

Resource Constrained Shortest Paths

Natashia Boland
Irina Dumitrescu

Mathematics and Statistics, University of Melbourne
&
HEC Montreal, Canada Research Chair in Distribution Management

Time Windows

- A path in a network $G=(V,A)$ may represent a route for a delivery vehicle
- In this case, the time that the vehicle makes each delivery may be constrained
- T_{ij} = time needed for vehicle to drive from node i to node j (may includes unloading at i)
- $[a_i, b_i]$ = time interval must arrive at node i in
- t_i = time arrive at node i
- If (i,j) in path, then $t_j = \max\{t_i + T_{ij}, a_j\}$
- Require $t_i \leq b_i$ for all $i \in V$

Outline

- Various constrained shortest path models
- Single additive resource constrained shortest paths
 - Problem complexity
 - Preprocessing
 - A label setting algorithm
 - Lagrangian relaxation
 - Using Lagrangian information within label setting
- Elementary constrained shortest paths

Integer LP Formulation

$$\begin{aligned} \min_{x \in \{0,1\}^{|A|}, t} \quad & \sum_{(i,j) \in A} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j: (i,j) \in A} x_{ij} - \sum_{j: (j,i) \in A} x_{ji} = \begin{cases} -1, & i=s \\ 0, & i \neq s,t \\ 1, & i=t \end{cases} \quad \forall i \in V \end{aligned}$$

$$t_j \geq t_i + T_{ij} - (b_i + T_{ij} - a_j)(1 - x_{ij}), \quad \forall (i,j) \in A$$

$$a_i \leq t_i \leq b_i, \quad \forall i \in V$$

Additive Arc Resources

- A path in a network $G=(V,A)$ may represent a route for a delivery vehicle
- The petrol consumed by the vehicle may be constrained by its tank capacity, Q
- D_{ij} = petrol consumed on arc (i,j)
- d_i = total petrol consumed visiting all customers on the route up to and including i
- If (i,j) in path, then $d_j = d_i + D_{ij}$
- Require $d_i \leq Q$ for all $i \in V$ (or just $d_t \leq Q$)
- Special case of time windows $[0,Q]$

Additive Node Resources

- A path in a network $G=(V,A)$ may represent a route for a delivery vehicle
- The quantity that can be carried on the vehicle may be constrained by its capacity, Q
- D_i = demand quantity for node i
- d_i = total demand for all customers preceding on the route, including i
- If (i,j) in path, then $d_j = d_i + D_j$
- Require $d_i \leq Q$ for all $i \in V$ (or just $d_t \leq Q$)
- Special case of additive arc resource

ILP Formulation

$$\begin{aligned} \min_{x \in \{0,1\}^{|A|}} \quad & \sum_{(i,j) \in A} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j: (j,i) \in A} x_{ji} - \sum_{j: (i,j) \in A} x_{ij} = \begin{cases} -1, & i=s \\ 0, & i \neq s,t \\ 1, & i=t \end{cases} \quad \forall i \in V \\ & \sum_{(i,j) \in A} D_{ij} x_{ij} \leq Q \end{aligned}$$

ILP Formulation

$$\begin{aligned} \min_{x \in \{0,1\}^{|A|}} \quad & \sum_{(i,j) \in A} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j: (j,i) \in A} x_{ji} - \sum_{j: (i,j) \in A} x_{ij} = \begin{cases} -1, & i=s \\ 0, & i \neq s,t \\ 1, & i=t \end{cases} \quad \forall i \in V \\ & \sum_{(i,j) \in A} D_i x_{ij} \leq Q - D_t \end{aligned}$$

Vector Resources

- Arc (i,j) may use r_{ij}^k units of resource k
- Resource limits R^k for each resource k
- $q_i^k =$ cumulative units of resource k used on path nodes from s to i
- If (i,j) in path, then $q_j^k = q_i^k + r_{ij}^k$
- Require $q_t^k \leq R^k$ for all k
- (Alternatively, some resources may be time window type, not just additive)

ILP Formulation

$$\min_{x \in \{0,1\}^{|A|}} \sum_{(i,j) \in A} c_{ij} x_{ij}$$

s.t.

$$\sum_{j: (i,j) \in A} x_{ij} - \sum_{j: (i,j) \in A} x_{ij} = \begin{cases} -1, & i=s \\ 0, & i \neq s,t \\ 1, & i=t \end{cases} \quad \forall i \in V$$

$$\sum_{(i,j) \in A} r_{ij}^k x_{ij} \leq R^k, \quad \forall \text{resources } k$$

Renewable Resources

- A path may represent a sequence of work for a crew in a “job network”, perhaps including several duty periods with rests in between
- The resource can be replenished at some special nodes, $R \subseteq V$
- If (i,j) in path and $j \notin R$, then $d_j = d_i + D_j$
- So if $j \in R$ then free to re-set $d_j = 0$
- Require $d_i \leq Q$ for all $i \in V$

Elementary Paths

- Shortest path problems often arise as subproblems in column generation
- In these cases there may be negative length cycles – a shortest elementary path (no repeated nodes) is sought
- This can be modelled by using a vector of additive resources, one for each node i
- Resource $r_{ij}^k = 1$ if $k=i$, 0 otherwise $\forall i,k \in V$
- Resource limit $R^k = 1$ for each $k \in V$

Other Variations

- Multi-criteria optimization
- Dynamic shortest paths e.g. with time-dependent travel times, such as due to peak-hour congestion
- Subset disjoint paths
- Many others.....

Notation

RCSPP:

$$\begin{array}{ll} \min & c(p) \\ \text{s.t.} & p \text{ a path in } G \text{ from } s \text{ to } t \\ & r(p) \leq R \end{array}$$

$c(p)$ = sum of costs of arcs in p

$r(p)$ = sum of resources consumed on arcs in p

Single Resource

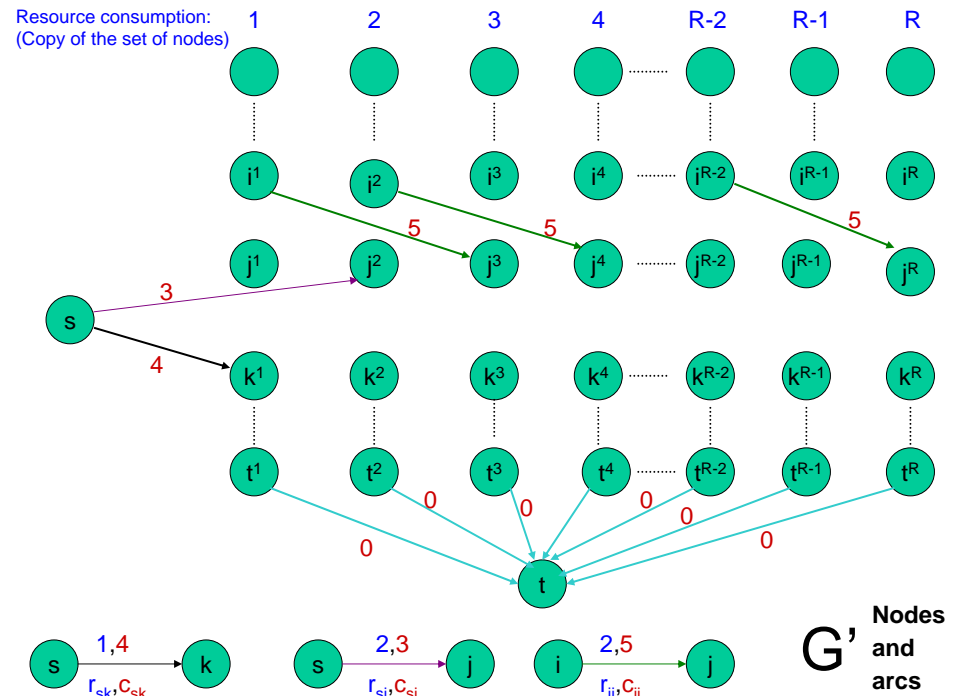
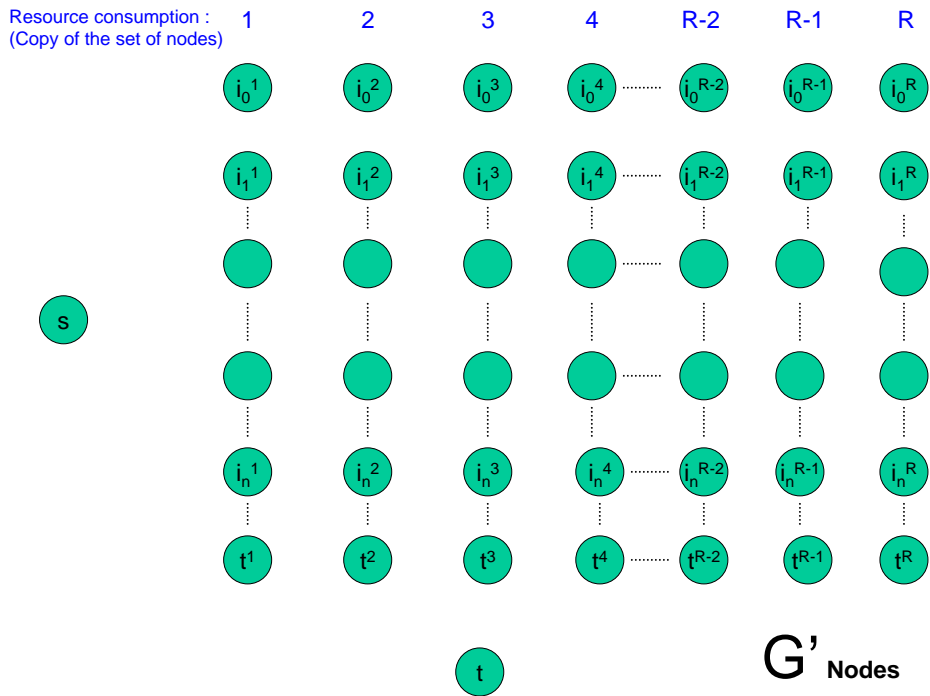
The Resource Constrained Shortest Path Problem (RCSPP)

- Directed graph $G=(V,A)$
- Start node s , end node t
- Costs c_a for arcs $a \in A$
- Resource consumption r_a for $a \in A$
- Resource limit R

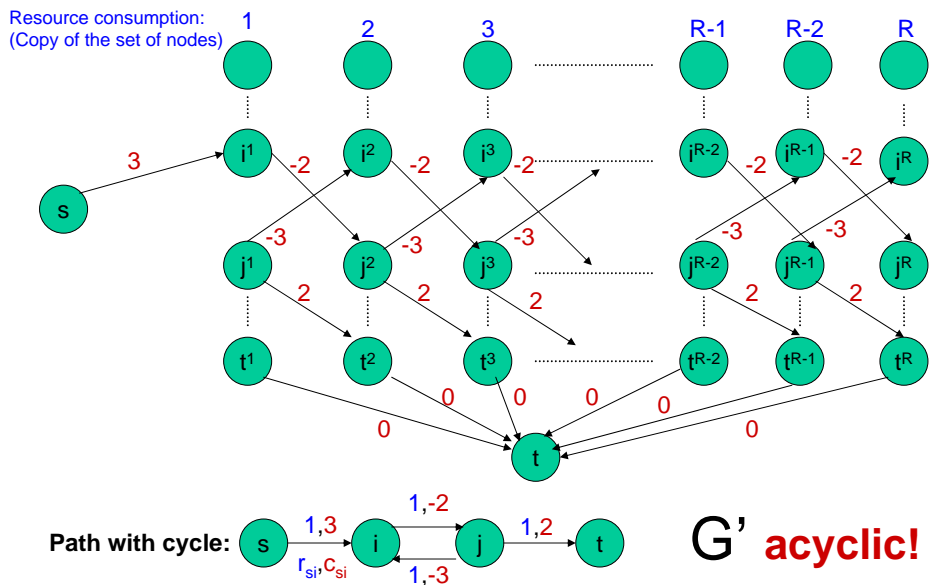
Find the least cost path from s to t consuming total resource no more than R

Complexity: RCSP and SPP

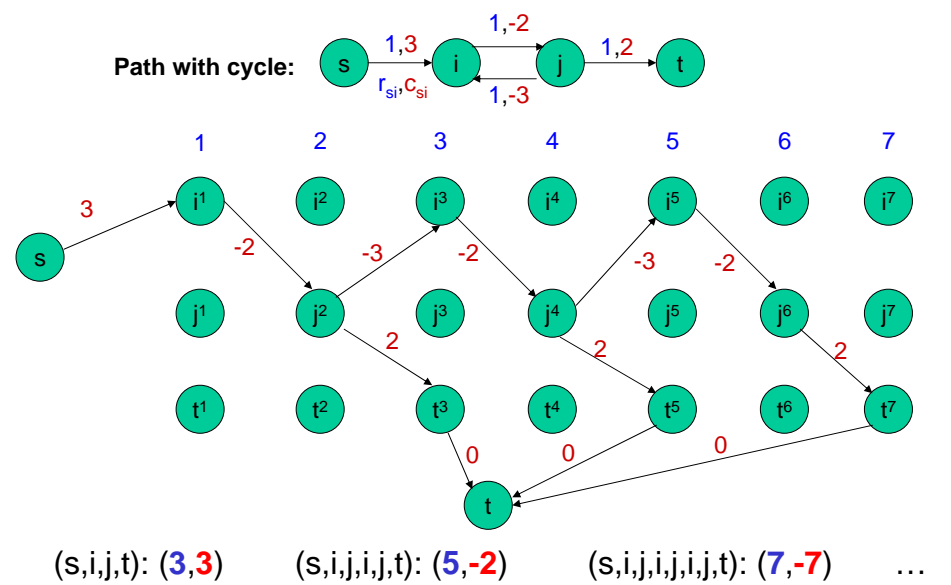
- The RCSPP can be transformed into an SPP on a new (acyclic !) graph
- Assumption: the resource consumption associated with every arc is positive
- The SPP is defined on the graph $G'=(V',A')$ obtained from the graph $G=(V,A)$
- Notation for G : $V=\{s,t,i_0,i_1,\dots,i_n\}$
- How do we obtain G' ?



RCSP: Negative Cost Cycles



RCSP: Negative Cost Cycles



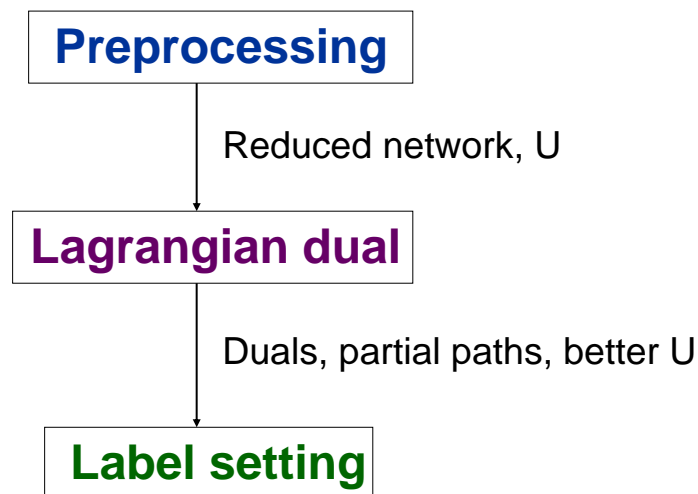
RCSPP Complexity

- Every resource-feasible path in G from s to t (possibly not simple) corresponds to a path in G' from s to t , of the same cost
- Every path in G' from s to t corresponds to a resource-feasible path in G from s to t (possibly not simple), of the same cost
- RCSPP in G equivalent to SPP in (acyclic) G'
- SPP solution time polynomial in $|V'|=O(|V|R)$
- Thus RCSPP is pseudopolynomially solvable

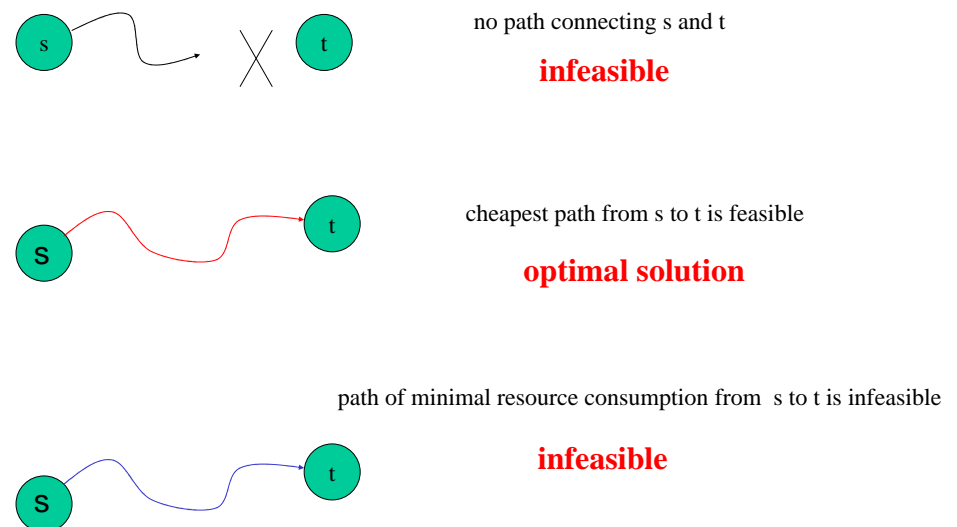
Previous approaches

- Exact
 - kth-shortest path methods: Eppstein (1997)
 - Dynamic programming/node labelling: Aneja et al (1983), Desrosiers et al (1983), Desrochers & Soumis (1988), Jaumard et al (1996)
 - Lagrangian relaxation: Handler & Zang (1980), Beasley & Christofides (1989), Mehlhorn & Ziegelmann (2000), Carlyle & Wood (2004)
- Approximate
 - cost scaling: Hansen (1979), Warburton (1987), Hassin (1992), Lorenz & Raz (2001)

Algorithm

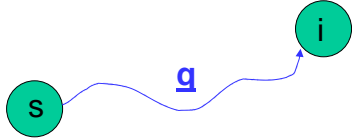


Preprocessing

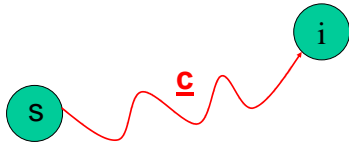


Preprocessing: Bounds

- U = upper bound (e.g. cost of least resource path)
- Lower bounds on paths (w.r.t. resource consumption and cost)

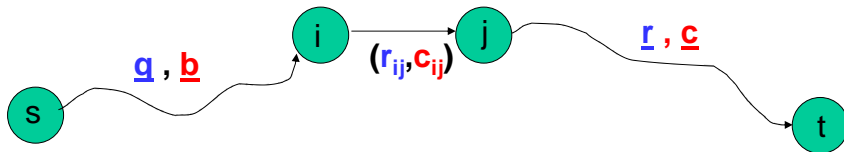


\underline{g} = resource consumed along the path of minimal consumption from s to i



\underline{c} = cost of the cheapest path from s to i

Preprocessing

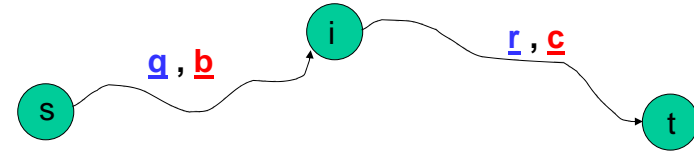


Eliminate arc (i,j) if either

$$\underline{g} + r_{ij} + \underline{r} > R \quad \text{or} \quad \underline{b} + c_{ij} + \underline{c} \geq U$$

Preprocessing

U = upper bound (e.g. cost of least resource path)



\underline{g} = lower bound on resource needed for $s \rightarrow i$ path

\underline{b} = lower bound on cost needed for $s \rightarrow i$ path

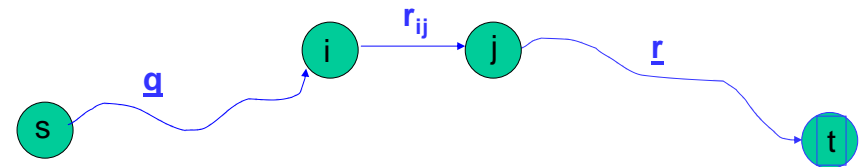
\underline{r} = lower bound on resource needed for $i \rightarrow t$ path

\underline{c} = lower bound on cost needed for $i \rightarrow t$ path

Eliminate node i if either

$$\underline{g} + \underline{r} > R \quad \text{or} \quad \underline{b} + \underline{c} \geq U$$

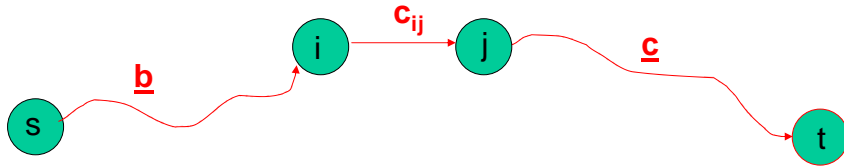
Upper Bound Update



Update upper bound if

$$\underline{g} + r_{ij} + \underline{r} \leq R \quad \text{and} \\ \text{cost of the path } (s, i, j, t) \text{ obtained} < U$$

Upper Bound Update



Update upper bound if

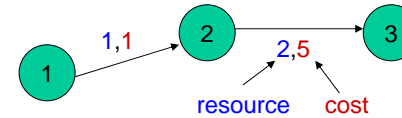
$$\underline{b} + \underline{c}_{ij} + \underline{c} < U \text{ and}$$

resource of the path (s, i, j, t) obtained $\leq R$

Repeat preprocessing until no change!!!

Labels and Dominance

- **Label** (at a node): a pair (resource, cost) associated with a path from s to that node

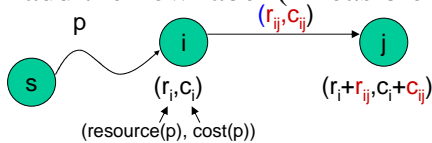


Path: (1,2,3), its corresponding label at node 3: (3,6)

- (resource₁, cost₁) **dominates** (resource₂, cost₂) if:
 - resource₁ \leq resource₂,
 - cost₁ \leq cost₂,
 - labels are not equal.
 - Examples: - (2,3) dominates (2,5) and (4,4);
- no dominance between (2,3) and (4,1).

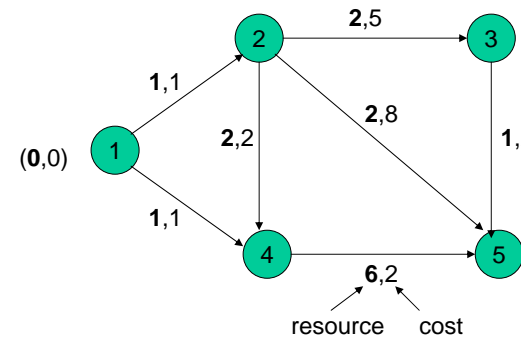
Label Setting Algorithm (LSA)

- **Treatment** of a label:
 - extend the path along outgoing arcs;
 - check dominance;
 - add the new label (if feasible and non-dominated)



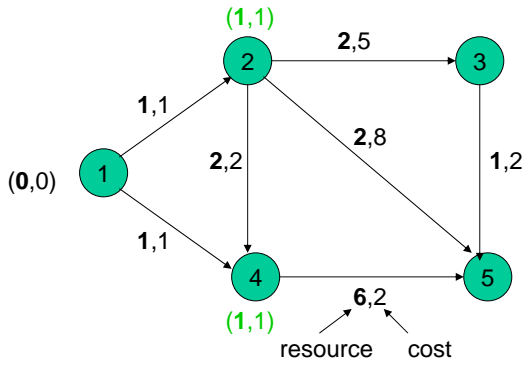
- LSA (Desrochers & Soumis, 1998):
 - Start with label (0,0) at s;
 - The labels not already visited are treated in increasing order of their resource consumption;

LSA - Example



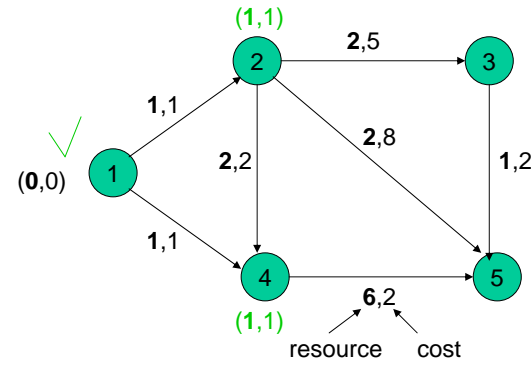
s=1
t=5
R=6

LSA - Example



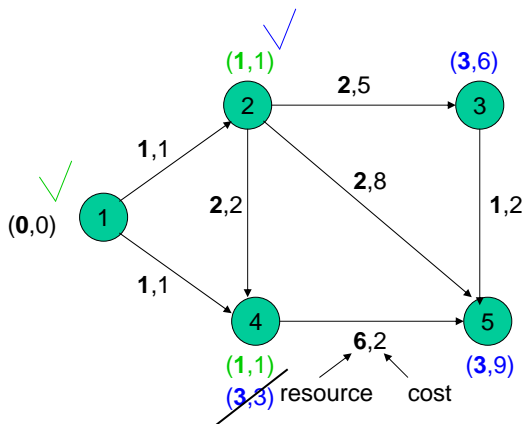
s=1
t=5
R=6

LSA - Example



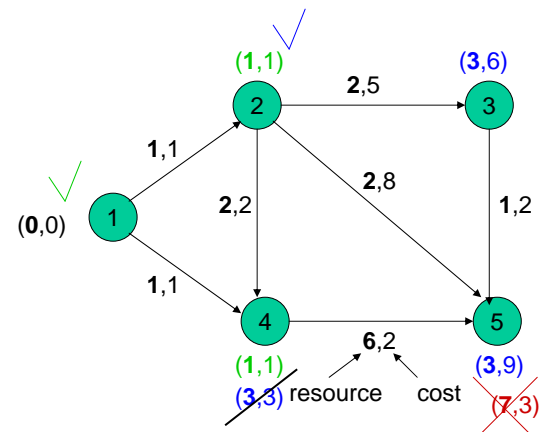
s=1
t=5
R=6

LSA - Example



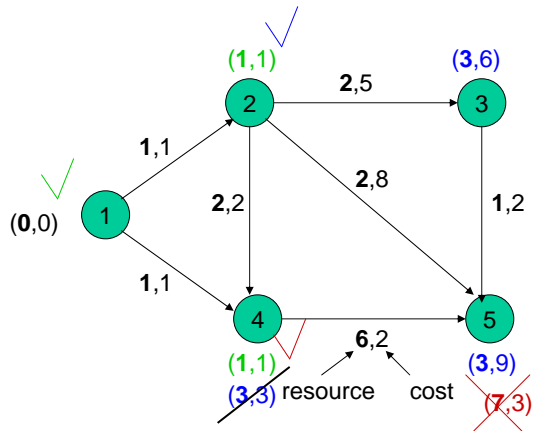
s=1
t=5
R=6

LSA - Example



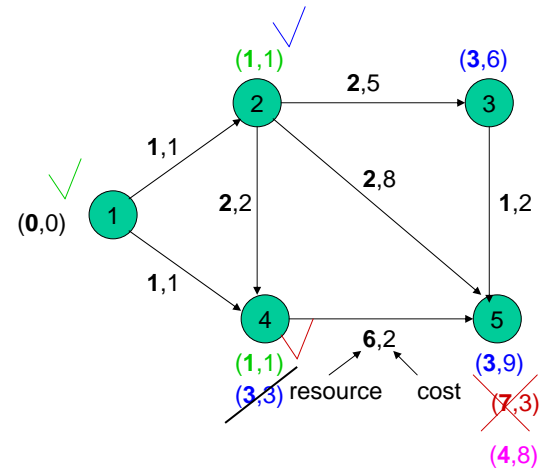
s=1
t=5
R=6

LSA - Example



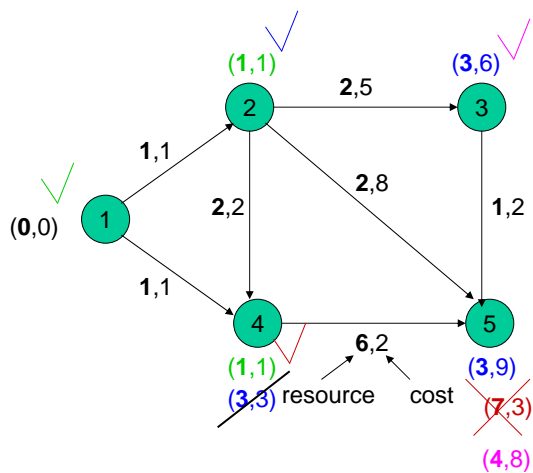
s=1
t=5
R=6

LSA - Example



s=1
t=5
R=6

LSA - Example



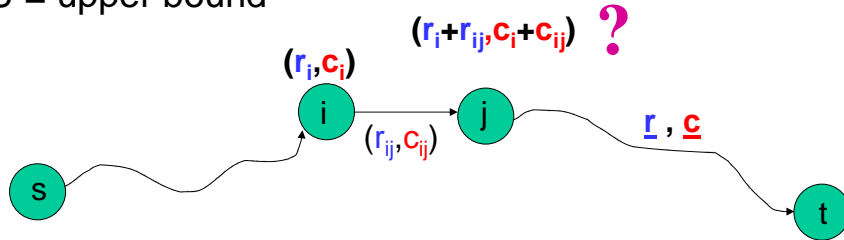
s=1
t=5
R=6

Notes on Algorithms

- Use of buckets yields $O(|V|^2R^2)$ algorithm
- Easily extended to multiple resources: treat labels in lexicographic resource vector order
- Label correcting: treat all labels on a node in one step
- See Handbook in OR & MS Vol. 8, Chapter 2, "Time-Constrained Routing and Scheduling", Desrosiers, Dumas, Solomon, Soumis
- Also Mehlhorn & Ziegelmann for treatment in increasing cost order, Carlyle & Wood for enumeration

Modified LSA (MLSA)

U = upper bound



\underline{r} = lower bound on resource needed for $j \rightarrow t$ path

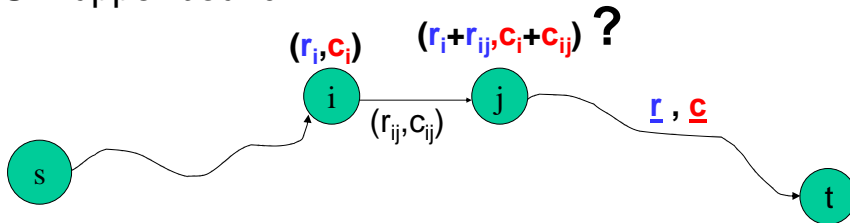
\underline{c} = lower bound on cost needed for $j \rightarrow t$ path

Extend (r_i, c_i) along arc (i, j) **only if**

$$r_i + r_{ij} + \underline{r} \leq R \quad \text{and} \quad c_i + c_{ij} + \underline{c} < U$$

Extending a path

U = upper bound



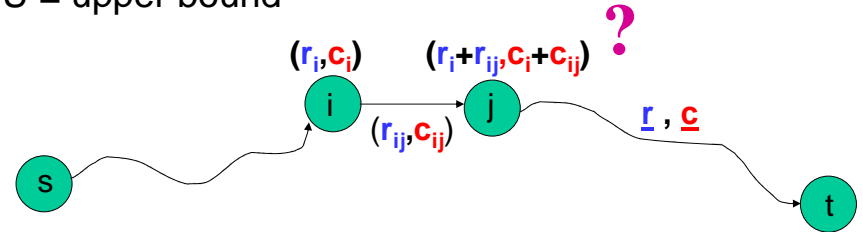
\underline{r} = lower bound on resource needed for $j \rightarrow t$ path

\underline{c} = lower bound on cost needed for $j \rightarrow t$ path

Can we do better?

Extending a path

U = upper bound



\underline{r} = least-resource $j \rightarrow t$ path

\underline{c} = least-cost $j \rightarrow t$ path

Extend (r_i, c_i) along arc (i, j) only if

$$r_i + r_{ij} + \underline{r} \leq R \quad \text{and} \quad c_i + c_{ij} + \underline{c} < U$$

Lagrangian Relaxation (LR)

$$z = \begin{cases} \min & cx \\ \text{s.t.} & x \text{ in } X \\ & Ax \leq b \quad \text{"hard constraints"} \end{cases}$$

- **Lagrangian relaxation**: lift the hard constraints into the objective function; obtain a problem that is easier to solve.

$$z_{LR}(\lambda) = \begin{cases} \min & cx + \lambda(Ax - b) \\ \text{s.t.} & x \text{ in } X \end{cases} \quad \lambda \geq 0$$

- Important property: $z_{LR}(\lambda) \leq z$ for any $\lambda \geq 0$.
- **Lagrangian dual** problem = finding the best lower bound:

$$\text{LD: } \max z_{LR}(\lambda), \lambda \geq 0$$

LR for RCSPP

- P^{ij} = set of paths from i to j

$$Z_{\text{RCSPP}} = \begin{cases} \min & c(P) \\ \text{s.t.} & P \text{ in } P^{\text{st}} \\ & r(P) \leq R \end{cases}$$

- **Lagrangian relaxation** for $\lambda \geq 0$:

$$Z_{\text{LR}}(\lambda) = \begin{cases} \min & c(P) + \lambda(r(P) - R) \\ \text{s.t.} & P \text{ in } P^{\text{st}} \end{cases}$$

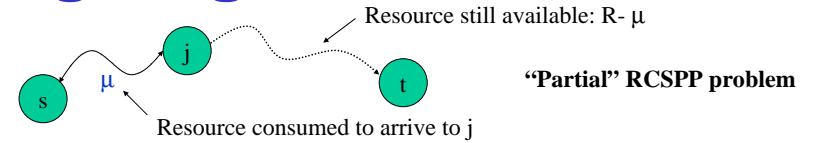
$$= \begin{cases} -\lambda R + \min & (c + \lambda r)(P) \\ \text{s.t.} & P \text{ in } P^{\text{st}} \end{cases}$$

- Calculating $Z_{\text{LR}}(\lambda)$ means calculating a shortest path problem with arc lengths given by $c + \lambda r$, solution P_λ

Finding them

- Using lengths $c_a + \lambda r_a$ for each $a \in A$, find the shortest path in G from j to t , for all nodes j .
- This yields $Z_{\text{LR}}^j(\mu, \lambda)$ for all j and any μ .
- Solve Lagrangian dual of RCSPP for paths from s to t using *Kelley's cutting plane method*.

Lagrangian lower bounds



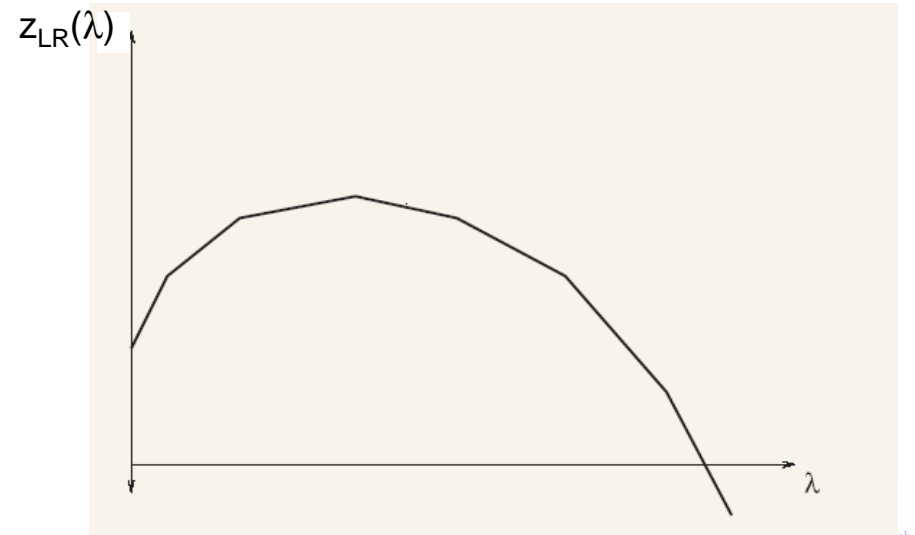
$$z^j(\mu) = \begin{cases} \min & c(p) \\ \text{s.t.} & p \text{ in } P^{jt} \\ & r(p) \leq R - \mu \end{cases}$$

Lower bound obtained for $\lambda \geq 0$:

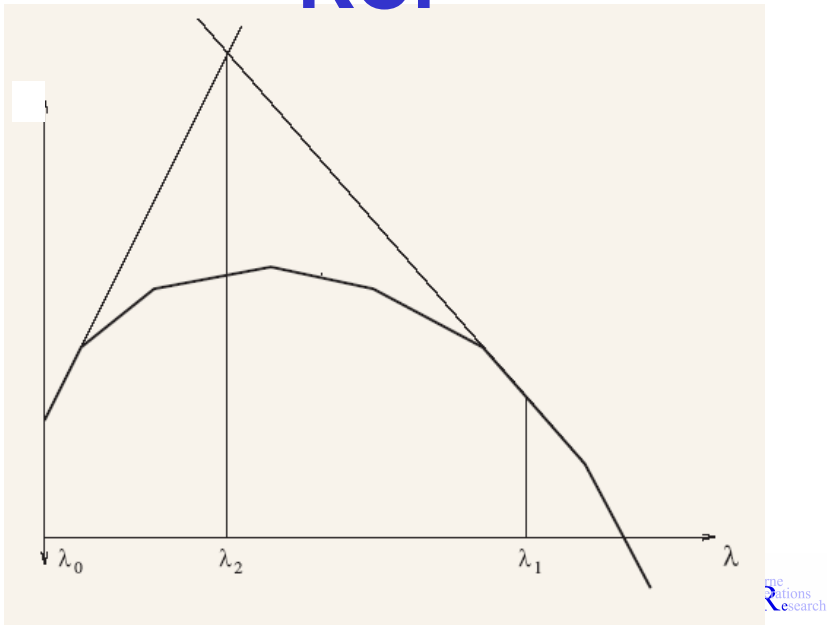
$$Z_{\text{LR}}^j(\mu, \lambda) = \begin{cases} \min & c(p) + \lambda(r(p) - (R - \mu)) \\ \text{s.t.} & p \text{ in } P^{jt} \end{cases}$$

$$= -\lambda(R - \mu) + \begin{cases} \min & c(p) + \lambda r(p) \\ \text{s.t.} & p \text{ in } P^{jt} \end{cases}$$

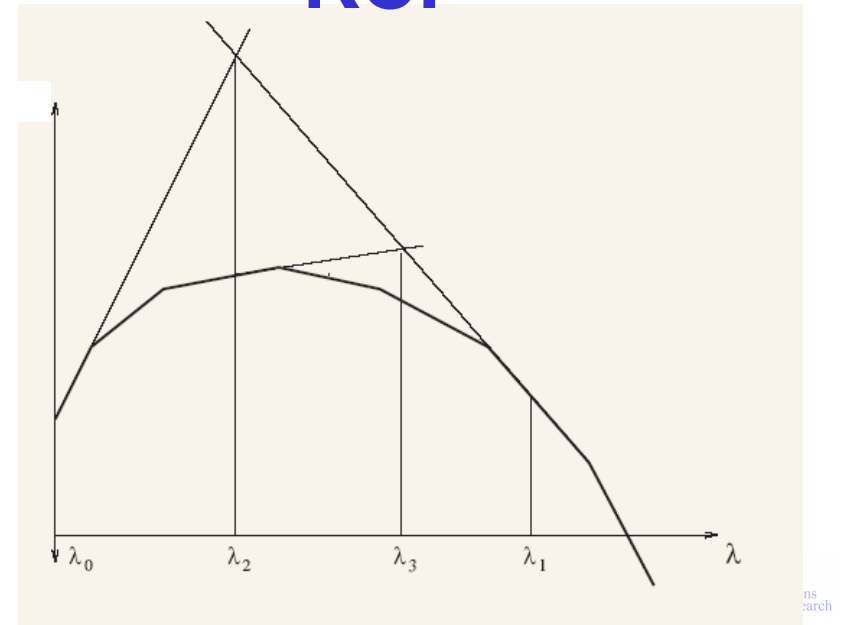
KCP



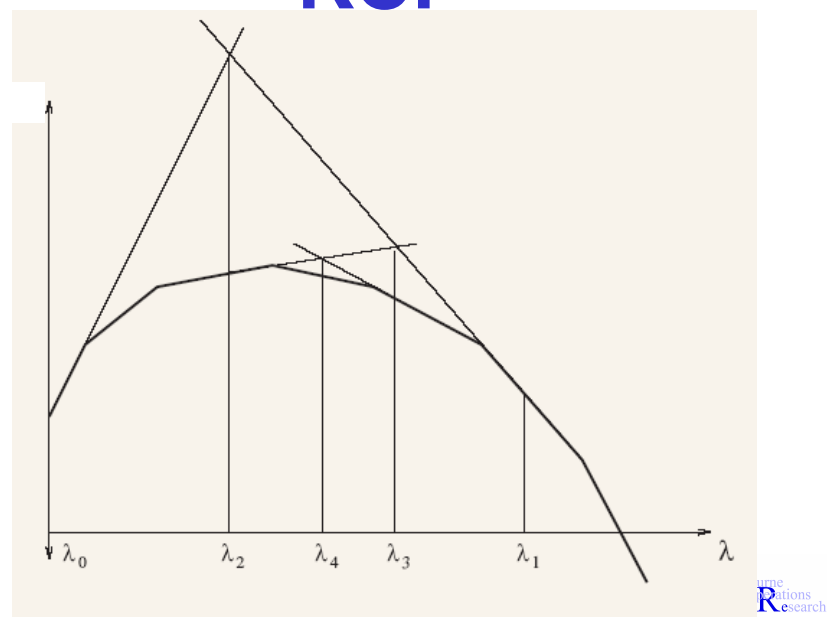
KCP



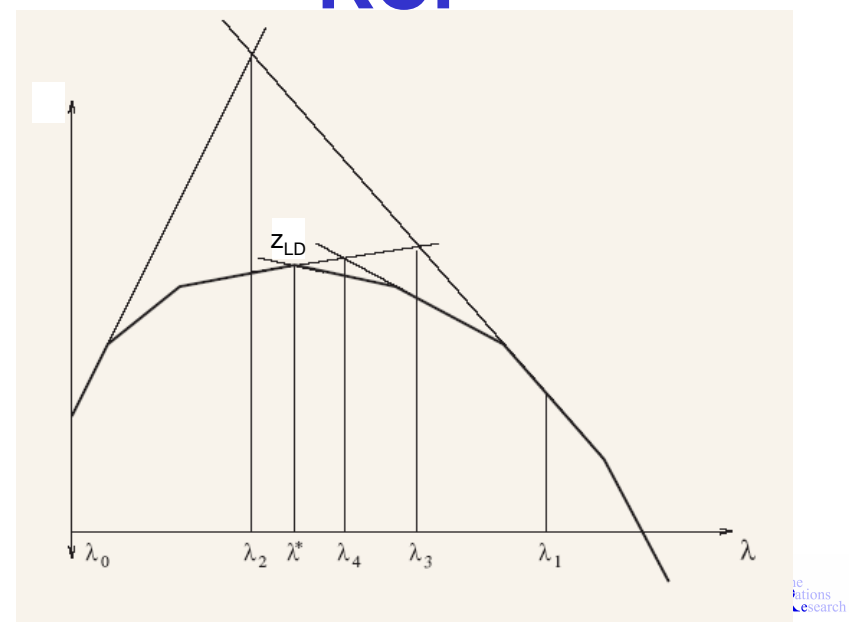
KCP



KCP



KCP



Kelley's Cutting Plane Method

Used to solve the Lagrangian dual:

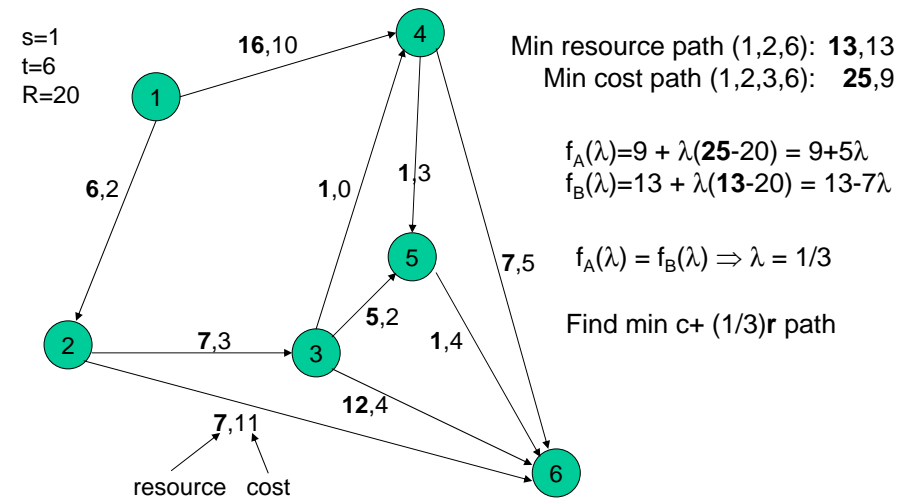
- keep two "active" values of λ : λ_A and λ_B such that $r(P_{\lambda_A}) - R > 0$ and $r(P_{\lambda_B}) - R < 0$.
- define two linear functions:

$$f_A(\lambda) = (r(P_{\lambda_A}) - R)\lambda + c(P_{\lambda_A})$$

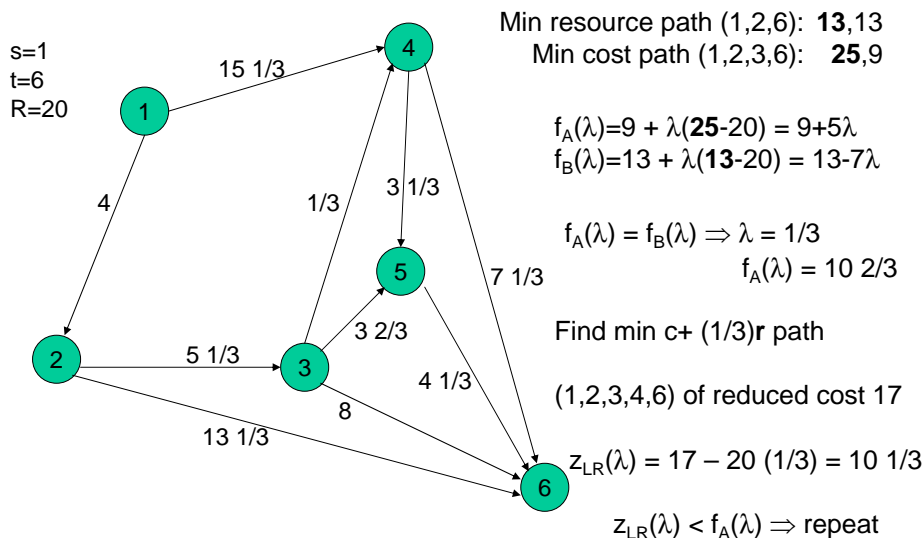
$$f_B(\lambda) = (r(P_{\lambda_B}) - R)\lambda + c(P_{\lambda_B})$$
- find λ' at the intersection of the two lines (i.e. $f_A(\lambda') = f_B(\lambda')$)
- STOP if $r(P_{\lambda'}) = R$ or $z_{LR}(\lambda') = f_A(\lambda')$.
- if $r(P_{\lambda'}) < R$ then $\lambda_B = \lambda'$, otherwise $\lambda_A = \lambda'$.

Initial λ values: $\lambda_A = 0$, $\lambda_B = U - c(\text{cheapest path } s \rightarrow t)$

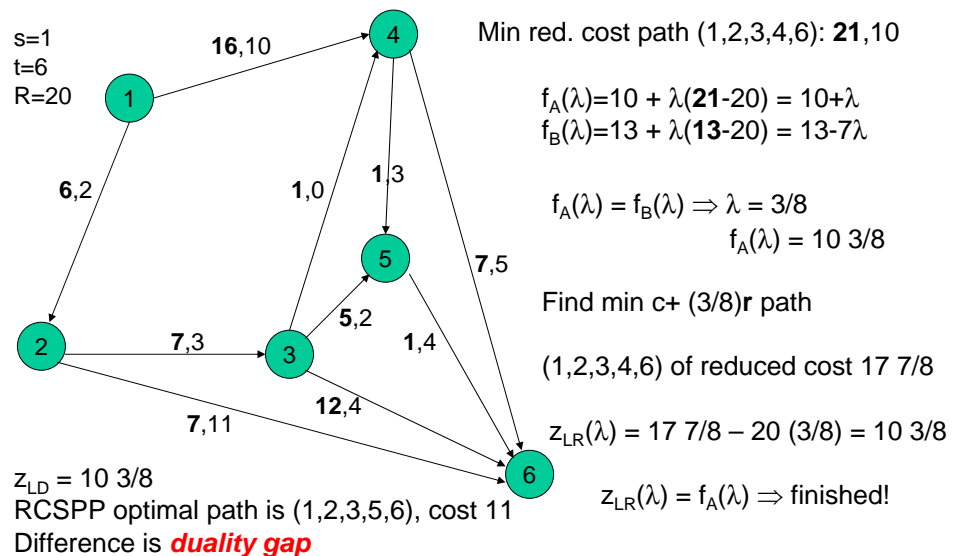
Lagrangian - Example



Lagrangian - Example



Lagrangian - Example



Finding them

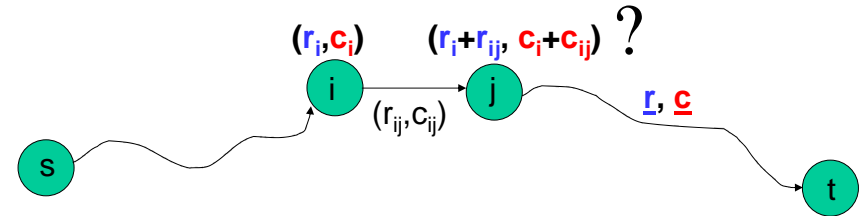
- Using lengths $c_a + \lambda r_a$ for each $a \in A$, find the shortest paths in G from j to t for all nodes j .
- This yields $z_{LR}^j(\mu, \lambda)$ for all j and any μ .
- Solve Lagrangian dual for paths from s to t using Kelley's cutting plane method.
- Save $\lambda_1, \dots, \lambda_K$ generated **and** length of shortest path from j to t for all j , at each iteration, L_1, \dots, L_K .
- $z_{LR}^j(\mu, \lambda_h) = -\lambda_h(R - \mu) + L_h$ is a *lower bound* on the least cost $j \rightarrow t$ path using resource no more than $R - \mu$.

Test Problems

- Class 3: randomly generated (Cherkassky et al.)
- Class 4: grid networks with random costs, resources
 - types 1 and 2 problems
- Class 5: US road networks
- Class 6: networks from digital elevation models

Extending a path

U = upper bound



\underline{r} = lower bound on resource needed for $j \rightarrow t$ path

$$\underline{c} = \max_{h=1, \dots, K} z_{LR}^j(R - r_i - r_{ij}, \lambda_h)$$

Extend (r_i, c_i) along arc (i, j) **only if**

$$r_i + r_{ij} + \underline{r} \leq R \quad \text{and} \quad c_i + c_{ij} + \underline{c} < U$$

Preprocessing results

Class	No. of nodes	No. of arcs	AAN		DB	
			node red. (%)	arc red. (%)	node red. (%)	arc red. (%)
4	602	1790	0.00	0.00	100	100
(1)	2502	7450	0.00	0.00	86.09	91.93
4	3002	8830	0.03	1.34	100	100
(2)	10002	29900	0.00	0.00	98.28	99.40
	40002	119800	0.00	0.00	100	100
	70002	209950	0.00	0.00	99.30	99.74
	135002	404850	0.00	0.00	98.35	96.67

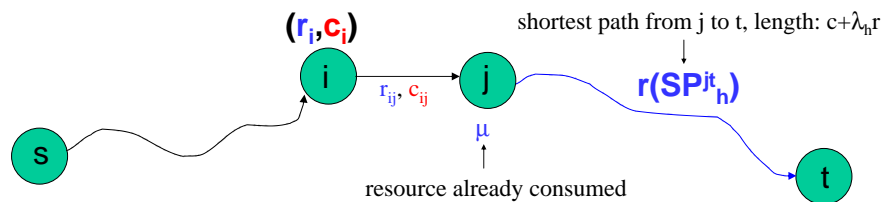
Extended Pruning

Problem Class	No. of nodes	LSA		MLSA	
		max. no. labels	time (sec)	max. no. labels	time (sec)
6-L	625	882	0.02	272	0.00
	2500	30707	1.72	7457	0.27
	5625	206311	27.89	37604	3.31
	15625 ₃	1416132 ¹	497.80 ¹	283287	70.29
	22500 ₃	1027544 ¹	491.98 ¹	472481	205.71
6-M	625 ₃	10562	0.16	2573	0.03
	2500	109976	7.63	20946	0.98
	5625	604935	89.63	97799	9.71
	15625	-	-	388537	156.47

With Lagrangian

Problem Class	No. of nodes	MLSA		MLSA with LR pruning				
		max. no. labels	time (sec)	LD-KCP		MLSA&LR		
				time (sec)	K	max. no. labels	% not pruned by 0, λ_K	time (sec)
4-L (Type 2)	3002	29949	1.61	0.49	6	26611	87.49	1.44
	10002	20368	2.57	0.62	7	5131	80.90	0.65
	40002	-	-	19.75	8	14458	77.19	22.01
	70002	-	-	29.18	9	1993	90.09	5.90
	135002	-	-	107.54	8	42564	99.84	246.79
4-M (Type 2)	3002	149770	8.10	0.52	6	25723	88.41	1.37
	10002	329087	58.30	3.03	7	837	94.56	0.14
	40002	-	-	19.62	8	4032	98.60	6.27
	70002	-	-	30.54	7	5177	99.92	14.77
	135002	-	-	123.42	10	16667	90.53	93.14

Extra Upper Bound Update



if $r_i + r_{ij} + r(SP^{jt}_h) \leq R$ then $c_i + c_{ij} + c(SP^{jt}_h)$ is an upper bound

Check for all h (i.e. for $\lambda_1, \dots, \lambda_K$), $\mu = R - (r_i + r_{ij})$.

Best partial upper bound:

$$\underline{c}(\mu) = \min_{h=1, \dots, K} \{ c(SP^{jt}_h) : r(SP^{jt}_h) \leq \mu \}$$

Inside MLSA update upper bound:

$$U = \min\{U, c_i + c_{ij} + \underline{c}(R - (r_i + r_{ij}))\}$$

Extra Upper Bound Update

Problem Class	No. of nodes	MLSA with LR pruning						
		MLSA&LR			MLSA&LR-UB			
		max. no. labels	% not pruned by 0, λ_K	time (sec)	max. no. labels	% not pruned by 0, λ_K	time (sec)	no. of times UB further updated
4-L (Type 2)	3002	26611	87.49	1.44	2827	52.21	0.16	23
	10002	5131	80.90	0.65	1274	62.05	0.17	8
	40002	14458	77.19	22.01	2009	57.15	3.15	10
	70002	1993	90.09	5.90	415	83.94	1.18	2
	135002	42564	99.84	246.79	1078	99.50	6.17	6
4-M (Type 2)	3002	25723	88.41	1.37	691	61.71	0.04	7
	10002	837	94.56	0.14	539	92.96	0.09	1
	40002	4032	98.60	6.27	981	99.90	1.53	3
	70002	5177	99.92	14.77	1985	99.97	5.67	2
	135002	16667	90.53	93.14	3727	76.86	20.93	3

Conclusions

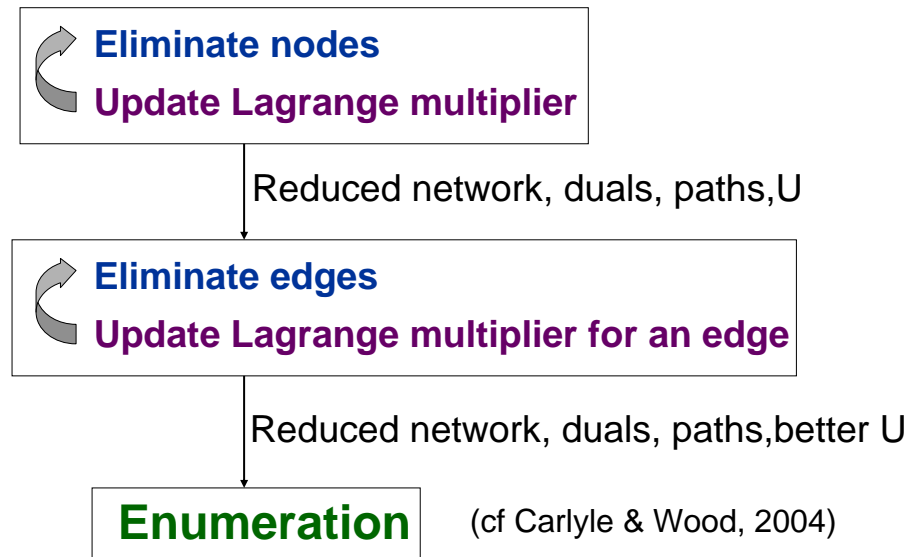
- Complete preprocessing is very effective
- Using lower bounds on path completion is very effective
- Lagrangian lower bounds can be efficiently calculated
- Aggressively updating upper bounds is best

Lagrangian Node Elimination

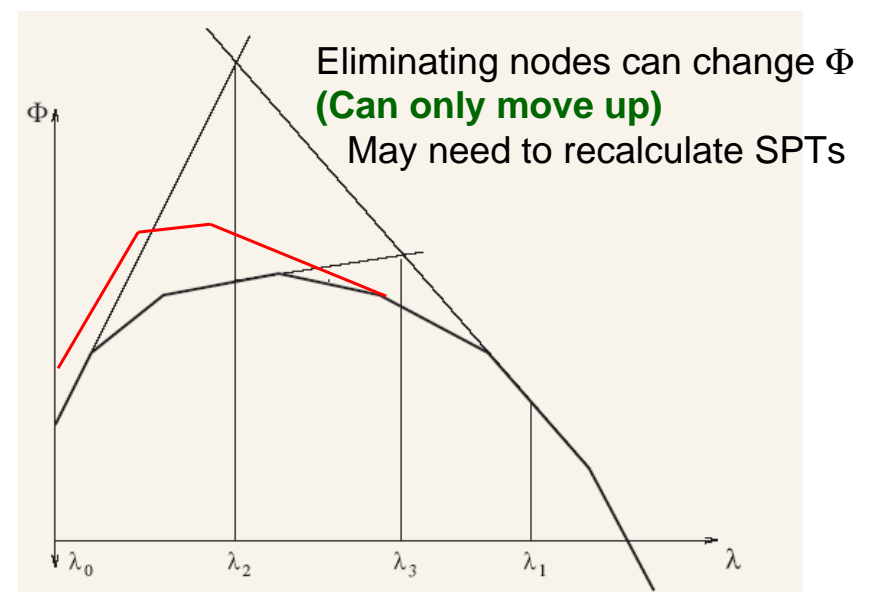
$$\begin{array}{l}
 \min c(p) \\
 \text{s.t. } \left. \begin{array}{l} p \text{ a path in } G \text{ from } s \text{ to } t \\ p \text{ passes through } i \end{array} \right\} X_i \\
 r(p) \leq R
 \end{array}$$

- $z_{LR}^i(\lambda) = \min \{ (c+\lambda r)(p) - \lambda R : p \in X_i \}$
- Can get lower bounds for individual nodes by combining forward and reverse shortest path trees
- Can eliminate nodes that violate upper bound
- Eliminate nodes at each value $\lambda_1, \dots, \lambda_K$

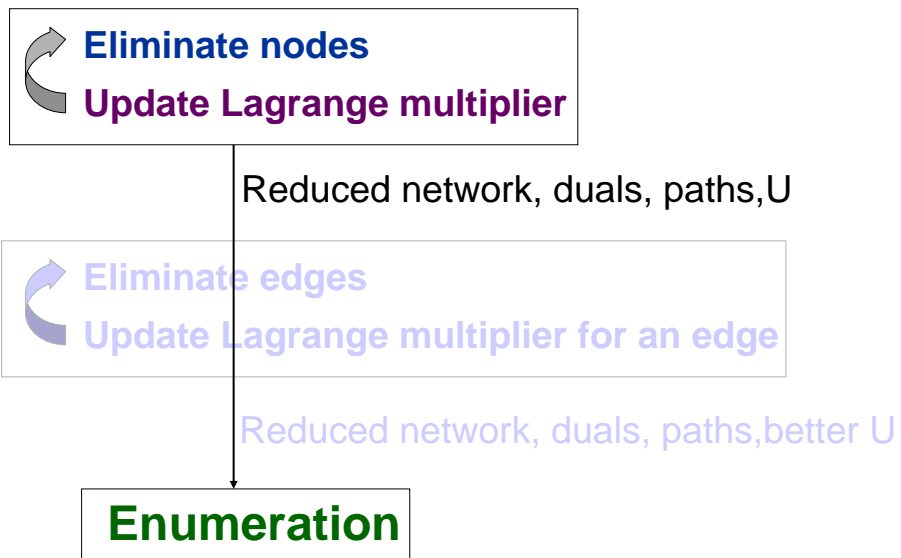
A new approach



Issue in KCP



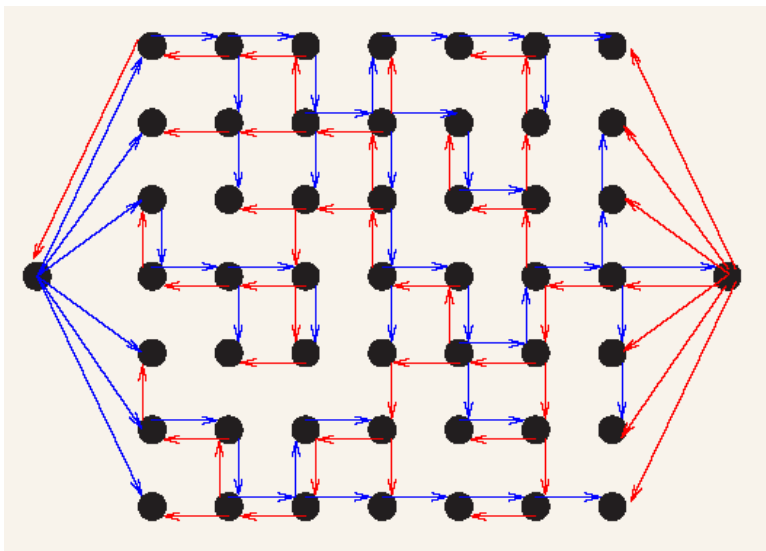
Algorithm



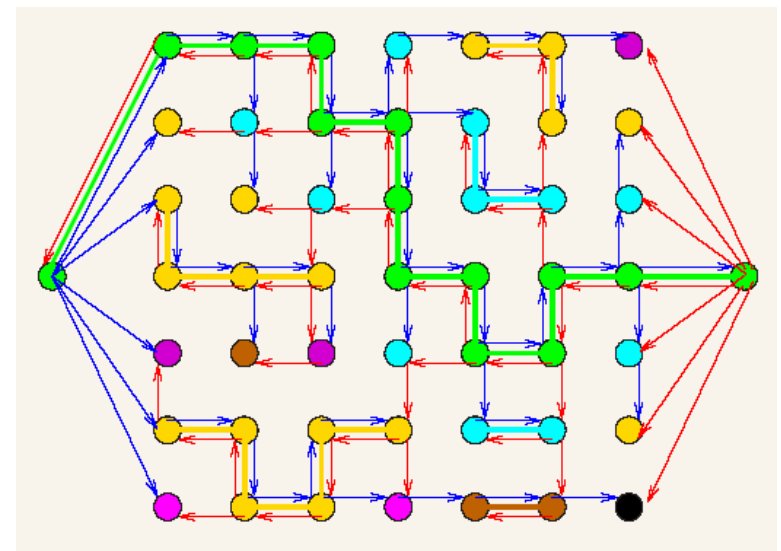
Node Elimination

- Nodes can be grouped into equivalence classes
- All nodes in the same equivalence class have same lower bound
- Equivalence classes have a tree-like structure with shortest path from s to t at root

Node Equivalence



Node Equivalence



Node Equivalence Theorem

- If nodes i and j are in the same equivalence class then

$$z_{LR}^i(\lambda) = z_{LR}^j(\lambda)$$

- So if can eliminate one node in equivalence class then can eliminate them all
- Equivalence classes themselves form a tree
- i in class higher in tree than class of j means

$$z_{LR}^i(\lambda) \geq z_{LR}^j(\lambda)$$

- So if can eliminate (class of) j then same for i

Test Problems

- Class 4, type 2

Performance: Small Networks

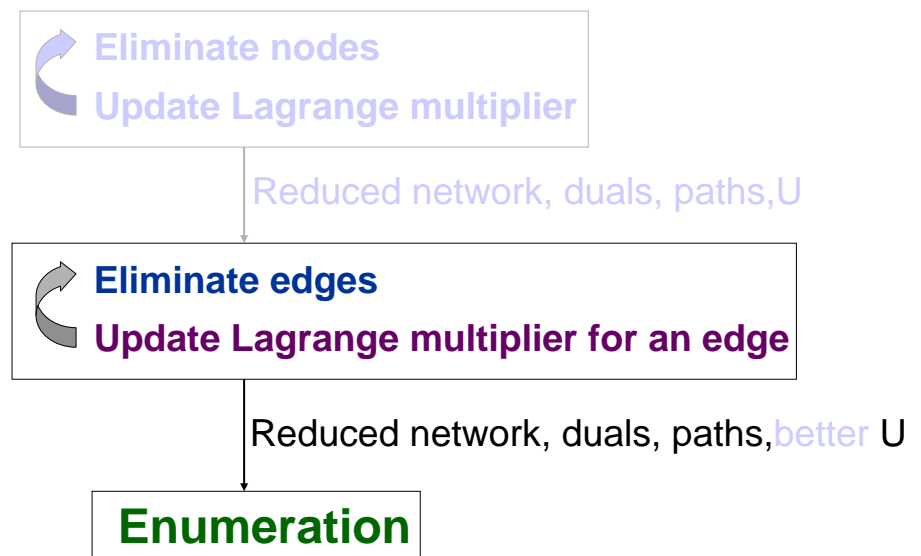
Problem				No Elim.(s)	Equiv. (s)	Simple.(s)
30x	V	3002	\bar{x}	0.44	0.3	0.28
100	A	8830	σ	0.17	0.07	0.07
100x	V	10002	\bar{x}	1.63	0.98	0.92
100	A	29990	σ	0.14	0.18	0.1
200x	V	40002	\bar{x}	8.5 ^a	5.09	4.93
200	A	119800	σ	4.78 ^a	2	2.11
200x	V	70002	\bar{x}	16.3 ^b	7.83	7.62
350	A	209950	σ	10.35 ^b	1.89	1.87

^aExcludes one run which took 671s

^bExcludes one run which took 1054s

350MHz PowerMac, 128 Mb RAM

Algorithm



Aggressive Edge Elimination

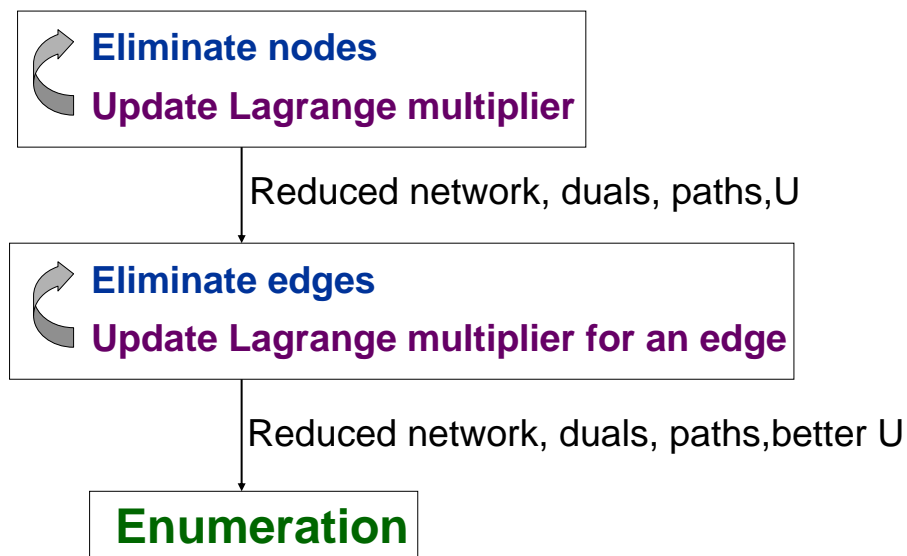
- Eliminate edges instead of nodes
- Optimize lower bound for every edge
- Update edges concurrently
- Search for upper bound at every step

Performance: Large Networks

Problem	500x500	1000x1350	
Nodes	250002	1350002	
Edges	749500	4049350	
Edges Elim	\bar{x} (s)	13.27	82.29
	σ (s)	9.96	26.7
	Timed-Out	0	0
Nodes Elim	\bar{x} (s)	8.2	54.22
	σ (s)	5.21	14.83
	Timed-Out	1	7
No Elim	\bar{x} (s)	10.74	110.77
	σ (s)	3.45	90.05
	Timed-Out	4	12

2.4GHz Pentium 4, 512 Mb RAM

Algorithm



Nodes Then Edges

- Eliminating nodes allows us to find optimal Lagrange multiplier fast
- Eliminating edges allows us to reduce intractable cases
- Enumeration effective at closing gap and proving optimality
- Combine all three together

Performance: Large Networks

Problem	500x500	1000x1350
Nodes	250002	1350002
Edges	749500	4049350
Nodes Elim \bar{x} (s)	8.2	54.22
σ (s)	5.21	14.83
Timed-Out	1	7
Edges Elim \bar{x} (s)	13.27	82.29
σ (s)	9.96	26.7
Timed-Out	0	0
Node then \bar{x} (s)	8.99	51.69
Edge Elim. σ (s)	9.45	7.48
Timed-Out	0	0

2.4GHz Pentium 4, 512 Mb RAM

Conclusions

- Preprocessing effective
- Reduces time to find optimal Lagrange multiplier
- Reduces number of intractable cases
- Finding optimal Lagrange multiplier dominates time in average case
- Enumeration dominates in intractable cases
- Node/arc equivalence classes = speed-up?

ERCSP

The Elementary Resource Constrained Shortest Path Problem (ERCSP)

- Directed graph $G=(V,A)$
- Start node s , end node t
- Costs c_a (\pm integers) for arcs $a \in A$
- Resource consumption r_a (+ integers) for $a \in A$
- Resource limit R
- The graph *may have* negative cost cycles

Find the least cost *elementary* path from s to t consuming total resource no more than R

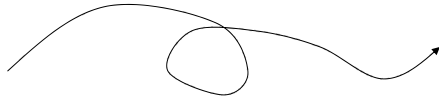
Previous approaches

- 2-cycle elimination
 - Houck, Picard, Queyranne, Vemuganti (1980)
 - Christofides, Mingozzi, Toth (1981)
- RCSP with node resources
 - Idea described by Beasley & Christofides (1989)
 - Implemented by Gueguen, Dejax, Dror, Feillet, Gendreau (2000) with strong dominance
 - State-space relaxation described by Kohl (1995)

The ERCSPP

$$\begin{array}{l}
 \min c(p) \\
 \text{s.t. } p \text{ a path in } G \text{ from } s \text{ to } t \\
 r(p) \leq R \\
 m_i(p) \leq 1 \text{ for all nodes } i \in V
 \end{array}$$

Multiplicity of node i :



$m_i(p)$ = number of times i appears in path p
 = number of arcs leaving i in p

A (state-space) relaxation

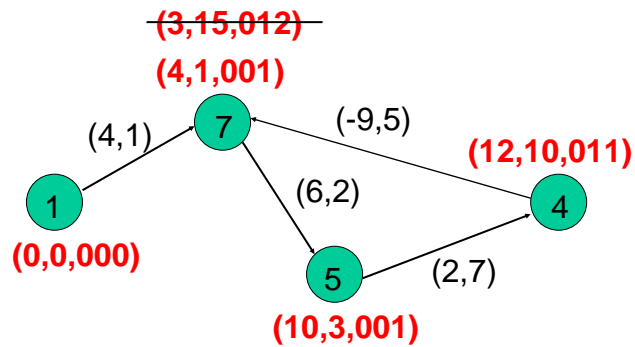
ERCSPP(S):

$$\begin{array}{l}
 \min c(p) \\
 \text{s.t. } p \text{ a path in } G \text{ from } s \text{ to } t \\
 r(p) \leq R \\
 m_i(p) \leq 1 \text{ for all nodes } i \in S
 \end{array}$$

$$S \subseteq V$$

Node resources

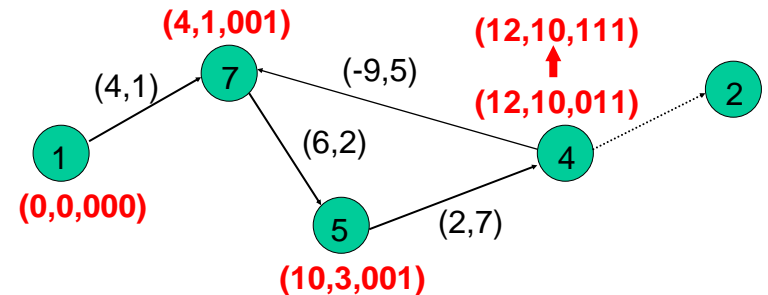
Each node in S ~ a resource with limit 1



$$S = \{2,4,7\}$$

Strong dominance

If $12 + r(\text{min resource } 4 \rightarrow 2 \text{ path}) + r(\text{min resource } 2 \rightarrow t \text{ path}) > R$



$$S = \{2,4,7\}$$

The general state-space augmenting algorithm

$S = \emptyset$

solve ERCSPP(S) using LSA

let p^* be optimal path

while (p^* not elementary) {

 augment S

 solve ERCSPP(S)

}

Augmenting S

- HMO: add some most repeated node on p^*
- HMO-All: add all most repeated nodes on p^*
- MO-All: add all repeated nodes on p^*
- M-All: add all nodes repeated on some efficient path
- All: $S = V$ initially

Memory usage

Problem Group Characteristics			# Labels	# Labels Ratio to HMO				
V	A	% neg. cost arcs	HMO	HMO-All	MO-All	M-All	All	
30	435	20	2328	2.53	2.86	4.47	(6) 82.65	
30	435	25	9879	1.85	2.53	3.51	(87) 111.50	
40	780	10	524	1.07	1.08	1.11	68.09	
40	780	15	1359	1.23	1.36	1.65	(1) 94.41	
40	1170	10	722	1.09	1.13	1.23	51.56	
30	435	30	(1) 6361	(14) 2.02	(25) 2.69	(32) 5.00	> 30 mins	
40	1560	10	7567	(2) 1.71	(2) 2.14	(2) 3.15	(99) 127.01	
50	1225	20	18417	(1) 1.86	(2) 2.35	(6) 3.89	-	
100	4950	15	28874	(2) 1.66	(2) 1.94	(10) 3.34	-	
150	16762	10	16041	1.44	(2) 1.55	(1) 2.80	-	

CPU time

Problem Group			Time (s)	Time Ratio to HMO				
V	A	% neg. cost arcs	HMO	HMO-All	MO-All	M-All	All	
30	435	20	0.07	1.26	1.52	2.01	(6) 19910.64	
30	435	25	2.11	3.55	7.10	11.44	(87) 16000.02	
40	780	10	0.01	2.38	2.08	2.29	3109.63	
40	780	15	0.03	1.45	1.48	2.06	(1) 15047.00	
40	1170	10	0.02	1.10	1.12	1.17	1451.27	
30	435	30	(1) 2.05	(14) 8.75	(25) 14.4	(32) 71.13	> 30 mins	
40	1560	10	6.39	(2) 2.94	(2) 4.82	(2) 11.9	(99) 13460.92	
50	1225	20	19.93	(1) 3.83	(2) 5.79	(6) 22.78	-	
100	4950	15	56.17	(2) 4.98	(2) 6.26	(10) 164.86	-	
150	16762	10	22.15	2.27	(2) 2.73	(1) 11.65	-	

Bounds

- Lower bounds: value of ERCSPP(S)
- Upper bounds: check all labels on node t after solving ERCSPP(S) for feasible paths

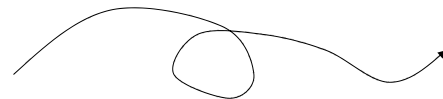
HMO: typical profile

S	Time	Max # Labels	Lower Bound	Upper Bound
0	0.72	5337	-167	-35
1	0.18	5379	-163	-35
2	0.38	8642	-131	-100
3	0.53	10869	-125	-109
4	0.92	14958	-124	-109
5	1.12	16287	-124	-109
6	1.13	16418	-124	-109
7	3.97	31117	-122	-114
8	6.95	40237	-120	-114
9	16.15	64385	-120	-120

The ERCSPP

$$\begin{array}{l}
 \min c(p) \\
 \text{s.t. } p \text{ a path in } G \text{ from } s \text{ to } t \\
 r(p) \leq R \\
 m_i(p) \leq 1 \text{ for all nodes } i \in V
 \end{array}$$

Multiplicity of node i:



$m_i(p)$ = number of times i appears in path p
 = number of arcs leaving i in p

State-space + Lagrangian relaxation

$$\begin{array}{l}
 \min c(p) + \sum_{i \in V} u_i (m_i(p) - 1) \\
 \text{s.t. } p \text{ a path in } G \text{ from } s \text{ to } t \\
 r(p) \leq R \\
 m_i(p) \leq 1 \text{ for all nodes } i \in S
 \end{array}$$

Gives a **lower bound** for $S \subseteq V$, $u \geq 0$

Can **solve** using costs $c_{ij} + u_i$ on all arcs $(i,j) \in A$

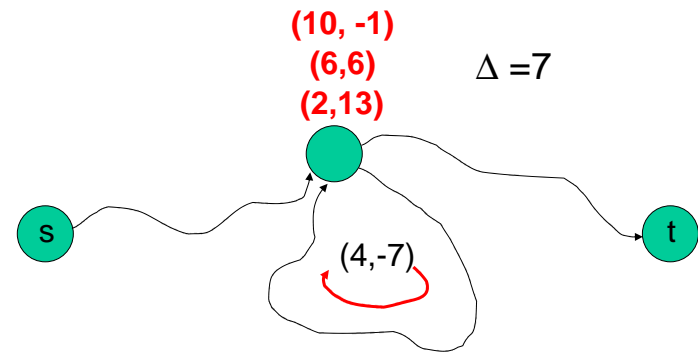
One approach

```

S = ∅, u = 0, initialize UB
solve ERCSPP(S,u), fix LB
let p* be optimal path
while (p* not elementary) {
  while (p* not elementary) {
    update u (coordinatewise)
    solve ERCSPP(S,u)
    update LB, UB
  }
  augment S, u = 0
  solve ERCSPP(S,u)
  update LB, UB
}
    
```

COULD
BE VERY
EFFICIENT

Updating u



Find smallest cost adjustment Δ so that last label on most repeated node i^* from optimal path would be dominated by some other label

Increase u_{i^*} by Δ

Which nodes?

- For HMO augmenting of S, often many “most repeated” nodes:
 - Closest to t?
 - Closest to s?
 - Lowest index?
- For increasing u, same story!

S	HMO With LR					HMO without LR				
	Time (s)	Labels	LB	UB	Htime (s)	Time (s)	Labels	LB	UB	
0	0.72	5337	-167	-35	0.72	0.72	5337	-167	-35	
	0.18	4231	-136	-100						
	0.18	4089	-136	-100						
	0.18	3892	-136	-100						
1	0.3	7371	-125	-109	0.3	0.18	5379	-163	-35	
	0.3	7378	-125	-112						
	0.3	7302	-125	-112						
2	0.53	10422	-124	-112	0.53	0.38	8642	-131	-100	
	0.53	10294	-124	-112						
	0.53	11086	-124	-112						
3	0.53	10880	-124	-112	0.53	0.53	10869	-125	-109	
	0.53	11947	-124	-112						
	0.53	10374	-123	-112						
4	1.1	16548	-123	-112	1.1	0.92	14958	-124	-109	
	1.1	15995	-123	-114						
	1.1	15752	-122	-114						
5	2	20559	-122	-114	2	1.12	16287	-124	-109	
	2	20632	-122	-114						
	2	19787	-122	-119						
6	3	27701	-122	-119	3	1.13	16418	-124	-109	
	3	26664	-122	-119						
	3	25812	-122	-119						
7	7	37190	-120	-119	7	3.97	31117	-122	-114	
	8	46906	-120	-120						
8						6.95	40237	-120	-114	
9						16.15	64385	-120	-120	
	38.64	46906			15.18	32.05	64385			

S	HMO with LR					HMO without LR			
	Time (s)	Labels	LB	UB	Htime	Time (s)	Labels	LB	UB
0	0.58	4376	-191	-40	0.58	0.58	4376	-191	-40
	0.58	5050	-191	-88					
	0.58	5537	-191	-88					
	0.58	4780	-191	-88					
1	0.37	8153	-140	-90	0.37	0.37	8153	-140	-90
	0.37	9500	-140	-90					
	0.37	8748	-140	-117					
2	0.58	11522	-140	-117	0.58	0.58	11699	-140	-90
	0.58	11010	-139	-117					
	0.58	11668	-139	-117					
3	1.05	16353	-139	-128	1.05	1.05	16978	-139	-117
	1.05	19267	-134	-128					
	1.05	19783	-134	-128					
4	2.08	29554	-134	-128	2.08	2.08	24705	-134	-117
	2.08	28606	-132	-128					
	2.08	26097	-132	-129					
5	4.63	36619	-129	-129	4.63	4.63	37031	-133	-121
6						6.32	42265	-132	-129
7						11.62	58426	-129	-129
	19.19	36619			9.29	27.23	58426		

Conclusions

- RCSP
 - Using lower bounds on path completion is very effective
 - Lagrangian lower bounds can be efficiently calculated
 - Aggressively updating upper bounds is best
- ERCSP
 - State-space relaxation is much more efficient
 - Conservative augmentation is best
 - Lagrangian relaxation looks promising

Further work

- Efficient re-optimization
- Alternative Lagrangian stopping conditions should be tested
- Alternative node selection strategies should be tested
- Alternative Lagrangian approaches should be explored
- Branch-and-bound

