Hybrid Methods
(here: CP+OR)

Willem-Jan van Hoeve

Tepper School of Business, Carnegie Mellon University

vanhoeve@andrew.cmu.edu

ACP Summer School on Practical Constraint Programming
June 16-20, 2014, Bologna, Italy

Motivation

Benefits of CP

- Modeling power
- Inference methods
- Advanced search
- Exploits local structure

Benefits of OR

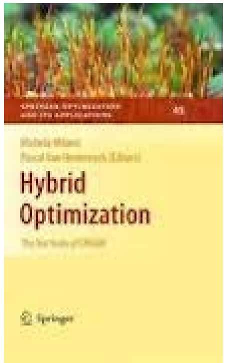
- Optimization algorithms
- Relaxation methods
- Duality theory
- Exploits global structure

Integrated methods can combine these complementary strengths

Can lead to several orders of magnitude of computational advantage

Some additional references

- Conference series CPAIOR
  - integration of techniques from CP, AI, and OR
  - http://www.andrew.cmu.edu/user/vanhoeve/cpaior/
  - online master classes/tutorials
  - book 'Hybrid Optimization' [Van Hentenryck&Milano, 2011]
- Other tutorials
  - CP summer school 2011: 'Integrating CP and mathematical programming' [John Hooker]
  - http://ba.gsia.cmu.edu/jnh/slides.html
- Success stories
  - http://moya.bus.miami.edu/~tallys/integrated.php



Outline

- Global constraint propagation
  - network flows
  - optimization constraints
- Integrating relaxations
  - Linear Programming relaxation
  - Lagrangean relaxation
- Decomposition methods
  - logic-based Benders
  - column generation

Propagation with Network Flows

J.-C. Régin. Generalized Arc Consistency for Global Cardinality Constraint. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 209-215, 1996.

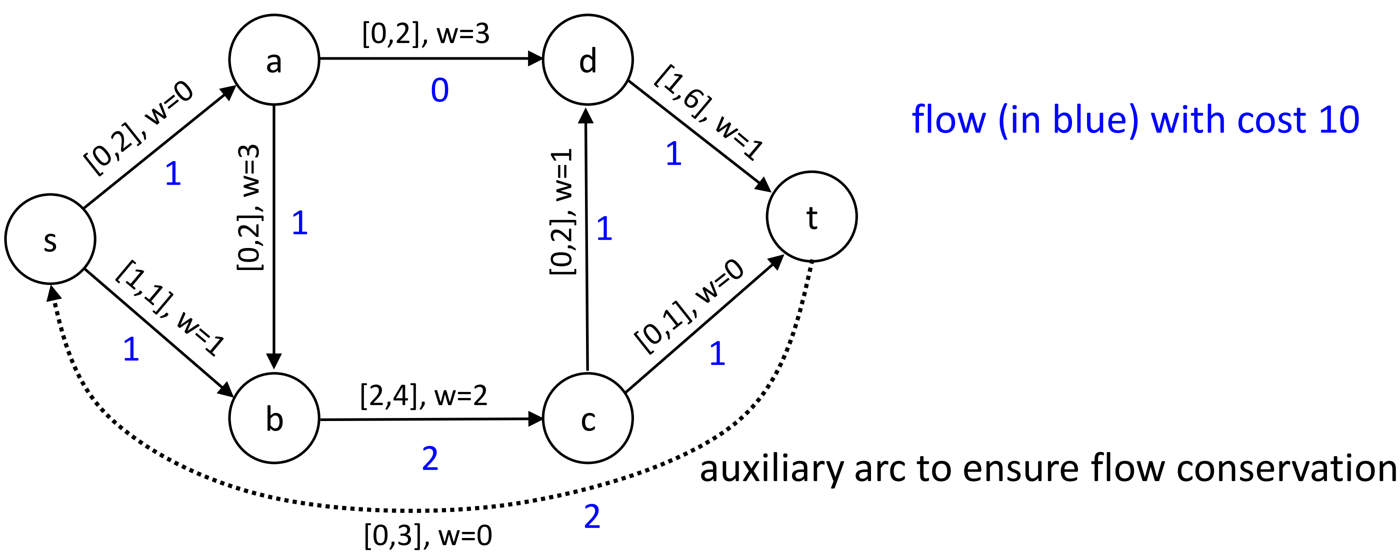
Network Flows

Let G=(V,A) be a directed graph with vertex set V and arc set A. To each arc $a \in A$ we assign a capacity function [d(a),c(a)] and a weight function w(a).

Let s,t $\in$ V. A function f: $A \to \mathbb{R}$ is called an s-t flow (or a flow) if

- $f(a) \geq 0$ for all $a \in A$
- $\sum_{a \text{ enters } v} f(a) = \sum_{a \text{ leaves } v} f(a)$ for all $v \in V$ (flow conservation)
- $d(a) \leq f(a) \leq c(a)$ for all $a \in A$

Define the cost of flow f as $\sum_{a \in A} w(a)f(a)$. A minimum-cost flow is a flow with minimum cost.



Example: Network flow for alldifferent

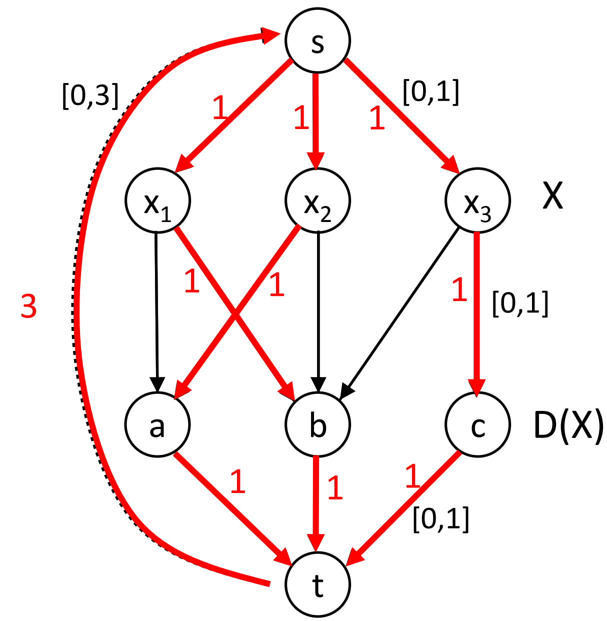
Fact: matching in bipartite graph $\Leftrightarrow$ integer flow in directed bipartite graph

Step 1: direct edges from X to D(X)

Step 2: add a source s and sink t

Step 3: connect s to X, and D(X) to t

Step 4: add special arc (t,s)



all arcs have capacity [0,1] and weight 0

except arc (t,s) with capacity [0, min{|X|,|D(X)|}]

Cardinality constraints

- The global cardinality constraint restricts the number of times certain values can be taken in a solution
- *Example:* We need to assign 75 employees to shifts. Each employee works one shift. For each shift, we have a lower and upper demand.

shift	1	2	3	4	5	6
min	10	12	16	10	6	4
max	14	14	20	14	12	8

$D(x_i) = \{1, 2, 3, 4, 5, 6\}$ for $i = 1, 2, \ldots, 75$

gcc($x_1, \ldots, x_{75}$, min, max)

Propagation for cardinality constraints

Definition*:* Let X be a set of variables with $D(x) \subseteq V$ for all $x \in X$ (for some set V). Let *L* and *U* be vectors of non-negative integers over V such that $L[v] \leq U[v]$ for all $v \in V$.
The constraint *gcc*(X, *L*, *U*) is defined as the conjunction

$$\wedge_{v \in V} \; \left(L[v] \leq \textstyle\sum_{x \in X} (x = v) \leq U[v] \right)$$

Questions:

1. Can we determine in polynomial time whether the constraint is consistent (satisfiable)?
2. Can we establish domain consistency (remove all inconsistent domain values) in polynomial time?

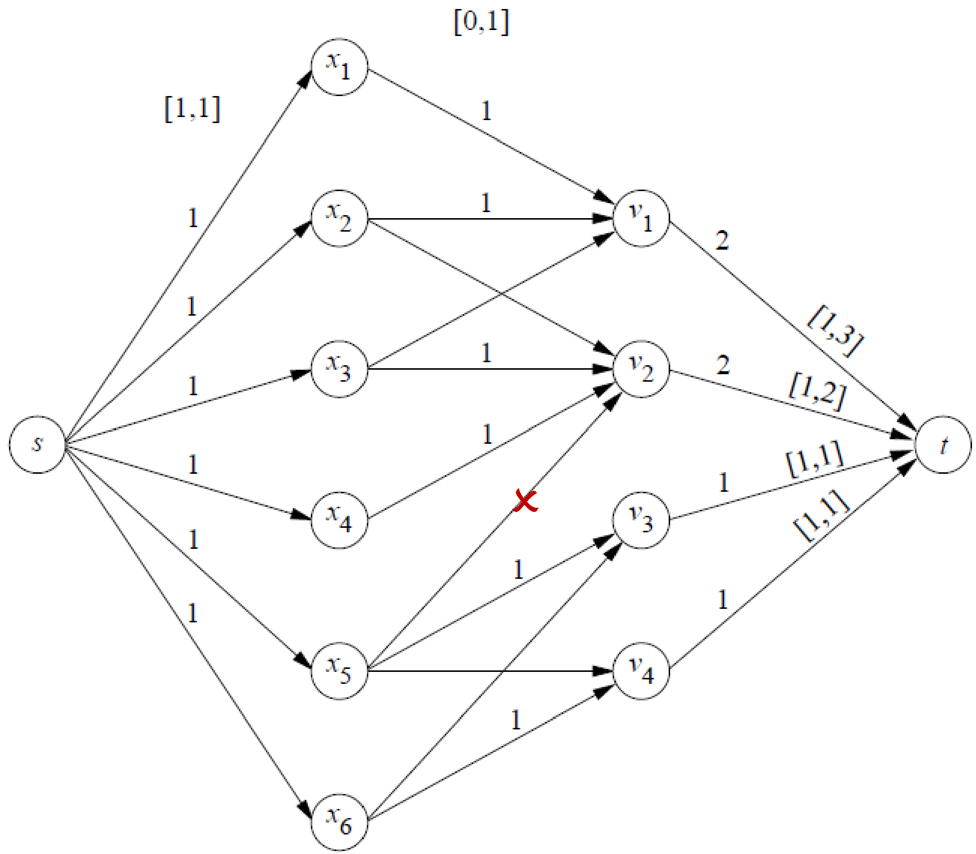
Network representation

- Lemma [Regin, 1996]: Solution to *gcc* is equivalent to particular network flow
  - similar to bipartite network for *alldifferent*
  - node set defined by variables and domain values, one source s and one sink t
  - define arc (x,v) for all $x \in X$, $v \in D(x)$ with capacity [0,1]
  - define arcs from s to x for all $x \in X$ with capacity [1,1]
  - define arcs from v to t for all $v \in V$ with capacity [L[v],U[v]]
- Feasible integer flow corresponds to solution to *gcc*
- *Note:* If L[v]=0, U[v]=1 for all $v \in V$ then *gcc* is equivalent to *alldifferent*

Example

$D(x_1)$	$D(x_2)$	$D(x_3)$	$D(x_4)$	$D(x_5)$	$D(x_6)$
{1}	{1,2}	{1,2}	{2}	$\{\cancel{2},3,4\}$	{3,4}

v	$[L(v), U(v)]$
1	[1,3]
2	[1,2]
3	[1,1]
4	[1,1]



gcc network

Domain consistency for gcc

- Determining consistency: compute network flow
  - Using Ford & Fulkerson's augmenting path algorithm, this can be done in O(mn) time for (n is number of variables, m is number of edges in the graph)
  - Can be improved to $O(m\sqrt{n})$ [Quimper at al., 2004]
- Naïve domain consistency
  - Fix flow of each arc to 1, and apply consistency check. Remove arc if no solution. $O(m^2\sqrt{n})$ time.
- More efficient algorithm: use residual network
  - similar to SCCs for alldifferent
  - domain consistency in O(m) time
  - maintain residual network incrementally

Optimization Constraints

- In the CP literature, ‘optimization’ constraints refer to constraints that represent a structure commonly identified with optimization
  - usually linked to the objective function (e.g., minimize cost)
  - sometimes stand-alone structure (budget limit, risk level, etc.)
  - for example, *knapsack* constraint
- For any constraint, a weighted version can be obtained by applying a weight measure on the variable assignments, and restricting the total weight to be within a threshold

GCC with costs

- The classical weighted version of the *gcc* is obtained by associating a weight $w(x,v)$ to each pair $x \in X$, $v \in V$. Let z be a variable representing the total weight. Then

  $cost\_gcc(X, L, U, z, w) =$

  $$gcc(X,L,U) \wedge \sum_{x \in X,\ x=v} w(x,v) \leq z$$

- In other words, we restrict the solutions to those that have a weight at most max(D(z))

Domain filtering for weighted gcc

1. Determine consistency of the constraint
2. Remove all domain values from X that do not belong to a solution with weight ≤ max(D(z))
3. Filter domain of z
   – i.e., increase min(D(z)) to the minimum weight value over all solutions, if applicable

Determining consistency of cost_gcc

- Once again, we can exploit the correspondence with a (weighted) network flow [Regin 1999, 2002]:

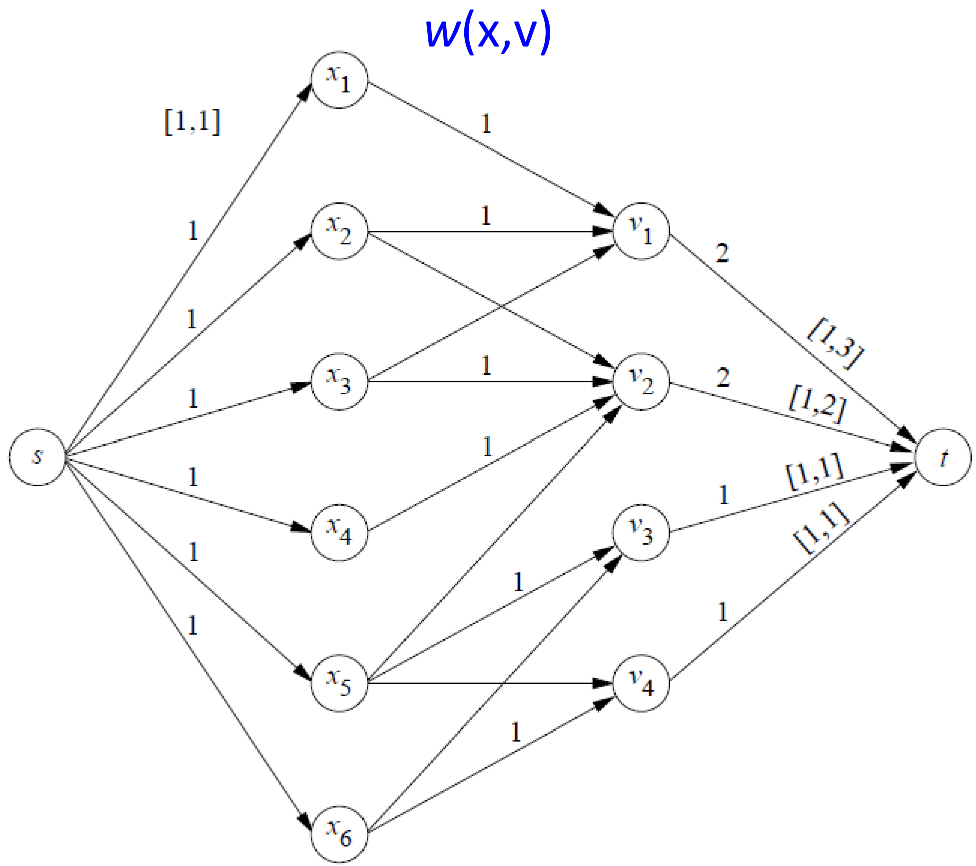
  A solution to *cost_gcc* corresponds to a weighted network flow with total weight ≤ max(D(z))

- We can test consistency of the *cost_gcc* by computing a minimum-cost flow

Example

$D(x_1)$	$D(x_2)$	$D(x_3)$	$D(x_4)$	$D(x_5)$	$D(x_6)$
{1}	{1,2}	{1,2}	{2}	{2,3,4}	{3,4}

v	$[L(v), U(v)]$
1	[1,3]
2	[1,2]
3	[1,1]
4	[1,1]



gcc network

Time complexity

- A minimum-cost flow can be found with the classical 'successive shortest paths' algorithm of Ford & Fulkerson
  - The flow is successively augmented along the *shortest* path in the residual network
  - Finding the shortest path takes O(m + n log n) time (for m edges, n variables)
  - In general, this yields a pseudo-polynomial algorithm, as it depends on the cost of the flow. However, we compute at most n shortest paths (one for each variable)
  - Overall running time is O(n(m + n log n)) time
- Naïve domain consistency in O(nm(m + n log n))
- Can be improved to O(*min*{n,|V|}(m + n log n))
  - all shortest paths in residual graph

Other constraints

- Network flows have been applied to several other global constraints
  - soft *alldifferent*
  - soft *cardinality* constraint
  - soft *regular* constraint
  - cardinality constraints in weighted CSPs
  - *sequence* constraint [Maher et al. 2008] [Downing et al. 2012]
  - resource scheduling [Baptiste et al. 2001] [Lombardi&Milano, 2012] [Bessiere et al. 2014]
  - …

[v.H. "Over-Constrained Problems", 2011]

- Very powerful and generic technique for handling global constraints

Outline

- Global constraint propagation
  - network flows
  - optimization constraints
- Integrating relaxations
  - Linear Programming relaxation
  - Lagrangean relaxation
- Decomposition methods
  - logic-based Benders
  - column generation

Linear Programming

- LP model is restricted to linear constraints and continuous variables
- Linear programs can be written in the following standard form:

$$
\begin{array}{rl}
\min & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\
\text{subject to} & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\
& a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\
& \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
& a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \\
& x_1, \ldots, x_n \geq 0
\end{array}
$$

or, using matrix notation:

$$\min \{c^\mathsf{T}x \mid Ax = b, x \geq 0\}$$

Benefits of Linear Programming

- Solvable in polynomial time
  - very scalable (millions of variables and constraints)
- Many real-world applications can be modeled and solved using LP
  - from production planning to data mining
- LP models are very useful as relaxation for integer decision problems
  - LP relaxation can be strengthened by adding constraints (cuts) based on integrality
- Well-understood theoretical properties
  - e.g., duality theory

LP-based domain filtering

- Suppose we have a LP relaxation available for our problem

$$\min \{c^{\mathsf{T}} x \mid Ax = b, x \geq 0\}$$

- We could establish "LP bounds consistency" on the domains of the variables:

  For each variable x_i

  change objective to $\min x_i$ and solve LP: lower bound LB_i

  change objective to $\max x_i$ and solve LP: upper bound UB_i

  $x_i \in [\mathrm{LB}_i\,, \mathrm{UB}_i]$

- Time-consuming (although it can pay off, e.g., in nonlinear programming problems)

LP-based domain filtering

- Instead of min/max of each variable, exploit <u>reduced costs</u> as more efficient approximation
  - marginal impact on objective for each variable

[Focacci, Lodi, and Milano, 1999, 2002]

- In the following, we consider 'optimization constraints' again:
  - associate a weight $c(x,v)$ to each pair $x \in X$, $v \in D(x)$
  - z is a variable representing the total weight

$$cost\_C(X, z, c) = C(X) \wedge \sum_{x \in X,\, x=v} w(x,v) \leq z$$

Creating a generic LP relaxation

- Create mapping between linear model and CP model by introducing binary variables y_{ij} for all $i \in \{1,\ldots,n\}$ and $j \in D(x_i)$ such that

$$x_i = j \Leftrightarrow y_{ij} = 1$$
$$x_i \neq j \Leftrightarrow y_{ij} = 0$$

- To ensure that each variable x_i is assigned a value, we add the following constraints to the linear model:

$$\sum_{j \in D(x_i)} y_{ij} = 1 \quad \text{for } i = 1, \ldots, n$$

- The objective is naturally stated as

$$\sum_{i=1}^{n} \sum_{j \in D(x_i)} c_{ij} y_{ij}$$

LP relaxation (cont'd)

- The next task is to represent the actual constraint, and this depends on the combinatorial structure
- For example, if the constraint contains a permutation structure (such as the *alldifferent*), we can add the constraints:

$$\sum_{i=1}^{n} y_{ij} \leq 1 \ \text{ for all } j \in \bigcup_{i=1}^{n} D(x_i)$$

- (Note that specific cuts known from MIP may be added to strengthen the LP)
- After the linear model is stated, we obtain the natural LP relaxation by removing the integrality condition on y_{ij} :

$$0 \leq y_{ij} \leq 1 \ \text{ for } i \in \{1, \ldots, n\}, j \in D(x_i)$$

Reduced-cost based filtering

- The output of the LP solution is an optimal solution value z^*, a (fractional) value for each variable y_{ij}, and an associated reduced cost $\overline{c}_{ij}$
- Recall that $\overline{c}_{ij}$ represents the marginal change in the objective value when variable y_{ij} is forced in the solution
  – e.g., if $y_{ij} = 1$ then z^* increases by $\overline{c}_{ij}$
- But y_{ij} represents $x_i = j$
- Reduced-cost based filtering:

$$\text{if } z^* + \overline{c}_{ij} > \max D(z) \text{ then } D(x_i) \leftarrow D(x_i) \setminus \{j\}$$

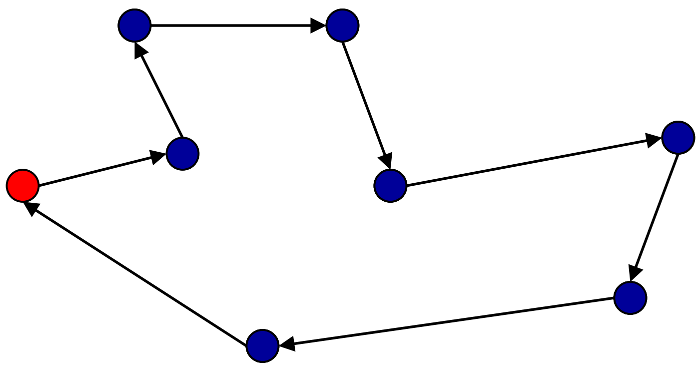
(This is a well-known technique in OR, called ‘variable fixing’)

Pros and Cons

- Potential drawbacks:
  - The filtering power depends directly on the quality of the LP relaxation, and it may be hard to find an effective relaxation
  - Solving a LP using the simplex method may take much more time than propagating the constraint using a combinatorial filtering algorithm
- Potential benefits:
  - It’s very generic; it works for any LP relaxation of a single constraint, a combination of constraints, or for the entire problem
  - Can be generated automatically from CP model (Xpress-Kalis)
  - New insights in MIP/LP solving can have immediate impact
  - For several constraint types, there exist fast and incremental combinatorial techniques to solve the LP relaxation
  - This type optimality-based filtering complements nicely the feasibility-based filtering of CP; several applications cannot be solved with CP otherwise

Example Application: TSP

- CP model
- LP relaxation
  - Assignment Problem
- Impact of reduced-cost based filtering



Graph G = (V,E) with vertex set V and edge set E

|V| = n

w(i,j): distance between i and j

CP models for the TSP

- Permutation model
  - variable pos_i represents the i-th city to be visited
  - (can introduce dummy node $pos_{n+1} = pos_1$)

$$\min \quad \textstyle\sum_i w(pos_i, pos_{i+1})$$
$$\text{s.t.} \quad \textit{alldifferent}(pos_1, \ldots, pos_n)$$

both models *decouple* the objective and the circuit

- Successor model
  - variable $next_i$ represents the immediate successor of city i

$$\min \quad \textstyle\sum_i w(i, next_i)$$
$$\text{s.t.} \quad \textit{alldifferent}(next_1, \ldots, next_n)$$
$$\textit{path}(next_1, \ldots, next_n)$$

(Hamiltonian Path, not always supported by the CP solver)

More CP models

- Combined model (still decoupled)

$$
\begin{aligned}
\min \ & \sum_{i \in V} w(i, next_i) \\
\text{s.t. } & \texttt{alldifferent}(next_1, \ldots, next_n) \\
& \texttt{alldifferent}(pos_1, \ldots, pos_n) \\
& pos_j = next_{pos_{j-1}} \qquad \forall j \in \{2, \ldots, n\} \\
& pos_1 = 1
\end{aligned}
$$

- Integrated model

min z

s.t. *alldifferent*($next_1$, …, $next_n$) [redundant]

WeightedPath(next, w, z) [Focacci et al., 1999, 2002]

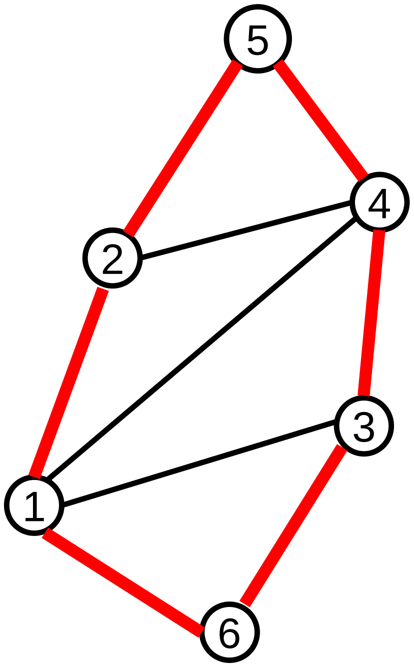
(Note: most CP solvers do not support this constraint)

Relaxations for TSP

- An integrated model using *WeightedPath*(next, w, z) allows to apply an LP relaxation and perform reduced-cost based filtering

- Observe that the TSP is a combination of two constraints
  - The degree of each node is 2
  - The solution is connected (no subtours)

- Relaxations:
  - relax connectedness: Assignment Problem
  - relax degree constraints: 1-Tree Relaxation



Assignment Problem

Binary variable y_{ij} represents whether the tour goes from i to j

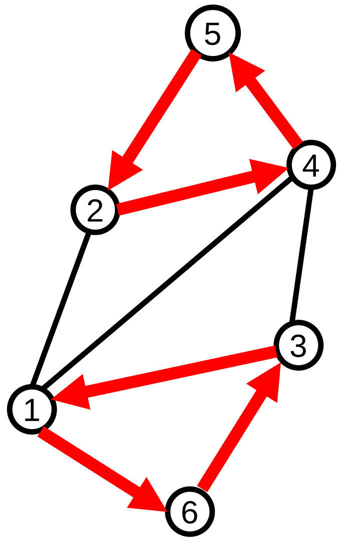
$$\min z = \sum_{i \in V} \sum_{j \in V} w_{ij} y_{ij}$$

s.t.

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

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

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



Mapping between CP and LP model

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

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

Assignment Problem

Binary variable y_{ij} represents whether the tour goes from i to j

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

s.t.

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

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

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

Benefits of AP relaxation

- Continuous relaxation provides integer solutions (total unimodularity)
- Specialized $O(n^3)$ algorithm (Hungarian method)
- Incremental $O(n^2)$ running time
- Reduced costs come for free
- Works well on asymmetric TSP

Mapping between CP and LP model

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

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

Computational results for TSP-TW

instance		Dyn.Prog. BS2000	Branch&Cut AFG2001	CP+LP FLM2002	
name	n	time	time	time	fails
rbg021.2	21	9.00	0.22	0.2	44
rbg021.3	21	9.60	27.15	0.4	107
rbg021.4	21	11.52	5.82	0.3	121
rbg021.5	21	127.97	6.63	0.2	55
rbg021.6	21	161.66	1.38	0.7	318
rbg021.7	21	N.A.	4.30	0.6	237
rbg021.8	21	N.A.	17.40	0.6	222
rbg021.9	21	N.A.	26.12	0.8	310
rbg034a	36	18.03	0.98	55.2	13k
rbg035a.2	37	N.A.	64.80	36.8	5k
rbg035a	37	7.67	1.83	3.5	841
rbg038a	40	8.64	4232.23	0.2	49
rgb040a	42	20.08	751.82	738.1	136k
rbg041a	43	24.57	N.A.	N.A.	
rbg042a	44	47.38	N.A.	149.8	19k
rbg050a	52	N.A.	18.62	180.4	19k
rbg067a	69	29.14	5.95	4.0	493
rbg152	152	37.90	N.A.	N.A.	

Langrangean Relaxation

Move subset (or all) of constraints into the objective with ‘penalty’ multipliers μ:

$$
\begin{array}{ll} \min & c^\top x \\ \text{s.t.} & A_1 x = b_1 \\ & A_2 x = b_2 \\ & x \geq 0 \end{array}
\quad \longrightarrow \quad
L(\mu) = \begin{array}{ll} \min & c^\top x + \mu^\top (b_2 - A_2 x) \\ \text{s.t.} & A_1 x = b_1 \\ & x \geq 0 \end{array}
$$

Weak duality: for any choice of μ, Lagrangean $L(\mu)$ provides a lower bound on the original LP

Goal: find optimal μ (providing the best bound) via

$$\max_{\mu \geq 0} \; L(\mu)$$

Motivation for using Lagrangeans

- Lagrangean relaxations can be applied to nonlinear programming problems (NLPs), LPs, and in the context of integer programming
- Lagrangean relaxation can provide better bounds than LP relaxation
- The Lagrangean dual generalizes LP duality
- It provides domain filtering analogous to that based on LP duality [Sellmann, CP 2004]
- Lagrangean relaxation can dualize 'difficult' constraints
  - Can exploit the problem structure, e.g., the Lagrangean relaxation may decouple, or $L(\mu)$ may be very fast to solve combinatorially

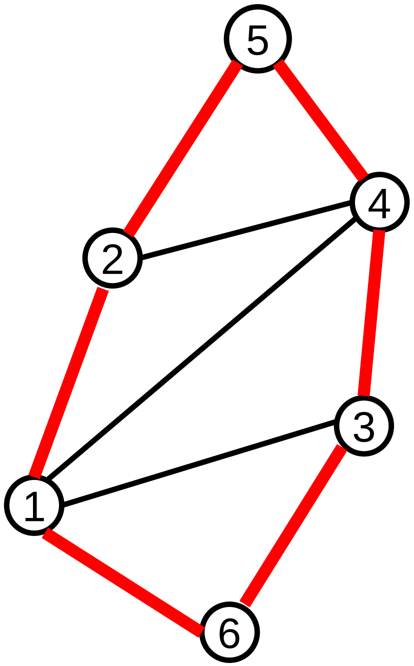
- Next application: Lagrangean relaxation for TSP

Recall: Relaxations for TSP

- An integrated model using *WeightedPath*(next, w, z) allows to apply an LP relaxation and perform reduced-cost based filtering

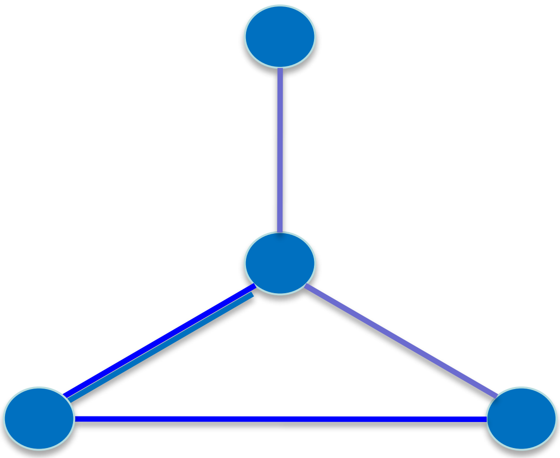
- Observe that the TSP is a combination of two constraints
  – The degree of each node is 2
  – The solution is connected (no sub tours)

- Relaxations:
  – relax connectedness: Assignment Problem
  – relax degree constraints: 1-Tree Relaxation



The 1-Tree Relaxation for TSP

- Relaxation of the degree constraints [Held&Karp, 1970, 1971]
- A minimum spanning tree gives such a relaxation
- A 1-tree is a stronger relaxation, which can be obtained by:
  - Choosing any node v (which is called the 1-node)
  - Building a minimum spanning tree T on G = (V\{v}, E)
  - Adding the smallest two edges linking v to T
- For *n* vertices, a 1-tree contains *n* edges



P.S. an MST can be found in
 $O(m\ \alpha(m,n))$ time

The Held and Karp bound for TSP

The 1-tree can be tightened through the use of Lagrangean relaxation by relaxing the degree constraints in the TSP model:

Let binary variable x_e represent whether edge e is used

$$\min \sum_{e \in E} w(e) x_e$$

$$\text{s.t.} \quad \boxed{\sum_{e \in \delta(i)} x_e = 2 \qquad \forall i \in V}$$

$$\sum_{i,j \in S, i<j} x_{(i,j)} \leq |S| - 1 \qquad \forall S \subset V, |S| \geq 3$$

$$x_e \in \{0, 1\} \qquad \forall e \in E$$

The Held and Karp bound for TSP

Lagrangean relaxation with multipliers π (penalties for node degree violation):

$$\min \sum_{e \in E} w(e) x_e + \sum_{i \in V \setminus \{1\}} \pi_i \Big(2 - \sum_{e \in \delta(i)} x_e\Big)$$

$$\text{s.t.} \quad \sum_{i,j \in S, i<j} x_{(i,j)} \le |S| - 1 \qquad \forall S \subset V \setminus \{1\}, |S| \ge 3$$

$$\sum_{e \in \delta(1)} x_e = 2$$

$$\sum_{e \in E} x_e = |V|$$

$$x_e \in \{0, 1\}$$

How to find the best penalties π?

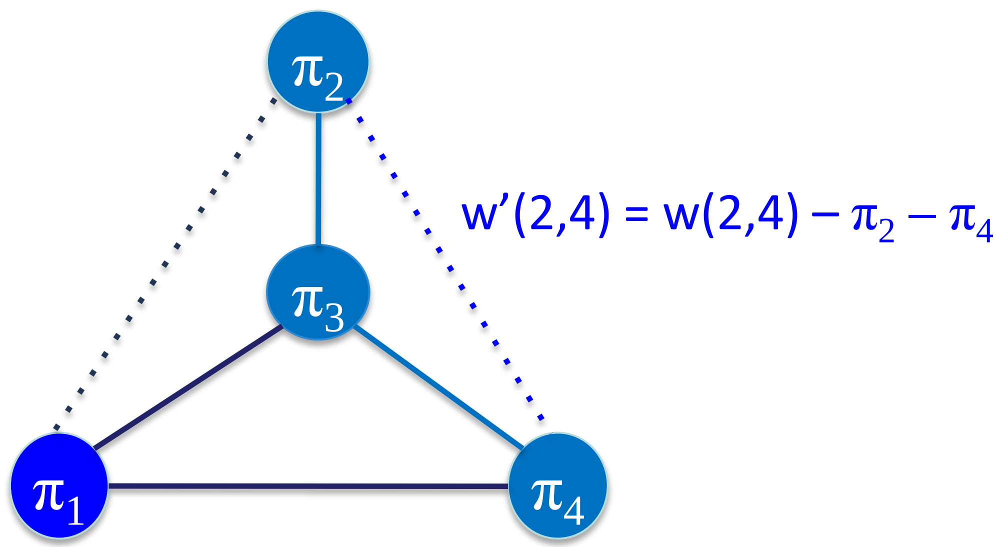
- In general, subgradient optimization
- But here we can exploit a combinatorial interpretation
- No need to solve LP

Held-Karp iteration

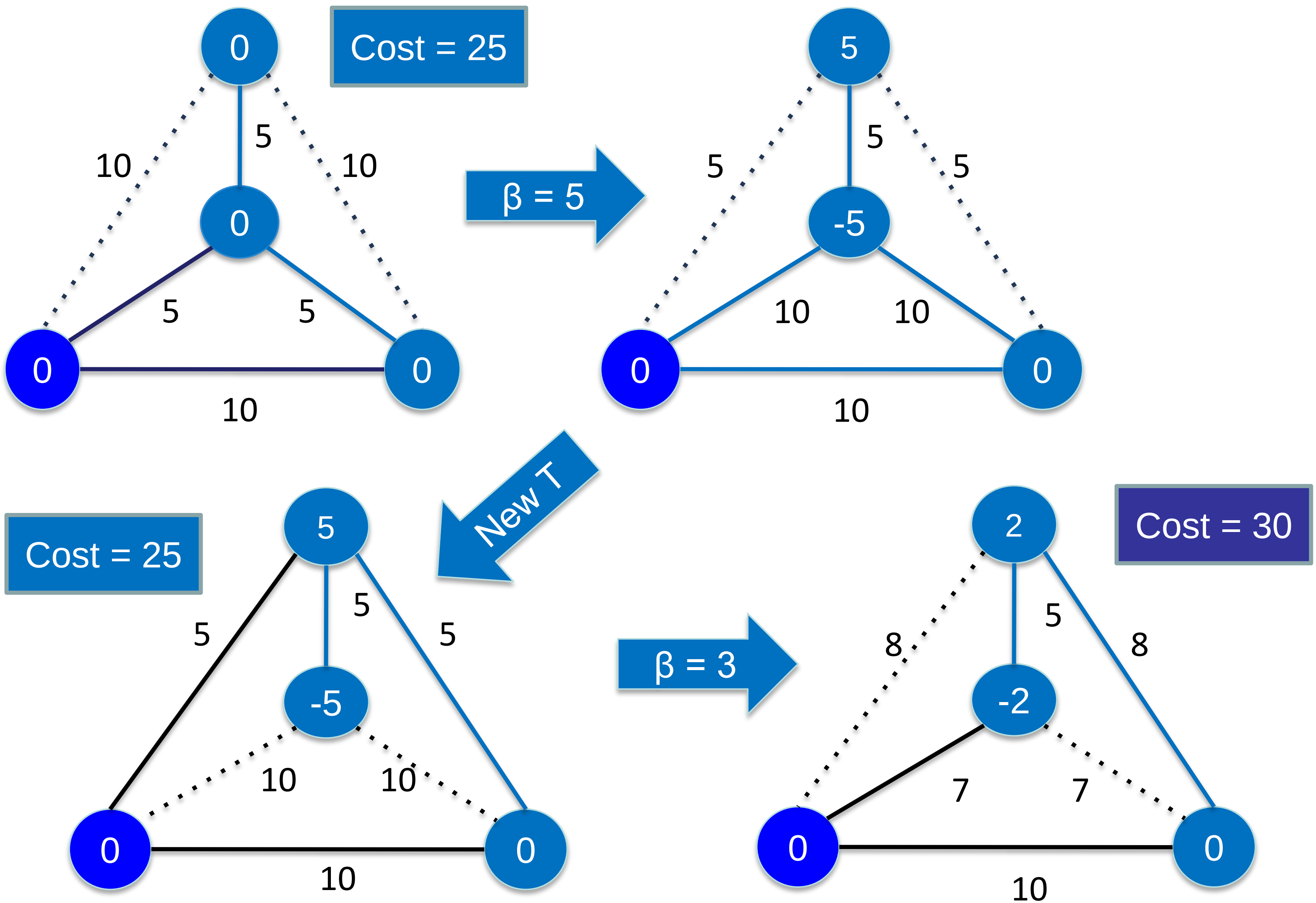
- Solve 1-tree w.r.t. updated edge weights $w'(i,j) = w(i,j) - \pi_i - \pi_j$
- Optimal 1-tree T gives lower bound: cost(T) + 2 $\sum_i \pi_i$
- If T is not a tour, then we iteratively update the penalties as

  π_i += (2-degree(i))*β (step size β different per iteration)

  and repeat



Example



Cost = 25
0
0
0
0
10
5
10
5
5
10
β = 5
5
-5
0
0
5
5
5
10
10
10
New T
Cost = 25
5
-5
0
0
5
5
5
10
10
10
β = 3
Cost = 30
2
-2
0
0
8
5
8
7
7
10

How can we exploit 1-tree in CP?

- We need to reason on the graph structure
  - manipulate the graph, remove costly edges, etc.
- Not easily done with 'next' and 'pos' variables
  - e.g., how can we enforce that a given edge e=(i,j) is mandatory?
  - ($next_i = j$ or $next_j = i$) ?
  - $(pos_k = i) \Rightarrow ((pos_{k+1} = j)$ or $(pos_{k-1} = j))$?
- Ideally, we want to have access to the graph rather than local successor/predecessor information
  - modify definition of global constraint

One more CP model for the TSP

Integrated model based on graph representation

min z

[Benchimol et al., 2012]

s.t. *weighted-circuit*(X, G, z)

- G=(V,E,w) is the graph with vertex set V, edge set E, weights w
- X is a set variable representing the set of edges that will form the circuit
  - Domain D(X) = [L(X), U(X)], with fixed cardinality |V| in this case
  - Lower bound L(X) is set of mandatory edges
  - Upper bound U(X) is set of possible edges
- z is a variable representing the total edge weight

Domain Filtering

- Given constraint

  weighted-circuit(X, G=(V,E,w), z)

- Apply the 1-tree relaxation to
  - remove sub-optimal edges from U(X)
  - force mandatory edges into L(X)
  - update bounds of z

- For simplicity, the presentation of the algorithms are restricted to G = (V\{1}, E)

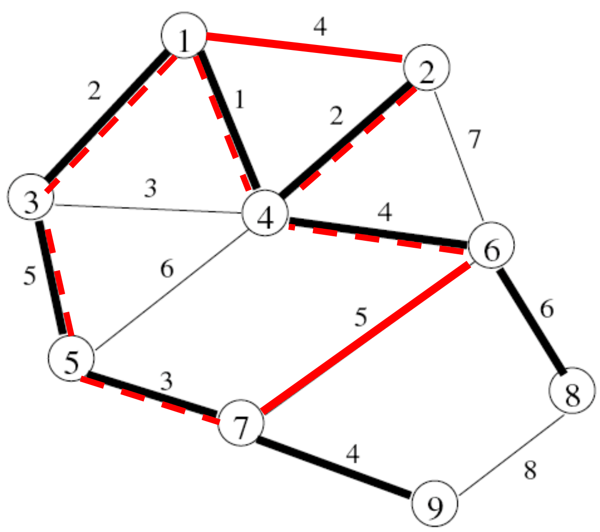
Removing non-tree edges

- The *marginal cost* of a non-tree edge e is the additional cost of forcing e in the solution:

  $$c'_e = \text{cost}(T(e)) - \text{cost}(T)$$

- Given a current best solution UB, edge e can be removed if cost(T(e)) > UB, or

  $$c'_e + \text{cost}(T) > UB$$



Replacement cost of

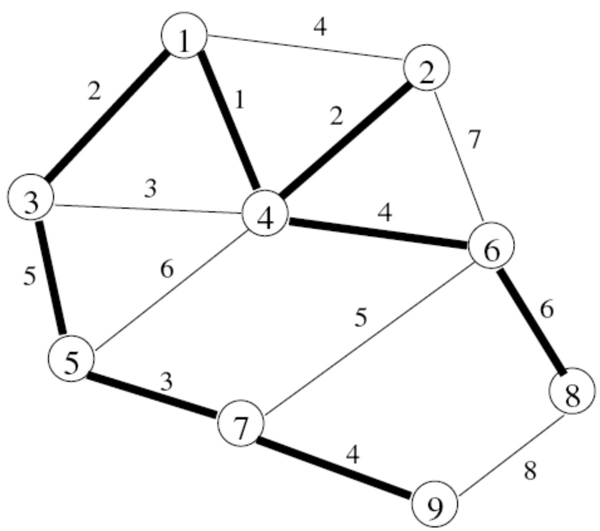
- (1,2) is 4 - 2 = 2
- (6,7) is 5 - 5 = 0

Computing marginal costs

Basic algorithm for computing marginal edge costs:

- For each non-tree edge $e=(i,j)$
  - find the unique i-j path P_e in the tree
  - the marginal cost of e is $c_e - \max(c_a \mid a \in P_e)$

Complexity: O(mn), since P_e can be found in O(n) time by DFS

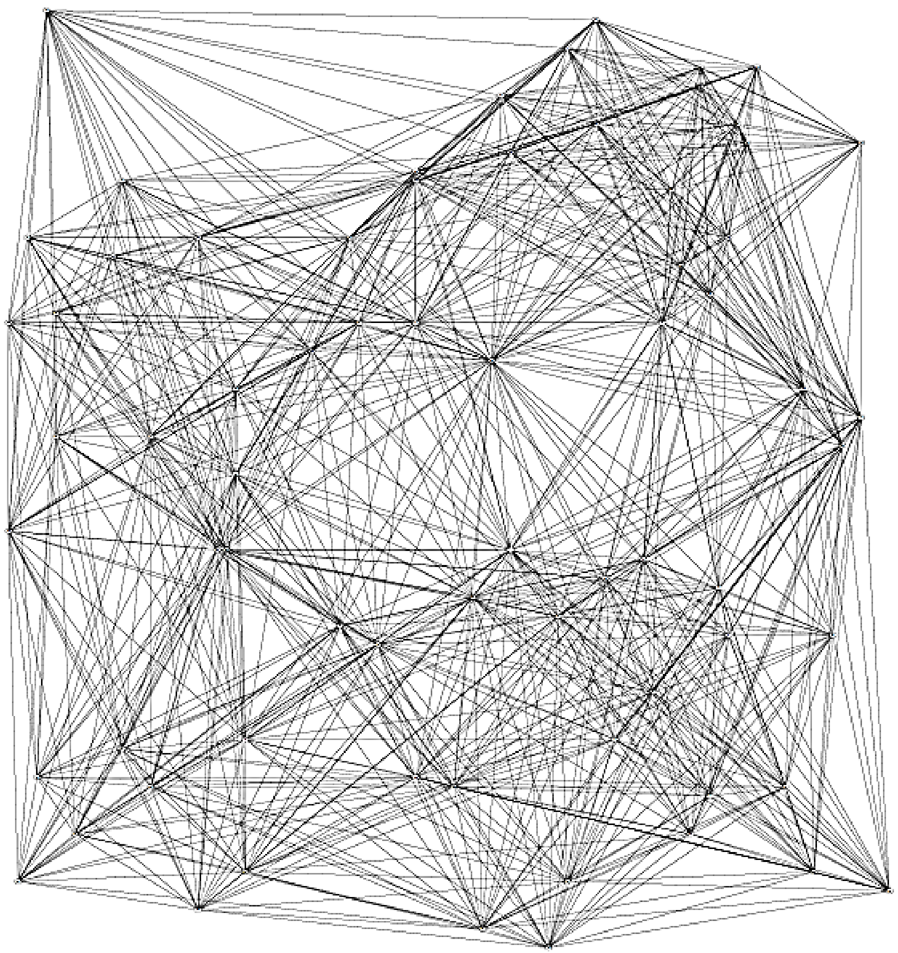


Can be further improved to O(m + n + n log n)

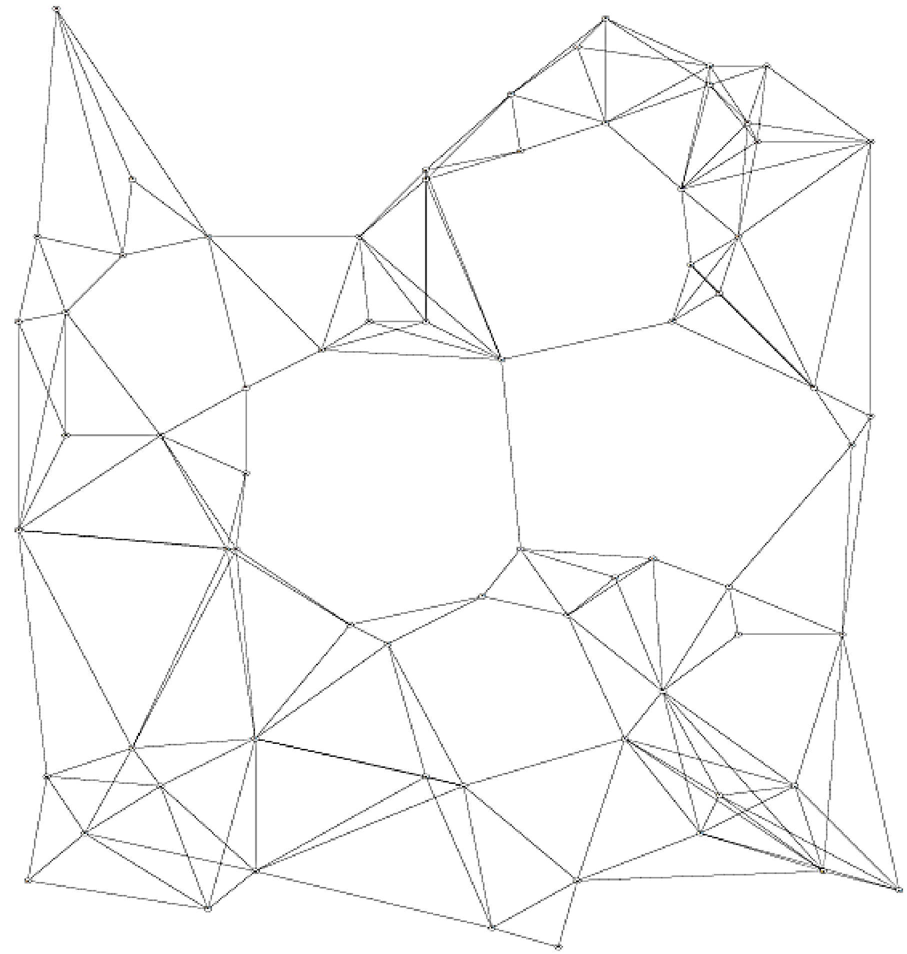
[Regin, 2008]

Impact of edge filtering

st70 from TSPLIB



upper bound = 700



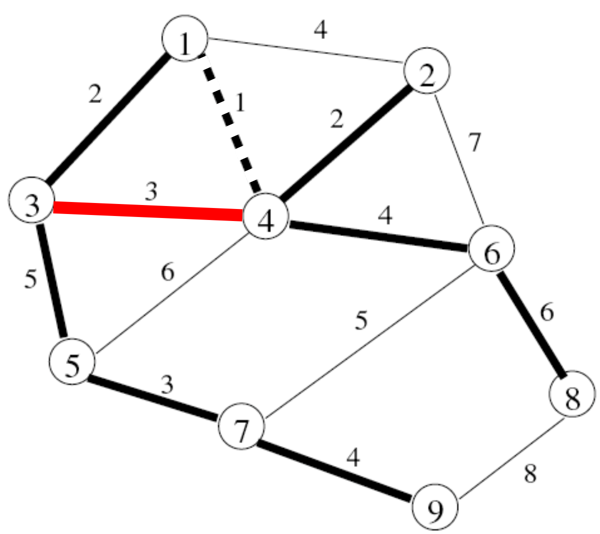
upper bound = 675

Forcing tree edges

- The *replacement* cost of a tree edge e is the additional cost when e is removed from the tree:

  c^r_e = cost(T \ e) – cost(T)

- Given a current best solution UB, edge e is mandatory if cost(T \ e) > UB, or c^r_e + cost(T) > UB

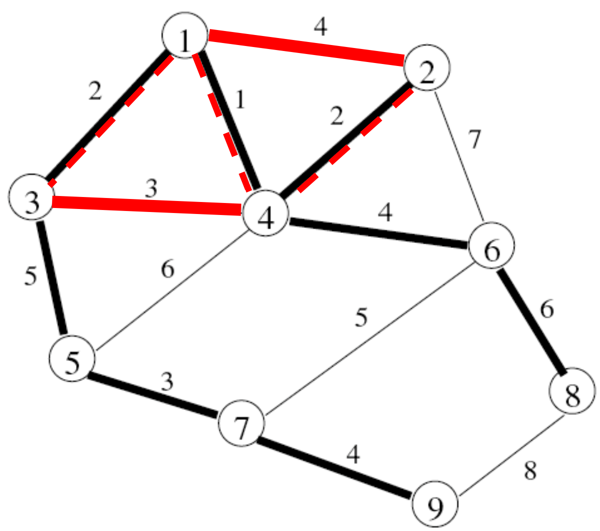


Replacement cost of (1,4)?

we need to find the cheapest edge to reconnect: 3 - 1 = 2

Computing replacement costs

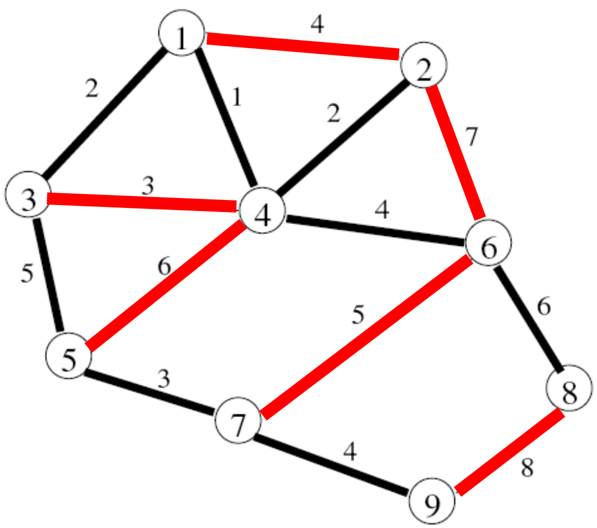
1. Compute minimum spanning tree *T* in *G*
2. Mark all edges in *T* as 'unmarked'
3. Consider non-tree edges, ordered by non-decreasing weight:
   - For non-tree edge (*i*,*j*), traverse the *i*-*j* path in T
   - Mark all unmarked edges *e* on this path, and assign $c^r_e = c_{ij} - c_e$
4. Basic time complexity O(*mn*), or, at no extra cost if performed together with the computation of marginal costs

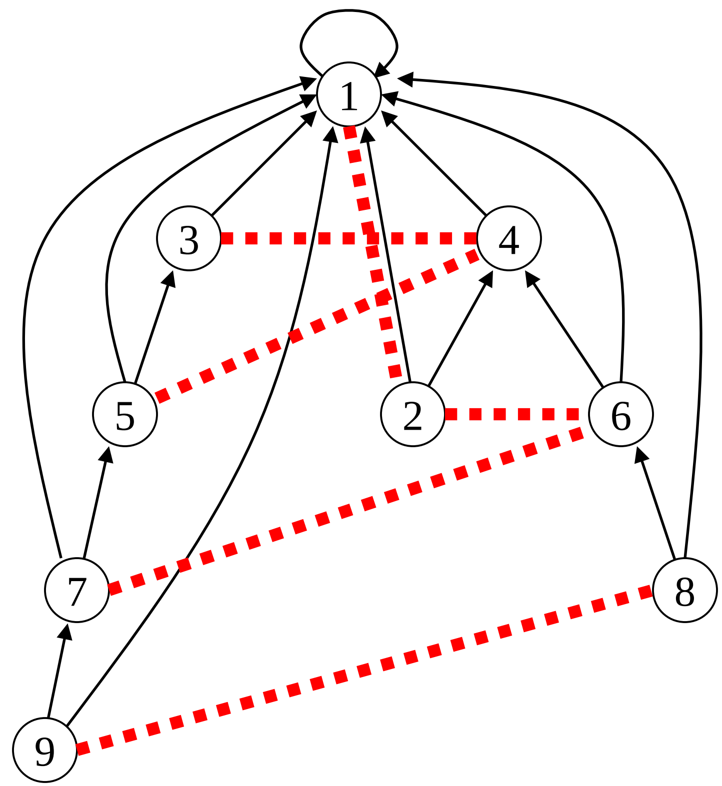


non-tree edge	mark edge	replacement cost
(3,4)	(1,4)	3 - 1 = 2
	(1,3)	3 - 2 = 1
(1,2)	(2,4)	4 - 2 = 2
	(edge (1,4) already marked)	
...		

Improving the time complexity

- We can improve this complexity by ‘contracting’ the marked edges (that is, we merge the extremities of the edge)
  - First, root the minimum spanning tree
  - Apply Tarjan’s ‘path compression’ technique during the algorithm
  - This leads to a time complexity of O($m\alpha(m,n)$)





Impact of filtering

	1-tree no filtering			1-tree with filtering			Concorde		
size	solved	time	nodes/s	solved	time	nodes/s	solved	time	nodes/s
50	1.00	0.13	299.26	1.00	0.03	712.39	1.00	0.18	19.59
100	1.00	3.19	55.10	1.00	0.34	160.65	1.00	0.31	6.10
150	1.00	18.31	13.83	1.00	1.42	46.91	1.00	0.59	4.52
200	1.00	132.30	5.16	1.00	4.68	33.00	1.00	0.97	3.18
250	0.97	409.88	2.13	1.00	10.98	25.76	1.00	1.98	2.83
300	0.80	770.67	1.38	1.00	24.35	20.29	1.00	2.32	2.15
350	0.67	1,239.25	0.61	1.00	39.54	15.96	1.00	3.74	1.92
400	0.33	1,589.71	0.42	0.97	108.45	11.04	1.00	4.57	1.64
450	0.17	1,722.56	0.34	1.00	121.08	12.16	1.00	4.99	1.68
500	0.00	1,800.00	0.21	0.97	194.32	8.81	1.00	6.42	1.38
550	0.00	1,800.00	0.20	0.97	206.99	7.98	1.00	5.00	1.00

randomly generated symmetric TSPs, time limit 1800s

average over 30 instances per size class

previous CP approaches could handle 100 cities maximum (if at all)

Comparison with ILOG CPO

		IBM ILOG CP Optimizer			1-tree with filtering		
instance	UB	best found	search nodes	time	best found	search nodes	time
burma14	3323	3323	9,455	0.76	3323	1	0.01
ulysses16	6859	6859	62,789	5.13	6859	1	0.00
gr17	2085	2085	608,220	66.34	2085	1	0.01
gr21	2707	2707	8,516	1.65	2707	1	0.01
ulysses22	7013	7013	11,028,276	1,800.00	7013	2	0.01
gr24	1272	1272	969,837	193.34	1272	6	0.01
fri26	937	937	11,402,433	1,800.00	937	2	0.01
bayg29	1610	1610	6,393,643	1,800.00	1610	6	0.01

Instances from TSPLIB, time limit 1800s

bayg29 was the largest instance for which CPO could find a solution

This relaxation-based filtering now allows CP to scale up to rbg443 (asymmetric TSP), resp. a280 (symmetric TSP) [Fages & Lorca, 2012]

Outline

- Global constraint propagation
  - network flows
  - optimization constraints
- Integrating relaxations
  - Linear Programming relaxation
  - Lagrangean relaxation
- Decomposition methods
  - logic-based Benders
  - column generation

Motivation

- Many practical applications are composed of several subproblems
  - *facility location*: assign orders to facilities with minimum cost, but respect facility constraints
  - *vehicle routing*: assign pick-up locations to trucks, while respecting constraints on truck (capacity, driver time, …)
- By solving subproblems separately we can
  - be more scalable (decrease solving time)
  - exploit the subproblem structure
- OR-based decomposition methods can preserve optimality

Motivation for integrated approach

Example: airline crew rostering

- Crew members are assigned a schedule from a huge list of possible schedules
  - this is a 'set covering' problem: relatively easy for IP/LP
- New schedules are added to the list as needed
  - many challenging scheduling constraints – difficult for MIP, but doable for CP

- Integrated OR/CP decompositions broaden the applicability to more complex and larger applications

Benders Decomposition

Benders decomposition can be applied to problems of the form:

$$\begin{array}{lll} \min & v = & f(x,y) \\ \text{s.t.} & & S(x,y) \\ & & x \in D_x, y \in D_y \end{array}$$

When fixing variables x, the resulting problem may become much simpler:

$$\begin{array}{ll} \min & f(\overline{x},y) \\ \text{s.t.} & S(\overline{x},y) \\ & y \in D_y \end{array}$$

Example: multi-machine scheduling

- variables x assign tasks to machines
- variables y give feasible/optimal schedules per machine
- when fixing x, the problem decouples into independent single-machine scheduling problems on y

Benders Decomposition (cont'd)

Iterative process

- Master problem: search over variables x
  - optimal solution x^k in iteration k
- Subproblems: search over variables y, given fixed x^k
  - optimal objective value v^k
- Add *Benders cut* to master problem

$$v \geq B_k(x) \qquad \text{(such that } B_k(x^k) = v^k \text{)}$$

Bounding

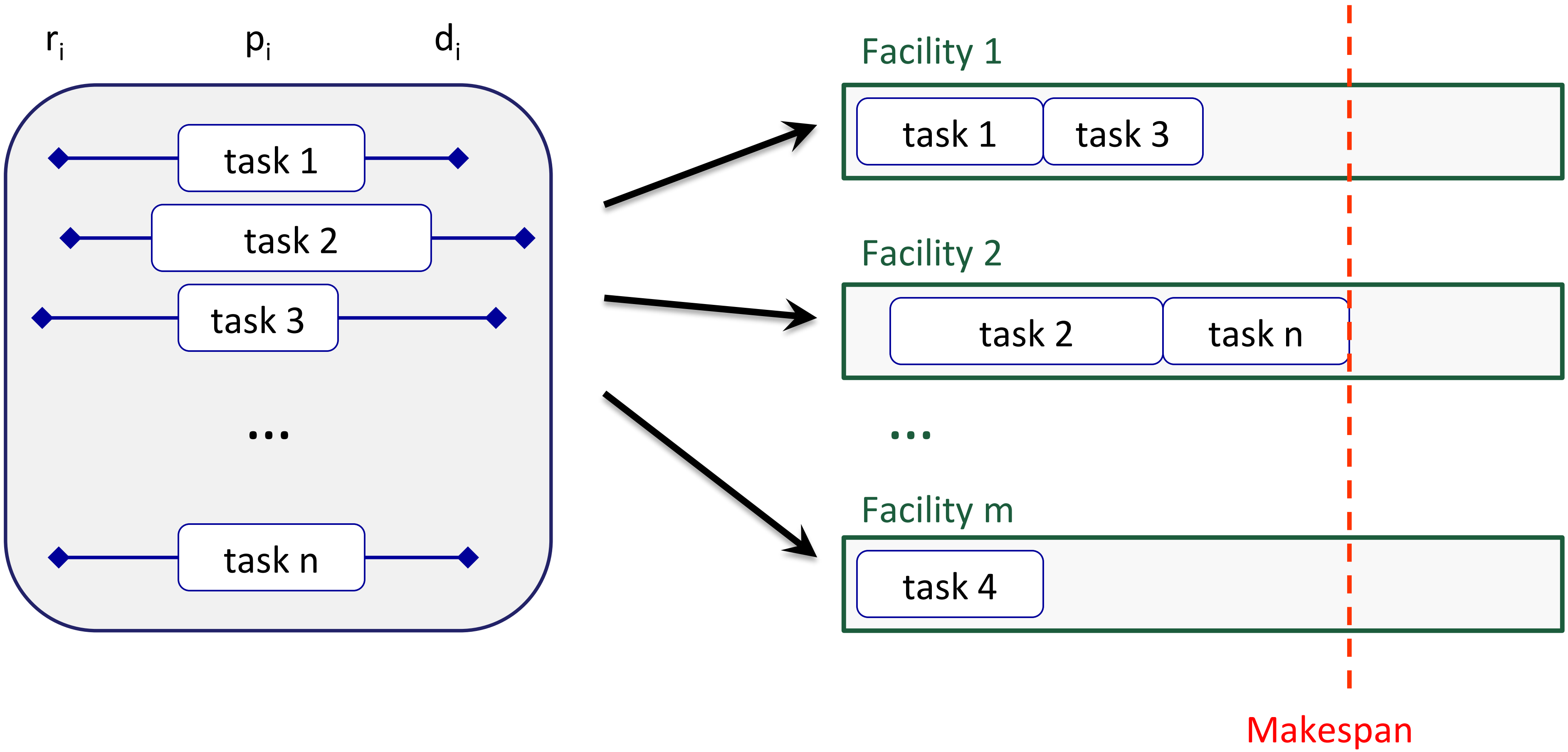
- Master is relaxation: gives lower bound
- Subproblem is restriction: gives upper bound
- Process repeats until the bounds meet

Logic-based Benders

- Original Benders decomposition applies to LP and NLP problems
  - Based on duality theory to obtain Benders cuts
- However, the concept is more general
  - Logic-based Benders: generalizes LP-based Benders to other types of inference methods, using ‘inference duality’
  - Also allows additional types of ‘feasibility’ cuts (nogoods)
  - Moreover, CP can be applied to solve the subproblems

[Jain & Grossmann, 2001] [Hooker & Ottoson, 2003]

Example: Task-Facility Allocation



Logic-Based Benders Scheme

Assign tasks to facilities (MIP)

task assign-ments T(f)

Benders cuts

Find schedule for each facility f (CP)

min Makespan

s.t. $\sum_f x_{if} = 1$, for all i

Makespan $\geq (\sum_i L_{if}\, x_{if})$ /Capacity(f), for all f

x_{if} in {0,1}

Benders cuts; LBs and feasibility

min Max(EndOf(T(f)))

s.t. ParallelSchedule(T(f), Capacity(f))

Cumulative Resource

[Hooker, 2007]

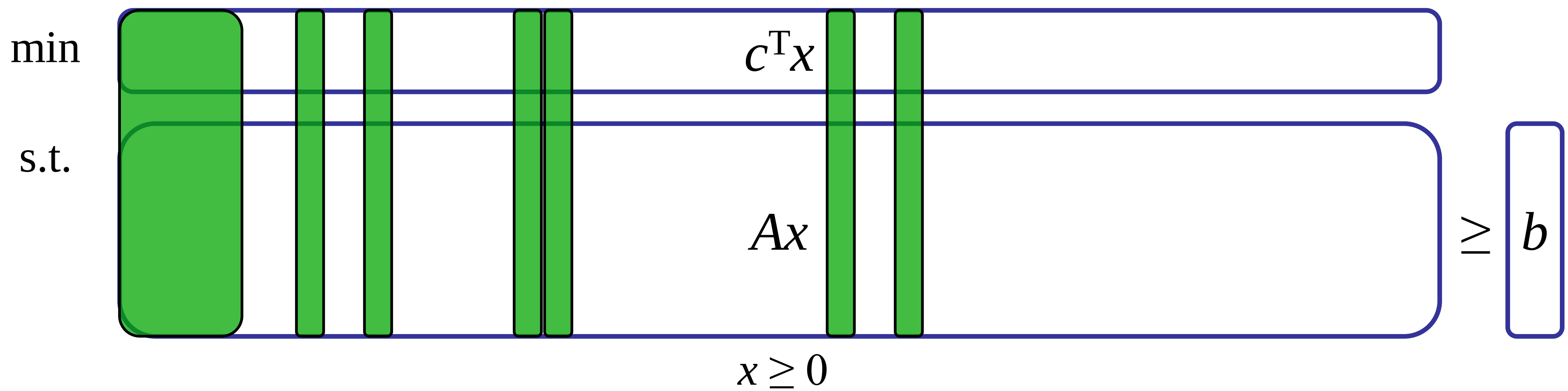
Pros and Cons

- Benefits of Logic-based Benders
  - reported orders of magnitude improvements in solving time w.r.t. CP and MILP [Jain & Grossmann, 2001], [Hooker, 2007]
  - CP models very suitable for more complex subproblems such as scheduling, rostering, etc.

- Potential drawbacks
  - finding good Benders cuts for specific application may be challenging
  - feasible solution may be found only at the very end of the iterative process

Column Generation

- One of the most important techniques for solving very large scale linear programming problems
  - perhaps too many variables to load in memory



- Delayed column generation (or variable generation):
  - start with subset of variables (‘restricted master problem’)
  - iteratively add variables to model until optimality condition is met

Column Generation (cont'd)

Column generation process:

- Solve for subset of variables S (assume feasible)
- This gives shadow prices λ for the constraints
- Use reduced costs to *price* the variables not in S

$$
\begin{aligned}
\min \quad & c_S^\mathsf{T} x_S \\
\text{s.t.} \quad & A_S x_S \geq b \\
& x_S \geq 0
\end{aligned}
$$

$$\overline{c}_i = c_i - \sum_j \lambda_j a_{ij}$$

- If $\overline{c}_i < 0$, variable x_i may improve the solution:

  add x_i to S and repeat

- Otherwise, we are LP-optimal (since all reduced costs are nonnegative)

How can we find the best variable to add?

Pricing Problem

- Solve *optimization problem* to find the variable (column) with the minimum reduced cost:

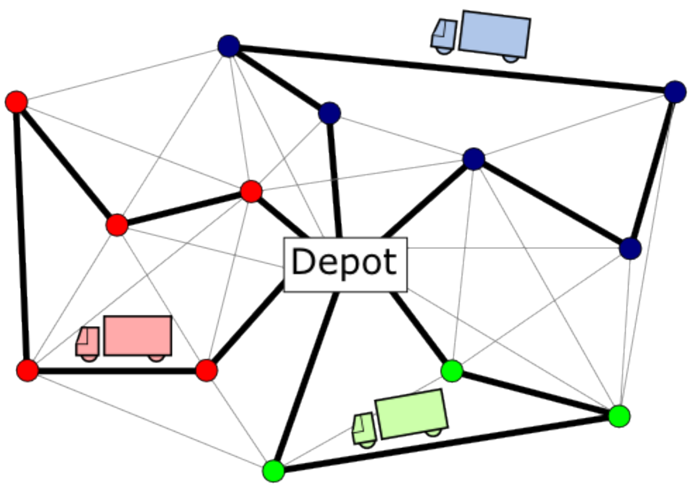
$$\begin{array}{ll} \min & c_y - \lambda^\mathsf{T} y \\ \text{s.t.} & y \text{ is a column of } A \end{array}$$

- In many cases, columns of A can be described using a set of (complicated) constraints

- Remarks:
  - *any* negative reduced cost column suffices (need not be optimal)
  - CP can be suitable method for solving pricing problem

Application: Capacitated Vehicle Routing

- Set of clients V, depot d
- Set of trucks (unlimited, equal)
- Parameters:
  - distance matrix D
  - load w_j for each client j in V (unsplittable)
  - truck capacity Q
- Goal:
  - find an allocation of clients to trucks
  - and a route for each truck
  - respecting all constraints
  - with minimum total distance



Problem Formulation: Restricted Master

- Let R be (small) set of feasible individual truck routes
  - parameter $a_{rj} = 1$ if client j is on route $r \in R$
  - parameter c_r represent the length of route $r \in R$
- Let binary variable x_r represent whether we use route $r \in R$
- Set covering formulation:

c	$c_{\{1\}}$	$c_{\{2\}}$	$c_{\{3\}}$	$c_{\{4\}}$	$c_{\{2,4\}}$	$\cdots$
client 1	1	0	0	0	0	
client 2	0	1	0	0	1	
client 3	0	0	1	0	0	
client 4	0	0	0	1	1	

$$
\begin{aligned}
\min \quad & \sum_{r \in R} c_r x_r \\
\text{s.t.} \quad & \sum_{r \in R} a_{rj} x_r \geq 1 && \forall j \in V \\
& 0 \leq x_r \leq 1 && \forall r \in R
\end{aligned}
$$

shadow price λ_j for all j

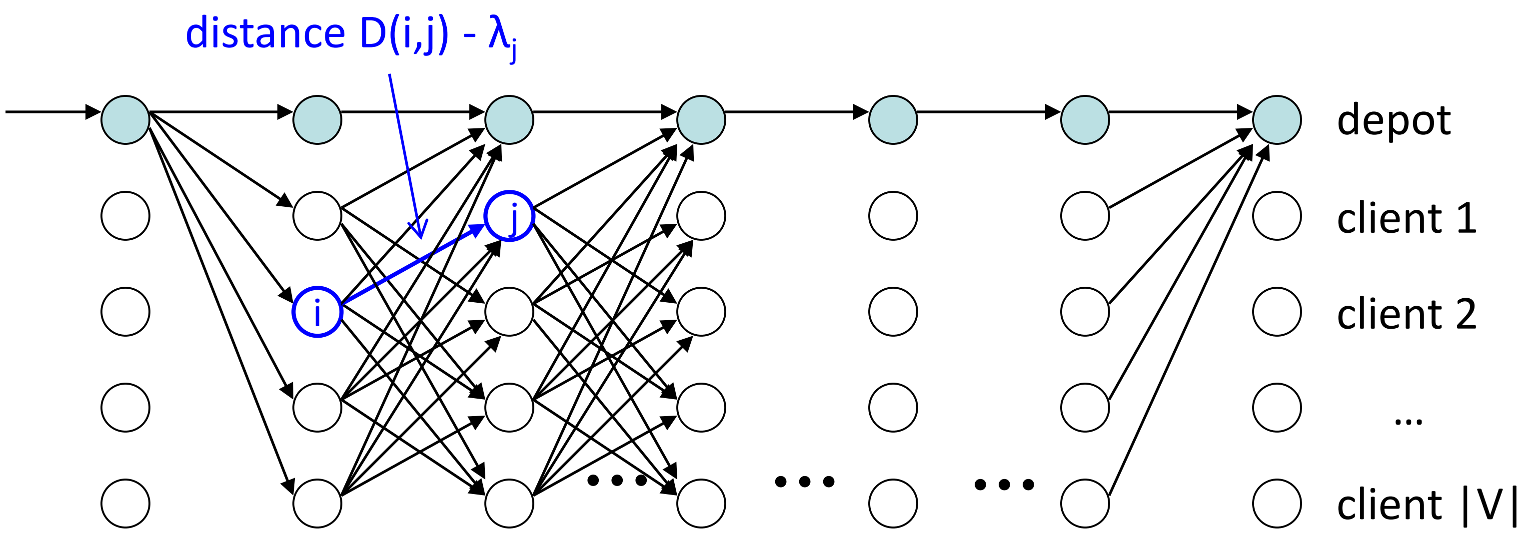
continuous LP relaxation

Pricing Problem

- Truck route similar to TSP, but
  - not all locations need to be visited
  - there is a capacity constraint on the trucks

- We can solve this problem in different ways
  - shortest path problem in a layered graph
  - single machine scheduling problem

Pricing as shortest path



Binary variable y_{ijk}: travel from location i to j in step k

Constraints:

- variables y_{ijk} must represent a path from and to the depot
- we can visit each location at most once
- total load cannot exceed capacity Q

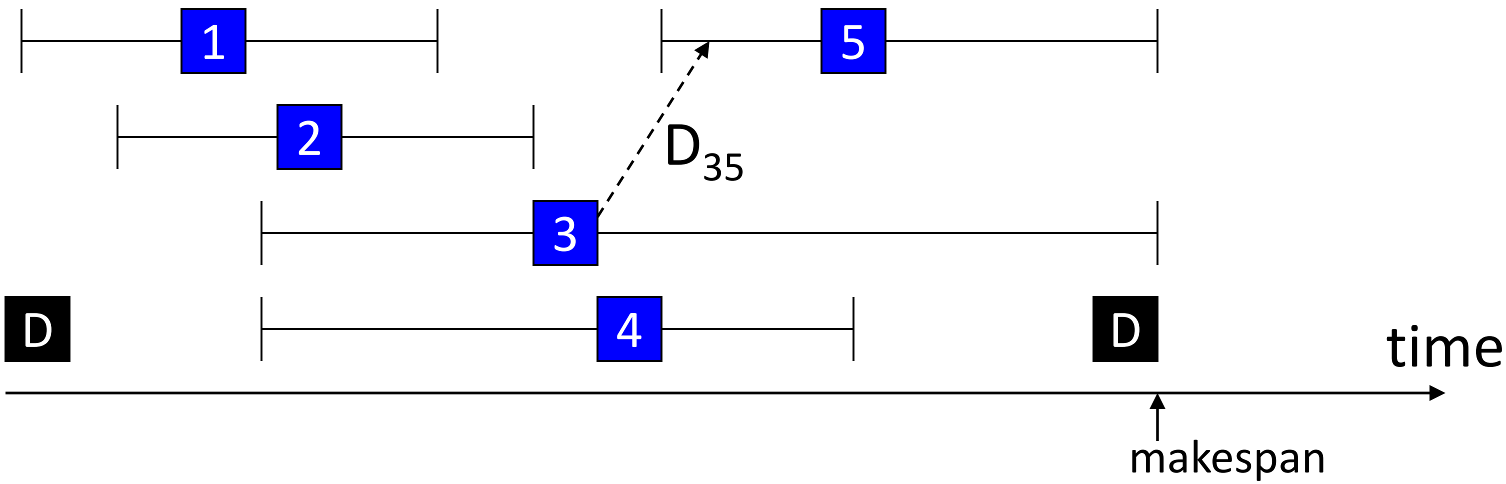
This model can be solved by IP (or dedicated algorithms)

Benefit of using CP

- We can use CP to solve the pricing problem:
  - represent the constrained shortest path as CP model,
  - or we can view the pricing problem as a single machine scheduling problem
- A major advantage is that CP allows to add many more side constraints:
  - time window constraints for the clients
  - precedence relations due to stacking requirements
  - union regulations for the drivers
  - …
- In such cases, other methods such as IP may no longer be effective

From TSP to machine scheduling

- Vehicle corresponds to 'machine' or 'resource'
- Visiting a location corresponds to 'activity'



- Sequence-dependent setup times
  - Executing activity j after activity i induces setup time D_{ij} (distance)
- Minimize 'makespan' (or sum of the setup times)
- Activities cannot overlap (disjunctive resource)

CP Model

- Activities (or interval variables):
  - Optional activity *visit*[j] for each client j (duration: 0)
  - *StartAtDepot*
  - *EndAtDepot*

- Transition times between two activities i and j
  - T[i,j] = D(i,j) – λ_j

CP Model (cont'd)

minimize $\quad EndAtDepot.\text{end} - \sum_j \lambda_j(\text{Visit[j].present})$

s.t. DisjunctiveResource(

Activities: Visit[j], StartAtDepot, EndAtDepot

Transition: T[i,j]

First: *StartAtDepot*

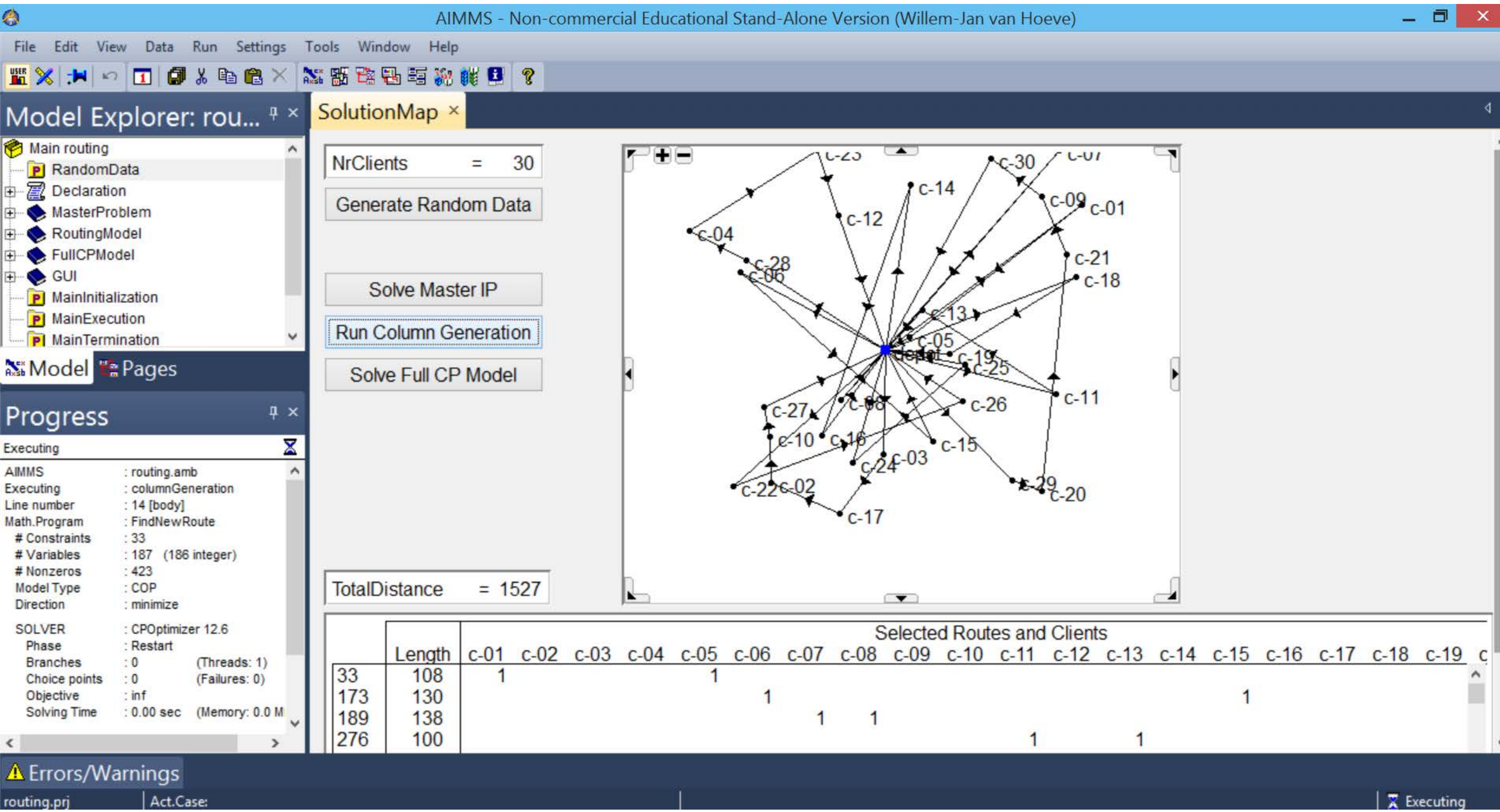
Last: *EndAtDepot*)

$\sum_j w_j(\text{Visit[j].present}) \le Q$

- Observe that this model naturally allows to add time windows (on Visit[j]), precedence relations, etc

Demo

Carnegie Mellon Tepper SCHOOL OF BUSINESS



Discussion

- Benefits of column generation
  - A small number of variables may suffice to prove optimality of a problem with exponentially many variables
  - Complicated constraints can be moved to subproblem
  - Can stop at any time and have feasible solution (not the case with Benders)
- Potential drawbacks / challenges
  - LP-based column generation still fractional: need branch-and-price method to be exact (can be challenging)
  - For degenerate LPs, shadow prices may be non-informative
  - Difficult to replace single columns: need *sets* of new columns which are hard to find simultaneously

Summary

- Various ways to integrate CP and OR
  - Global constraint propagation (e.g., network flows)
  - Integrating relaxations (LP, Lagrangian, SDP)
  - Decomposition methods (Benders, column generation)
- Very active research area
  - SCIP solver [Achterberg et al. 2007-]
  - CP/OR + local search [Michel&Van Hentenryck 2005-] [Benoist et al, 2011]
  - SAT and CP reasoning in MIP solving [Achterberg et al. 2013]
  - SAT+CP in Lazy Clause Generation [Ohrimenko, Stuckey, et al., 2007-]
  - SAT+OR techniques for MaxSAT problems [Davies 2013]
  - CP+machine learning [Bartolini et al. 2011] [Lombardi&Milano, 2013]
  - …many more examples by Hadrien Cambazard tomorrow