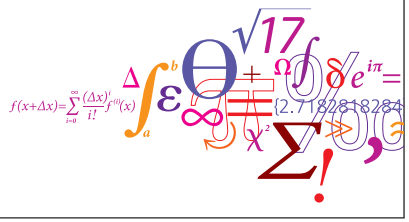
# Resource Constrained Shortest Path

Jesper Larsen[1]

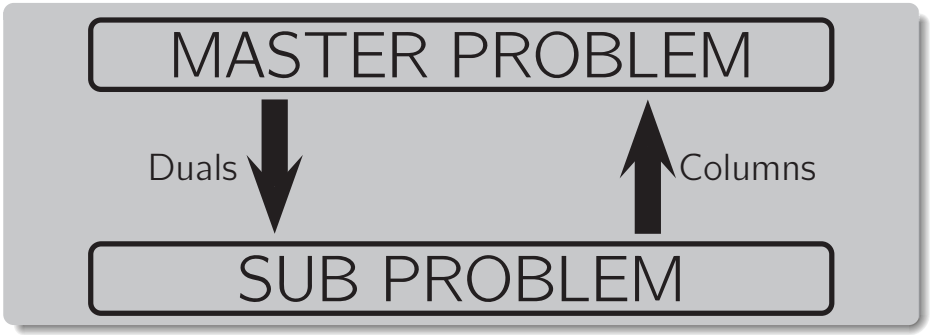
[1]Department of Management Engineering
Technical University of Denmark

42134 Advanced Topics in Operations Research



DTU Management Engineering
Department of Management Engineering

## The Column Generation Framework




### Specific for our SPP/SCP

- Let $P_j$ be the set of indices for which we for column $j$ has a 1 in the column.
- Now the reduced cost can be written as $\tilde{c}_j = c_j - \alpha^T A_j = c_j - \sum_{i \in P_j} \alpha_i$



## A scheduling problem (again)

- We have 6 assignments A, B, C, D, E and F, that needs to be carried out. For every assignment we have a start time and a duration (in hours).
- Minimize the total cost of carrying out the 6 assignments.

### A Workplan

- A workplan is a set of assignments. Now we want to formulate a mathematical model that finds the cheapest set of workplans that fulfills all the assignments.
- The cost of a workplan is $\max(4.0, L)$ where $L$ is the length of the workplan.
- *Let us consider a slightly modified problem by simple letting the cost of a workplan be equal to L.*



## The Integer Linear Programming Model

We get the following model:

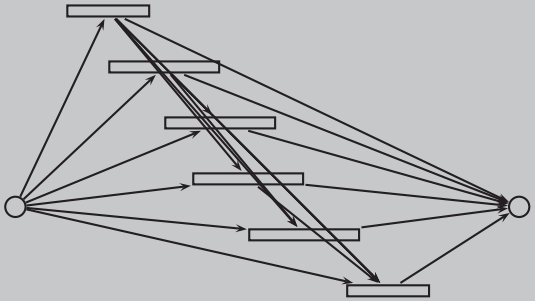
$$\begin{array}{ll} \min & \sum_{j=1}^{N} c_j x_j \\ \text{s.t.} & \sum_{j=1}^{N} a_{ij} x_j = 1 \quad \forall i \\ & x_j \in \{0,1\} \end{array}$$

- $x_j = 1$ if we use workplan $j$ and 0 otherwise (variable).
- $c_j$ is equal to the cost of the plan (parameter).
- $a_{ij} = 1$ if assignment $i$ is included in workplan $j$ (parameter).

## Finding and pricing the assignments

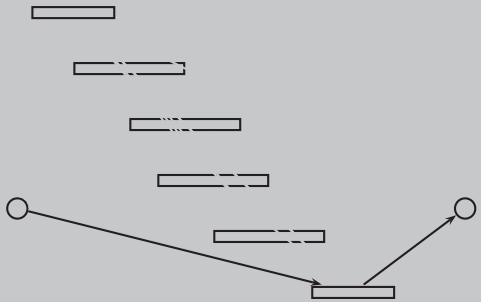
### Assignment data

| Assignment | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| Start | 0.0 | 1.0 | 2.0 | 2.5 | 3.5 | 5.0 |
| Duration | 1.5 | 2.0 | 2.0 | 2.0 | 2.0 | 1.5 |





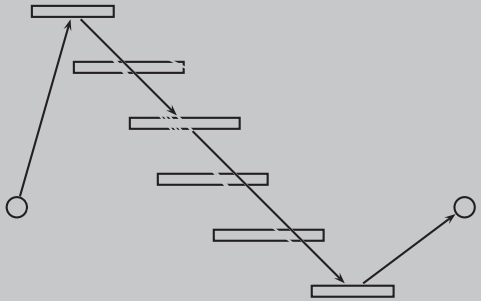
## Pricing a simple workplan



- What is the price of the path $St \rightarrow F \rightarrow Fi$?
- $\tilde{c} = 0.5 + 1.5 - \alpha_F$ where $\alpha_F$ is the dual variable of the constraint corresponding to assignment F.



## Pricing another workplan



$$\begin{aligned}\tilde{c} &= 0.5 + (1.5 + 0.5) + (2.0 + 1.0) + 1.5 - \alpha_A - \alpha_C - \alpha_F \\ &= 0.5 + (2.0 - \alpha_A) + (3.0 - \alpha_C) + (1.5 - \alpha_F)\end{aligned}$$



## Structure of the subproblem

### Constructing the network

- We add start and finish nodes and add edges from start to all assignments and from all assignments to finish.
- We add edge $(i, j)$ to the network if assignment $j$ can follow directly after assignment $i$.
- For each outgoing edge of assignment $i$ set the cost equal to the duration of the assignment *plus* waiting time *minus* the associated dual variable.
- Check-in and -out can be picked up by the start node.

## Structure of the subproblem

### Objective

- Find the most favorable path from Start to Finish.
- This is a shortest path problem.

### Note

Some edges can have negative length, but the graph is acyclic.



## Crew Scheduling

### Generalization

- The example can be generalized to solve one of the biggest success stories for Operations Research, namely, aircrew scheduling.
- In crew scheduling we want to assign pilots and cabin crew to flights.
- Due to the vast complexity of the problem it is split into two phases: the crew pairing problem and then the crew rostering problem.
- In the crew pairing problem we build anonymous "blocks" of work that are assigned to specific people in the crew rostering problem.



## Crew Pairing I

### Aim

The aim is to produce pairings (work plans from home base and back to home base) of low cost.

### The Master Problem

- Each constraint make sure we cover a given flight.
- May also include some home base constraints.
- Each variable (column) is a feasible pairing.



## Crew Pairing II

### The Subproblem

- Can be viewed as a graph. Each flight is a node. In addition we have a start and a finish node for each home base in the problem.
- Edge costs are based on crew cost, overnight cost etc.
- Again we need to find a path from start to finish with a favourable cost.
- The subproblem can be posed as a extension of the shortest path problem

### Note

Some edges can have negative length, but the graph is acyclic.

## Crew Rostering I

### Aim

The aim is to produce personal rosters for each individual crew member in order to maximize some common or personal "happiness".

### The Master Problem

- Each constraint make sure we cover a given pairing.
- Furthermore we need a GUB for each person.
- Each variable (column) is a feasible roster.



## Crew Rostering II

### The Subproblem

- The subproblem will consist of nodes (pairings and other types of duties). In addition we have a start and a finish node for each home base in the problem.
- Cost is "happiness".
- The subproblem can be posed as an extension of the shortest path problem.

### Note

- Some edges can have negative length, but the graph is acyclic.
- One subproblem for each crew member.

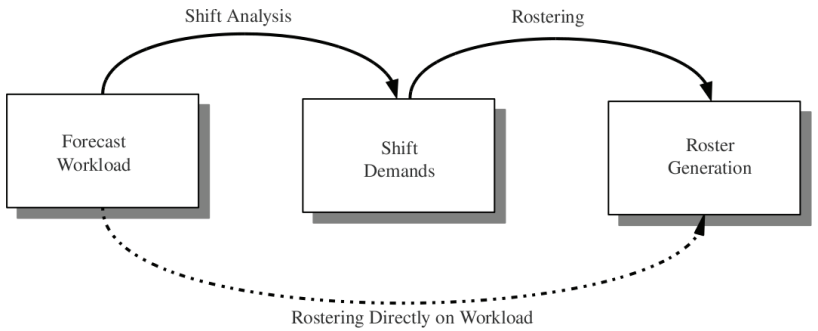


## The Ground Crew Rostering Problem with Work Patterns

- Manpower planning problem arising in the ground operations of airlines
- Construct a *roster* that covers the anticipated workload as well as possible
- A *roster-line* is a sequence of *shifts* that an employee works
- A shift is associated with a specific task and is usually 9 hours in duration
- Roster-lines must conform to a *work pattern* (i.e. 6&3)
  - Specifies both the *on-stretch* and the *off-stretch*
  - Can be staggered across employees giving different *pattern groups*
- Problem is similar in structure to nurse rostering
  - Less concerned with individual preferences
- Rostering horizon is 6 months



## Roster Generation Process

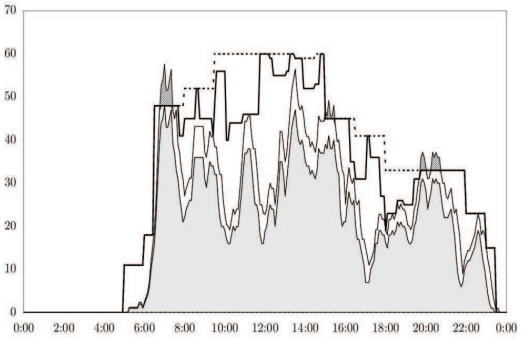



- Using the forecast workload directly circumvents shift demand calculation
  - Number of employees working any shift is obtained as a "bi-product"
  - More flexible from a robustness modelling perspective

# Example

- Input: A set of shifts with given demands, forecast workload
- Ouput: Covered workload (subject to breaks) in the form of roster-lines

| Shift | Start | End | Demand |
|---|---|---|---|
| N1 | 04:45 | 13:15 | 0 |
| E1 | 05:00 | 13:30 | 13 |
| E2 | 06:00 | 14:30 | 13 |
| E3 | 06:30 | 15:00 | 16 |
| D1 | 08:00 | 16:30 | 14 |
| M1 | 09:30 | 18:00 | 5 |
| L1 | 13:30 | 22:00 | 11 |
| L2 | 14:30 | 23:00 | 10 |
| L3 | 15:00 | 23:30 | 8 |




| | Mon | Tue | Wed | Thu | Fri | Sat | Sun | Mon | Tue | Wed | Thu | Fri | Sat |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Person | 06-aug | 07-aug | 08-aug | 09-aug | 10-aug | 11-aug | 12-aug | 13-aug | 14-aug | 15-aug | 16-aug | 17-aug | 18-aug |



# Mathematical Model - Shift Demand

$$\min \sum_{r\in\mathcal{R}} c_r x_r + \sum_{s\in\mathcal{S}} \check{c}_s u_s$$

s.t.

$$\sum_{r\in\mathcal{R}} a_{sr} x_r + u_s \geq q_s \qquad \forall s \in \mathcal{S} \quad (\pi_s)$$

$$\sum_{r\in\mathcal{R}} a_{gr} x_r \leq m_g \qquad \forall g \in \mathcal{G} \quad (\mu_g)$$

$$\sum_{r\in\mathcal{R}} x_r \leq n \qquad (\gamma)$$

$$u_s \geq 0 \qquad \forall s \in \mathcal{S}$$

$$x_r \in \mathbb{Z}_+ \qquad \forall r \in \mathcal{R}$$



# Column Generation

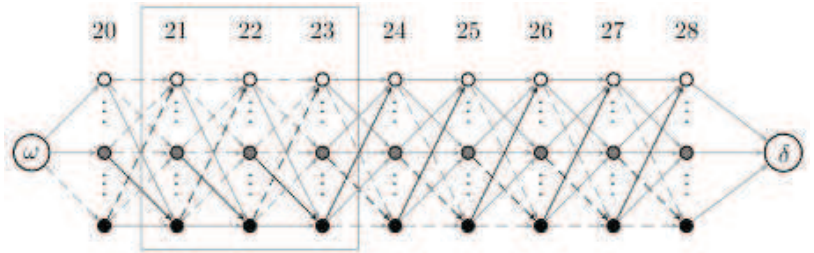
- Decomposition technique for solving large scale linear programmes
- Problem is split into a *Master* problem and a *Pricing Problem*
- The master problem finds an optimal allocation of a roster-lines
  - Considers only a subset of roster-lines
  - $x_r \in \mathbb{Z}_+$ is replaced with $x_r \in \mathbb{R}_+$
- The pricing problem dynamically identifies favourable roster-lines
  - Modelled using an acyclic network
  - Nodes correspond to shifts while arcs correspond to shift transitions
  - Enforces all hard requirements for a roster-line (e.g. pattern)
  - Attempts to satisfy soft constraints (e.g. late to early sequence)

$$\min_{r\in\mathcal{R}} \left( c_r - \sum_{s\in\mathcal{S}} a_{sr}\pi_s - \sum_{g\in\mathcal{G}} a_{gr}\mu_g - \gamma \right)$$



# Pricing Problem

- 6&3 pattern gives 9 independent subproblems
- Consider the following pricing problem and group [1,1,1,0,0,0,1,1,1]




- Subproblem network collapses at first day of an off-stretch
- Must solve a resource constrained shortest path (max consec nights)

## Vehicle Routing with Time Windows

### Problem definition

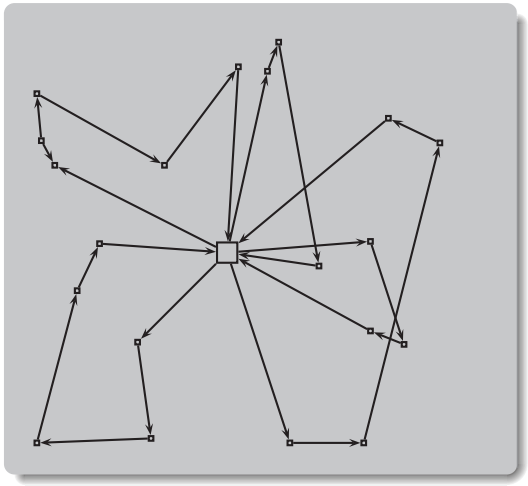
- Given is a fleet of vehicles $F$ and a set of customers $N$.
- Each customer (node) $i \in N$ has to be supplied from a depot with some quantity $q_i$, and the delivery has to take place in the time window $[a_i; b_i]$.
- All vehicles are identical with a capacity of $Q$.

### Objective

The aim is to produce routes of minimal accumulated length that does not violate capacity restrictions and that visits all customers exactly ones.



## An example



- The depot
- The customers
- ... and some routes



## The Master Problem

### A Set Partitioning Formulation

We get the following model:

$$\begin{array}{ll} \min & \sum_{j=1}^{N} c_j x_j \\ \text{s.t.} & \sum_{j=1}^{N} a_{ij} x_j = 1 \quad \forall i \\ & x_j \in \{0,1\} \end{array}$$

- $x_j = 1$ if we use route $j$ and 0 otherwise (variable).
- $c_j$ is equal to the cost of route $j$ (parameter).
- $a_{ij} = 1$ if customer $i$ is visited by $j$ (parameter).

The billions and billions of feasible paths makes column generation necessary.



## Solving the subproblem

### Structure

- Like in scheduling if we evaluate the reduced cost of a path we get $\sum_{\{(i,j) \text{ in the path}\}} c_{ij} - \sum_{\{i \text{ in the path}\}} \alpha_i$.
- A column is a path from depot to depot visiting certain customers, therefore what we are looking for is a "favorable" path from depot to depot.
- Terms can be moved around so that we get that the reduced cost of a path is the sum of "reduced costs of the edges" in the path.
- The pricing problem is therefore a **shortest path problem with capacity constraints and time windows**.

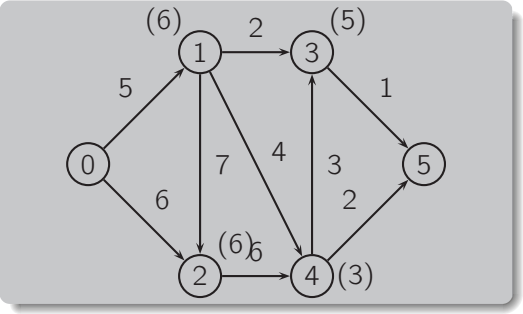
### Note

Edges can have negative length, and the graph can be **cyclic**.

## Resource Constrained Shortest Path

### An example

Let us assume we are solving a subproblem that can be stated as a resource constrained shortest path. Let us furthermore assume the underlying graph is **acyclic**.



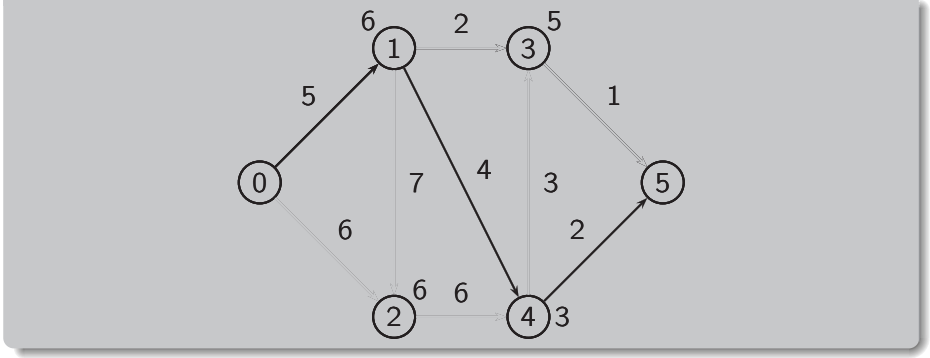

### Resource limit

Assume there is a resource limit of 10 units.



## Using Dijkstra

As an initial idea let us simply forget about the resource. Then the problem is reduced to an ordinary shortest path problem where we can use Dijkstra.






## Dynamic Programming for RCSP

### Dijkstra's algorithm

- The Dijkstra algorithm can be described by the dynamic program:

$$\begin{aligned} d_s &= 0, \\ d_j &= \min_{(i,j)\in\mathcal{A}} \{d_i + l_{ij} \,:\, j \in V \setminus \{s\}\}. \end{aligned}$$

- Dijkstra constructs the shortest path by extending "good" paths
- By checking whether $d_i + l_{ij} < d_j$ holds or not we try to extend only the "good"paths, the remaining paths are not extended any further.



## Label Setting Algorithm

### Introduction

- Labelling algorithms associate labels with partial paths and creates new ones extending the paths.
- When no more labels can be generated, the optimal path is given by the label at the end node with the lowest cost.

### A Label

- At some node $i$, a partial path $p$ is associated with a label $E_p = (rc_p, D_p^0, D_p^1, \ldots, D_p^L)$.
- $D_p^0, D_p^1, \ldots, D_p^L$ are called **resources**
- Labels "consumes" or "accumulates" resources as they pass edges.

## Basic algorithm

### Simple algorithm

1. Make the first label
2. Find an untreated label with *smallest* accumulated consumption of a strictly increasing resource, and extend it to new labels as long as all resource limits are not exceeded.
3. If more untreated labels exist go to step 2, else stop

### Our Example

For our small example we need:

- One resource for the accumulated demand.
- Furthermore in order to reconstruct the final path we also need a reference back to the immediate predecessor and the current node.



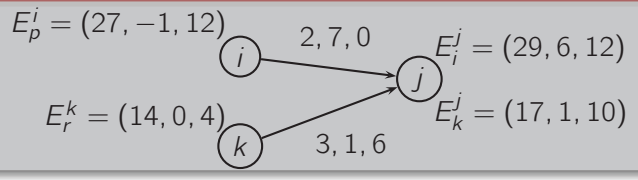
## Dominance

### Dominance rule

- Certain rules are used to fathom partial partial paths $p$ that are known not to be part of the searched optimal path.
- A partial path $p_1$ dominates another partial path $p_2$ ending at the same node $i$, if the following conditions are respected:

$$rc_1 \leq rc_2$$
$$D_1^l \leq D_2^l \qquad \forall l \in \{0, 1, \ldots, L\}$$

### Example






## What if the graph is cyclic?

### Challenges

- When we solve resource constrained shortest path problems as subproblems in column generation edge costs might be negative.
- In acyclic graphs this is not a problem.
- but in graphs with cycles there might be cycles with negative costs.

### Elementary Shortest Path Problem....

In the Elementary Shortest Path Problem with Resource Constraints each node can at most appear once in a path.



## Implementing ESPPRC

### Using Resources

- Assign a resource to every node in the graph.
  - The resource is 0 if the node has not been visited
  - and 1 is the node has been visited once.
- the resource has an upper bound of 1.

### Dominance

- Now it gets much more difficult to dominate.
- Given two labels $L_1$ and $L_2$. $L_1$ dominates $L_2$ if $rc_1 \leq rc_2$ and $D_1^l \leq D_2^l$ for all resources $l$.
- This means that if $L_1$ is dominating $L_2$ the nodes visited by partial path represented by $L_1$ is a subset of the nodes visited by the partial path represented by $L_2$.

## Elementary Shortest Path with Resource Constraints

### Capacity Constraints and Time Windows

- Let us focus on the subproblem where we want to find the shortest path with capacity constraints and time windows.
- So each customer has a demand $q_i$ and a time window for delivery $[a_i; b_i]$.
- With this subproblem and without changing the master problem we are solving the Vehicle Routing Problem with Time Windows.

### Capacity constraints and Time Windows

For routing with time windows we need the following resources:

- One resource for accumulated capacity and one for accumulated time
- A resource for visiting each node
- Furthermore a reference back to the immediate predecessor and the



## Mathematical Model of ESPPTWCC

### The Integer Linear Programming Model

$$\min \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} c_{ij}x_{ij}, \qquad s.t. \tag{1}$$

$$\sum_{i\in\mathcal{C}} d_i \sum_{j\in\mathcal{N}} x_{ij} \leq q \tag{2}$$

$$\sum_{j\in\mathcal{N}} x_{0j} = 1 \tag{3}$$

$$\sum_{i\in\mathcal{N}} x_{ih} - \sum_{j\in\mathcal{N}} x_{hj} = 0 \qquad \forall h \in \mathcal{C} \tag{4}$$

$$\sum_{i\in\mathcal{N}} x_{i,n+1} = 1 \tag{5}$$

$$s_i + t_{ij} - K(1 - x_{ij}) \leq s_j \qquad \forall i,j \in \mathcal{N} \tag{6}$$

$$a_i \leq s_i \leq b_i \qquad \forall i \in \mathcal{N} \tag{7}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in \mathcal{N} \tag{8}$$



## Further Remarks

### Solve to Optimality

- Preprocessing (eg. node removal)
- Strengthen Dominance
- Bi-directional search based on a monotone increasing resource

### Other approaches

- Dominate more thoroughly than theoretically possible.
- Use inherent structure in the problem to make the problem easier to solve (this typically means strengthen dominance further).
- Use heuristics/metaheuristics.



## Solving a relaxation I

One way of making the problem easier to solve is by relaxing the "Elementary" constraints. That is, we allow a node to be visited more than once.

### Why?

- The Elementary problem is NP-hard and computationally hard to solve. The relaxed problem is only pseudo-polynomial.
- By allowing revisiting customers we are allowing cycles.
- Infinite cycling is prohibited by the resources either the capacities or the time windows.

## Solving a relaxation II

### Advantages

- Resources for the each node is not necessary.
- We have only two resources in our labels: capacity and time.
- If accumulations of resources respect the triangle inequalities and are strictly positive, an optimal integer solution to the master problem only contains elementary paths.

### Disadvantages

- Quality of bound decreases
- Cycles are introduced



## Building on the idea of Dijkstra

- In order to build the SPPTWCC algorithm we have to make two assumptions:
  1. Time is always increasing along the edges, i.e. $t_{ij} > 0$.
  2. Time and capacity are discretized.
- The label in SPPTWCC contains the accumulated reduced cost, current time $t$ of arrival and the accumulated demand $d$, and accumulated cost.
- Labels are treated in order of increasing time ($t$).



## Elimination of 2-cycles

### Challenge

- The problem in the relaxation is that although we cannot cycle indefinitely we might get many labels due to the cycling.
- So we will generate paths that contain:
  - $\ldots \rightarrow i \rightarrow j \rightarrow i \ldots$
  - $\ldots \rightarrow i \rightarrow j \rightarrow k \rightarrow i \ldots$
  - etc etc

### 2-cycles

- As there are typically most 2-cycles it could be "enough" to just do something about them.
- In order to eliminate 2-cycles we associate a type with each of the labels. A label can be either strongly, semi-strongly or weakly dominated.



## Label Dominance I

### Strongly dominant

- A label is denoted strongly dominant if it is not dominated by any other label and at least one of the following conditions are satisfied:
  1. $t + t_{i,pred} > b_{pred}$
  2. $q + q_{pred} > Q$.
- This implies that a strongly dominant label can not participate in a two-cycle due to either time or capacity constraints (or both).

## Label Dominance II

### Semi-strongly dominant

The label is called semi-strongly dominant if it is not dominated by any other label and none of the conditions

1. $t + t_{i,pred} > b_{pred}$
2. $q + q_{pred} > Q$

are satisfied, which implies that a semi-strongly dominant label has the potential of being part of a two-cycle.

### Weakly dominant

A label is weakly dominated if it is only dominated by semi-strongly dominant labels, **and** the semi-strongly dominant labels have the same predecessor **and** this predecessor is different from the predecessor of the weakly dominated label.



## Computational Complexity

- It should be noted that the two-cycle elimination scheme does *not* change the computational complexity of the SPPTWCC algorithm.
- Let "potential" values for a label be given, then exactly one of the following three cases is true:
  1. There is no label for these values.
  2. There is one strongly dominant label for these values.
  3. There is one semi-strongly dominant label and **at most** one weakly dominant label for these labels.
- So the total number of labels is growing at most by a factor 2, thereby retaining the computational complexity.



## Labels and Extensions

- As semi-strongly dominant labels can be part of a two-cycle, they are not permitted to be extended back to the predecessor.
- Instead we allow for the existence of weakly dominant labels. They are dominated by semi-strongly dominant labels **and** they can be extended "back" to the predecessor of the semi-strongly dominant labels.
- The weakly dominant label will only be extended to the predecessor of a semi-strongly label as it is dominated by a semi-strongly dominant label for all other possible extensions.



## Discarding a label

If a new label is dominated by an old label it can be discarded if:

1. The old label is not semi-strongly dominant. If the old label is not semi-strongly dominant it is either strongly dominant or weakly dominant. In both cases dominated labels are not allow, therefore the new label can be discarded.
2. The old label is semi-strongly dominant **and**
   1. the old and the new label have the same predecessor.
   2. the new label is dominated by two or more labels with different predecessors.
   3. the new label can not be extended to the predecessor of the old label.

- The same rules can also be applied if a new label is dominating an old label.
- Additionally a dominated label that is not discarded can change type to weakly dominant.

## Details on implementation I

### Dominance

- In order to allow for an effective check of dominance we maintain a list of labels for each vertex.
- These labels are sorted lexicographically according to arrival time, accumulated demand and accumulated cost.
- The list of labels at the same vertex is scanned checking whether the label we want to insert into the list is dominated.
- This process terminates as the lexicographically right position in the list is reached.
- If the new label is dominated it might either be discarded straight away (for example if the new label is dominated by a weakly dominated label) or "marked" as being dominated if it is possible to keep it as a weakly dominated label according to the dominance rules.



## Details on implementation II

### Dominance (cont.)

- If the label is inserted in the list the remaining part of the list can not dominate the label, but instead the newly inserted label might dominate one or more of the remaining ones.

### Bucket Datastructure

- In each iteration we need the label with the smallest accumulated time (or another strictly increasing resource).
- Now let $t_{\min}$ be the minimal "time-length" of any edge.
- So whenever a label is extended, the time of the new label is at least increased by $t_{\min}$.
- So if a label is extended from a label with presently smallest time $t'$, it can not dominate any of the labels in the interval $[t'; t' + t_{\min}[$.



## Concluding Remarks

- ESPPRC typically has longer running times due to weak dominance.
- But ESPPRC results in better bounds giving a smaller Branch and Bound tree.
- It is always important to keep the number of resources at a minimum.
- Performance for both types of algorithms deteriorates drastically as the number of resources increase.