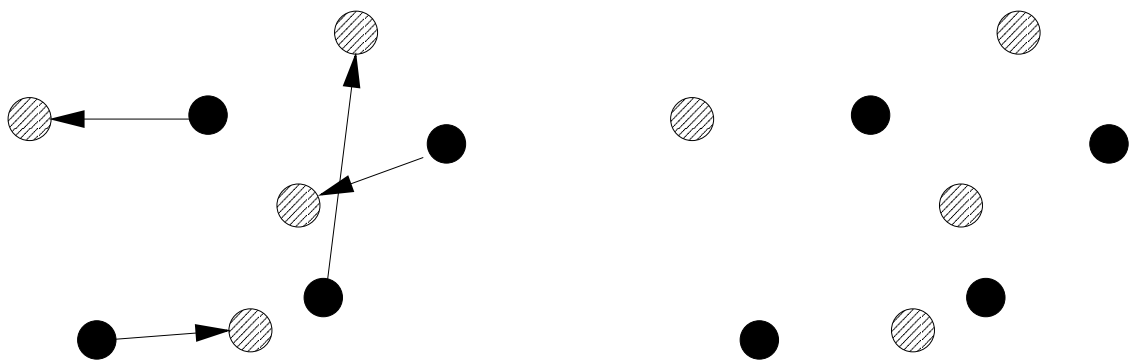
Literature, homework assignments and slides on homepage after lecture.

# The Pickup and Delivery Problem with Time Windows (PDPTW)

- The PDPTW is a model of a transportation problem that often occurs in real life.
- Assume that you own a fleet of vehicles. Your customers want goods to be transported between locations in Denmark.
- Examples
  - 3x34
  - Transportation of lifestocks (for example pigs)
  - Transportation of raw material and goods between production facilities and warehouses

# The pickup and delivery problem with time windows

- More formally, we are given:
  - A set of  $n$  **requests**. Each request consists of a pickup and a delivery.
  - A request involves transporting an amount of goods.
  - Vehicles have a certain maximum capacity.
  - The pickup and delivery of a request are performed by the same vehicle.
  - A time window is associated with both the pickup and the delivery.



- We usually want to either
  1. minimize travelled distance (number of vehicles to use is free) or
  2. minimize number of vehicles used.

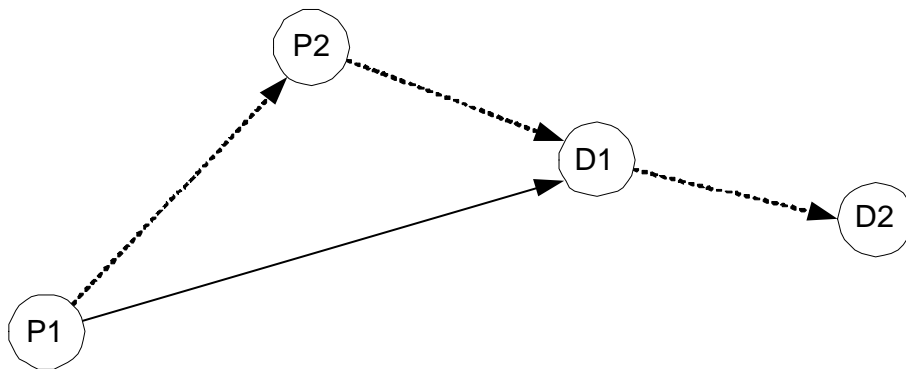
# Introduction to DARP

Door-to-door transportation of elderly and disabled persons (*the users*).

- Several users are transported in the same vehicle (think of a mini-bus).
- The users specify when they wish to be picked up and when they have to be at their destination (a *request*).
- The users do not specify an exact time of day, but a time window. Example: Instead of requesting a pickup at 9:11 the users request a pickup between 9:00 and 9:30.
- Often the user only specifies either the pickup or delivery time window. An operator would assign the other time window.

# Introduction to DARP

- Users don't like to be taken on long detours even if it helps the overall performance of the transportation system. Consequently a maximum ride time constraint is specified for each request.



- Time windows are not enough for ensuring that the maximum ride time constraint is enforced. Example: pickup [8:00; 8:15], delivery [8:45; 9:00], max ride time 45 minutes. Pickup at 8:00 and delivery at 9:00 violates max ride time constraint. Pickup time window could be shrunk to [8:15; 8:15]. This would ensure that ride time constraint is enforced, but it rules out perfectly good solutions like pickup at 8:05 and delivery at 8:45.
- Each vehicle has a certain capacity (only a limited amount of seats).
- The vehicles have to start and end their tours at a given start and end terminal.
- Objective: minimize driving cost subject to the constraints mentioned above.
- Problem is NP-Hard.

# IP model

Notation:

|                                   |   |
|-----------------------------------|---|
| $n$                               | Number of requests.   |
| $P = \{1, \dots, n\}$             | Pickup locations  |
| $D = \{n + 1, \dots, 2n\}$        | Delivery locations  |
| $N = P \cup D \cup \{0, 2n + 1\}$ | The set of all nodes in the graph. 0 and $2n + 1$ are the start and end terminal respectively. Request $i$ consist of pickup $i$ and delivery $n + i$ . |
| $K$                               | Set of vehicles   |
| $G = (N, A)$                      | Directed graph on which the problem is defined. $A$ is the set of edges.  |
| $Q$                               | Capacity of a vehicle   |
| $q_i$                             | Amount loaded onto vehicle at node $i$ . $q_i = -q_{n+i}$ .   |
| $[e_i, l_i]$                      | time window of node $i$   |
| $d_i > 0$                         | duration of service at node $i$   |
| $L$                               | Max ride time of a request.   |
| $c_{ij}$                          | Cost of traveling from node $i$ to node $j$ . It is Assumed that $c_{ij}$ satisfies the triangle inequality.  |
| $t_{ij}$                          | Time needed for going from node $i$ to node $j$ . It is assumed that $t_{ij}$ satisfies the triangle inequality and that $t_{ij} > 0$ .                 |

# IP model

## Decision variables

Binary variables

$x_{ij}^k$  1 iff the  $k$ th vehicle goes straight from node  $i$  to node  $j$ .

Fractional variables

$B_i^k$  When vehicle  $k$  starts visiting node  $i$

$Q_i^k$  The load of vehicle  $k$  after visiting node  $i$ .

$L_i^k$  The ride time of request  $i$  on vehicle  $k$ .

# IP model (3-index)

Objective:

$$\min \sum_{k \in K} \sum_{i \in N} \sum_{j \in N} c_{ij}^k x_{ij}^k$$

Every request is served exactly once:

$$\sum_{k \in K} \sum_{j \in N} x_{ij}^k = 1 \quad \forall i \in P$$

Same vehicle services pickup and delivery:

$$\sum_{j \in N} x_{ij}^k - \sum_{j \in N} x_{n+i,j}^k = 0 \quad \forall i \in P, k \in K$$

Every vehicle leaves the start terminal:

$$\sum_{j \in N} x_{0j}^k = 1 \quad \forall k \in K$$

The same vehicle that enters a node leaves the node:

$$\sum_{j \in N} x_{ji}^k - \sum_{j \in N} x_{ij}^k = 0 \quad \forall i \in P \cup D, k \in K$$

Every vehicle enters the end terminal:

$$\sum_{i \in N} x_{i,2n+1}^k = 1 \quad \forall k \in K$$

# Standard model (3-index)

Setting and checking visit time:

$$B_j^k \geq (B_i^k + d_i + t_{ij})x_{ij}^k \quad \forall i \in N, j \in N, k \in K$$
$$e_i \leq B_i^k \leq l_i \quad \forall i \in N, k \in K$$

Linearization of first equation ( $M_{ij}^k$  is a large constant):

$$B_j^k \geq B_i^k + d_i + t_{ij} - M_{ij}^k(1 - x_{ij}^k) \quad \forall i \in N, j \in N, k \in K$$

Setting and checking vehicle load:

$$Q_j^k \geq (Q_i^k + q_j)x_{ij}^k \quad \forall i \in N, j \in N, k \in K$$
$$Q_i^k \leq Q \quad \forall i \in N, k \in K$$

Binary variables:

$$x_{ij}^k \in \{0, 1\} \quad \forall i \in N, j \in N, k \in K$$

**If DARP:** Setting and checking ride time:

$$L_i^k = B_{n+i}^k - (B_i^k + d_i) \quad \forall i \in P, k \in K$$
$$L_i^k \leq L \quad \forall i \in N, k \in K$$

# Overview of recent literature

- From my point of view: Two main branches - exact methods and heuristics
- PDPTW heuristics:
  - *Li & Lim, 2001*, tabu search + simulated annealing, up to 50 requests, 30-4000 seconds on a i686.
  - *Bent & Van Hentenryck, 2003*, LNS, up to 500 requests, Largest problems was solved in 135 minutes on 1.2 Ghz AMD.
  - *Ropke & Pisinger, 2005*, ALNS, up to 500 requests, avg. time to solve largest problems was 90 minutes, but some of the problems took up to eight hours to solve (on a Pentium IV, 1.5 Ghz).
- Generally the later methods dominate the earlier in terms of solution quality.
  - *Xu, Chen, Rajagopal, 2003*, column generation heuristic, up to 500 requests, largest problems was solved in 2.1-7.5 hours on PIII, 450 Mhz. Problem was different from the problem solved in the three previous papers so a direct comparison is not possible.
- State of the art heuristics seems quite robust and yields high quality solutions.

# Overview of recent literature

- PDPTW exact methods
- IP Column generation: Branch and bound algorithm with linear programming (LP) lower bound based on a model with an exponential number of variables.
- Branch & Cut: Branch and bound algorithm with (LP) lower bound based on a model with an exponential number of constraints.
  - *Dumas, Desrosiers, Soumis*, 1991, column generation, up to 55 requests, largest problem was solved (\*) in 313 seconds on a Cyber 855 computer (whatever that is).
  - *Savelsbergh, Sol*, 1998, column generation, up to 30 requests, 5 seconds - 15 hours IBM/RS6000.
  - *Lu, Dessouky*, Branch & Cut, 2003, up to 15 requests, 4.3 seconds - 4.8 hours on 900Mhz Sun computer. Without time windows, slightly larger problems could be solved.
  - *Sigurd, Pisinger, Sig*, 2004, column generation, upto 205 requests, largest problem solved in 1.5 hours, 933Mhz PIII.
  - Ropke, Cordeau, Laporte, 2005 (work in progress), branch and cut, up to 20 requests in 2 hours on 2.5 Ghz Pentium IV.

- Ropke, Cordeau, 2005 (work in progress), column generation, up to 500 requests. Largest problem solved in 350 seconds on 2.5 Ghz Pentium IV. Only very “nice” problems of these sizes can be solved though.
- The last method is the topic of this talk.
- It’s hard to compare algorithms. No standard benchmark instances + small variations in the problem solved.

# DARP literature

- Heuristics, for an overview see:
  - Jean-François Cordeau, Gilbert Laporte, *The Dial-a-Ride Problem: Variants, Modelling issues and Algorithms*, Technical Report, University of Montreal, G-2002-25.
  - It's hard to compare heuristics due to lack of common test instances.
- Exact methods
  - Several single vehicle algorithms from the eighties.
  - Cordeau, 2004, Branch & Cut, instances with up to 30 requests are solved within 4 hours on 2.5Ghz Pentium IV.
  - Ropke & Cordeau, 2005 (work in progress), Branch & Cut, instances with up to 50 requests are solved within 4 hours on 2.5Ghz Pentium IV.

# PDPTW, Set Partitioning Problem (SPP)

- Enumerate all feasible routes.
- Construct an IP model where each decision variable  $x_j$  corresponds to a feasible route. If  $x_j = 1$  then route  $j$  is used in the solution.
- $c_j$  gives the cost of the  $j$ 'th route.
- $a_{ij}$  indicates if request  $i$  is served on the  $j$ 'th route.

$$\min \sum_{j=1}^q c_j x_j \quad (1)$$

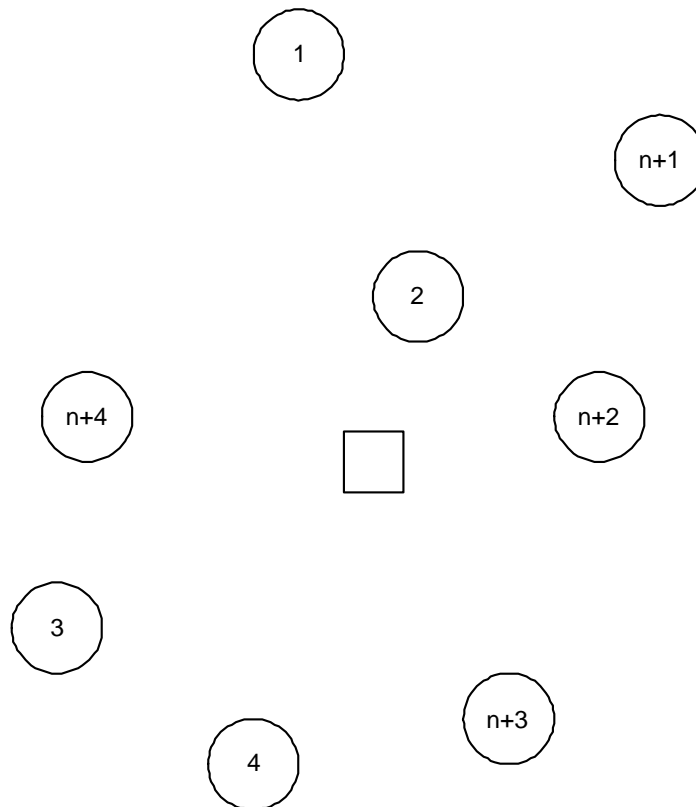
Subject to:

$$\sum_{j=1}^q a_{ij} x_j = 1 \quad \forall i \in V_0 \quad (2)$$

$$x_j \in \{0, 1\} \quad \forall j \in \{0, 1\} \quad (3)$$

- Each column corresponds to a feasible path from node 0 to node  $n + 1$ .
- It can be proven that the LP-relaxation of the set partitioning formulation dominates the LP-relaxation of the 3-index formulation.

# Set partitioning example



- Feasible routes (requests in the feasible routes):  $\{1\}$ ,  $\{2\}$ ,  $\{3\}$ ,  $\{4\}$ ,  $\{1,2\}$ ,  $\{1,4\}$ ,  $\{2,3\}$ ,  $\{3,4\}$ ,  $\{1,2,4\}$ .

$$\min 15x_1 + 8x_2 + 14x_3 + 13x_4 + \dots + 23x_6 + 20x_7 + 19x_8 + 27x_9$$

st

$$\begin{array}{cccccccccc} x_1 & & & & & \dots & + & x_6 & & + & x_9 & = & 1 \\ & x_2 & & & & \dots & & + & x_7 & & + & x_9 & = & 1 \\ & & x_3 & & & \dots & & + & x_7 & + & x_8 & & = & 1 \\ & & & x_4 & & \dots & + & x_6 & & & x_8 & + & x_9 & = & 1 \end{array}$$

# Solving the set partitioning problem

- When the number of feasible routes is rather small we can use the set partitioning model directly, and solve the problem using an IP-solver. The number of feasible routes can be small if the problem only contains a few customers, or if the problem is very constrained.
- Usually the number of feasible routes grows very rapidly as the number of customers increase. In a problem with 40 requests, where all routes with up to 7 requests are valid we would get about 4.6 million columns. Such a set partitioning problem is impossible to solve with the IP solvers of today, as the set partitioning problem is NP-hard.
- We could try to solve the LP relaxation of the problem, but even though we have polynomial time algorithms for the linear programming problem, this approach would also be futile if we apply linear programming directly.
- What we can do, is applying column generation techniques.

# Delayed Column Generation for VRPTW

- Basic idea:
  1. Solve the LP-relaxation using only a subset of all the columns.
  2. Use the theory from the Simplex algorithm to determine if the LP solution to the reduced problem also is the optimal LP solution to the complete problem with all the columns.
  3. If so, then we are “done” (we have a LP relaxation to the complete set partition problem). If not, then we use the theory of the Simplex algorithm to add one (or more) columns to our reduced problem. Step 1-3 is repeated until we are “done”.
- In practice step 2 and 3 are performed together.

# Column Generation - details

- The initial set of columns should be chosen such that a feasible solution to the LP-problem exists. This can for example be done by creating  $n$  columns, each representing a route with one request.
- When the reduced LP-problem has been solved we should determine if our LP-solution also is optimal for the complete LP, or if more columns must be added to the problem.
- This is done by looking at how the simplex algorithm works. In each iteration of the simplex algorithm we have a basic solution. In order to progress we have to choose a new variable to *enter the basis*. In a minimization problem like ours we have to choose a variable with negative *reduced cost*.

$$\min \quad cx \quad (4)$$

$$s.t. \quad Ax = b \quad (5)$$

$$x \geq 0 \quad (6)$$

- Reduced cost:

$$c_j^r = c_j - \pi A_j$$

where  $c_j$  is the  $j$ 'th coefficient in the objective,  $\pi$  is the vector of dual variables and  $A_j$  is the  $j$ 'th column in the matrix  $A$ .

## Column Generation cont.

- In our problem the reduced cost of a column (variable) is:

$$c_j^r = c_j - \sum_{i=1}^n \pi_i a_{ij}$$

- Thus we can generate all feasible columns one by one and check their reduced cost. In the end we can pick the column with lowest reduced cost. If that value is negative the column should be added to our problem, while if it is nonnegative the simplex algorithm is done and our solution is the optimal LP-solution to the complete LP-relaxation of the set-partitioning problem.
- In practice it is not possible to go through all columns one by one. Instead we define an IP-problem whose solution identifies the column with smallest reduced cost. We denote such a problem the *pricing problem* or the *sub-problem*.
- Recall that each column in our set partitioning problem corresponds to a feasible path from node 0 to node  $2n+1$ . That is, a path that respects time windows and capacities. Thus we should look for these paths.

## Column Generation cont.

- We are looking for a path from node 0 to  $2n + 1$ . The path should respect the constraints of the PDPTW / DARP.
- This problem can be expressed as an IP problem with a binary decision variables  $x_{pq}$  for each edge  $(p, q)$  in the graph. We would also need some fractional variables, but let us forget about them for now.  
The binary decision variable is 1 if the edge is used in the path. We have the following problem:

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & Bx = b \\ & x \in \{0, 1\}^{(2n+1)(2n+1)} \end{array}$$

- How should  $f(x)$  look like? Recall from last slide: Reduced cost of a variable / column:

$$c_j^r = c_j - \sum_{i=1}^n \pi_i a_{ij}$$

$a_{ij}$  indicated if request  $i$  was used in route  $j$ . Given a solution to the IP above we can find the  $i$ th entry in its column  $a_{i*}$  as follows:

$$a_{i*} = \sum_{q=0}^{2n+1} x_{iq}$$

Thus we can write the reduced cost of a column:

$$\begin{aligned} c_*^r &= c_* - \sum_{i=1}^n \pi_i a_{i*} \\ &= \sum_{p=0}^{2n+1} \sum_{q=0}^{2n+1} c_{pq} x_{pq} - \sum_{i=1}^n \pi_i \left( \sum_{q=0}^{2n+1} x_{iq} \right) \\ &= \sum_{p=0}^{2n+1} \sum_{q=0}^{2n+1} (c_{pq} - \pi'_p) x_{pq} \end{aligned}$$

where

$$\pi'_p = \begin{cases} \pi_p & \text{if } p \in \{1, \dots, n\} \\ 0 & \text{otherwise} \end{cases}$$

## Column Generation cont.

- Our shortest path problem becomes

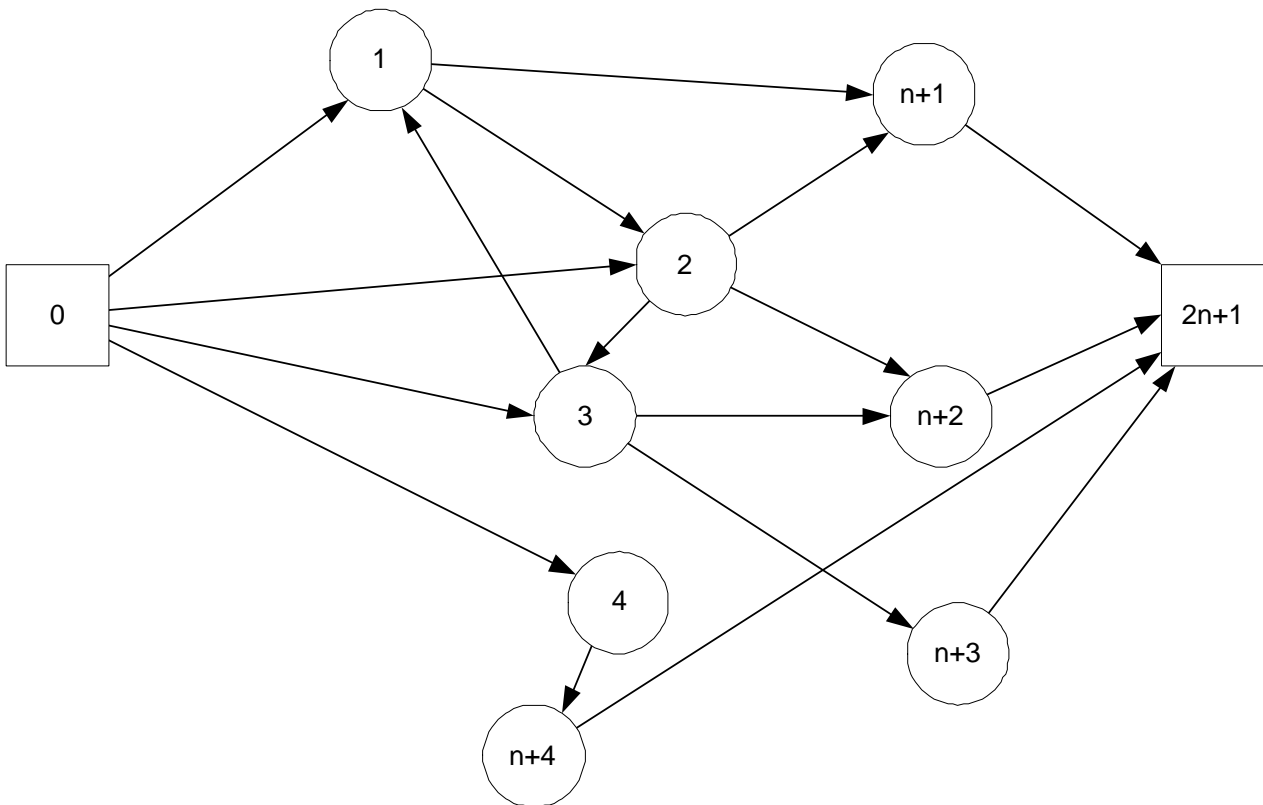
$$\min \sum_{p=0}^{2n+1} \sum_{q=0}^{2n+1} (c_{pq} - \pi'_p) x_{pq}$$

$$s.t. \quad Bx = b$$

$$x \in \{0, 1\}^{(2n+1)(2n+1)}$$

- This is an *elementary* shortest path problem with pickup and deliveries, time windows, capacity constraints and negative edge weights (*ESPPDTWCC*). Such a problem is NP-hard.
- A problem that is slightly easier is the shortest path problem with pickup and deliveries, with time windows, capacity constraints and negative edge weights (*SPPDTWCC*). In this problem, we are allowed to visit each customer several times, and our paths can consequently contain cycles. The problem is still NP-hard though. Most column generation algorithms for the PDPTW have done this, but we aim at solving the *ESPPDTWCC*.

# Shortest path example



- All edges have length 1.
- Dual variables:  $\pi_1 = 3, \pi_2 = 4, \pi_3 = 2, \pi_4 = 1$
- Notice: Cycling!
- When  $\forall (i, j) \in E \ t_{ij} > 0$  or  $\forall i \in V \ d_i > 0$  infinite cycling is impossible.

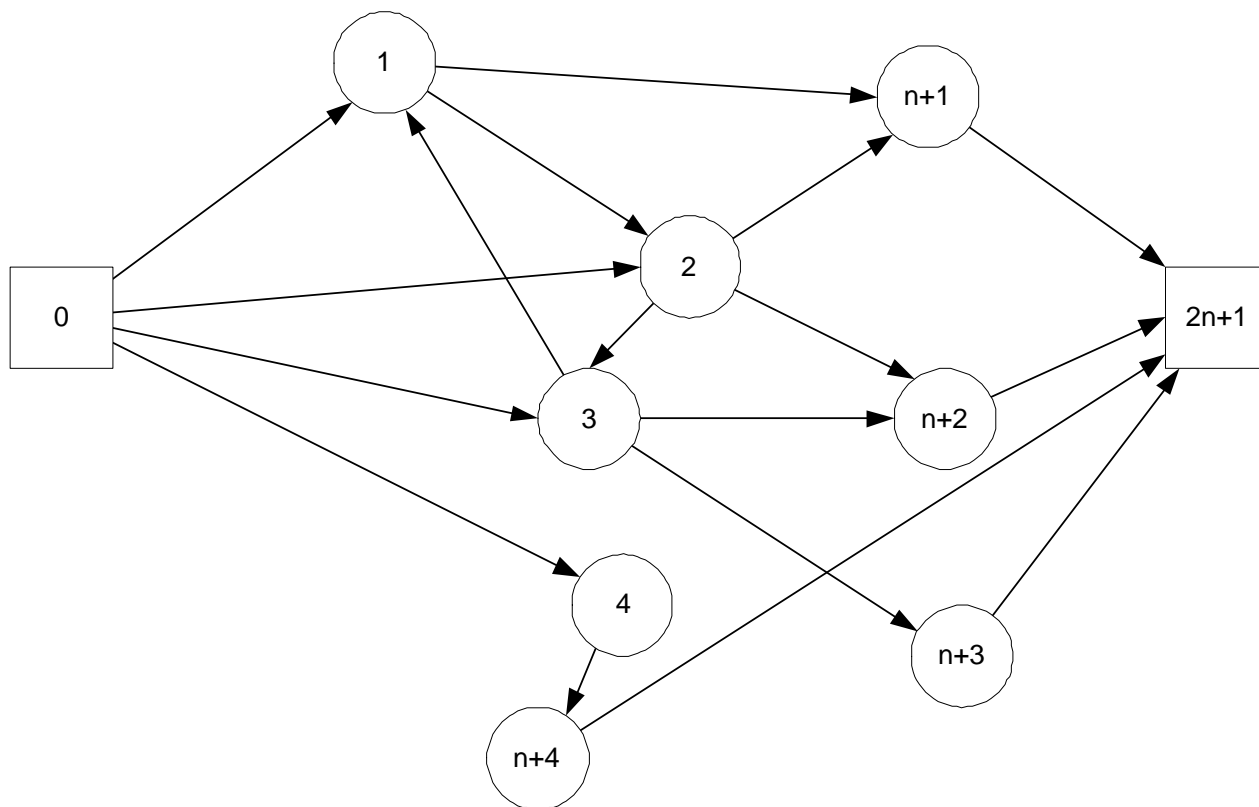
# Solving the SPPDWTWCC

- Solution approach: *labeling algorithm*.
- Brute force: Enumerate all paths:
  - Labels are associated with each node  $i$  in the graph. A label identifies a path from 0 to node  $i$ . Various info can be stored in the label, like the cost of the path, the sequence of nodes visited in the path, the time node  $i$  is visited in the path.
  - We start out with only one label. This label is associated with node 0 and has cost zero.
  - concept - *extending labels*: Consider one specific label in node  $i$  and extend the path corresponding to the label to all nodes that can be reached directly from  $i$ . For each of the constructed paths, check if it is feasible. If it is, create a new label in the end node corresponding to the new path.

- The brute force algorithm works as follows:
  - While labels in nodes  $0, \dots, 2n$  exists:
    - \* Select a label  $l$ , do not select labels in node  $2n + 1$ .  
Extend the label to all nodes that can be reached directly from the label and erase the label.
  - Find the cheapest label in node  $2n + 1$ . Return the path stored in the label, this is the shortest path.
  
- Every time a label is extended the “childs” of the label will have a later start time because of assumptions so process will end.
  
- But... the running time is exponential and we get non-elementary paths.
  
- We can get rid of nonelementary paths by keeping track of the nodes visited in the label/path. We should not extend labels to already visited nodes.

# Solving the ESPPDWTWCC

- Example again:



# Solving the ESPPDWTWCC

- Speeding up labeling algorithm: *Dominance*.
- Is it possible to discard some labels during the execution of the algorithm?
- Assume that the labels  $x$  and  $y$  both corresponds to node  $i$ . We say that  $x$  *dominates*  $y$  if the cheapest feasible path to node  $2n + 1$  that can be created from label  $y$  is more expensive than the cheapest feasible path to node  $2n + 1$  that can be created from label  $x$ .
- If  $x$  dominates  $y$  then  $y$  can be discarded (the paths we get from  $y$  will be more expensive than the best path we get from  $x$ ).
- How can we know if  $x$  dominates  $y$ ?

- First try:  $x$  dominates  $y$  if cost of path in  $x$  is cheaper than cost of path in  $y$ .
  - Doesn't work: If label  $x$  visits node  $i$  later than label  $y$  then we might visit some edges with high negative reduced cost that can't be visited from label  $x$  because of time windows.
- Second try:  $x$  dominates  $y$  if cost of path in  $x$  is cheaper than cost of path in  $y$  **and** node  $i$  is visited at least as early in label  $x$  as in label  $y$ .
  - Getting better. Problem above solved, but still not good enough. If  $x$  has visited some requests that  $y$  hasn't visited then the best path extended from  $y$  might end up being better than the best path extended from  $x$ . The path extended from  $y$  might visit some of the nodes that  $x$  already has visited and thereby use some negative cost edges that paths extended from  $x$  can't reach.
- Third try ....

- Final solution:  $x$  dominates  $y$  if

$$x.cost \leq y.cost$$

$$x.time \leq y.time$$

$$x.nodesVisited \subseteq y.nodesVisited$$

$$x.openRequests \subseteq y.openRequests$$

- *openRequests* are the requests that the path in the label has started serving (pickup serviced) but hasn't finished (delivery not done yet).
- This dominance relation works if visiting deliveries never lowers the the cost of the paths. This can be achieved by moving the dual values from the edges leaving the pickups to the edges entering the pickups.
- Several tricks can be used to speed up the computation more.

# Hard and Easy ESPPPDTWCC

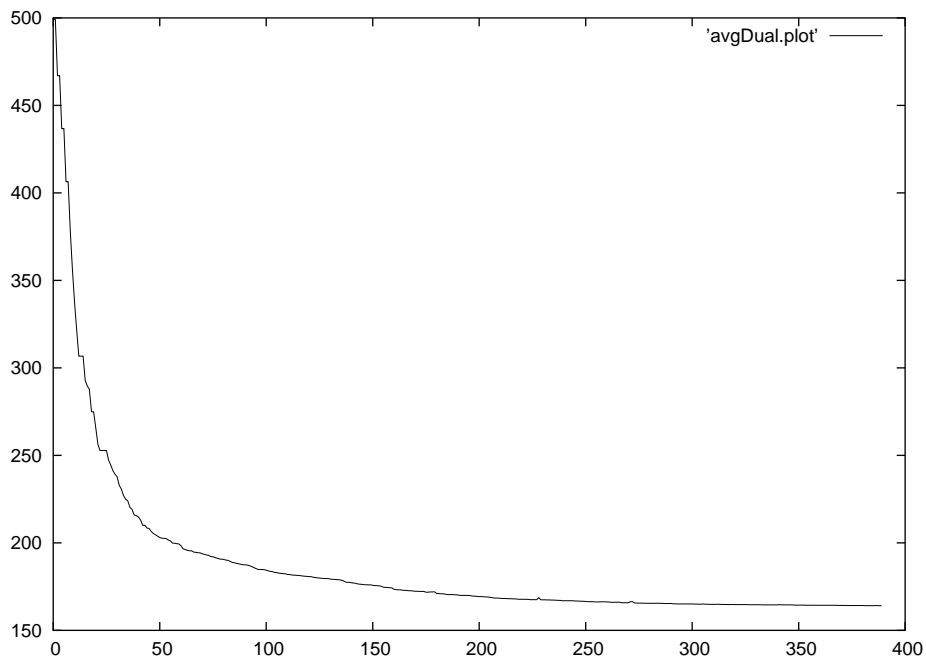
- Several properties of a SPPTWCC problem influences the practical hardness of the problem when solved by the algorithm described above. The table below summarizes these properties.

| Easier                             | Harder                              |
|------------------------------------|-------------------------------------|
| Small problems                     | Large problems                      |
| Sparse graph                       | Dense graph                         |
| Tight time windows                 | Large time windows                  |
| few open requests at the same time | Many open requests at the same time |
| Few edges with negative cost       | Many edges with negative cost       |

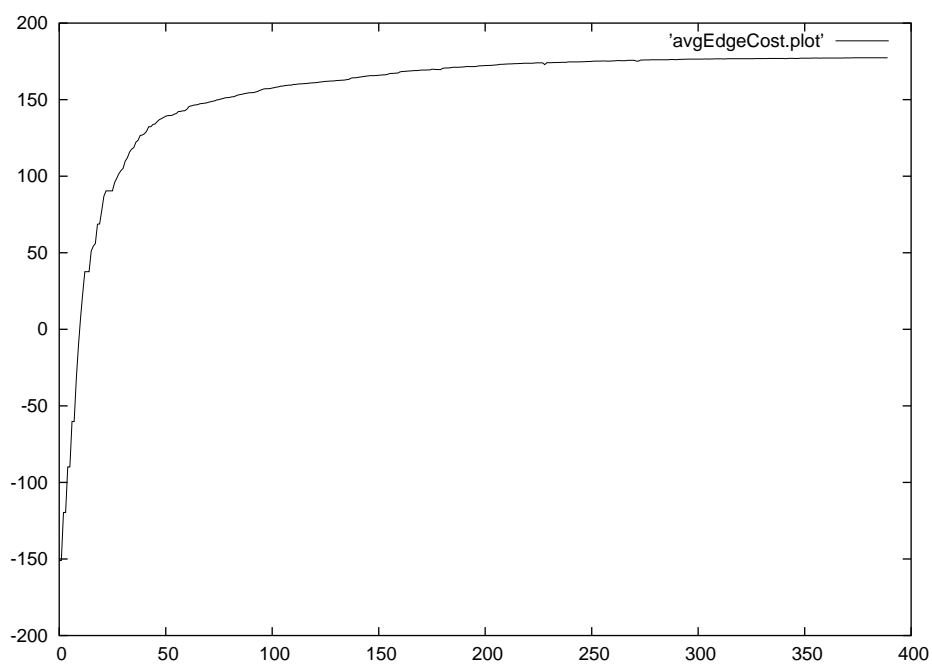
- It is hard to do much about the first four properties, although the preprocessing described at the end of this lecture can help a bit.
- The number of edges with negative cost and the magnitude of the negative costs are determined by the dual variables (recall that  $\hat{c}_{pq} = (c_{pq} - \pi'_p)$ ). Let's see how the dual variables and the edge cost behave as our column generation algorithm progresses.

# Column generation R101, 100 cust. (VRPTW examples)

$$\sum_{i=1}^n \pi_i / n:$$

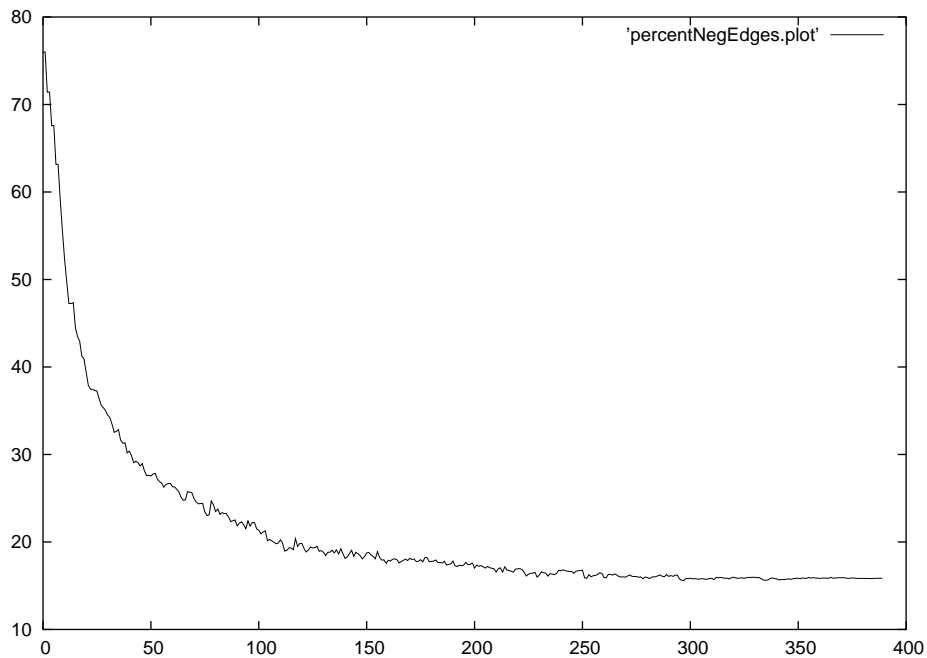


Average edge cost in graph for shortest path problem:

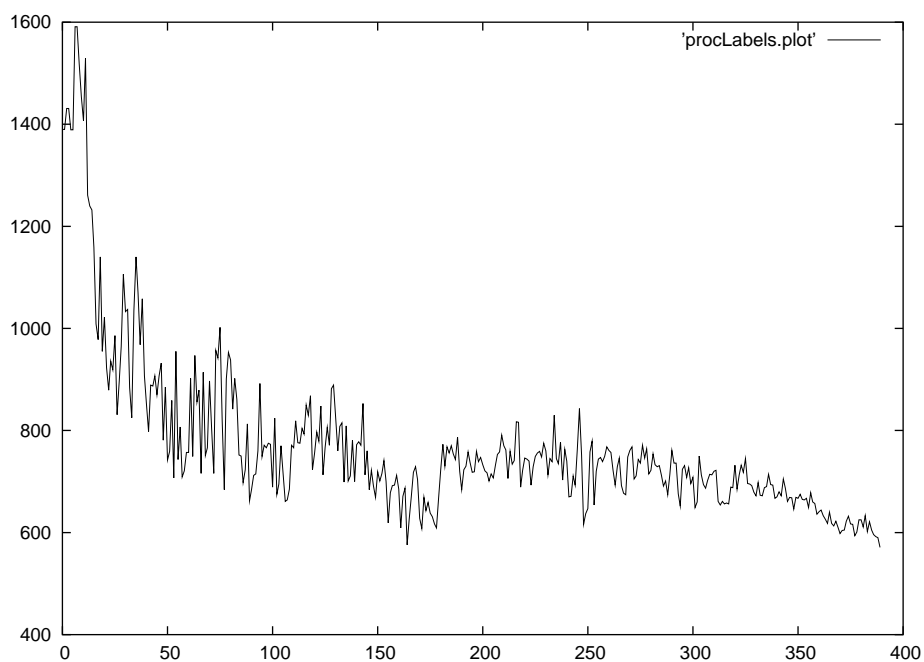


# Column generation R101, 100 cust.

Percent negative edges in shortest path graph.

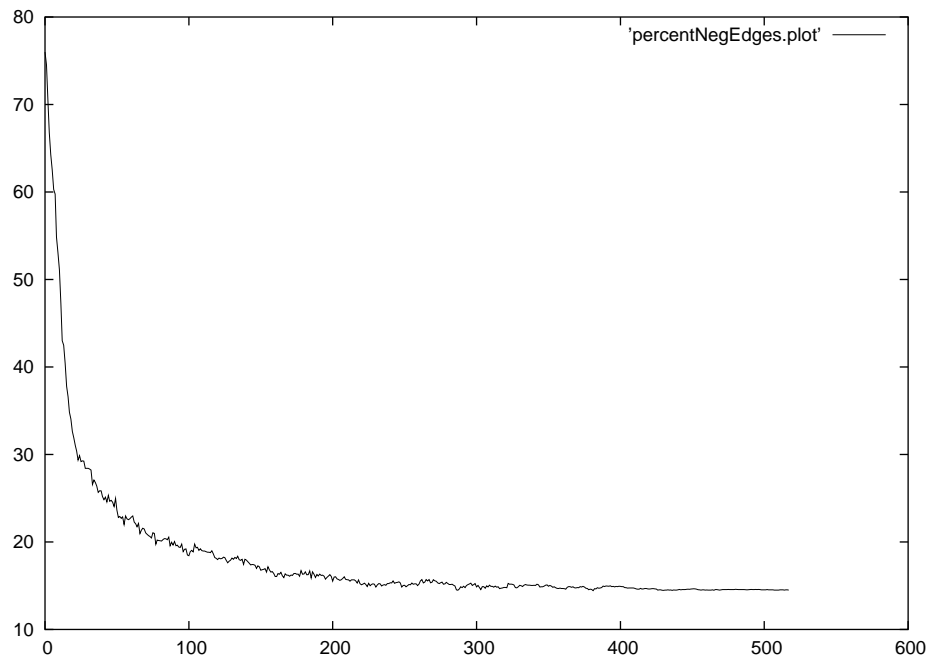


Number of labels processed in shortest path algorithm.

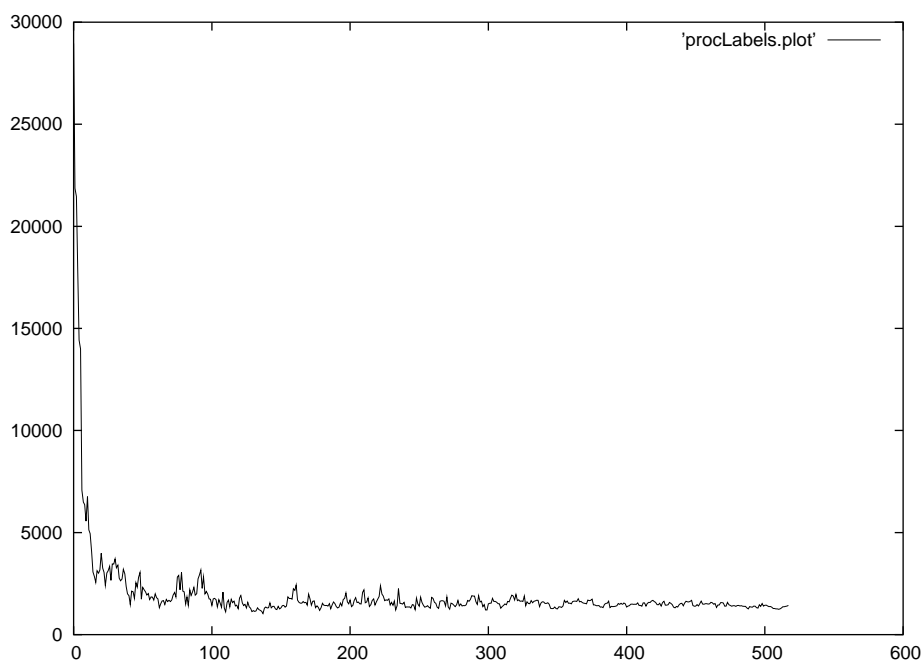


# Column generation R102, 100 cust.

Percent negative edges in shortest path graph.

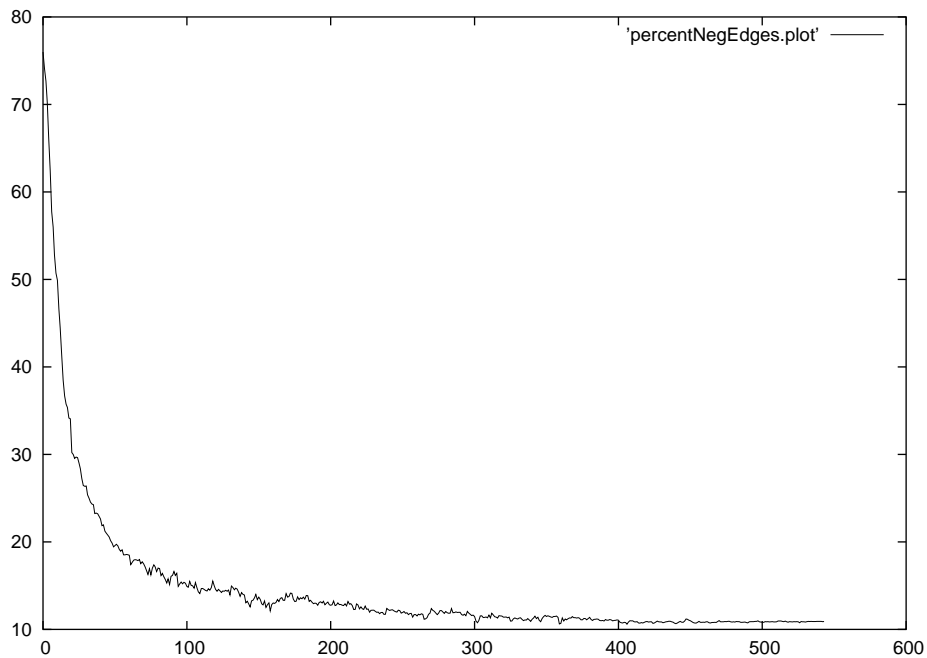


Number of labels processed in shortest path algorithm.

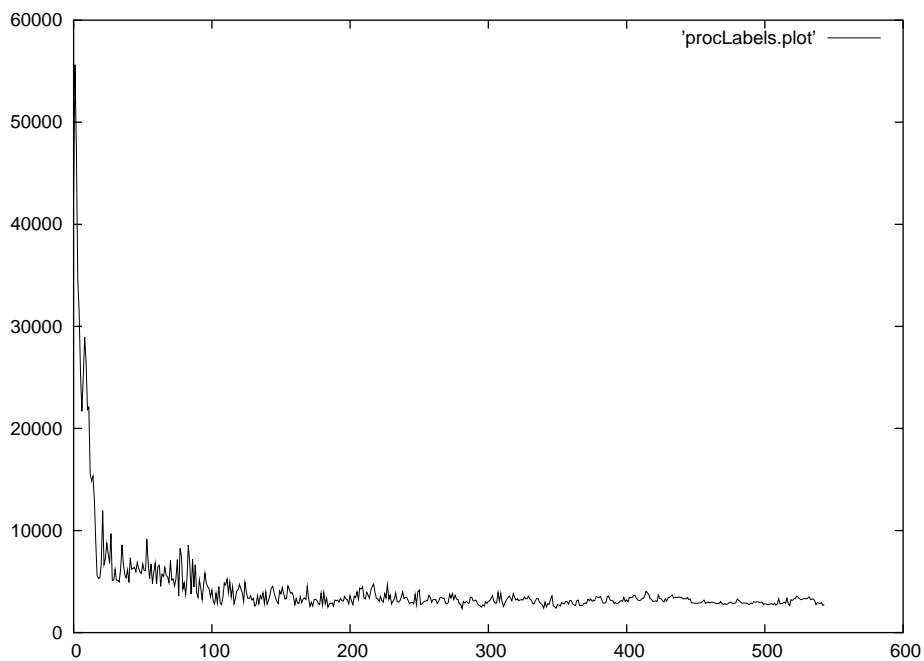


# Column generation R103, 100 cust.

Percent negative edges in shortest path graph.



Number of labels processed in shortest path algorithm.



- Lesson learned: The shortest path problem should be solved with a heuristic initially. Only when the heuristic

no longer can produce columns with reduced cost is it necessary to switch to the exact method. We must always solve the last pricing problem to optimality to prove that the lower bound is valid.

# Adding more inequalities to the model

- It would be nice to be able to add inequalities to our set partitioning problem. This would allow us to add valid inequalities in a cutting plane fashion. Furthermore it can make it easy to implement a branch and bound method.
- In general this is not easy though as the addition of a new inequality can change the structure of our subproblem.
- We will now see how inequalities in the  $x_{ij}^k$  variables in the original 3-index formulation can be used in the column generation.
- Define flow variables:

$$f_{ij} = \sum_{k \in K} x_{ij}^k \quad \forall i, j \in V$$

- Given a solution  $(x_p)$  to the LP relaxation of our SPP we can find the flow variables as follows:

$$f_{ij} = \sum_{p=1}^q y_{ij}^p x_p \quad \forall i, j \in V$$

where  $y_{ij}^p$  indicates the number of times that path  $p$  uses edge  $(i, j)$ . <TEGN>

# Adding more inequalities to the model

- Any valid inequality in the  $x_{ij}^k$  variables can be written as:

$$\sum_{k \in V} \sum_{i \in V} \sum_{j \in V} \rho'_{ijk} x_{ij}^k \geq \tau$$

Since the vehicles are identical this can be rewritten using flow variables:

$$\sum_{i \in V} \sum_{j \in V} \rho_{ij} f_{ij} \geq \tau$$

In the column generation context the left hand side becomes

$$\sum_{i \in V} \sum_{j \in V} \rho_{ij} \left( \sum_{p=1}^q (y_{ij}^p x_p) \right) = \sum_{p=1}^q \sum_{i \in V} \sum_{j \in V} (\rho_{ij} y_{ij}^p x_p)$$

Thus the coefficient of the  $p$ 'th column is

$$\sum_{i \in V} \sum_{j \in V} (\rho_{ij} y_{ij}^p)$$

If the dual variable corresponding to the constraint is named  $\theta$  then the cost of an edge in our subproblem becomes:  $\hat{c}_{ij} = c_{ij} - \pi_j - \theta p_{ij}$ , and we can solve this shortest path problem using the algorithms discussed previously.

# Method for solving DARP

- Now that we can add inequalities to the master problem, we can use the PDPTW pricing problem when solving DARP problems. We don't have to make a specific pricing algorithm.
- Infeasible path inequality:
  - For all paths  $i_1 \rightarrow i_2 \rightarrow \dots \rightarrow i_\alpha$  that are infeasible because of ride time constraints, add the following constraints to the master problem:

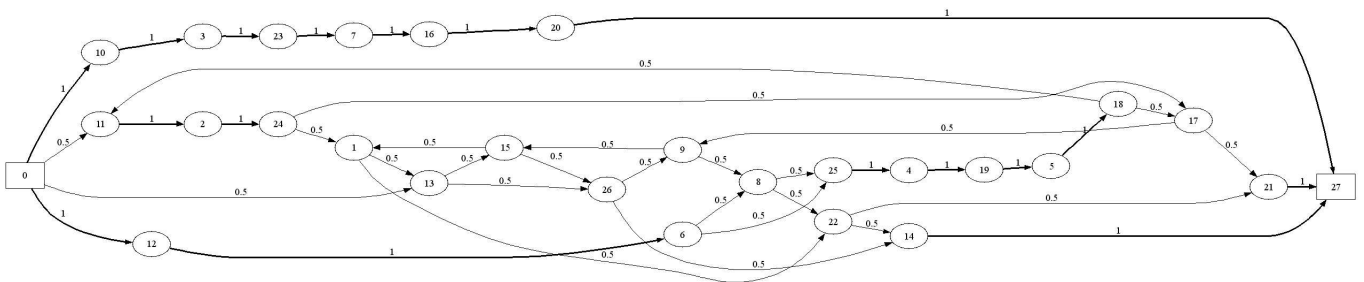
$$\sum_{j=1}^{\alpha-1} x_{i_j i_{j+1}} \leq \alpha - 2$$

## <TEGN>

- There are an exponential amount of these constraints, but they can be generated on the fly. They can even be separated in polynomial time.

# Additional cuts

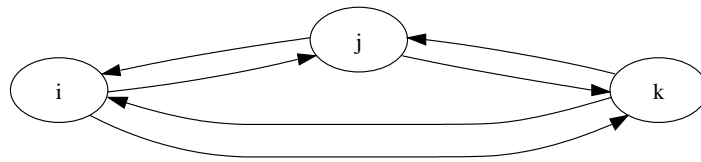
- Even though our LP relaxation is composed of elementary shortest paths we might get some “ugly” LP solutions as the example below illustrate.



- The following slides introduces some valid inequalities from Branch and Cuts algorithms for the PDPTW. It is not known if these inequalities will be useful for the column generation algorithm.

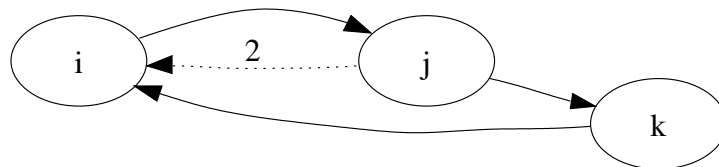
# Valid inequalities - some examples

Subtour elimination constraints



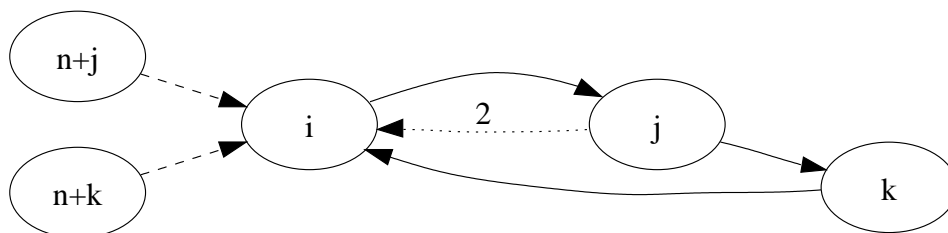
$$x_{ij} + x_{ji} + x_{jk} + x_{kj} + x_{ki} + x_{ik} \leq 2$$

Lifting for directed case:



$$x_{ij} + 2x_{ji} + x_{jk} + x_{ki} \leq 2$$

Lifting for PDPTW case:

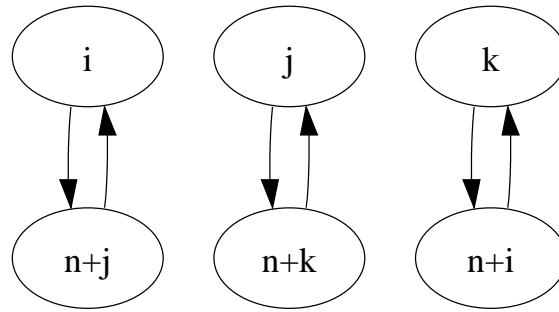


$$x_{ij} + 2x_{ji} + x_{jk} + x_{ki} + x_{n+j,i} + x_{n+k,i} \leq 2$$

General expression and more liftings described in paper.

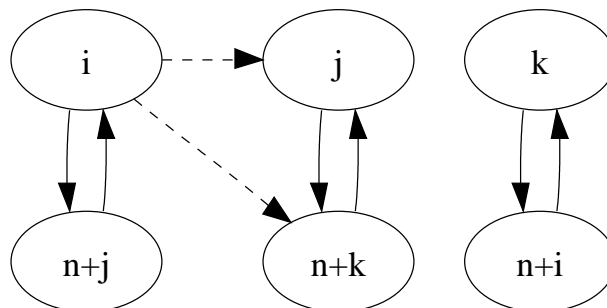
Separation algorithms?

# Generalized order constraints



$$x_{i,n+j} + x_{n+j,i} + x_{j,n+k} + x_{n+k,j} + x_{k,n+i} + x_{n+i,k} \leq 2$$

Lifting for directed case:



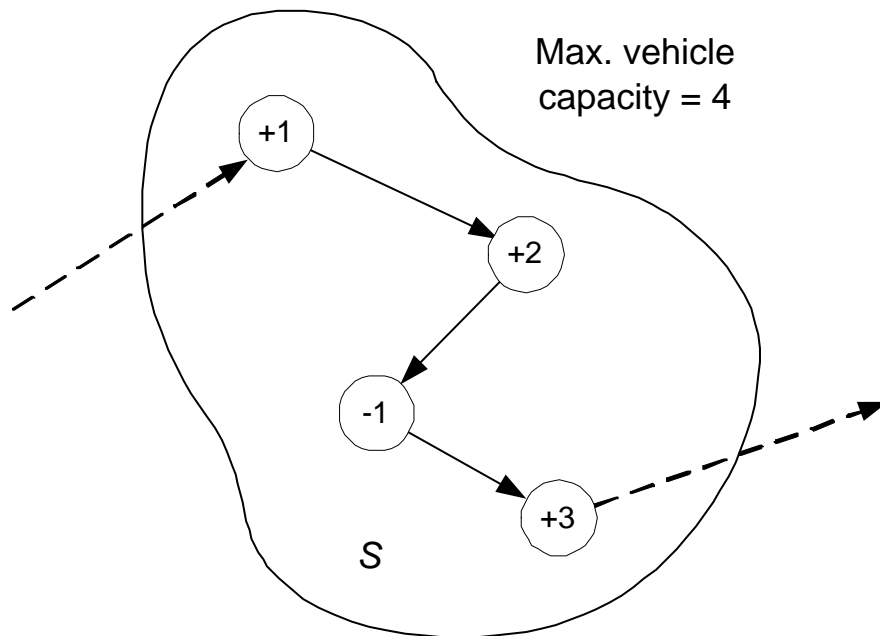
$$x_{i,n+j} + x_{n+j,i} + x_{j,n+k} + x_{n+k,j} + x_{k,n+i} + x_{n+i,k} + x_{ij} + x_{i,n+k} \leq 2$$

Separation algorithms?

# Capacity constraints

$$\sum_{i \in S} \sum_{j \in N \setminus S} x_{ij} \geq \left\lceil \frac{q(S)}{Q} \right\rceil \quad \forall S \subseteq P \cup D$$

$$q(S) = \sum_{i \in S} q_i$$



Separation algorithms?

# Branching

- Many branching schemes have been proposed in the literature, here we will only look at two (and only one that actually works!).
- The most obvious branching scheme is to branch on the fractional  $x_j$  variables in the SPP, this corresponds to saying that the  $j$ 'th route must be used in one subproblem and must not be used in the other subproblem.
- Unfortunately this branching scheme does not work very well with the column generation method:
- The  $x_j = 1$  branch is easy. Here we can remove the requests served by the  $j$ 'th route from the problem and add the cost of the route to the objective function.
- The  $x_j = 0$  branch is harder. In this case we must ensure that our column generation does not return the column we have forbidden. This can only(?) be done by making a shortest path algorithm that not only can return the shortest path but also the second shortest path. And this is a bit more tricky than it seems (see exercise). When  $p$  columns have been fixed to zero the shortest path algorithm must be able to return the  $p$ 'th shortest path.

## Branching cont.

- The second branching scheme branch on the  $f_{ij}$  variables (remember that  $f_{ij}$  indicates if an arc is used in the solution or not). In this case it is easy to handle both the  $f_{ij} = 1$  and  $f_{ij} = 0$  branches.
- Fixing  $f_{ij}$  to zero is simply done by removing the edge  $(i, j)$  from the graph.
- Fixing  $f_{ij}$  to one is done by removing all edges originating in  $i$  unless  $i=0$  and all edges ending in  $j$  unless  $j = n + 1$  from the graph. The only edge originating in  $i$  and ending in  $j$  left in the graph is  $(i, j)$ . **<TEGN>**
- The problem with this branching scheme is that it creates unbalanced sub problems.

# Preprocessing

- General idea: Reduce the size or complexity of a problem before applying a time consuming method like branch and bound.

<TEGN>

# PDPTW Results

Branch & Cut results (Ropke, Cordeau & Laporte, 2005  
- Work in progress):

| name  | n  | time     | init UB | best IP | Opt | root LB | best LB | B&B nodes |
|-------|----|----------|---------|---------|-----|---------|---------|-----------|
| a3-24 | 24 | 120      |         |         |     |         |         |           |
| a3-30 | 30 | 120      |         |         |     |         |         |           |
| a3-36 | 36 | 120      |         |         |     |         |         |           |
| a4-16 | 16 | 120      |         |         |     |         |         |           |
| a4-24 | 24 | 120      |         |         |     |         |         |           |
| a4-32 | 32 | 120      |         |         |     |         |         |           |
| a4-40 | 40 | 120      |         |         |     |         |         |           |
| a4-48 | 48 | 120      |         |         |     |         |         |           |
| b2-16 | 16 | 120      |         |         |     |         |         |           |
| b2-20 | 20 | 0.1335   | 284.83  | 284.82  | ×   | 277.183 | 284.82  | 47        |
| b2-24 | 24 | 120      |         |         |     |         |         |           |
| b3-18 | 18 | 6.11367  | 286.83  | 286.82  | ×   | 254.665 | 286.82  | 3511      |
| b3-24 | 24 | 120      |         |         |     |         |         |           |
| b3-30 | 30 | 120.088  | 452.42  |         |     | 416.291 | 441.334 | 7120      |
| b3-36 | 36 | 120      |         |         |     |         |         |           |
| b4-16 | 16 | 0.051167 | 293.09  | 293.08  | ×   | 286.562 | 293.08  | 30        |
| b4-24 | 24 | 120      |         |         |     |         |         |           |
| b4-32 | 32 | 120      |         |         |     |         |         |           |
| b4-40 | 40 | 120      |         |         |     |         |         |           |
| b4-48 | 48 | 120      |         |         |     |         |         |           |

Column Generation (Ropke & Cordeau, 2005 - Work in progress):

| name  | n  | Opt | LB | root | best IP | time    | nodes | LP-time |
|-------|----|-----|----|------|---------|---------|-------|---------|
| a2-16 | 16 | X   |    | X    | 219.06  | 2.57    | 9     | 0.01    |
| a2-20 | 20 | X   |    | X    | 254.4   | 142.69  | 265   | 2.5     |
| a2-24 | 24 | X   |    | X    | 274.72  | 206.33  | 129   | 1.4     |
| a3-18 | 18 | X   |    | X    | 228.67  | 1.41    | 1     | 0.05    |
| a3-24 | 24 | X   |    | X    | 230.87  | 30.17   | 3     | 0.07    |
| a3-30 | 30 | X   |    | X    | 361.42  | 508.85  | 125   | 1.77    |
| a3-36 | 36 | X   |    | X    | 366.22  | 194.44  | 3     | 0.4     |
| a4-16 | 16 | X   |    | X    | 217.34  | 1.01    | 5     | 0.02    |
| a4-24 | 24 | X   |    | X    | 286.87  | 88.97   | 57    | 0.36    |
| a4-32 | 32 | X   |    | X    | 353.23  | 286.74  | 31    | 0.79    |
| a4-40 | 40 | -   |    | X    | 416.82  | 7200.16 | 256   | 13.36   |
| a4-48 | 48 | -   |    | X    | 483.14  | 7200.59 | 18    | 1.35    |
| b2-16 | 16 | X   |    | X    | 255.12  | 0.54    | 11    | 0.06    |
| b2-20 | 20 | X   |    | X    | 284.82  | 0.1     | 1     | 0.02    |
| b2-24 | 24 | X   |    | X    | 376.8   | 3.85    | 31    | 0.32    |
| b3-18 | 18 | X   |    | X    | 286.82  | 0.21    | 5     | 0       |
| b3-24 | 24 | X   |    | X    | 343.58  | 3.04    | 19    | 0.07    |
| b3-30 | 30 | X   |    | X    | 452.41  | 0.78    | 3     | 0.07    |
| b3-36 | 36 | X   |    | X    | 515.93  | 2.25    | 3     | 0.11    |
| b4-16 | 16 | X   |    | X    | 293.08  | 0.03    | 1     | 0       |
| b4-24 | 24 | X   |    | X    | 343.6   | 0.23    | 1     | 0.03    |
| b4-32 | 32 | X   |    | X    | 467.09  | 1.13    | 3     | 0.07    |
| b4-40 | 40 | X   |    | X    | 596.83  | 2.16    | 1     | 0.13    |
| b4-48 | 48 | X   |    | X    | 588.1   | 1910.41 | 2189  | 126.84  |

Column Generation (Ropke & Cordeau, 2005 - Work in progress):

|         | #Req | Opt | LB | Opt | best IP | time    | nodes | LP-time |
|---------|------|-----|----|-----|---------|---------|-------|---------|
| LC121   | 106  | X   |    | X   | 2698.6  | 1.07    | 1     | 0.04    |
| LR121   | 105  | X   |    | X   | 4810.1  | 6.28    | 1     | 0.08    |
| LRC121  | 106  | X   |    | X   | 3599    | 30.95   | 3     | 0.18    |
| LC141   | 211  | X   |    | X   | 7138.8  | 36.37   | 1     | 0.08    |
| LR141   | 208  | X   |    | X   | 10620.4 | 142.41  | 1     | 0.33    |
| LRC141  | 208  | -   |    | X   | 8929.4  | 7201.43 | 192   | 21.62   |
| LC161   | 315  | X   |    | X   | 14076.6 | 93.04   | 1     | 0.22    |
| LR161   | 317  | X   |    | X   | 22482.1 | 1400.46 | 1     | 2.54    |
| LRC161  | 313  | -   |    | X   | 17765.3 | 7204.49 | 18    | 5.24    |
| LC181   | 420  | X   |    | X   | 25156.9 | 270.03  | 1     | 0.34    |
| LR181   | 421  | -   |    | -   | 39180.9 | 7210.61 | 1     | 6.95    |
| LRC181  | 411  | -   |    | -   | 31804.6 | 7209.75 | 1     | 4.23    |
| LC1101  | 527  | X   |    | X   | 42454.3 | 346.65  | 1     | 0.45    |
| LR1101  | 527  | -   |    | -   | 56516.8 | 7223.61 | 1     | 14.41   |
| LRC1101 | 527  | -   |    | -   | 48671.1 | 7224.18 | 1     | 12.7    |

# Conclusion

- Hard problems!
- Interesting problem. Lot of stuff to do!
- Column generation seems better suited than branch & cut for the PDPTW at the moment. It's going to be interesting to see the results for the DARP.