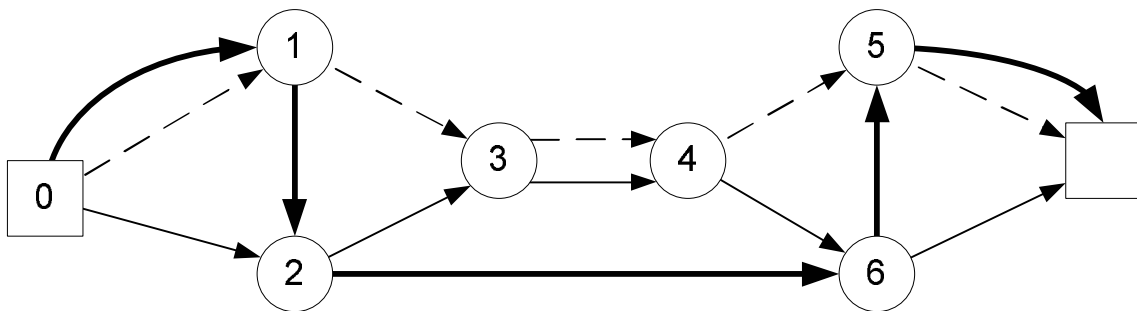


The Vehicle Routing Problem With Time Windows

Kombinatorisk optimering

2. april 2004

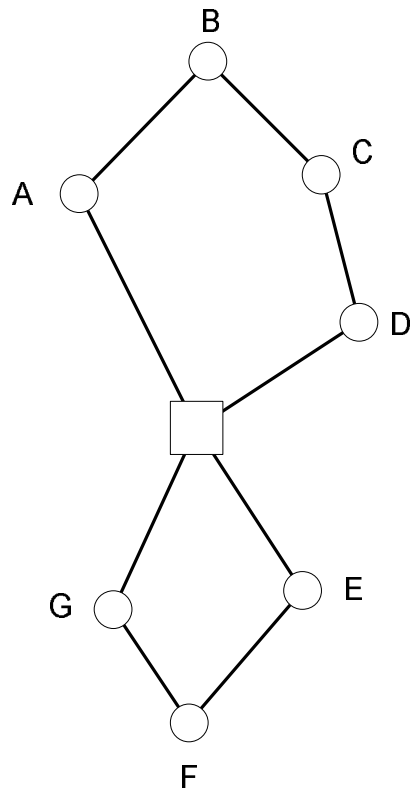


SCVRP

Recall from lecture one month ago:

- n customers that each demand a certain amount of goods.
- k identical vehicles, all located at one depot that supplies the goods to the customers. All vehicles should be used to serve the customers. Each vehicle has a certain capacity.
- A symmetric cost matrix that indicates the cost of traveling between two nodes in the graph. It is assumed that the cost matrix satisfies the triangle inequality.
- Our task is to assign the customers to vehicles such that the capacity of the vehicles are obeyed and such that all customers are served. Furthermore we must construct routes for the vehicles. Our objective is to minimize the total cost of the routes (as given by the cost matrix). Each customer should be visited precisely once.

CVRP Example



The Vehicle Routing Problem With Time Windows (VRPTW)

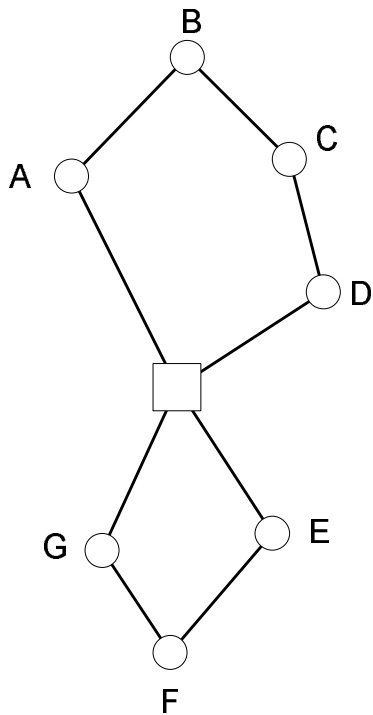
- An extension to CVRP.
- In the CVRP we had a cost c_{ij} for traveling between customers. In the VRPTW we both have a cost c_{ij} and a time t_{ij} needed for travelling between the customers i and j . Both c_{ij} and t_{ij} satisfies the triangle inequality.
- Each customer i has a time window $[a_i, b_i]$. A vehicle can only service a customer within the customer's time window.
- A vehicle is allowed to arrive to early at a customer i , but it must wait untill the start a_i of the customer's time window before the service operation can start.
- A vehicle must never arrive at a customer i after the end of the customer's time window (b_i).
- The time needed to service a customer i is s_i . In practice we represent this service time by adding it to the travel time of all edges leaving node i .
- The vehicles cannot leave the depot before a^d and must return to the depot before b^d .

VRPTW objectives

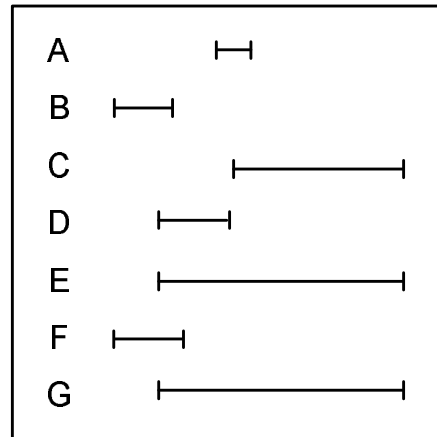
- Two objectives are typically considered in the literature:
 1. Minimize the number of vehicles needed for servicing the customers as first priority. Break ties by choosing the solution with lowest total travel cost (as given by c_{ij}).
 2. Minimize the total travel cost (as given by c_{ij}). The number of vehicles to use can be chosen freely.
- The heuristics described in the literature usually use an objective of the first type while exact methods use an objective of the second type.

VRPTW example

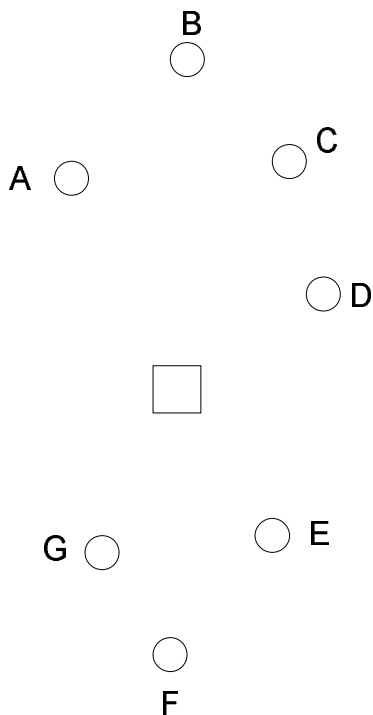
CVRP Solution



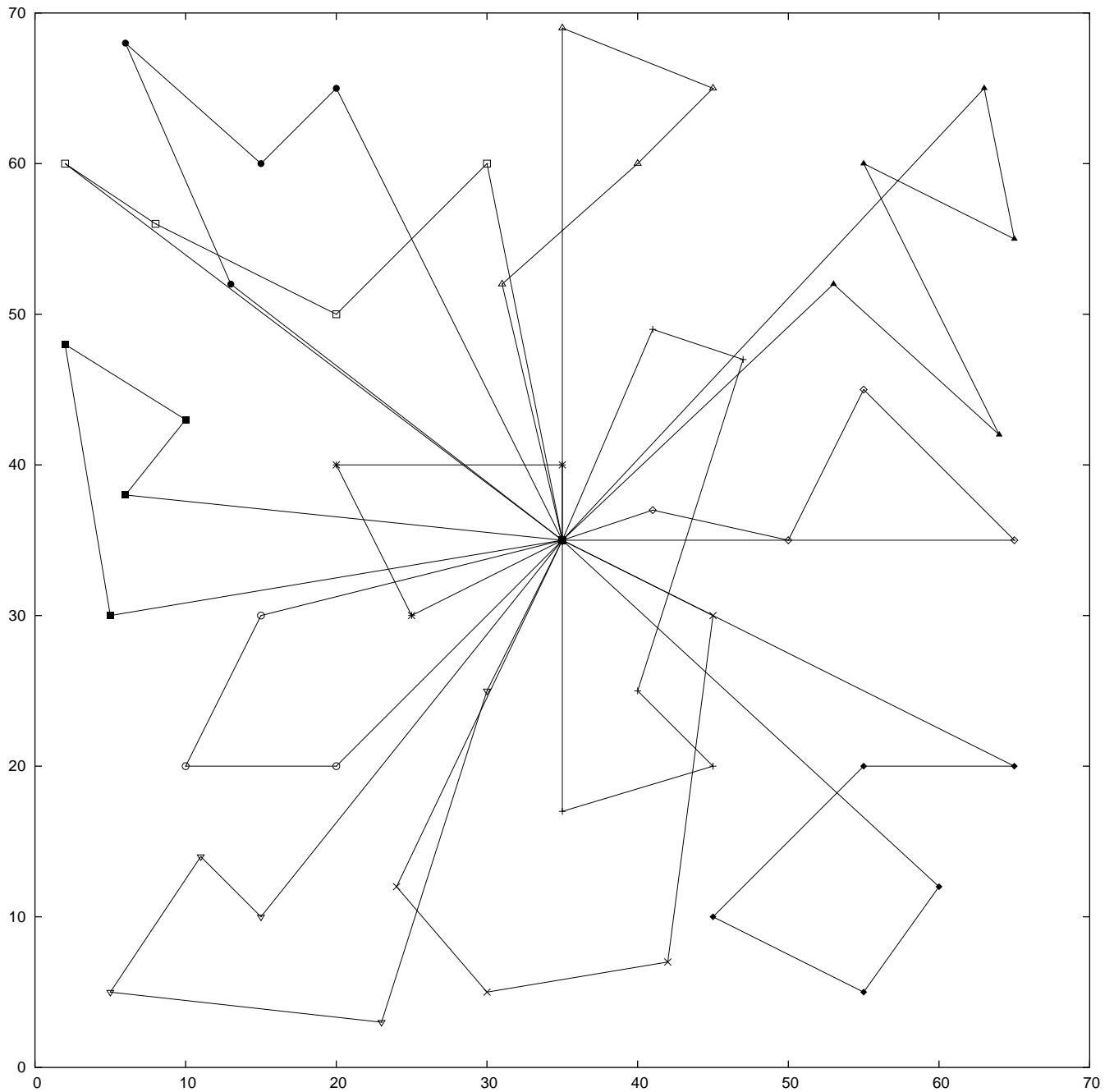
Time Windows



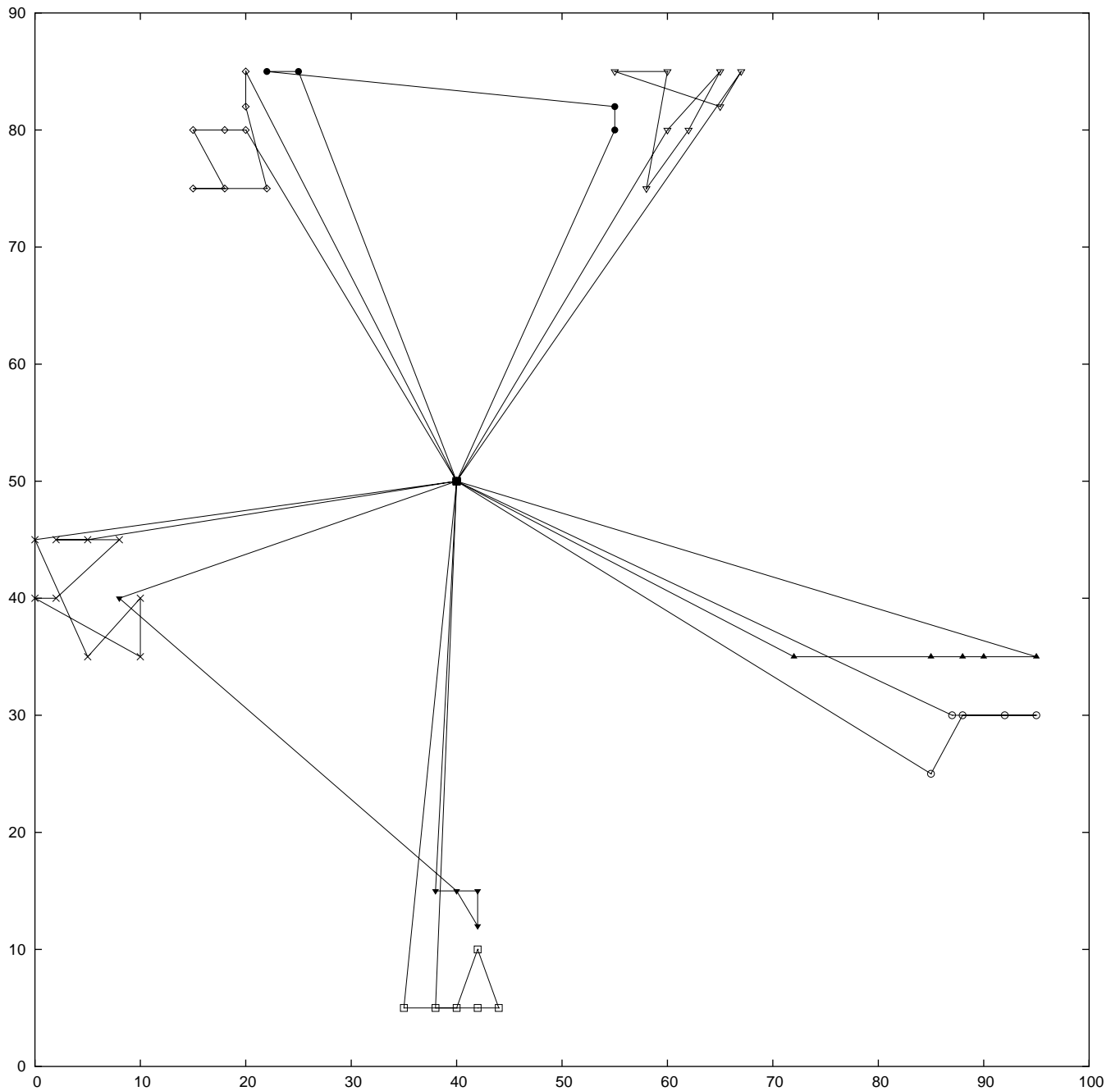
VRPTW Solution



VRPTW, 50 customers (R101)



VRPTW, 50 customers (RC101)



Notation

- $V = \{0, 1, \dots, n, n + 1\}$: The set of all nodes (customers), Node 0 is the depot. Node $n + 1$ is a copy of the depot (start and end terminal).
- $V_0 = V \setminus \{0, n + 1\}$
- $E = \{(i, j) : i, j \in V\}$: The set of all edges.
- c_{ij} : The elements of cost matrix that defines the cost of traveling between nodes (we assume that the cost matrix satisfies the triangle inequality).
- $G = (V, E)$: The graph on which the problem is defined.
- d_i : demand of customer i . It is assumed that $\forall i \in V_0 : d_i > 0$
- $d(S) = \sum_{i \in S} d_i, S \subseteq V$
- C : Capacity of a vehicle.
- $\delta(S), S \subseteq V$: The set of edges with one endpoint in S and one endpoint in $V \setminus S$.
- $\Delta^+(i) = \{j : j \in V, (i, j) \in E\}$: The *forward star* of i .
- $\Delta^-(i) = \{j : j \in V, (j, i) \in E\}$: The *backward star* of i .

VRPTW, 3-index model

Variables:

x_{ijk} : 1 if edge (i, j) is traversed by vehicle k , 0 otherwise.

w_{ik} : Time when vehicle k starts service at customer i . If vehicle k does not visit customer i then the variable is “undefined”.

Constants:

K : The set of vehicles.

c_{ij} : Cost of travelling from node i to node j . $i, j \in V$

$t_{ij} > 0$: The time needed to travel from i to j . $i, j \in V$

$d_i > 0$: The amount of goods to deliver at customer i .

a_i : Earliest arrival time at customer i . $i \in N$

b_i : Latest arrival time at customer i . $i \in N$

a^d : Earliest departure time from the depot.

b^d : Latest arrival time at the depot.

$M_{ij} = \max\{0, b_i + s_i + t_{ij} - a_j\}$: Large constants.

$$\min \sum_{k \in K} \sum_{(i,j) \in E} c_{ijk} x_{ijk} \quad (1)$$

Subject to:

$$\sum_{k \in K} \sum_{j \in \Delta^+(i)} x_{ijk} = 1 \quad \forall i \in V_0 \quad (2)$$

$$\sum_{j \in \Delta^+(0)} x_{0jk} = 1 \quad \forall k \in K \quad (3)$$

$$\sum_{i \in \Delta^-(j)} x_{ijk} - \sum_{i \in \Delta^+(j)} x_{jik} = 0 \quad \forall k \in K, \forall j \in V_0 \quad (4)$$

$$\sum_{i \in \Delta^-(n+1)} x_{i,n+1,k} = 1 \quad \forall k \in K \quad (5)$$

$$w_{ik} + t_{ij} - w_{jk} \leq (1 - x_{ijk}) M_{ij} \quad \forall k \in K, \forall (i, j) \in E \quad (6)$$

$$a_i \sum_{j \in \Delta^+(i)} x_{ijk} \leq w_{ik} \quad \forall k \in K, \forall i \in V_0 \quad (7)$$

$$b_i \sum_{j \in \Delta^+(i)} x_{ijk} \geq w_{ik} \quad \forall k \in K, \forall i \in V_0 \quad (8)$$

$$a^d \leq w_{ik} \quad \forall k \in K, \forall i \in \{0, n+1\} \quad (9)$$

$$b^d \geq w_{ik} \quad \forall k \in K, \forall i \in \{0, n+1\} \quad (10)$$

$$\sum_{i \in N} d_i \sum_{j \in \Delta^+(i)} x_{ijk} \leq C \quad \forall k \in K \quad (11)$$

$$x_{ijk} \in \{0, 1\} \quad \forall k \in K, \forall (i, j) \in E \quad (12)$$

$$w_{ik} \geq 0 \quad \forall k \in K, \forall i \in v_0 \quad (13)$$

VRPTW, 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 customer i is served on the j 'th route.

$$\min \sum_{j=1}^q c_j x_j \quad (14)$$

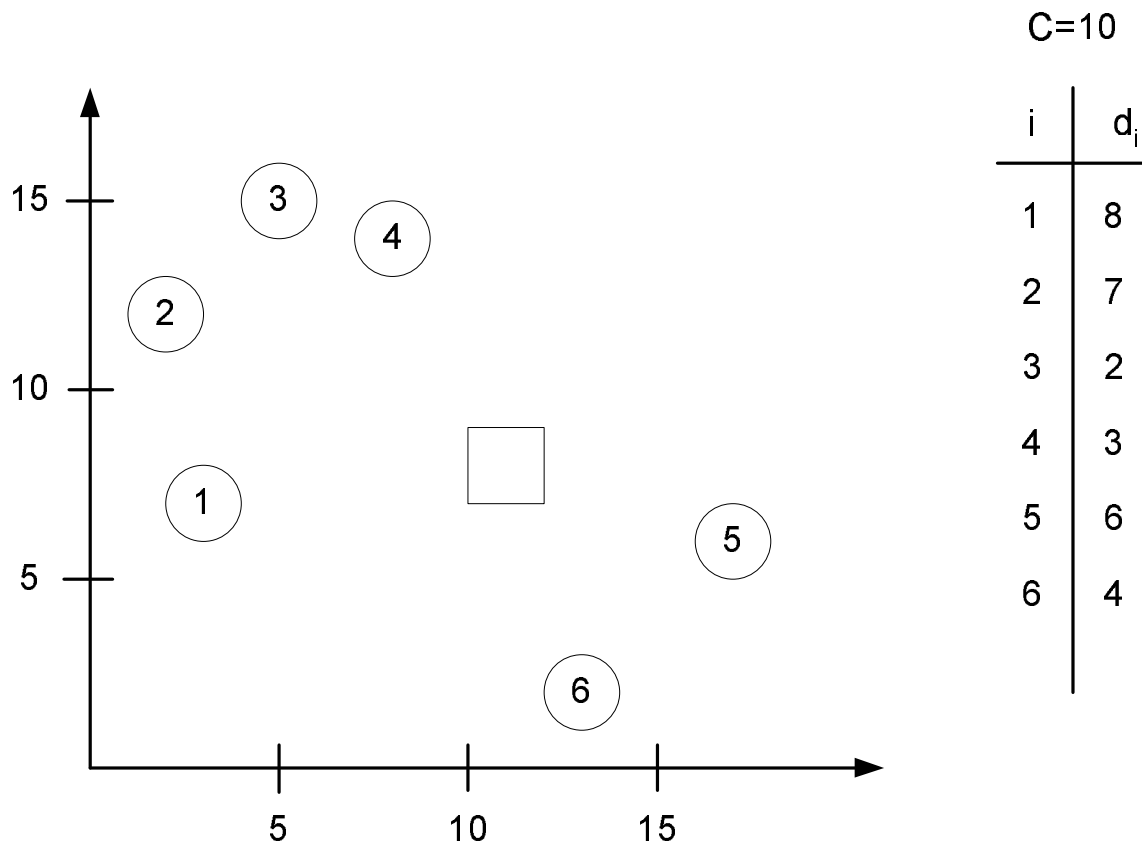
Subject to:

$$\sum_{j=1}^q a_{ij} x_j = 1 \quad \forall i \in V_0 \quad (15)$$

$$x_j \in \{0, 1\} \quad \forall j \in \{1, \dots, q\} \quad (16)$$

- 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 (see exercise).

Set partitioning example



- Feasible routes: $\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{1,3\}, \{2,3\}, \{2,4\}, \{3,4\}, \{3,5\}, \{3,6\}, \{4,5\}, \{4,6\}, \{5,6\}, \{3,4,6\}$.

$$\min 18x_1 + 26x_2 + 26x_3 + \dots + 34x_{13} + 34x_{14} + 24x_{15} + 42x_{16}$$

st

$$\begin{array}{rcccccc} 1x_1 + & & & & & & = 1 \\ & 1x_2 & & & & & = 1 \\ & & 1x_3 + & \dots & & & + 1x_{16} = 1 \\ & & & \dots + 1x_{13} + 1x_{14} & & & + 1x_{16} = 1 \\ & & & \dots + 1x_{13} & & + 1x_{15} & = 1 \\ & & & \dots & & + 1x_{14} + 1x_{15} + 1x_{16} & = 1 \end{array}$$

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 might be small because the problem only contains a few customers, or because 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 customers, where all routes with up to 7 customers 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 customer.
- 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*.

$$\min \quad cx \quad (17)$$

$$s.t. \quad Ax = b \quad (18)$$

$$x \geq 0 \quad (19)$$

- Reduced cost:

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

where c_j is the j 'th coefficient in the objective, π 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 $n + 1$. That is, a path that respects time windows and capacities. Thus we should look for these paths.

Column Generation cont.

- The feasible paths from node 0 to $n + 1$ are defined by:

$$\sum_{j \in \Delta^+(0)} x_{0j} = 1 \quad (20)$$

$$\sum_{i \in \Delta^-(j)} x_{ij} - \sum_{i \in \Delta^+(j)} x_{ji} = 0 \quad \forall j \in V_0 \quad (21)$$

$$\sum_{i \in \Delta^-(n+1)} x_{i,n+1} = 1 \quad (22)$$

$$w_i + t_{ij} - w_j \leq (1 - x_{ij})M_{ij} \quad \forall (i, j) \in E \quad (23)$$

$$a_i \sum_{j \in \Delta^+(i)} x_{ij} \leq w_i \quad \forall i \in V_0 \quad (24)$$

$$b_i \sum_{j \in \Delta^+(i)} x_{ij} \geq w_i \quad \forall i \in V_0 \quad (25)$$

$$a^d \leq w_i \quad \forall i \in \{0, n + 1\} \quad (26)$$

$$b^d \geq w_i \quad \forall i \in \{0, n + 1\} \quad (27)$$

$$\sum_{i \in N} d_i \sum_{j \in \Delta^+(i)} x_{ij} \leq C \quad (28)$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in E \quad (29)$$

$$w_i \geq 0 \quad \forall i \in V \quad (30)$$

- We want to find a column with minimum reduced cost

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

Column Generation cont.

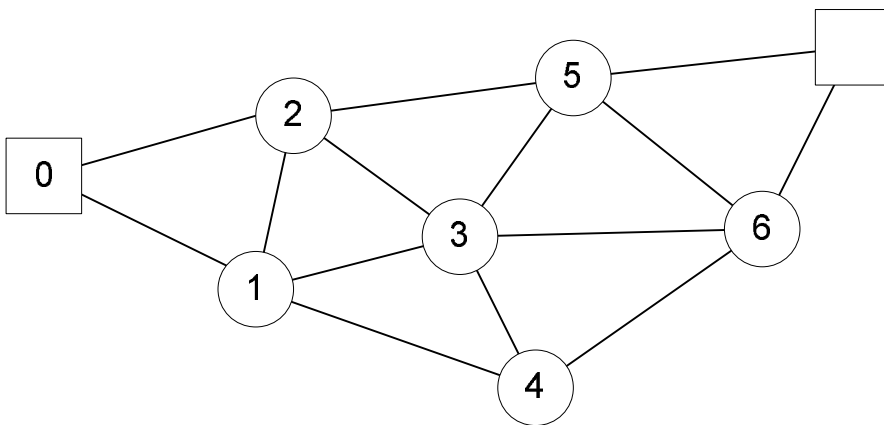
- So the objective of our subproblem is:

$$\begin{aligned} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} - \sum_{j=1}^n \pi_j \sum_{i \in V} x_{ij} \\ = \sum_{i \in V} \sum_{j \in V} (c_{ij} - \pi_j) x_{ij} \end{aligned}$$

- Where π_0 and π_{n+1} by definition is set to 0.
- Thus we have to solve an *elementary* shortest path problem with time windows, capacity constraints and negative edge weights (*ESPPTWCC*). Such a problem is NP-hard.
- A problem that is slightly easier is the shortest path problem with time windows, capacity constraints and negative edge weights (*SPPTWCC*). 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, but pseudo polynomial algorithms for the problem exists.
- When solving the SPPTWCC we must change our set partitioning to a set covering problem, and we notice that the lower bound potentially deteriorates as our solution space is widened.

Shortest path example

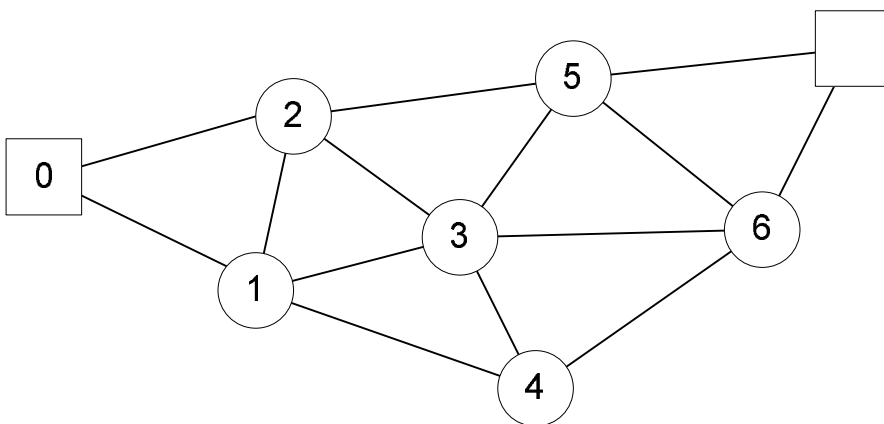
- Travel time and distance between nodes with drawn edges is set to 1. Travel time and distances between all other nodes is ∞ .



$$A_{\text{new}} = [\quad]$$

C=10

i	d_i	π_i	$[a_i, b_i]$
1	5	2	[0,100]
2	2	0	[0,100]
3	1	1	[0,100]
4	2	3	[3,6]
5	1	0	[0,100]
6	1	2	[0,100]

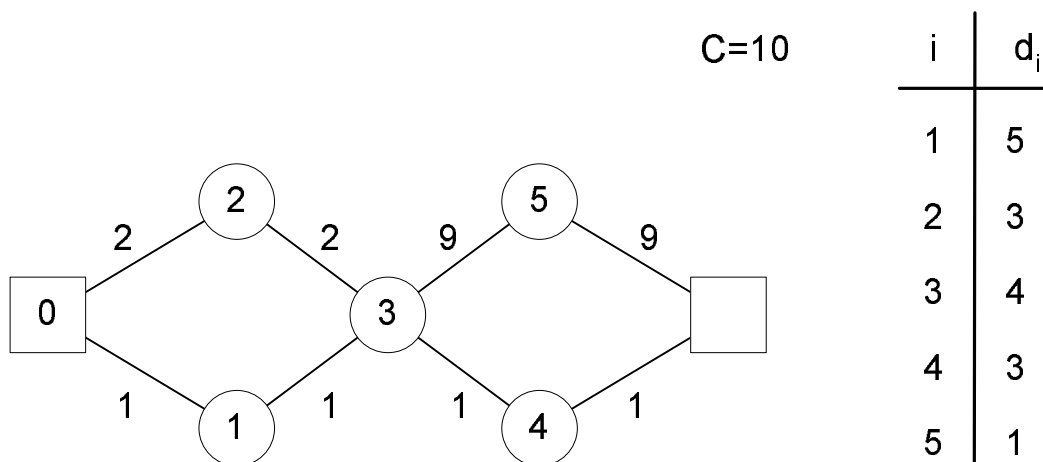


$$A_{\text{new}} = [\quad]$$

- Notice that when $\forall (i, j) \in E \ t_{ij} > 0$ or $\forall i \in V_0 \ d_i > 0$ infinite cycling is impossible.

Solving the SPPTWCC

- Normal shortest path from s to t where edge weights are positive - use Dijkstras algorithm:
 - $d[v]$: upper bound on shortest path from s to v . Initially $\forall i \neq s \quad d[v] = \infty$ and $d[s] = 0$.
 - Repeat until node t is processed:
 - * Process the node i among the unprocessed nodes where $d[i]$ is minimal.
 - (When processing a node i we check all the edges leaving the node to see if some of the $d[v]$'s at the endpoints can be improved by using the path from s to i).
- Dijkstras algorithm won't work when time windows or capacities are introduced even if all distances are non-negative.



Solving the SPPTWCC

- Instead we introduce labels (corresponding to $d[v]$). Each label represents a partial, feasible path. A label contains five elements:
 - i : The node where the path ends.
 - c : The cost of the path so far.
 - t : The time when service starts at node i when this path is taken.
 - d : The capacity used along the path.
 - p : A pointer to the label that this label was *extended* from.
- Given a label l and an edge e leaving the node of the label. We say that we create a new label by *extending* l along e . The new label represents the path we obtain by extending the path represented by l with the edge e . [Figure].
- We represent a label as the 5-tuple (i, c, t, d, p) .
- A label $x = (i_x, c_x, t_x, d_x, p_x)$ is said to *dominate* a label $y = (i_y, c_y, t_y, d_y, p_y)$ iff

$$\begin{aligned} & i_x = i_y \\ & \wedge (c_x \leq c_y \wedge t_x \leq t_y \wedge d_x \leq d_y) \\ & \wedge (c_x < c_y \vee t_x < t_y \vee d_x < d_y) \end{aligned}$$

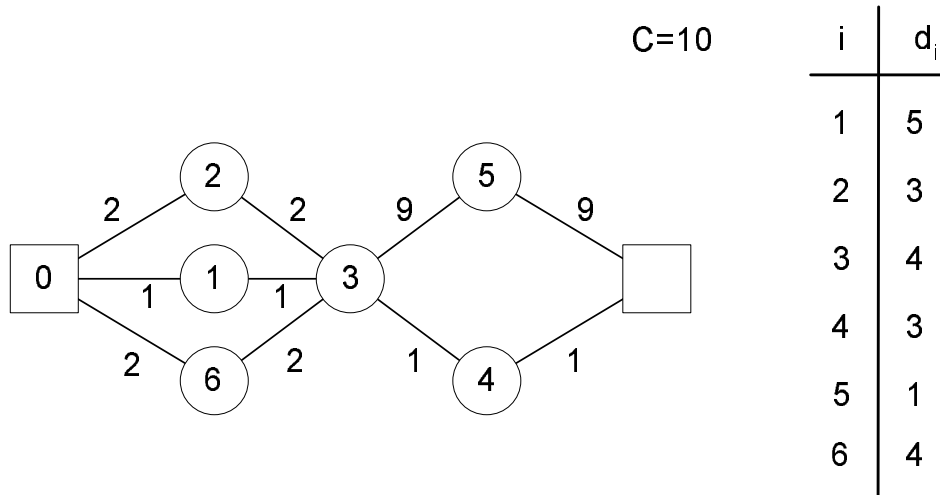
Solving the SPPTWCC

- If we encounter two labels x and y where x dominates y when searching for a SPTWCC then we can discard the y label as the corresponding path always could be exchanged with the path corresponding to label x without creating a path which is worse.
- In the SPPTWCC algorithm, labels are stored in a set of unprocessed labels and in a set containing processed labels. Initially the first set only contain the label $(0,0,\alpha^d,0, \text{NULL})$ which corresponds to the path from 0 to 0 while the second set is empty.
- In each iteration of the algorithm the unprocessed label with lowest time component is removed from the set of unprocessed labels and moved to the set of processed labels.
- The chosen label is *extended* along all edges leaving the node of the label. We thereby create a set of new labels. If any of the new labels are dominated by existing labels then these new labels are thrown away.
- If some of the unprocessed labels are dominated by any of the new labels then these unprocessed labels are discarded. Notice that the new labels cannot dominate the already processed labels.
- Finally the undominated, new labels are added to the set of unprocessed labels.

Solving the SPPTWCC

- The algorithm terminates when the set of unprocessed labels is empty. The shortest path from 0 to $n + 1$ can be found by going through the labels with node id $n + 1$.
- The algorithm is guaranteed to terminate due to the choice of next label to process and assumption on travel times.
- If travel times and demands are given by integer, then the maximum number of processed labels (and stored labels) is bounded by $O(n(b_d - a_d)C)$. The running time of the algorithm is $O(\text{poly}(n(b_d - a_d)C))$.
- An algorithm with such a running time is said to be *pseudo-polynomial*. (An algorithm is *pseudo-polynomial* if it is polynomial in $|I_u|$, where I_u is the instance coded such that all numbers are written in unary, and $|I_u|$ is the number of bits needed to represent the instance).

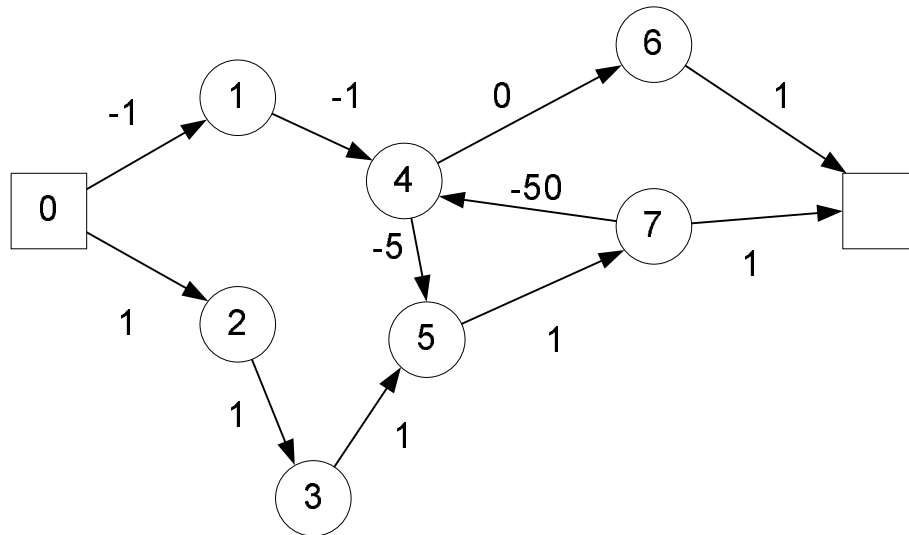
Solving the SPPTWCC



- As noted before, paths containing cycles can easily be constructed by our SPPTWCC algorithm. This lead to lower bounds that are worser compared to what could be obtained by using a ESPPTWCC algorithm.
- To improve the bounds one typically has used a modified version of the SPPTWCC algorithm that eliminates two-cycles. That is cycles of the form $i - j - i$.
- In order to do this, one has to introduce some extra dominance criterreas, making the algorithm slightly more complex. The overall running time of the shortest path algorithm is not increased much by eliminating two cycles while the lower bounds can be improved considerably.
- Recently [Irnich03] the cycle elimination principle has been generalized to eliminating k -cycles for $k \geq 3$. This can improve lower bounds even more. [Table]

Solving the ESPPTWCC

- For several years, most researchers have considered the ESPPTWCC to be too hard to solve for practical use in a column generation algorithm.
- Recently (2000-2003) it has been shown that the problem can be solved for many of the subproblems arising in a column generation algorithm for the VRPTW and that the bounds obtained make it possible to solve problems which couldn't be solved by methods using SPPTWCC.
- In order to solve a ESPPTWCC we have to extend our labels such that they contain n binary fields, one for each customer. Each binary field indicates if the corresponding customer has been visited by the path represented by the label.
- A label x can only be extended to a node i if the path represented by x has not visited i already.
- Ultimately, a label x can only dominate another label y if x dominates y in the traditional way and if the set (S_x) of customers visited by the path corresponding to x is a subset of the set of customers visited by the path corresponding to y (S_y),



- It's the new, weaker dominance criteria that make the ESPPTWCC so much harder to solve.
- Several “tricks” have been proposed to reduce the running time of the algorithm without sacrificing optimality.
- Example 1: If x dominates y in the traditional sense but x visits one customer i , not visited by y then we cannot discard either x or y . But if we can show that y cannot be extended to i without violating the capacity constraint or the time window of i then we can discard i . The same trick can be applied if x visits more than one customer not visited by y .

Solving the ESPPTWCC

- Example 2: Again, assume x dominates y in the traditional sense but x visits one customer i , not visited by y . If y can be extended to a path y^* to $n + 1$ with cost $c(y^*)$ then we would also be able to extend x^* to $n + 1$ by visiting the same customers as y^* but skipping node i . The triangle inequality and the fact that x dominates y in the traditional sense ensures that x^* is feasible. If we can show that $c(x^*) \leq c(y^*)$ then we can discard label y .
- We have that (p and q are the predecessor and successor of i in the path of y^*).

$$c(y^*) - c(x^*) = c_{pi} + c_{iq} - c_{pq} - \pi_i + c(y) - c(x)$$

If this number is greater or equal to zero then y can be discarded. If we have that

$$\pi_i + c(y) - c(x) \geq 0$$

then this would be satisfied (as c_{ij} satisfies the triangle inequality). This gives us another rule for discarding y .

Hard and Easy (E)SPPTWCC

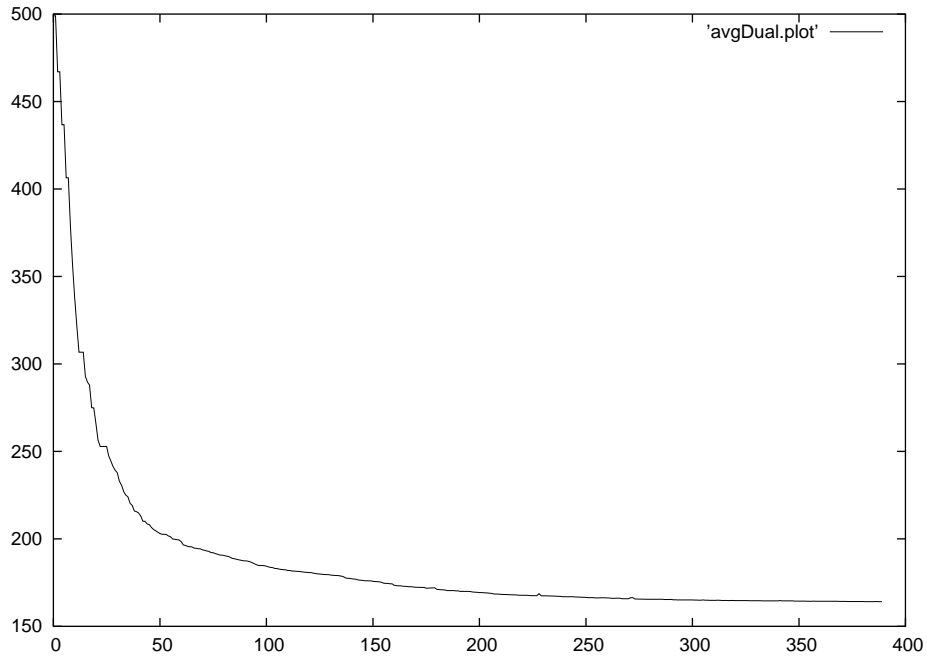
- Several properties of a SPPTWCC problem influences the practical hardness of the problem when solved by the algorithm described above. The table below summarizes these properties.

Easier	Harder
Small problems	Large problems
Sparse graph	Dense graph
Tight time windows	Large time windows
Few edges with negative cost	Many edges with negative cost

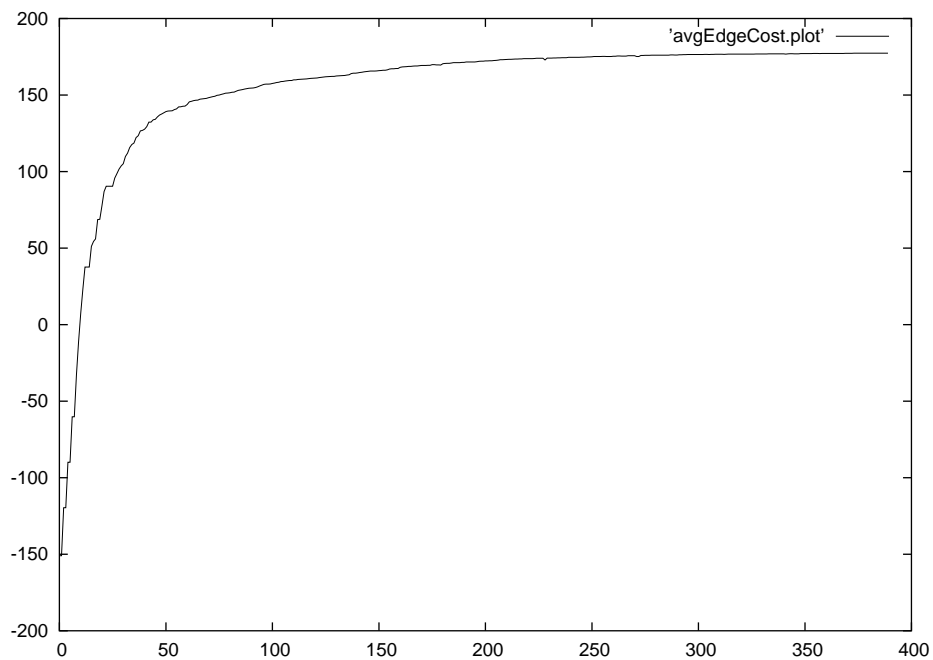
- It is hard to do much about the first three properties, although the preprocessing described at the end of this lecture can help a bit.
- The number of edges with negative cost and the magnitude of the negative costs are determined by the dual variables (recall that $\hat{c}_{ij} = (c_{ij} - \pi_j)$).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

Column generation R101, 100 cust.

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

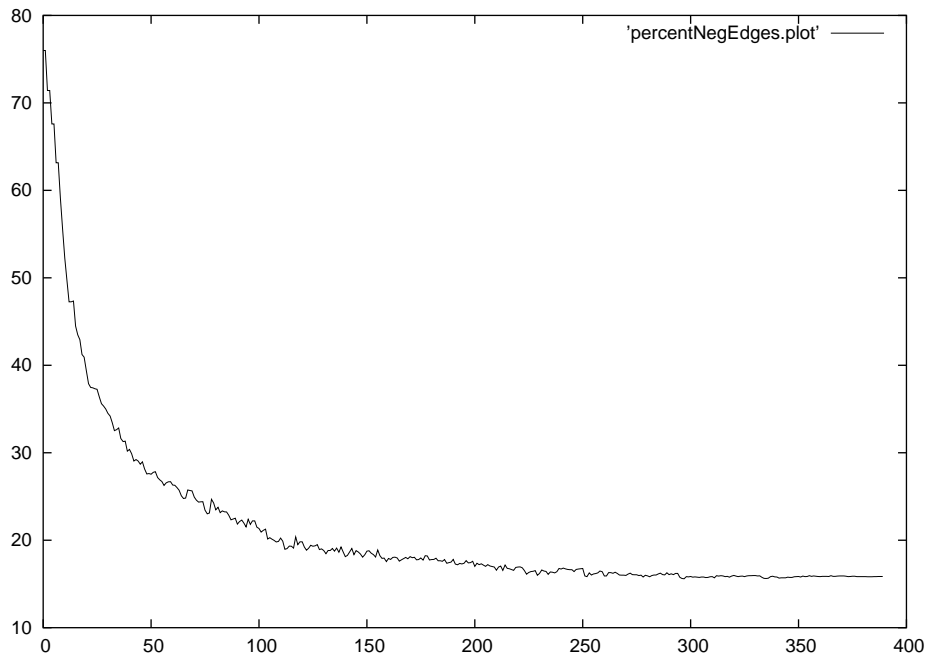


Average edge cost in graph for shortest path problem:

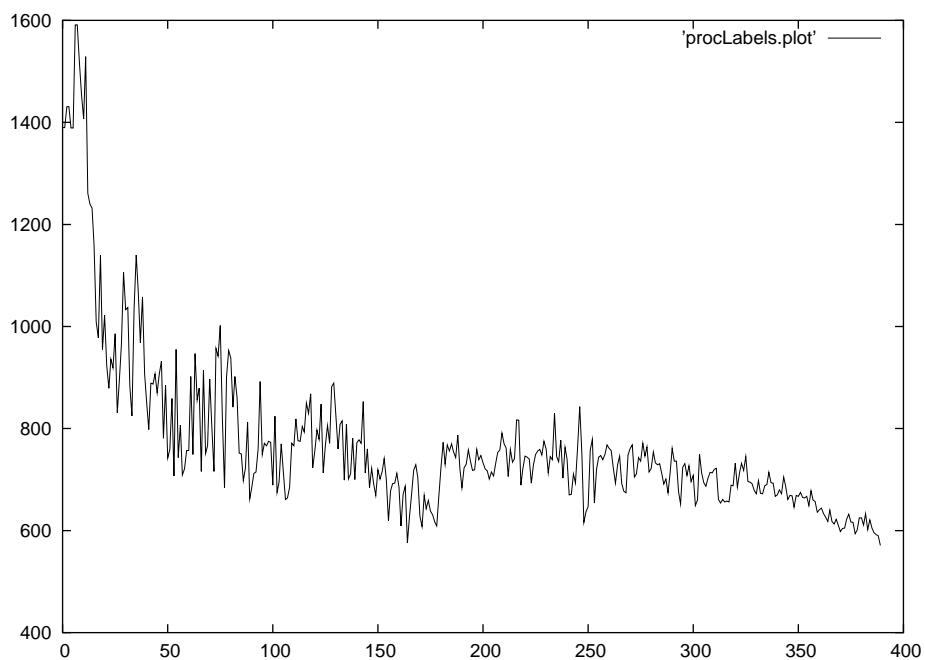


Column generation R101, 100 cust.

Percent negative edges in shortest path graph.

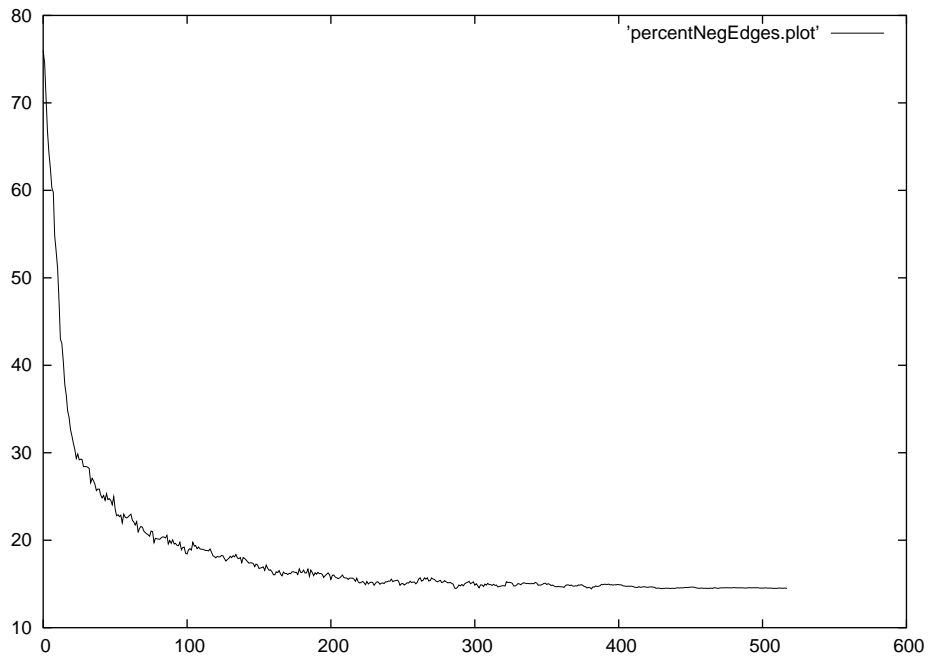


Number of labels processed in shortest path algorithm.

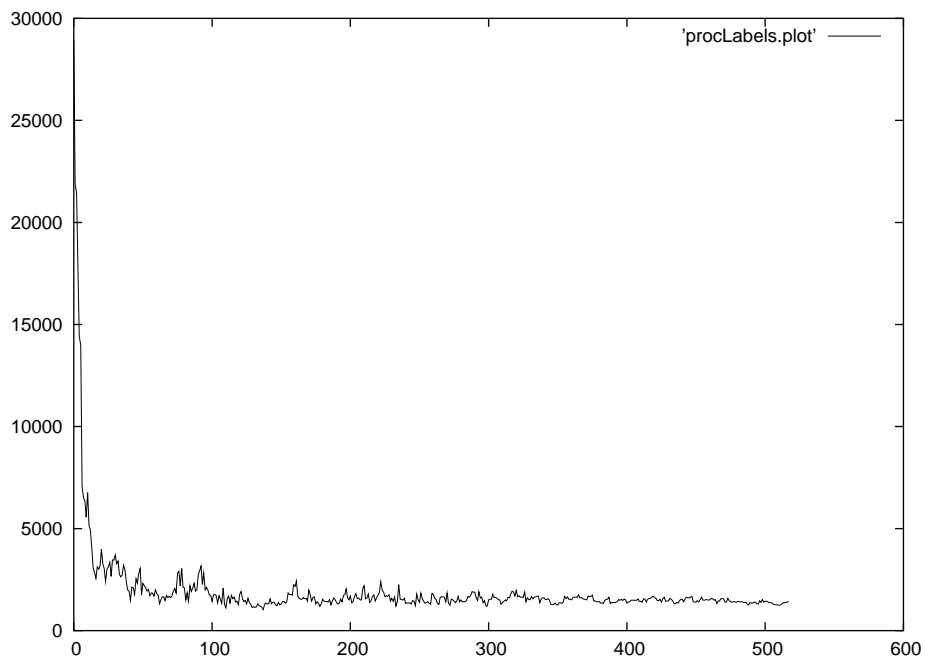


Column generation R102, 100 cust.

Percent negative edges in shortest path graph.

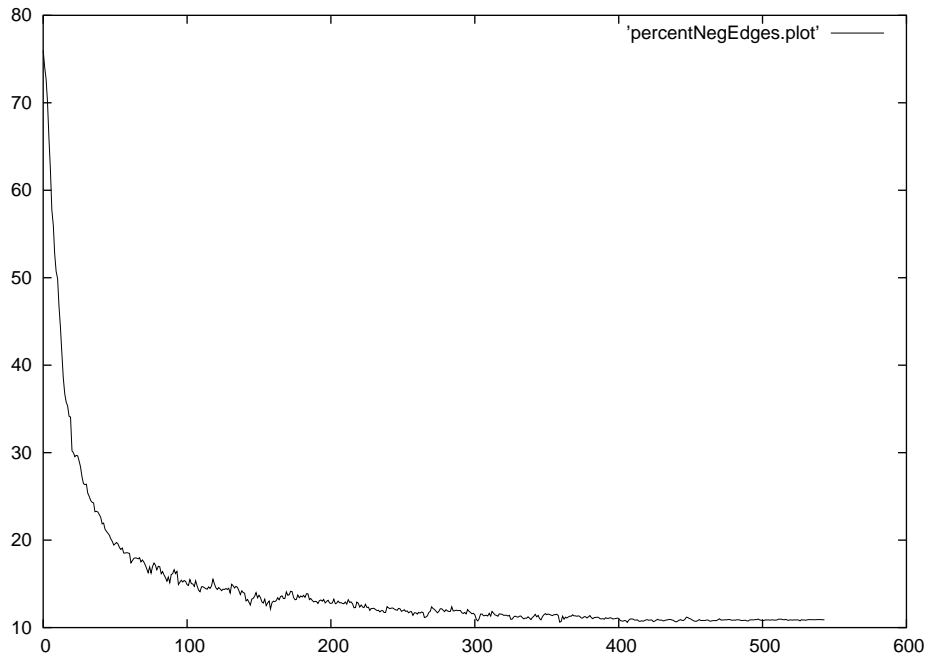


Number of labels processed in shortest path algorithm.

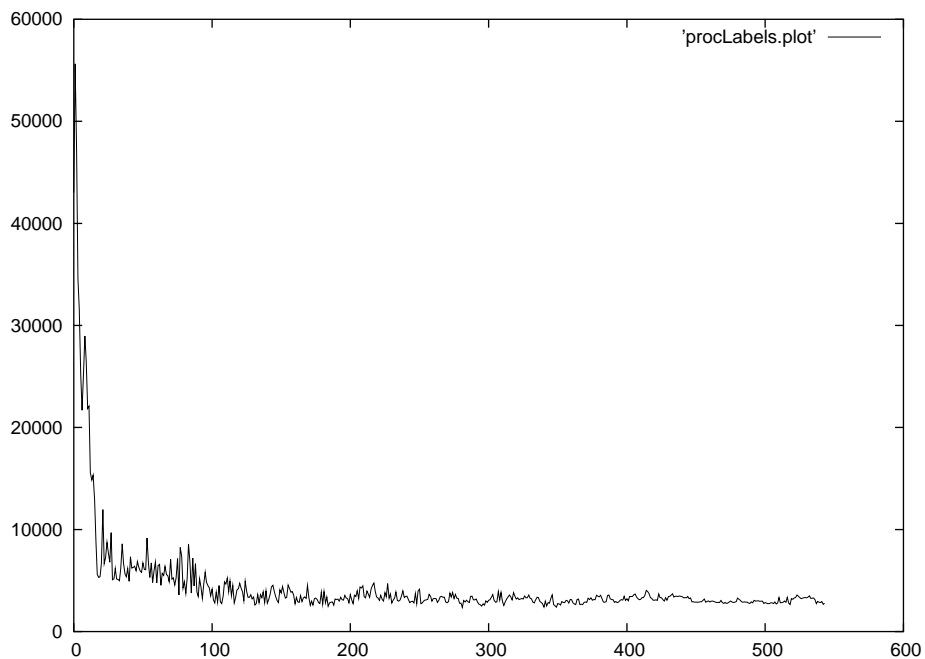


Column generation R103, 100 cust.

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 makes 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_{ijk} variables in the original 3-index formulation can be used in the column generation.
- Define flow variables:

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

- 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) .

Adding more inequalities to the model

- Any valid inequality in the x_{ijk} variables can be written as:

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

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

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

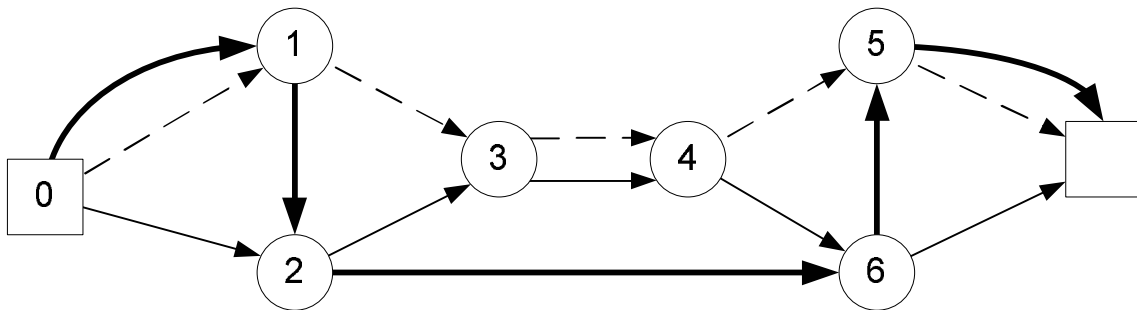
Thus the coefficient of the p 'th column is

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

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

k-path cuts

- Even though our LP relaxation is composed of elementary shortest paths we might get some “ugly” LP solutions as the example below illustrate.
- In this example, the LP relaxation uses the three routes $\{(0,1,2,6,5,7), (1,3,4,5,7), (2,3,4,6,7)\}$. The x_j variables corresponding to the three routes are all set to 0.5 such that the six customers are served by 1.5 vehicles.



- We can get rid of such solutions by adding valid inequalities.
- Recall *capacity inequalities* from SCVRP lecture one month ago.

$$\sum_{e \in \delta(S)} x_e \geq 2k(S) \quad \forall S \subseteq V \setminus \{0\}$$

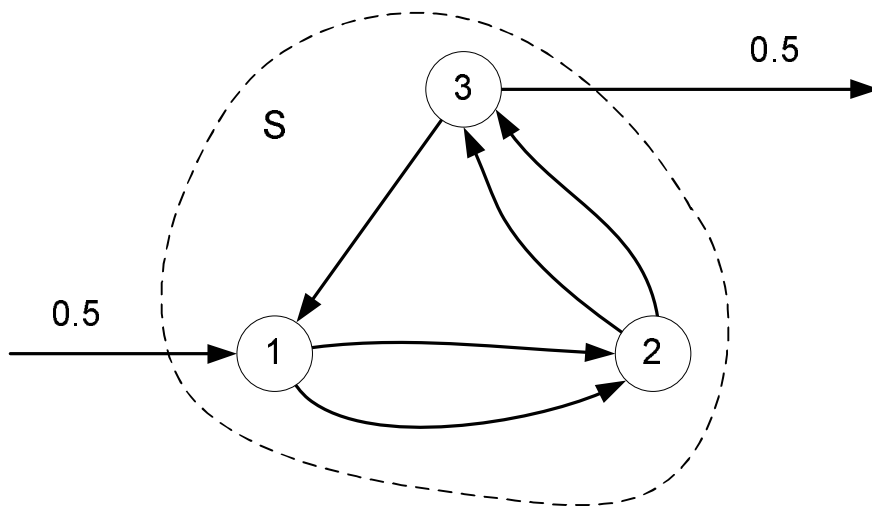
Where $k(S)$ is a lower bound on the minimum number of vehicles needed to serve S , for example $k(S) = \lceil d(S)/C \rceil$.

k-path cuts

- In the VRPTW problem where edges are directed, this inequality becomes:

$$\sum_{k \in K} \sum_{i \in S} \sum_{j \in V \setminus S} x_{ijk} = \sum_{i \in S} \sum_{j \in V \setminus S} f_{ij} \geq k(S)$$

- We can use the same $k(S)$ as used for CVRP. This would for example allow us to get rid of some situations that occur when using SPPTWCC calculations:



- Instead of just using the methods from SCVRP when calculating $k(S)$ it seems wise to also let the time windows be part of the calculation of $k(S)$, as the time windows often can be more constraining than the capacity constraints.

2-path cuts

- Proving that an inequality

$$\sum_{k \in K} \sum_{i \in S} \sum_{j \in V \setminus S} x_{ijk} = \sum_{i \in S} \sum_{j \in V \setminus S} f_{ij} \geq k(S)$$

is valid can be computational hard when $k(S)$ is determined using time window information as we will see below. Therefore we will restrict our attention to the situation where $k(S) = 2$.

- Given a set S how do we determine if at least two vehicles are necessary to serve the customers in S ?
- First we check capacities like in SCVRP.
- Secondly we see if it is possible to solve a *travelling salesman problem with time windows* (TSPTW) on the nodes in $S \cup \{0\}$. If a feasible solution to the TSPTW is found then we cannot use S as the set can be served by only one vehicle.
- On the other hand, if it turns out that no feasible TSPTW tour exists in the set, then we know that $k(S)$ at least is two, and we have identified a valid inequality.

2-path cuts

- The TSPTW problem is NP-hard, but small instances of the problem can be solved quickly by using dynamic programming.
- Notice that we can start by solving the TSPTW heuristically to see if a TSPTW tour can be found - we only need to use an exact method when the heuristic cannot find a tour, this can speed up the search for violated valid inequalities.
- Violated inequalities can be found using a heuristic like the one described when discussing SCVRP. [Kohl95] describes two other (but related) separation algorithms.
- If we want to identify cuts where the $k(S) \geq 3$ then it is no longer enough to solve the TSPTW. In that case we have to solve a VRPTW on the set of nodes given by $S \cup \{0\}$ with the objective of minimizing the number of vehicles needed to serve the customers.
- If the objective of such a minimization problem is z vehicles, then we know that $k(S) = z$ provides a valid inequality.
- This approach was used in [Cook99] where inequalities with $k(S) = 5$ was identified. These valid inequalities allowed the authors to solve several previously unsolved problems, but the time requirements were quite high. [Table]

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 customers 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.

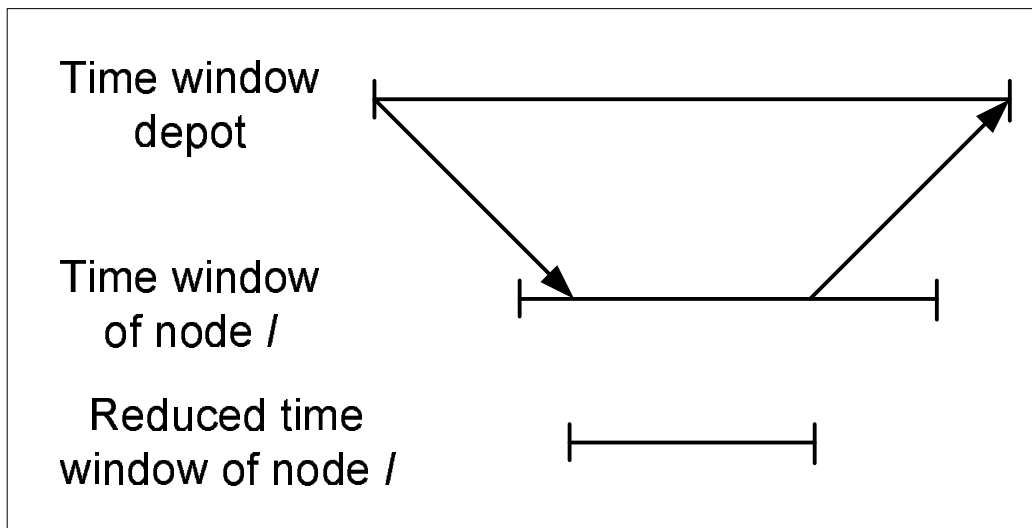
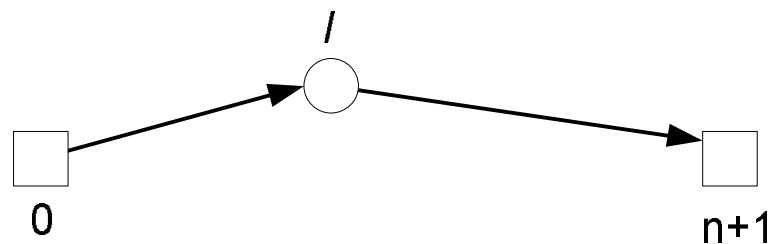
Branching cont.

- 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) .
- The problem with this branching scheme is that it creates unbalanced sub problems.

Preprocessing

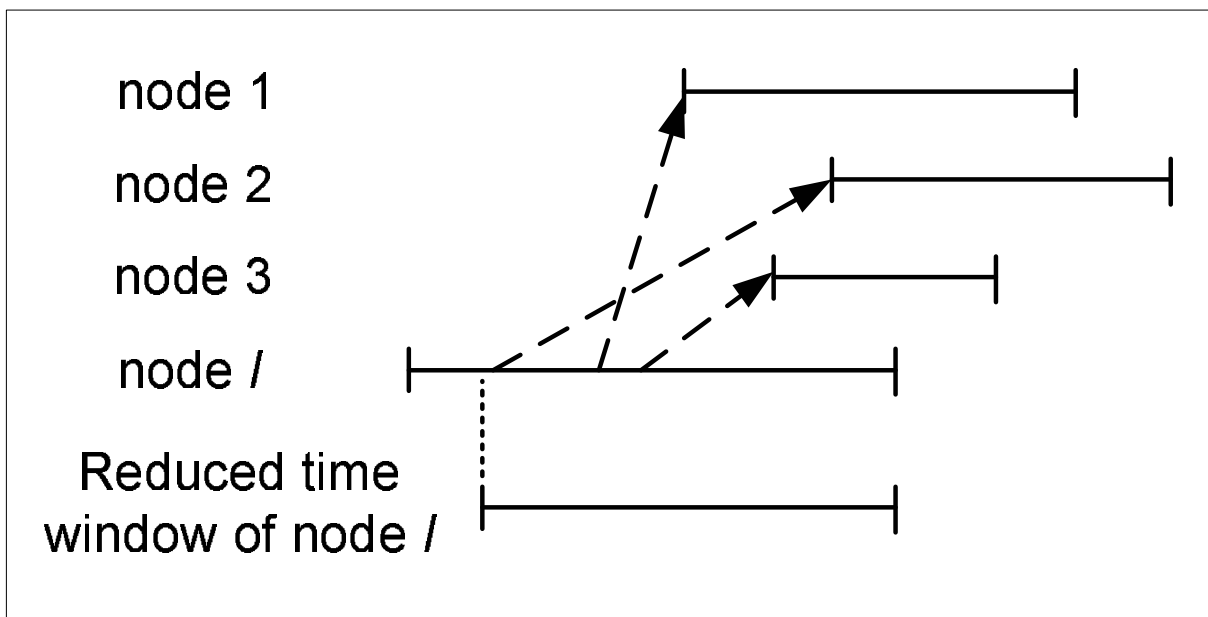
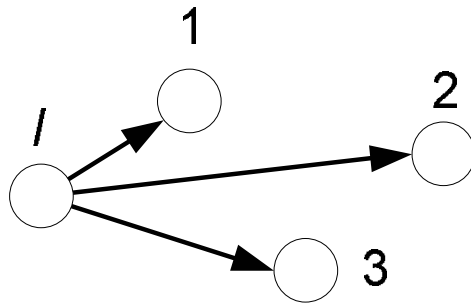
- General idea: Reduce the size or complexity of a problem before applying a time consuming method like branch and bound.
- In VRPTW problems, the size of the time windows can often be reduced by some simple rules (remember that travel times satisfy the triangle inequality):
- Rule 1: Depot time window:

$$\forall l \in N : [a_l, b_l] = [\max\{a_l, a_0 + t_{0l}\}, \min\{b_l, b_{n+1} - t_{l,n+1}\}]$$



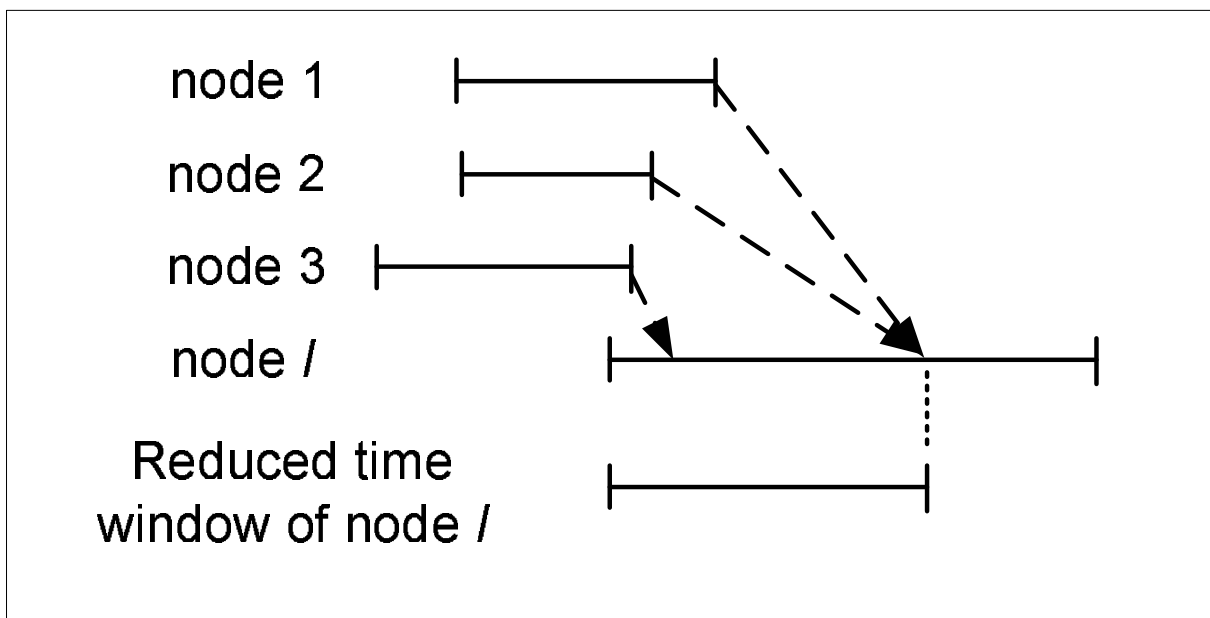
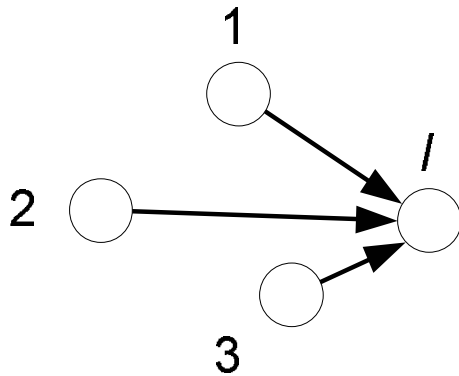
- Rule 2: Arrival time to successors:

$$\forall l \in N : a_l = \max \left\{ a_l, \min \left\{ b_l, \min_{(l,j) \in E(N)} \{ a_j - t_{lj} \} \right\} \right\}$$



- Rule 3: Departure time from predecessors:

$$\forall l \in N : b_l = \min \left\{ b_l, \max \left\{ a_l, \max_{(i,l) \in E(N)} \{b_i + t_{il}\} \right\} \right\}$$



- Rule 2+3 are applied repeatedly until no more reductions can be found.

Results

Ref	Type1	Type2	Large
[Des92]	50	-	
[Kohl95]	70	-	
[Larsen99]	75	17	
[Cook99]	80	30	7x200
[Kalle01]	80	42	7x200, 1x400, 1x1000
[Irnich03]	84	59	
[Chabrier03]	-	59	
[Danna03]	-	+1	
#probs	87	81	

Conclusion

- VRPTW is a hard problem!
- Designing a successful algorithm requires many components.
- Recent “inventions” have led to improved results. Solving the elementary shortest path problem or eliminating “large” cycles seems promising.
- Lagrangean decomposition proposed by Kallehauge et al. is another promising direction (we have skipped this topic in this lecture).

Reading

The following chapters can be skipped.

3.4.2

4.2.3

4.3

Furthermore, chapter 3.3.2 can be read briefly.

References

[Des92] M. Desrochers, J. Desrosiers, M. Solomon, *A New Optimization Algorithm for the vehicle Routing Problem with Time Windows*, Operations Research vol. 40, No. 2, 1992.

[Kohl95] N. Kohl, *Exact Methods for Time Constrained Routing and Related Scheduling Problems*, Ph.D. Thesis, IMM, DTU (IMM-PHD-1995-16).

[Cook99] W. Cook, J.L. Rich, *A parallel cutting-plane algorithm for the vehicle routing problem with time windows*, Technical Report, TR99-04, Department of Computational and Applied Mathematics, Rice University, 1999.

[Larsen99] J. Larsen, *Parallelization of the Vehicle Routing Problem with Time Windows*, IMM, DTU, 1999.

[Kalle00] B. Kallehauge, *Lagrange dualitet og ikke-differentiabel optimering*, Eksamensprojekt, IMM, DTU, 2000.

[Kalle01] B. Kallehauge, J. Larsen, O.B.G. Madsen, *Lagrangian Duality Applied to Vehicle Routing with Time Windows*, Technical report, 2001.

[Chabrier03] A. Chabrier, *Vehicle Routing Problem with Elementary Shortest Path based Column Generation*, Technical Report, 2003.

[Danna03] E. Danna, *Accelerating Branch-and-Price with Local Search: A Case Study on the Vehicle Routing Problem with Time Windows*. ILOG Technical Report 03-006.

[Feillet03] D. Feillet, P. Dejax, M. Gendreau, C. Gueguen, *An Exact Algorithm for the Elementary Shortest Path Problem with Resource Constraint: Application to some Vehicle Routing Problems*. Technical Report, 2003.

[Irnich03] S. Irnich, D. Villeneuve, *The Shortest Path Problem with Resource Constraints and k -Cycle Elimination for $k \geq 3$* , Technical Report, 2003.