

TIME DISCRETIZATION IN INTEGER PROGRAMMING

Martin Savelsbergh

Georgia Institute of Technology

Joint work with

Minh Vu Duc – University of Tours

Mike Hewitt – Loyola University

Luke Marshall – Microsoft Research

Edward He & Felipe Lagos – PhD students, Georgia Institute of Technology

Natashia Boland & George Nemhauser – Georgia Institute of Technology



It's About Time



Outline

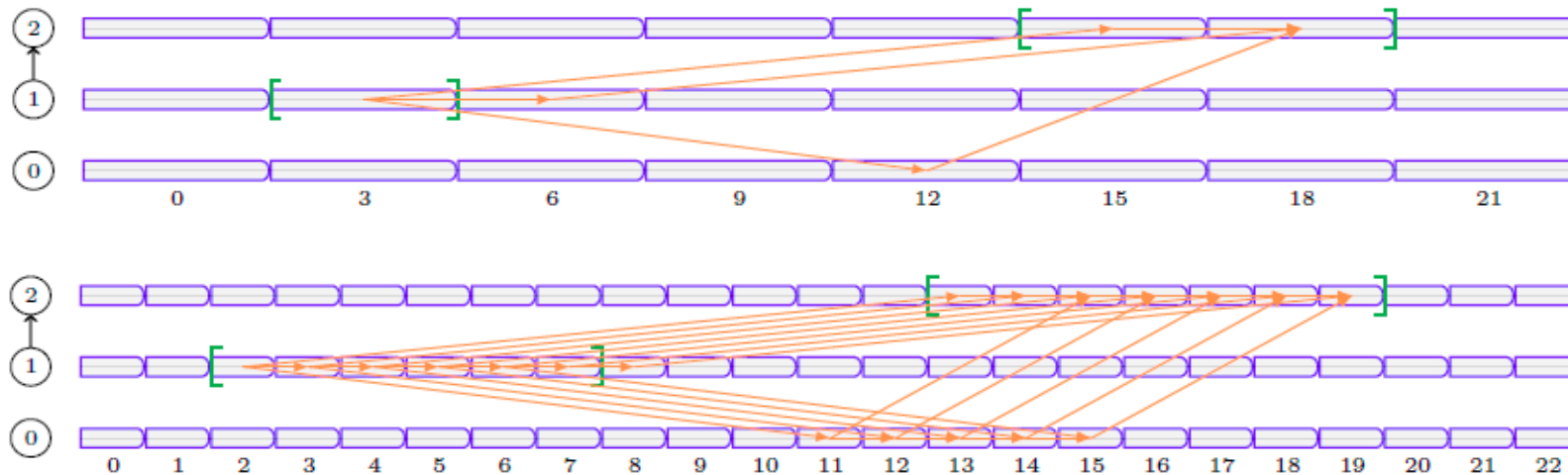
- Motivation
- Dynamic Discretization Discovery
- Dynamic Discretization Discovery for Time-dependent Shortest Paths
- Dynamic Discretization Discovery for the Time-dependent Traveling Salesman Problem with Time Windows
- Final Remarks

Motivation

- Time-dependent models
 - Occur whenever a schedule of activities needs to be constructed
 - Involve decisions about the times at which activities occur and/or resources are utilized
- Time-dependent models are pervasive in applications
 - Production planning
 - Public transit scheduling
 - Space flight logistics
 - Service network design
 - Vehicle routing
 - ...

Motivation

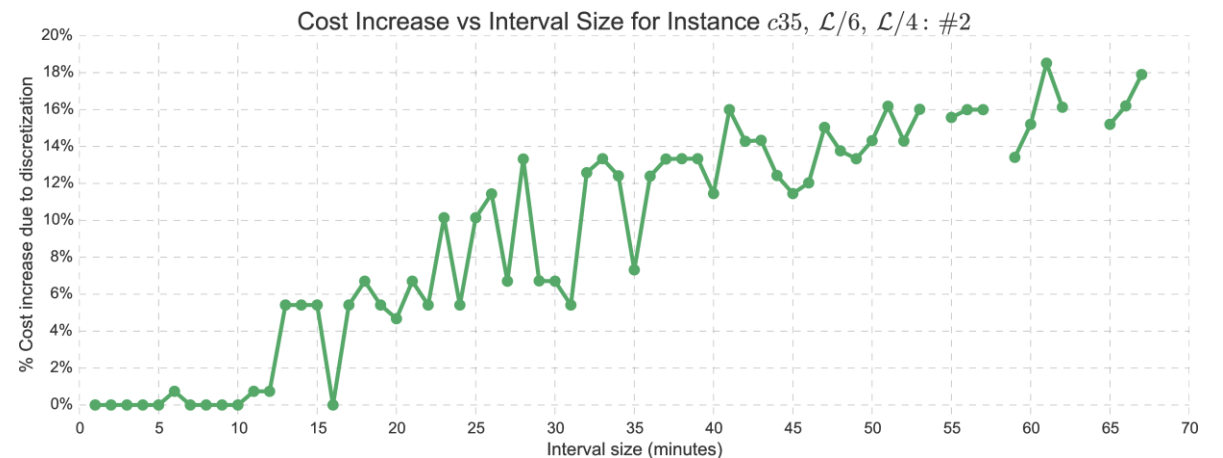
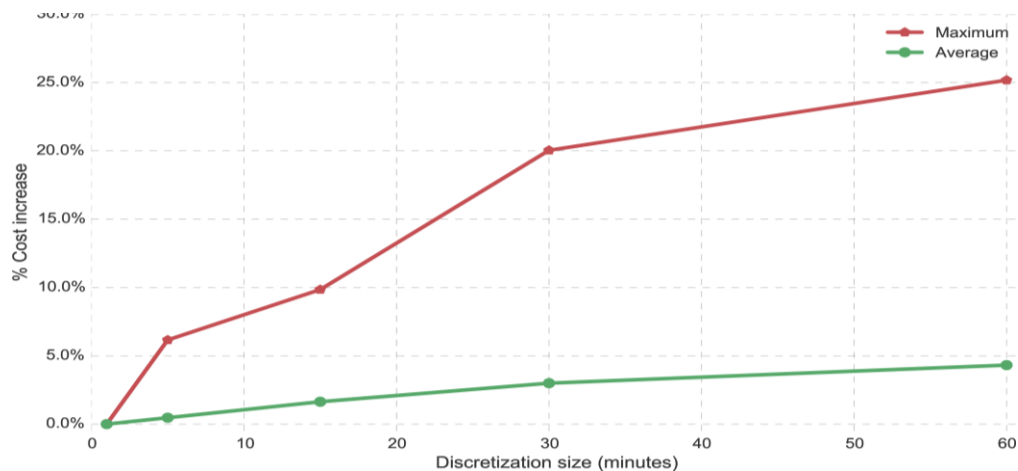
- Integer programming formulations based on time-expanded networks are often considered/used for solving time-dependent models
- Typically these use a regular discretization of time
 - What granularity to choose (minutes, hours, days, weeks, ...)?



Motivation

- Time-expanded network trade-off:
 - Coarse discretization: small network, low fidelity
 - Fine discretization: large network, high fidelity
- Observation:
 - Coarsening the discretization (to increase computational efficiency) can significantly increase the cost of a service network design

Note: Not always easy to see/argue that a time-expanded network formulation exists that will produce a continuous-time optimal solution

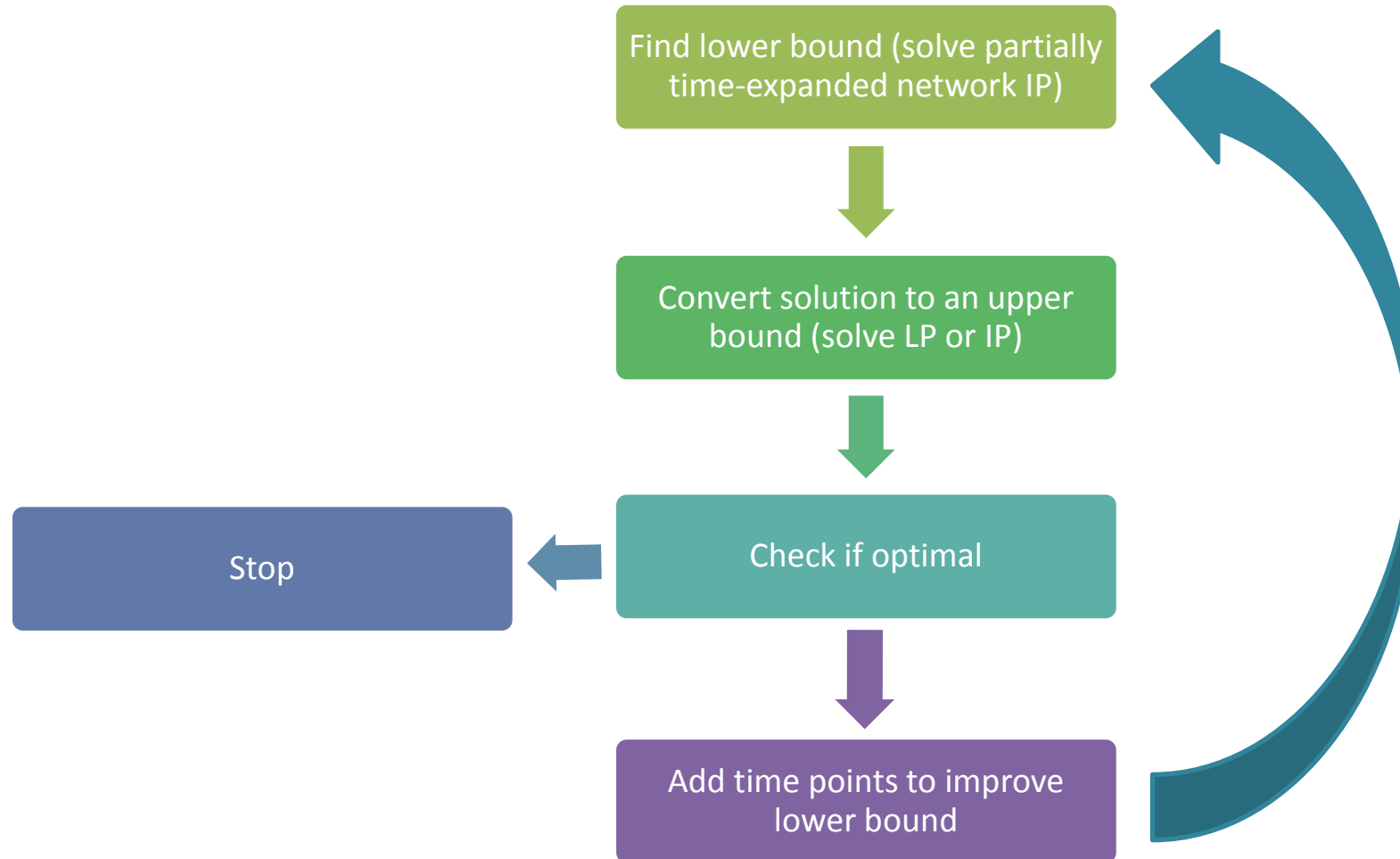


Dynamic Discretization Discovery

Solving integer programming formulations based on time-expanded networks efficiently and effectively

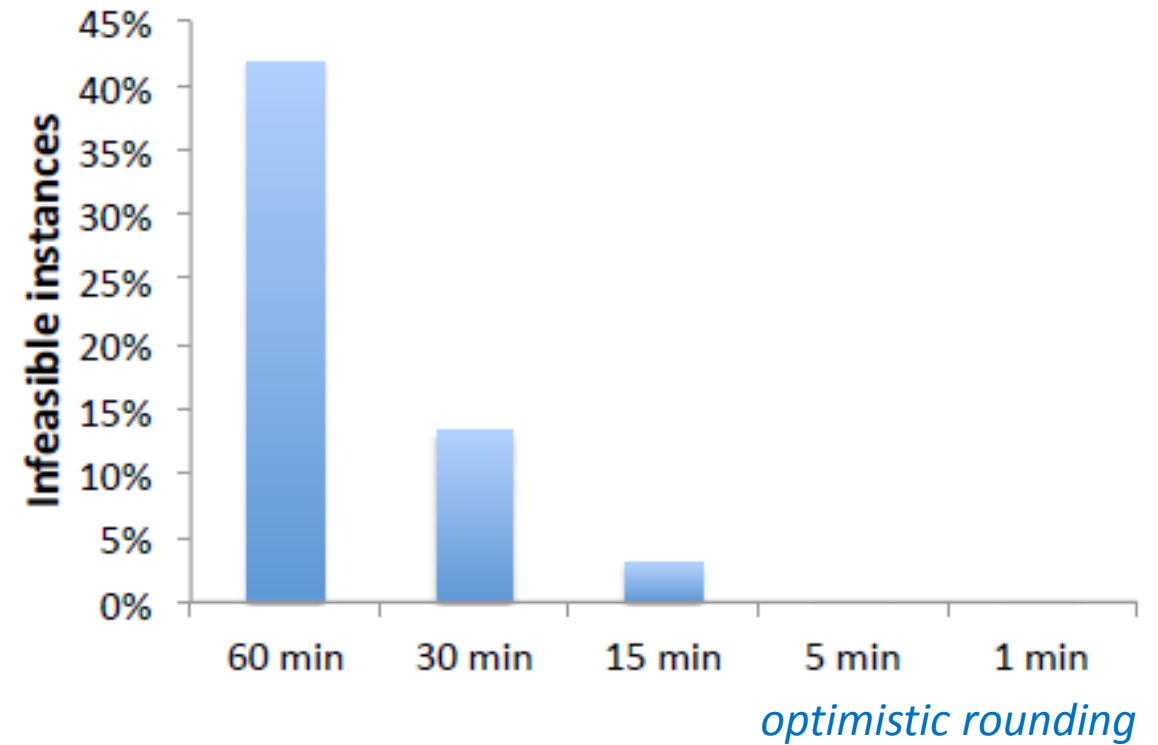
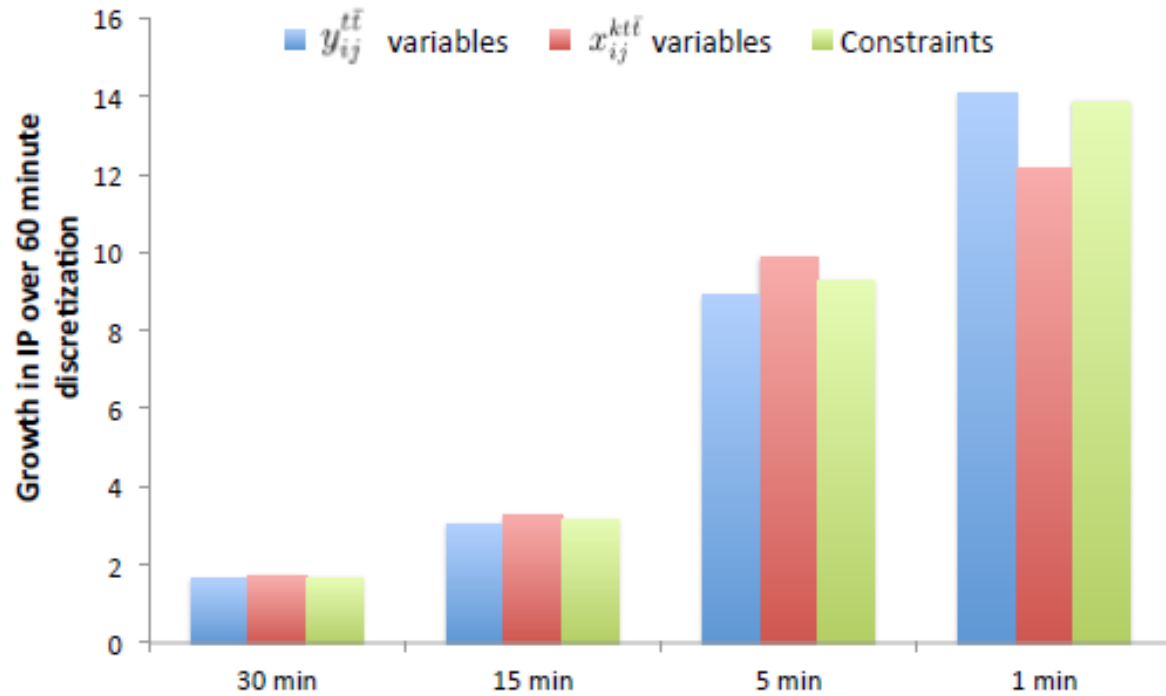
Dynamically discovering time points that are needed to find and prove (continuous-time) optimal solutions

Dynamic Discretization Discovery



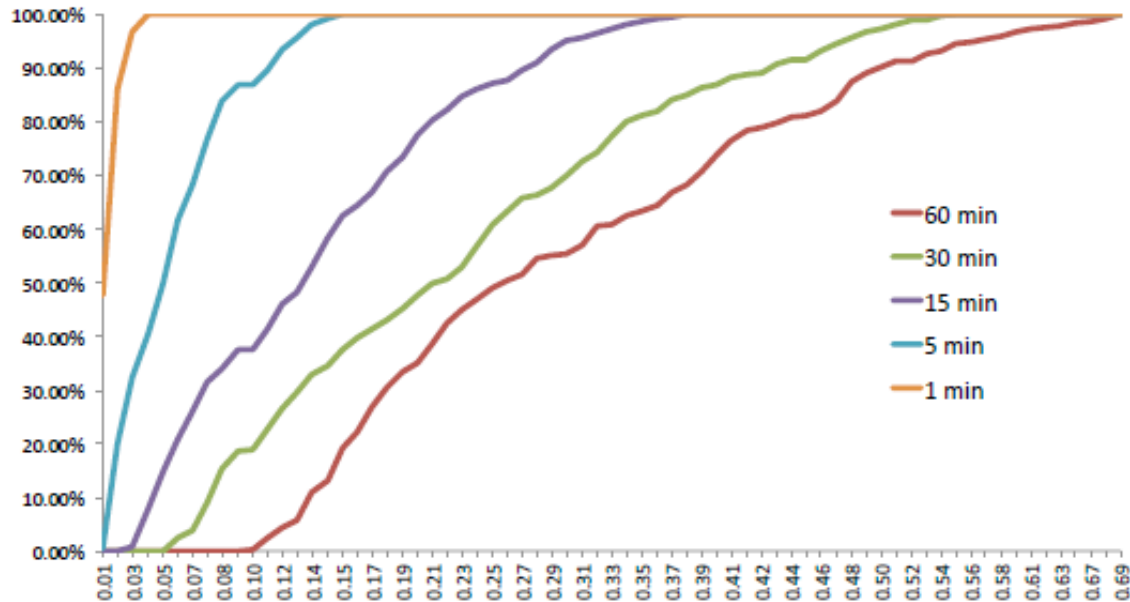
Service Network Design

Full discretization IP sizes

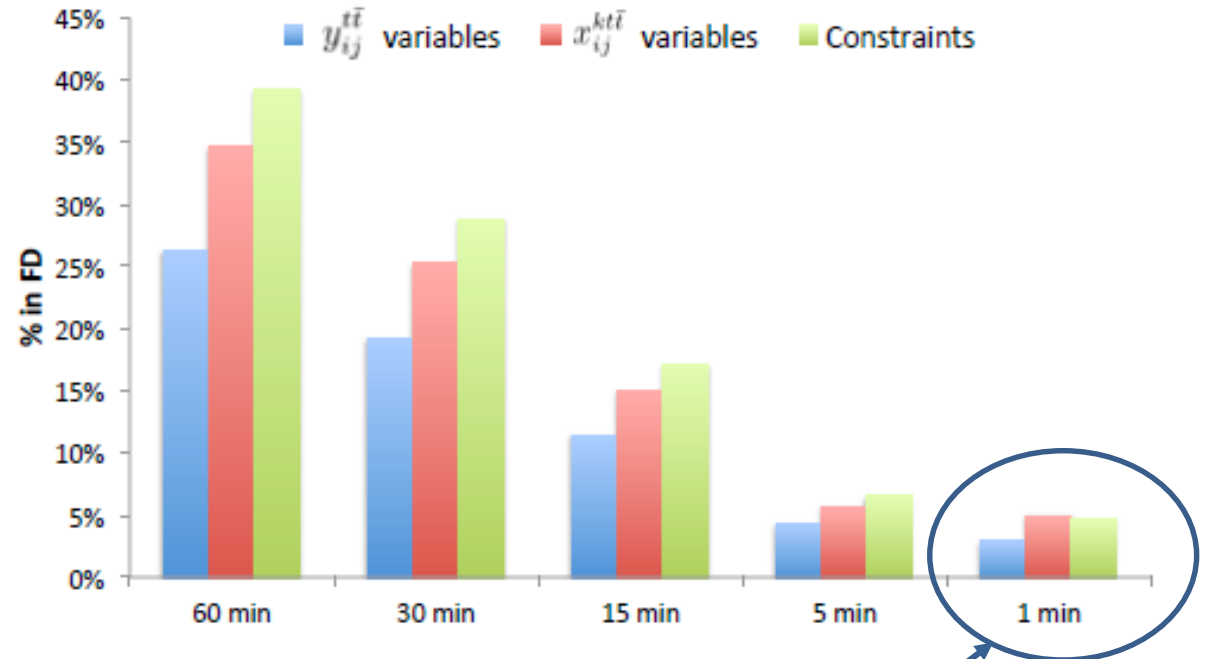


Service Network Design

DDD algorithm performance



Relative size of partial time-expanded network in final iteration

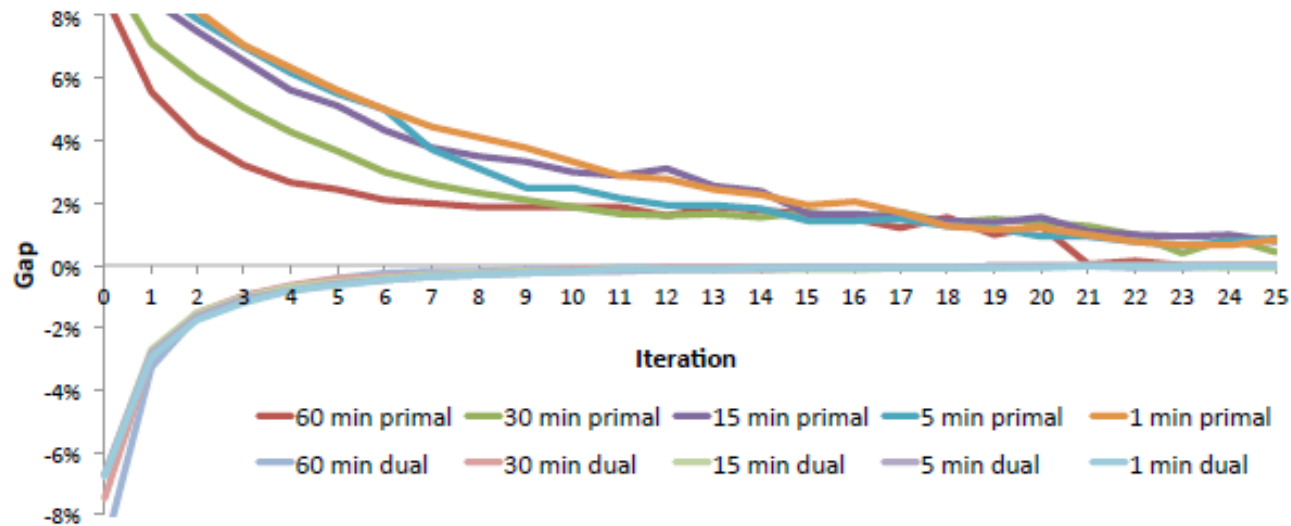


Relative size of integer program in final iteration

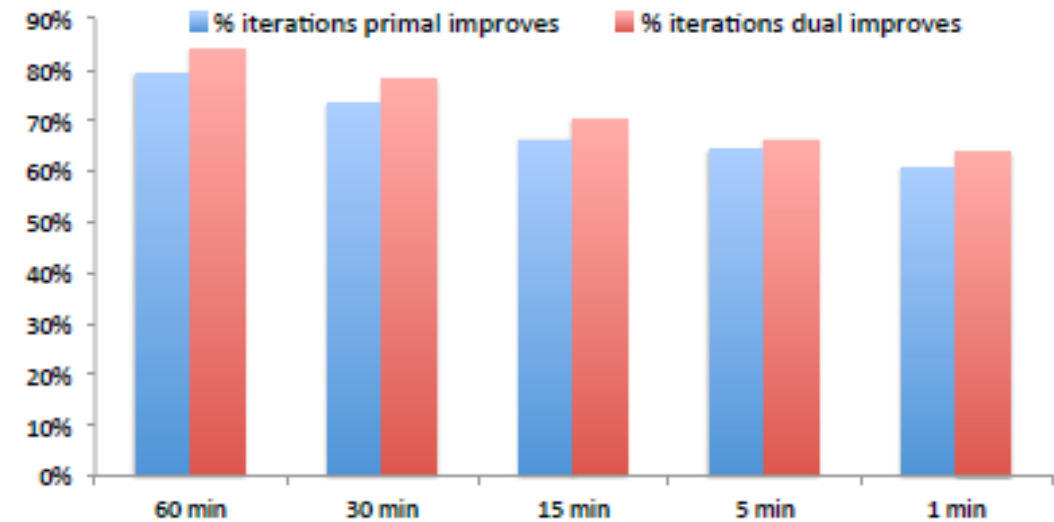
- CTSNDP-Solve maintains a much smaller network than that of the full discretization
- **Final network is never more than 4% of the full size for the 1-minute case**

Service Network Design

DDD algorithm performance



Primal and dual gaps by iteration



Frequency of improvement

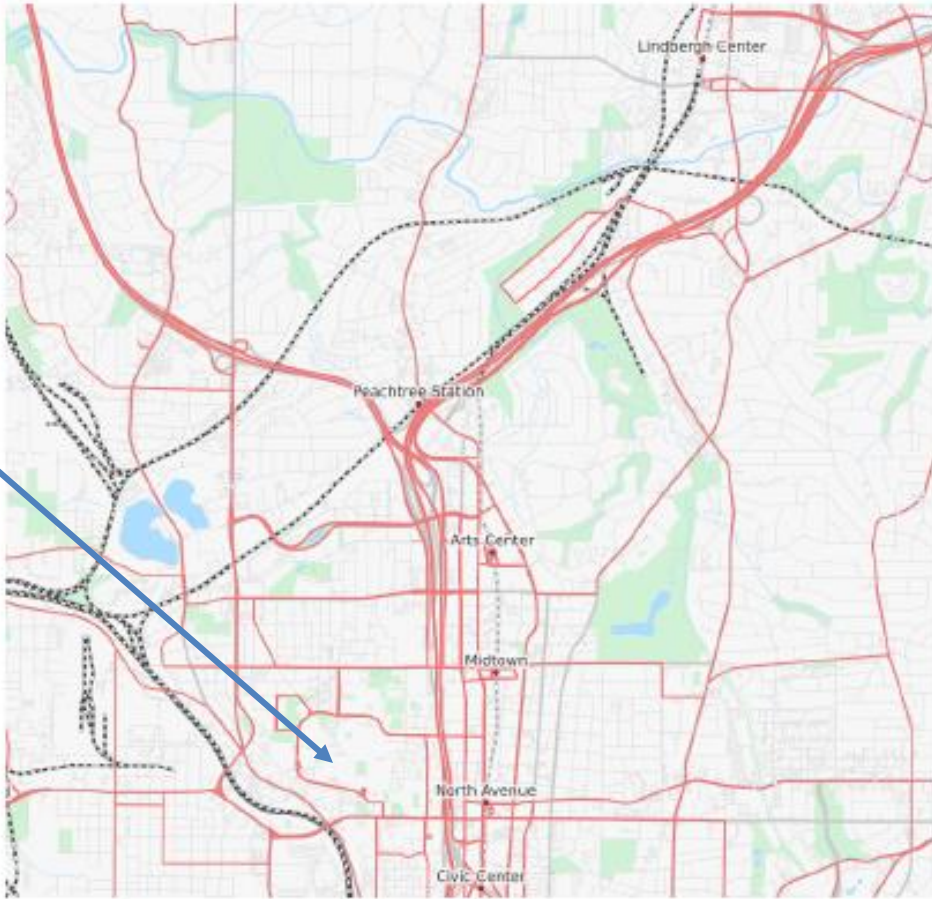
Dynamic Discretization Discovery – Key Ideas

- Focus on **partially** time-expanded networks
- Define an optimization problem over the partially time-expanded network that provides a dual bound
- Define an optimization problem that attempts to convert a solution to the optimization problem over the partially time-expanded network into a continuous-time feasible solution
- Iteratively refine the partially time-expanded network until a continuous-time feasible solution of desired quality has been obtained

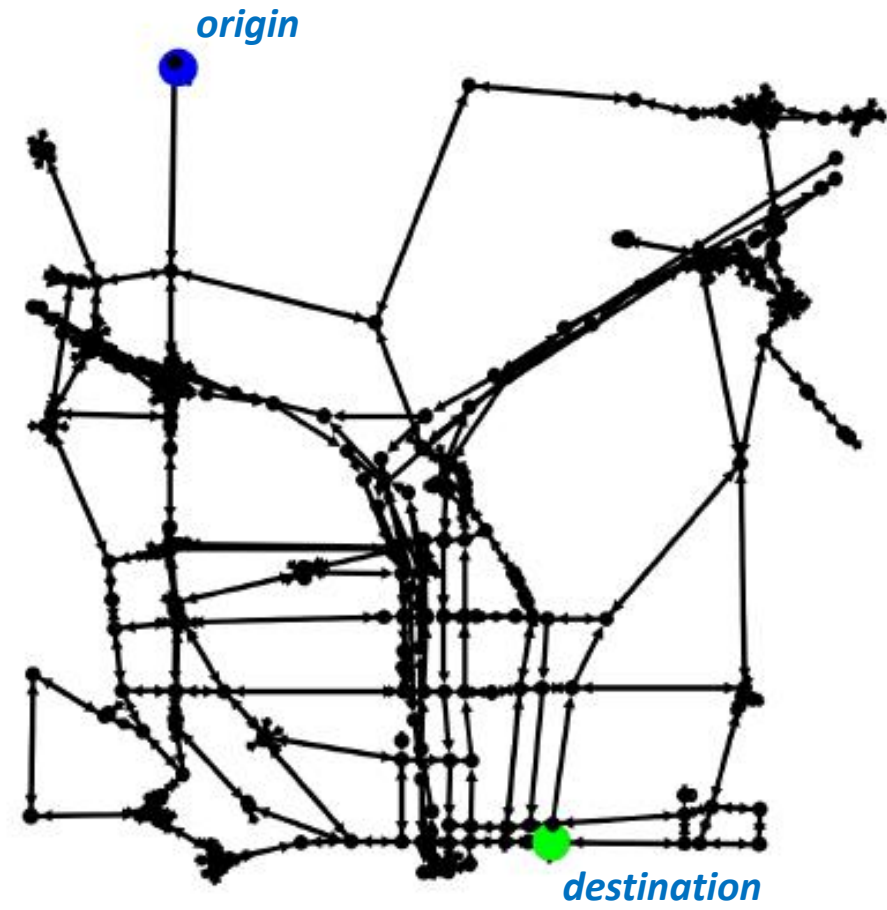
Time-Dependent Shortest Path Problems

Motivation: Time-Dependent Travel in Atlanta

Georgia
Tech



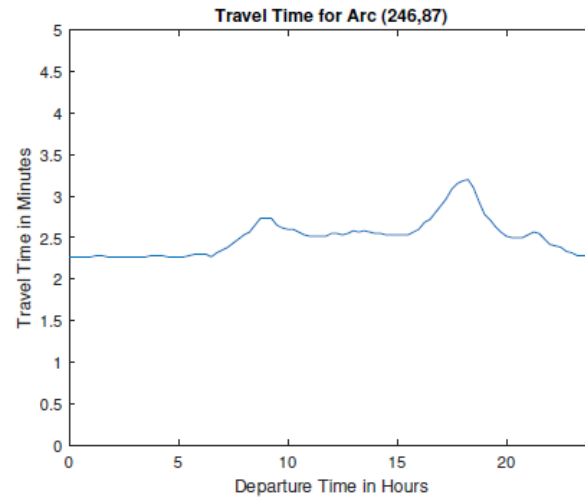
(a) Map of section of Midtown and North Atlanta



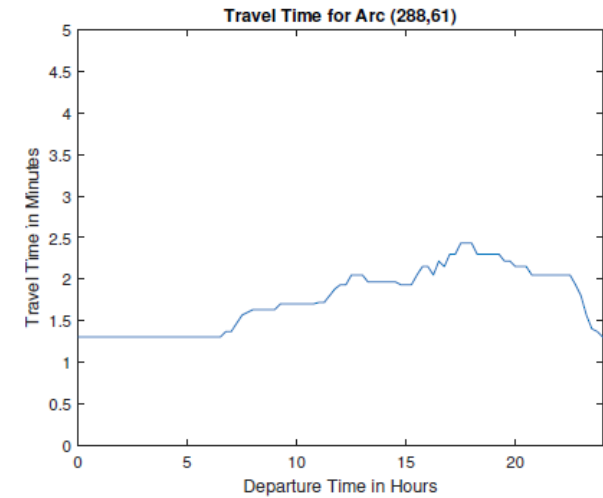
(b) Network representation

Travel Time Functions

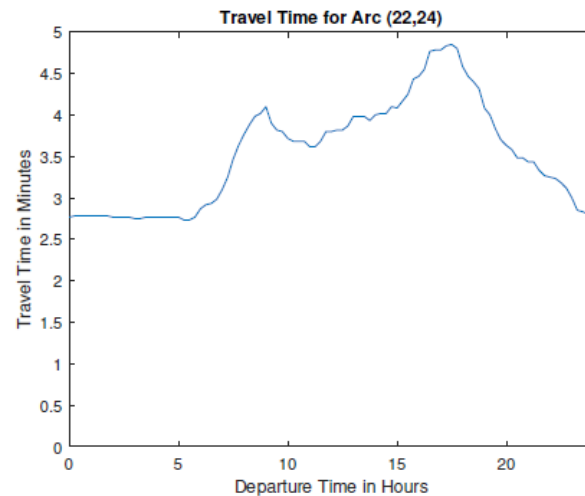
Travel time functions obtained from GPS traces from different sources (car navigation systems & cell phones)



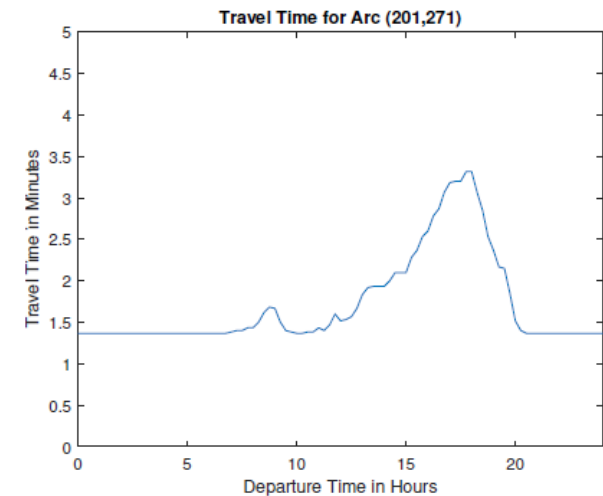
(a) Travel time of an arc along Peachtree St.



(b) Travel time of an arc along Ponce de Leon Ave. NE



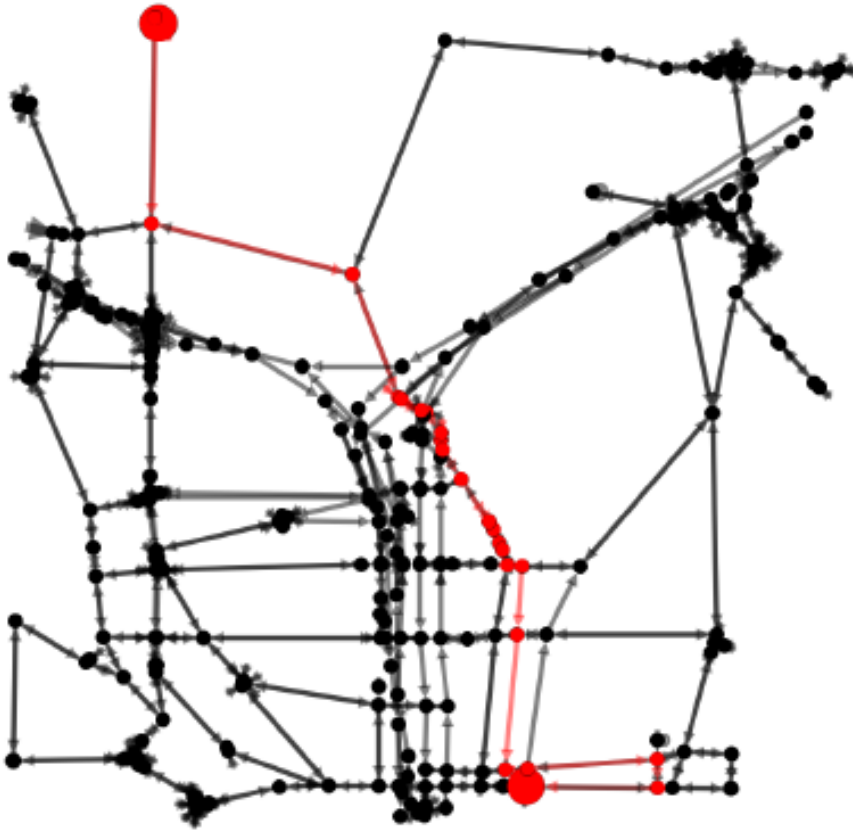
(c) Travel time of an arc along North Ave. NE



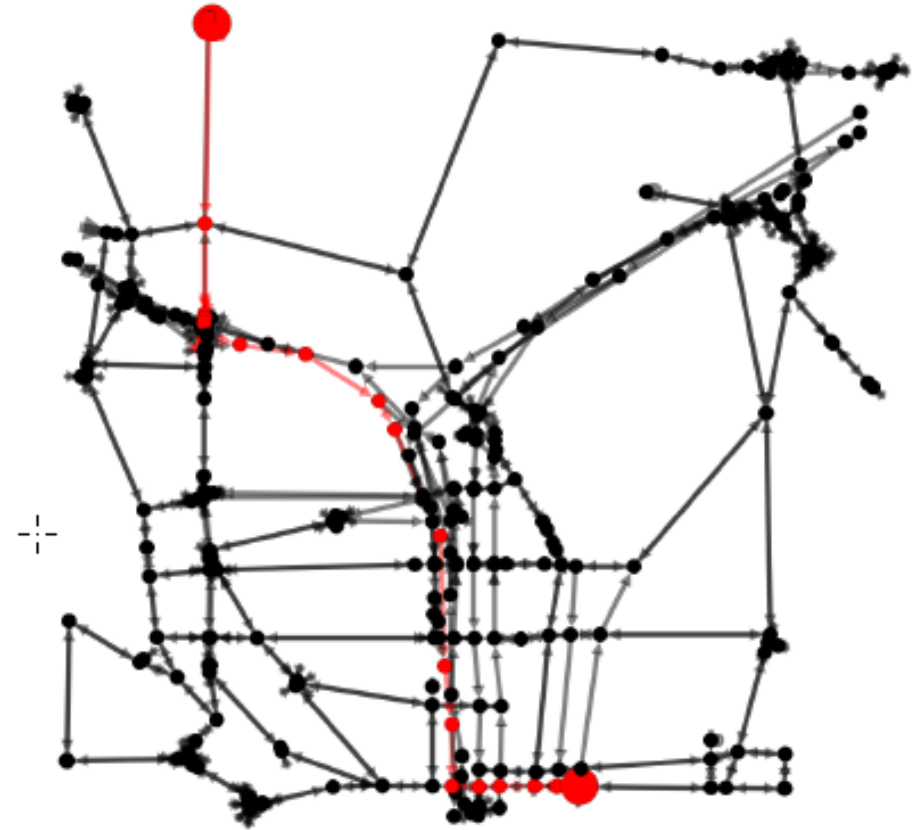
(d) Travel time of an arc along Midtown section of I-

Minimum Paths

Earliest arrival path, latest departure path, and minimum duration path in the period 9:00 – 13:00



(a) Earliest arrival / latest departure path



(b) Minimum duration path

Earliest arrival path: 47.5 min Latest departure path: 47.8 min Minimum duration path: 45.8 min (dep: 10:37)

Time-Dependent Shortest Paths

- Given
 - directed graph $D = (V, A)$
 - start node 1, end node n ,
 - a time horizon $[0, T]$, and
 - time-dependent travel time $c_{i,j}(t)$ for each arc $(i, j) \in A$ and time t , giving the time to traverse the arc if starting at time t , which
 - satisfy the FIFO property **FIFO = no overtaking; start earlier \rightarrow arrive earlier**
- Find
 - start time $t_1^* \geq 0$ and
 - a time-dependent path $P(t_1^*)$ that
 - arrives at node n by time T

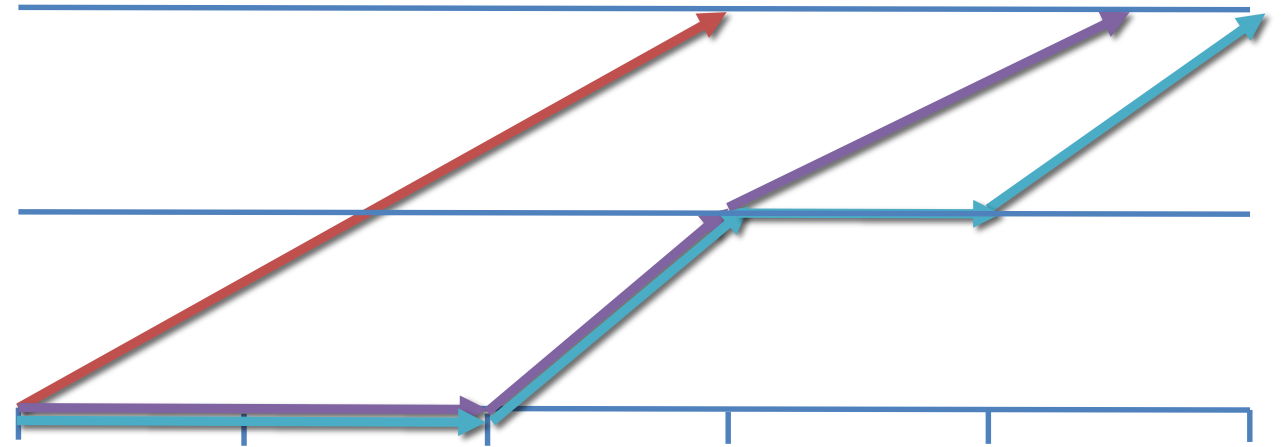
Objectives:




- Minimize arrival time
- Minimize duration
- Minimize travel time

Time-Dependent Shortest Paths

Arcs	Arc Travel Times					
	0	1	2	3	4	5
(1,2)	(3)	2	(1)	2	3	(4)
(1,3)	(3)	3	3	3	3	3
(2,3)	(3)	2.5	2	1.5	(1)	(1.5)

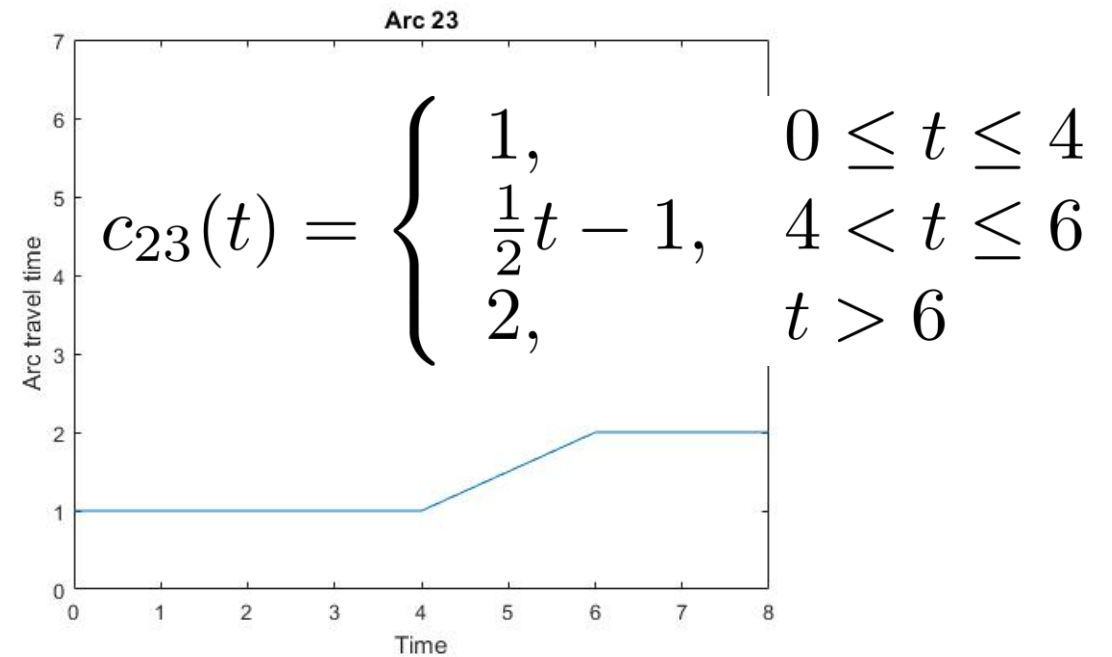
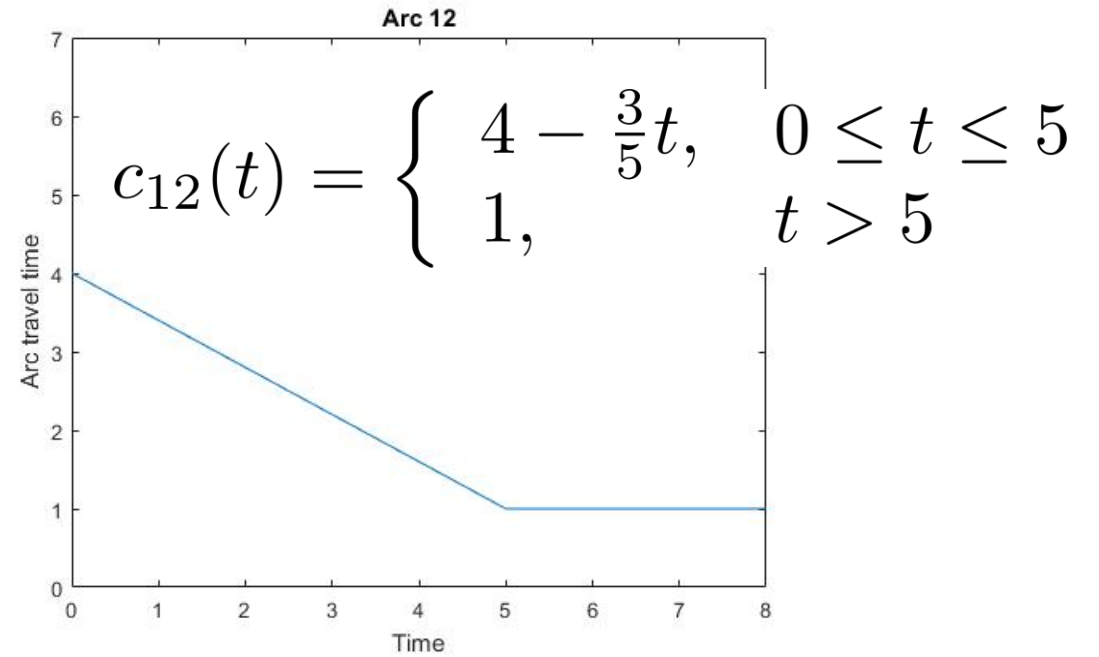
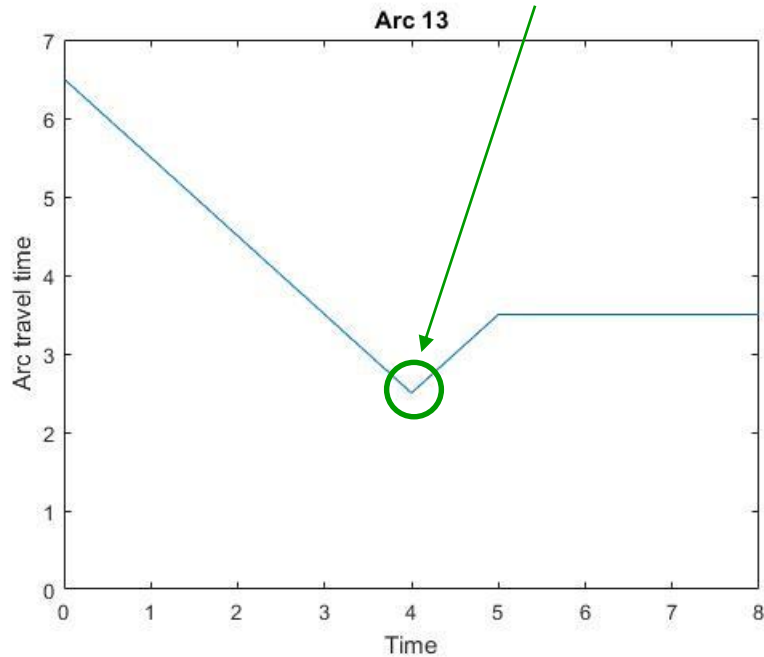
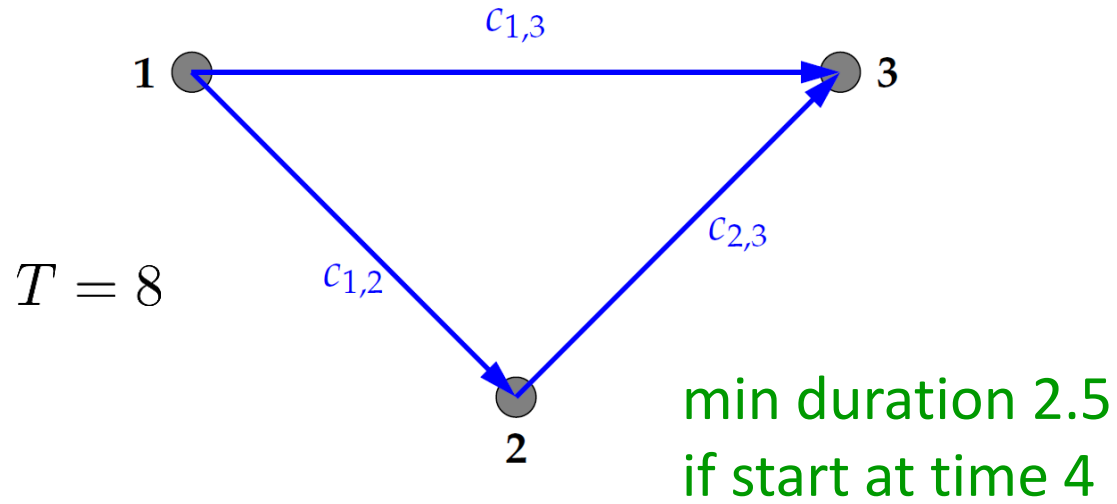
Table: Arc travel times $c_{i,j}(t)$,
columns are t , rows are (i,j)



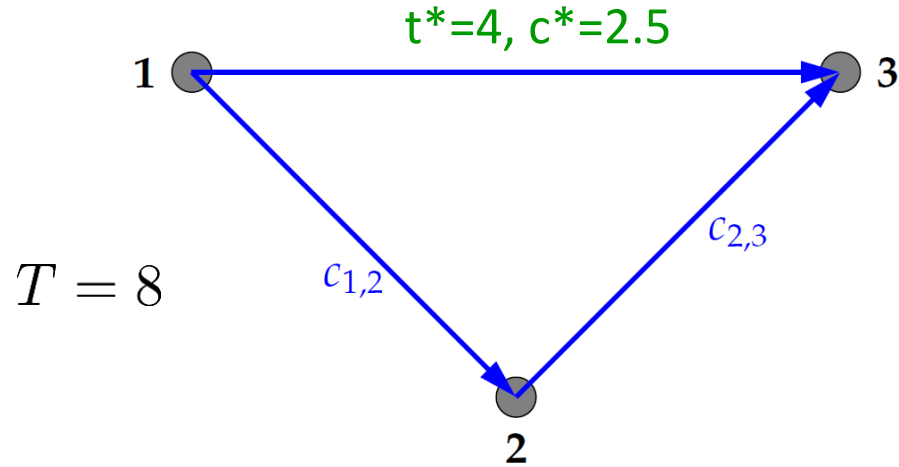
- Minimum Arrival Time Path 
- Minimum Duration Path 
- Minimum Travel Time Path 

Minimum Duration Time-Dependent Shortest Path Problem

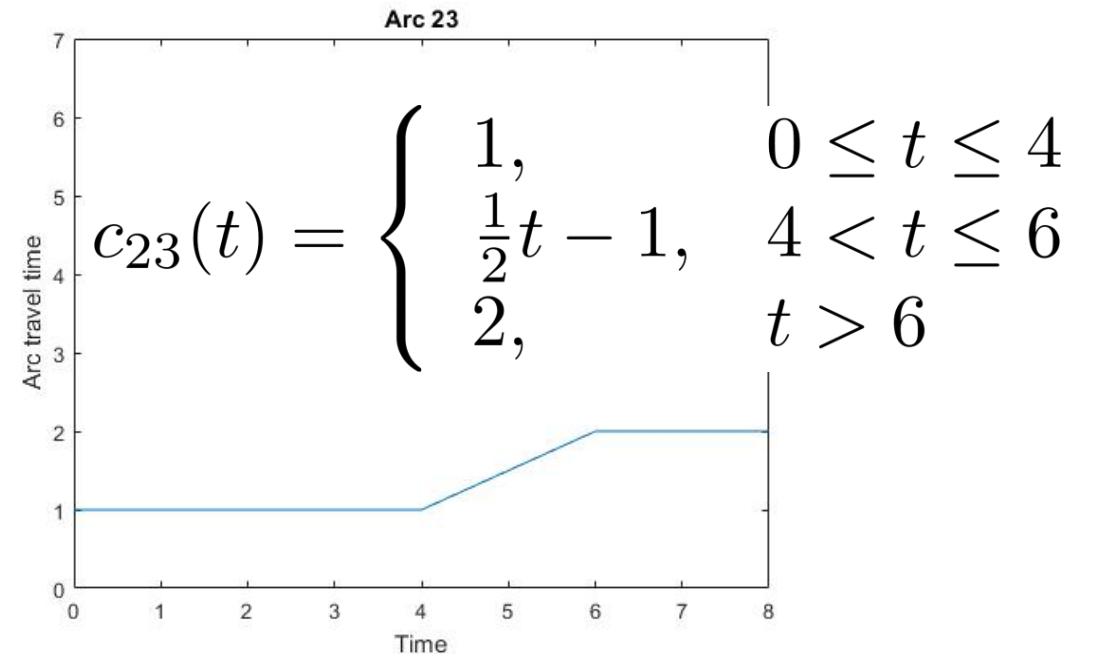
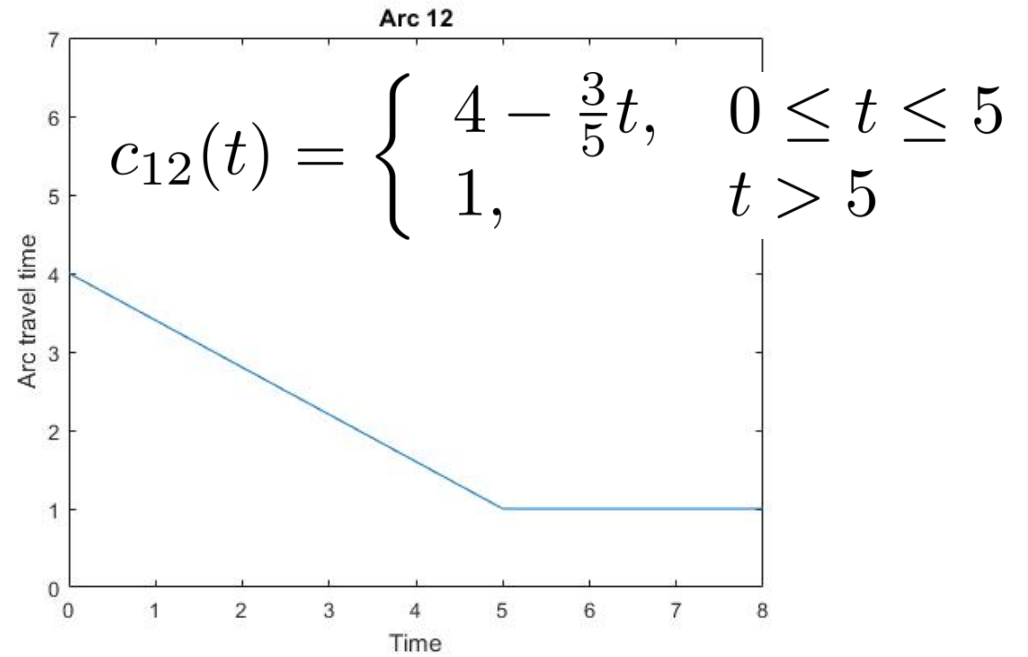
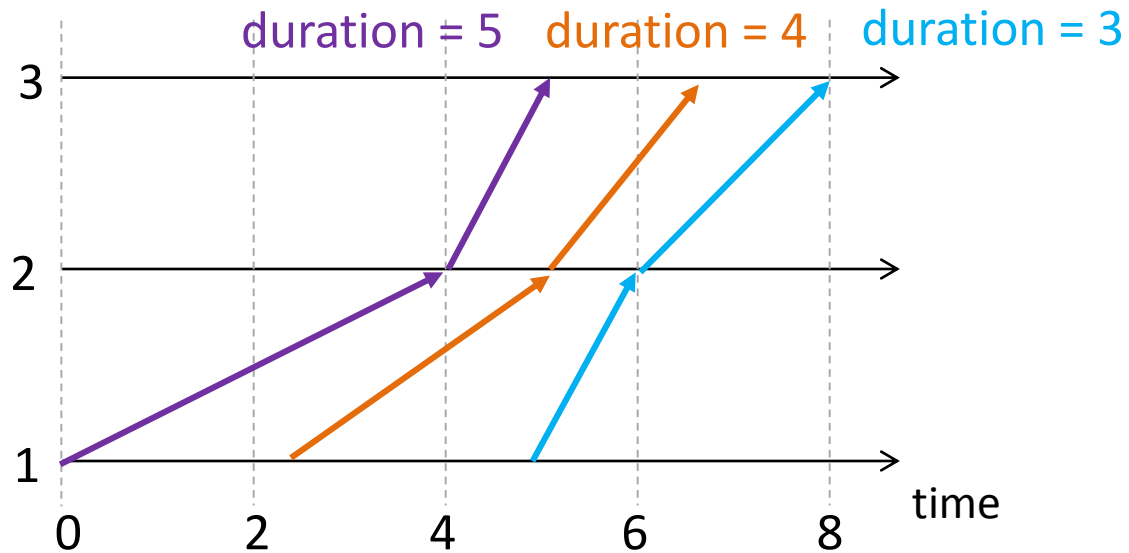
Example



Example



Path (1,2,3)



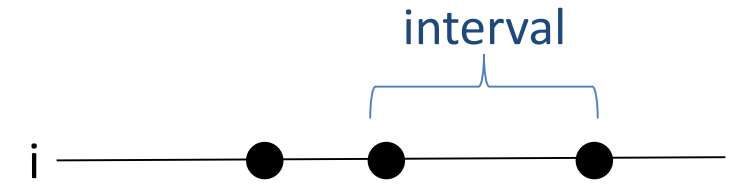
Existing Approaches

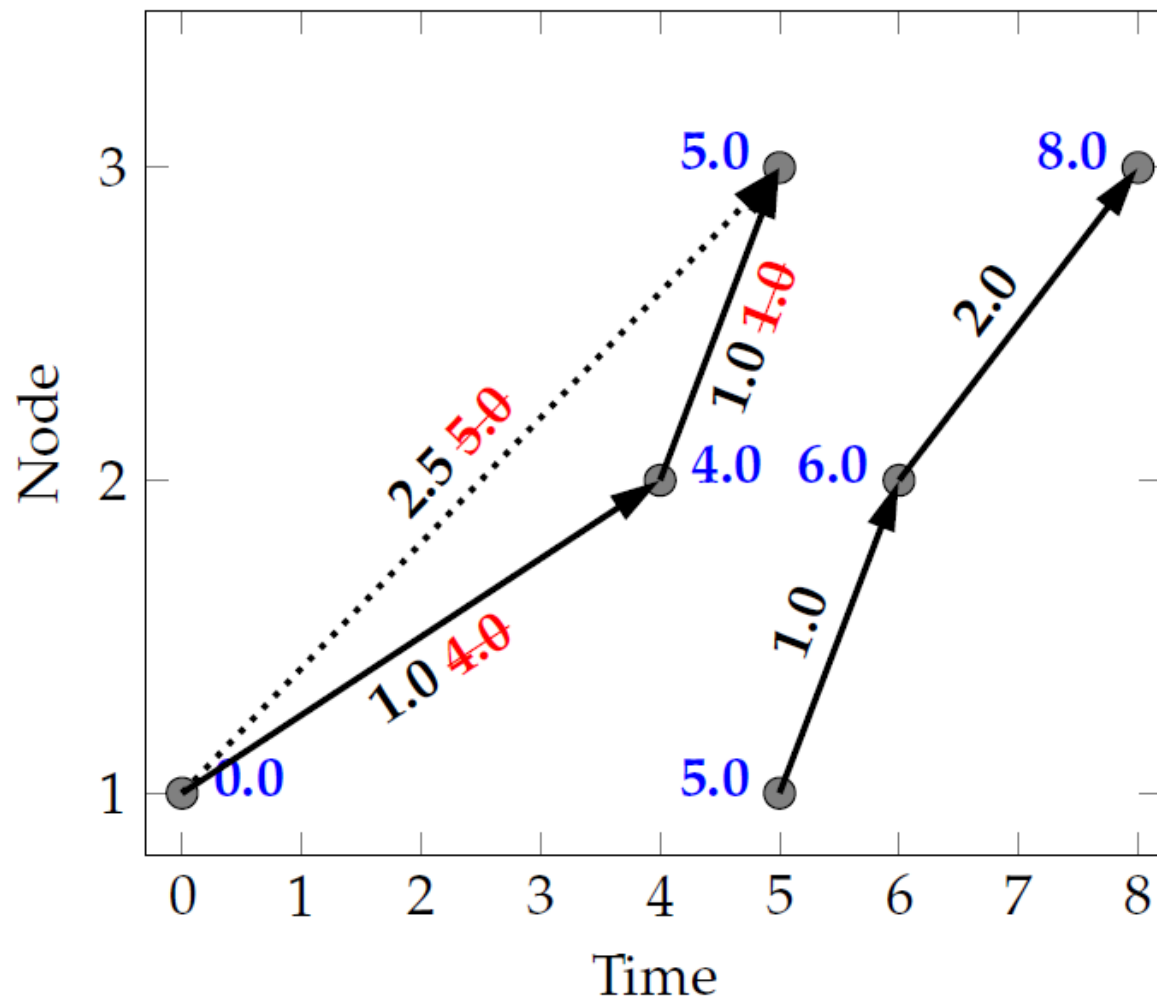
- Given a **fixed** start time, the minimum arrival time path can be found by a time-dependent (TD) version of Dijkstra's algorithm
- Orda and Rom (1990) first described the minimum duration path problem
- **Discrete algorithms**
 - discretize the possible start times, and apply TD Dijkstra's to each start time
 - quality of solution depends on quality of discretization
- **Continuous algorithms**
 - use variants of Dijkstra's (Nachtigall 1995) or A* algorithm (Kanoulas 2006)
 - solution is exact, however, requires repeated functional operations
- Foschini and Suri (2014) observe that for **piecewise linear** continuous travel time functions
 - there always exists an optimal path that contains a departure at a breakpoint, and
 - gave an exact algorithm that considers all start times associated with such breakpoints

Concept for a DDD Approach

- Foschini and Suri (2014) investigates **all** the breakpoints.
- Is there a way to dynamically decide which breakpoints to investigate?
- Dynamic discretization discovery (DDD) idea:

- Each node has a time discretization
- The time between consecutive time points is an interval
- Time points are created using **Arc-Completed Backward Shortest Path Trees** (ABSPTs)
- Given a time, t , at the end node
 - Find the latest time that a path could depart at a node to reach the end node at time t (using backward TD Dijkstra); the travel time associated with the path from the start node gives an upper bound
 - For each node create a time point at that time (at that node)
 - Create a timed copy of each arc, using the time points created (may be “too short”)
- Give each arc a **cost**: the **minimum** travel time for that arc over the subsequent interval
- The least cost path in any ABSPT must be a lower bound on the minimum duration

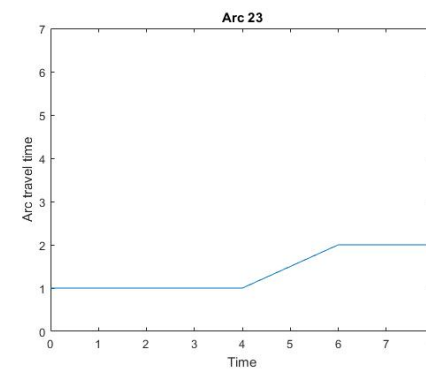
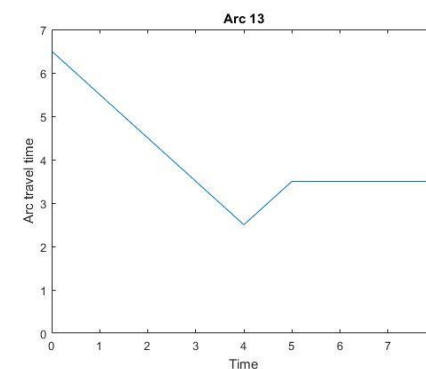
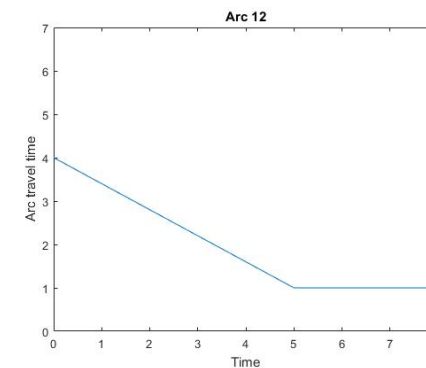




UB = 3.0

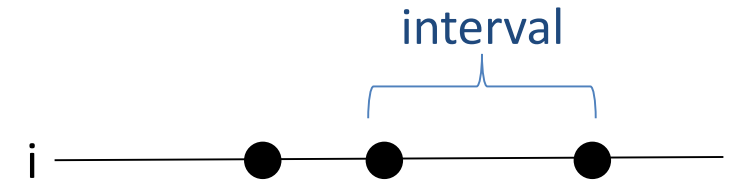
LB = 2.0

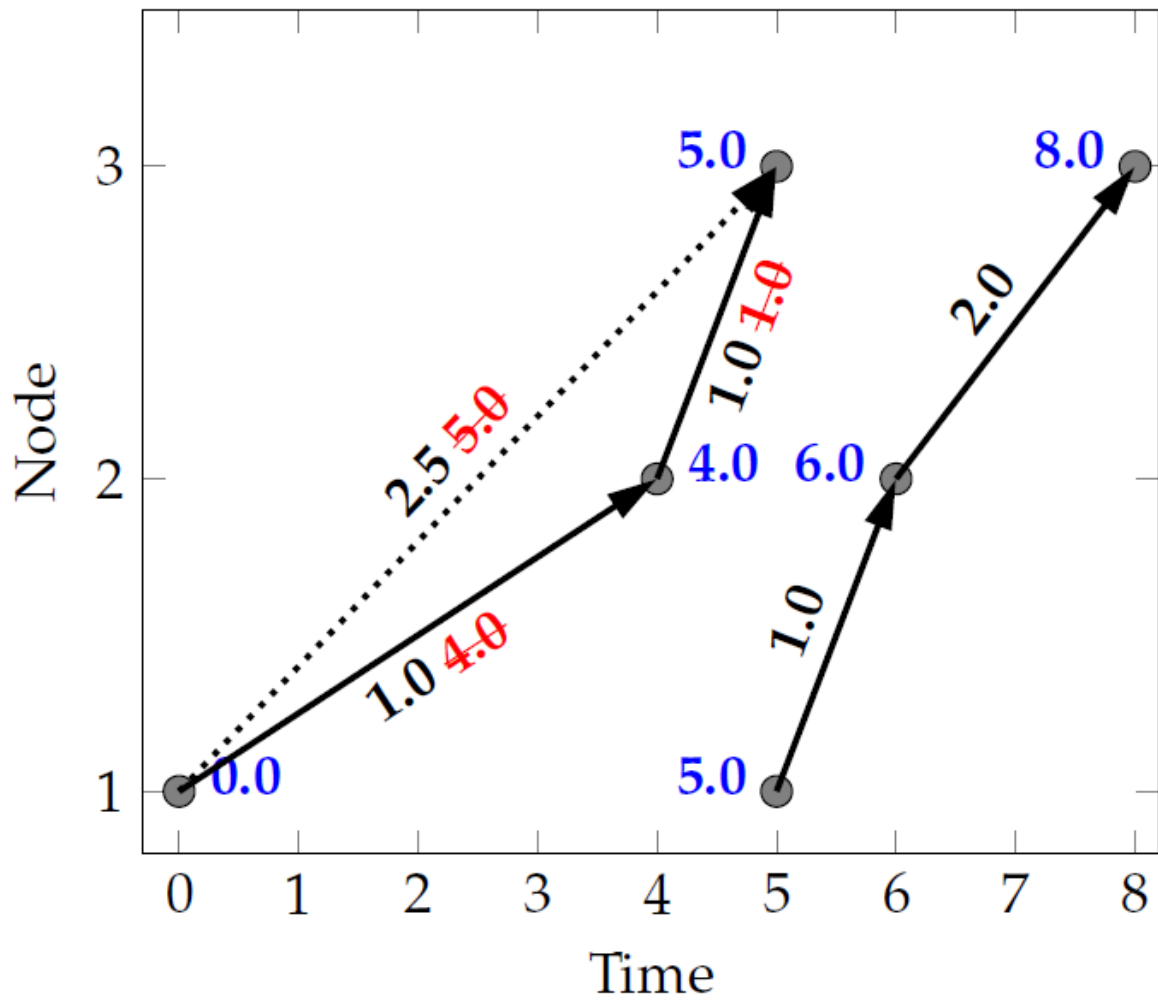
$T = 8$



Concept for a DDD Approach

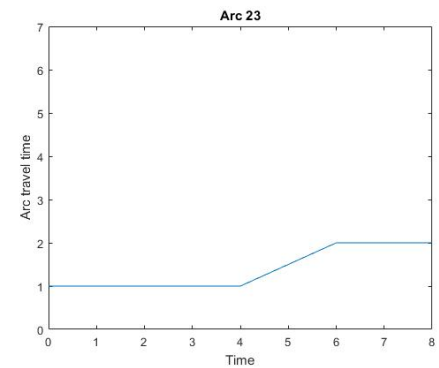
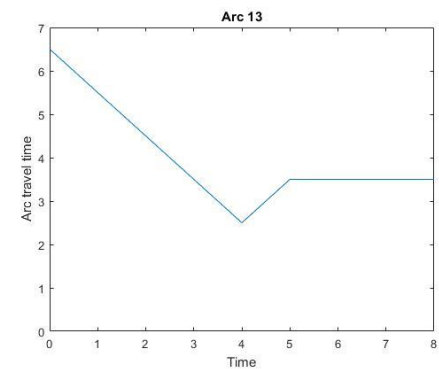
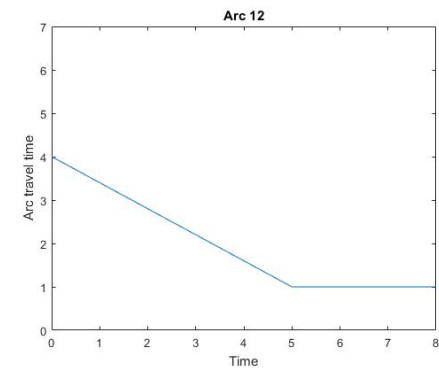
- Dynamic discretization discovery (DDD) idea:
 - Each node has a time discretization
 - The time between consecutive time points is an interval
 - Time points are created using **Arc-Completed Backward Shortest Path Trees** (ABSPTs)
 - Given a time, t , at the end node
 - Find the backward shortest path tree
 - Create a copy of each arc, using the time points created (some may be “too short”)
 - Give each arc a **cost**: the **minimum** travel time for that arc over the subsequent interval
 - The least cost path in any ABSPT must be a lower bound on the minimum duration
 - *If there is a gap between lower and upper bound, look for an arc in the ABSPT that gave the lower bound whose next breakpoint is strictly within the next interval*
 - *Find the path from that breakpoint arc’s origin node at the breakpoint time to the end*
 - *This gives a time on the end node: construct the ABSPT*
 - *Update the costs on arcs in the preceding ABSPT*

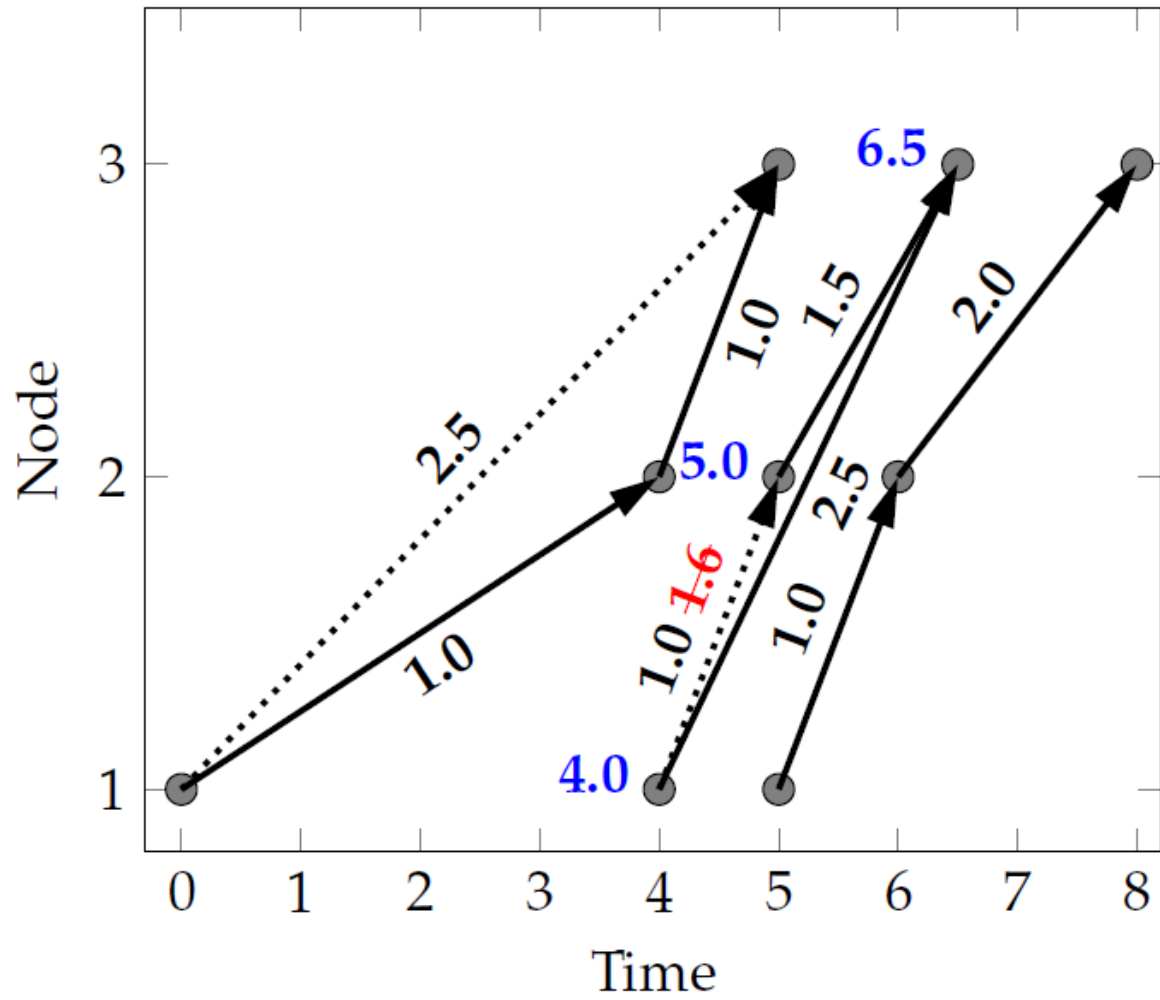




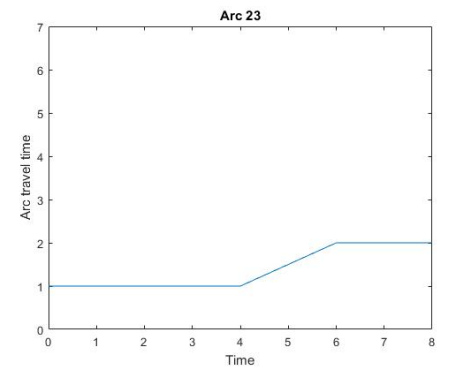
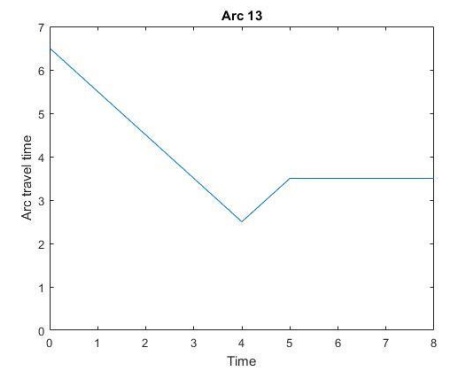
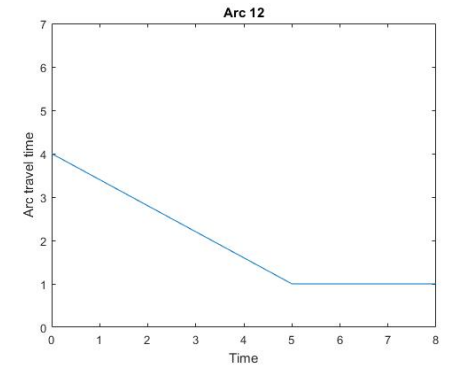
UB = 3.0

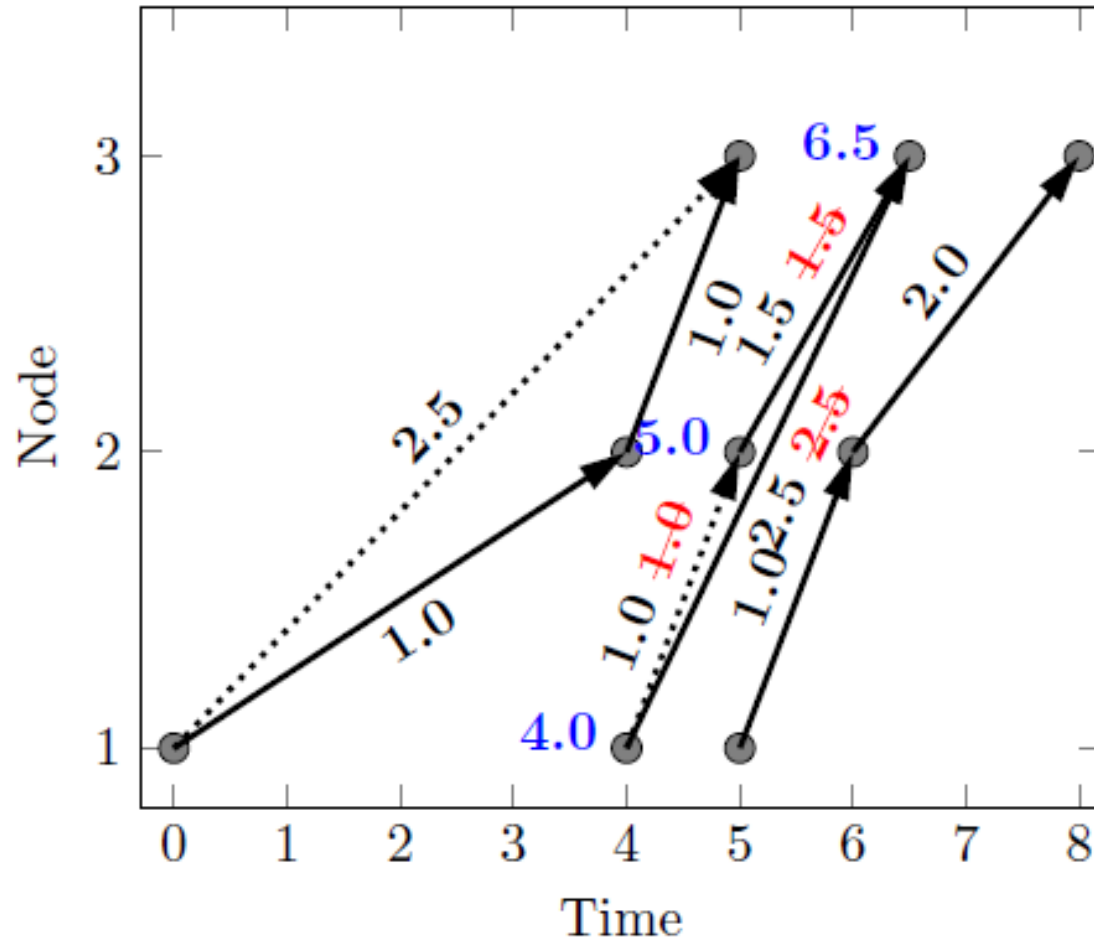
LB = 2.0



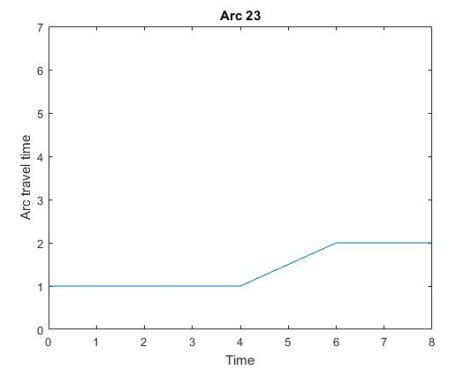
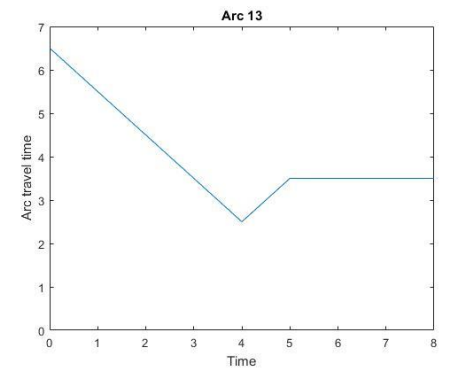
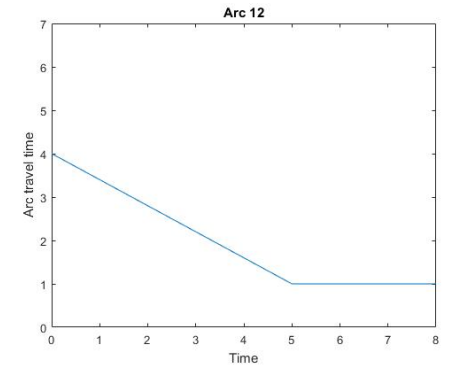


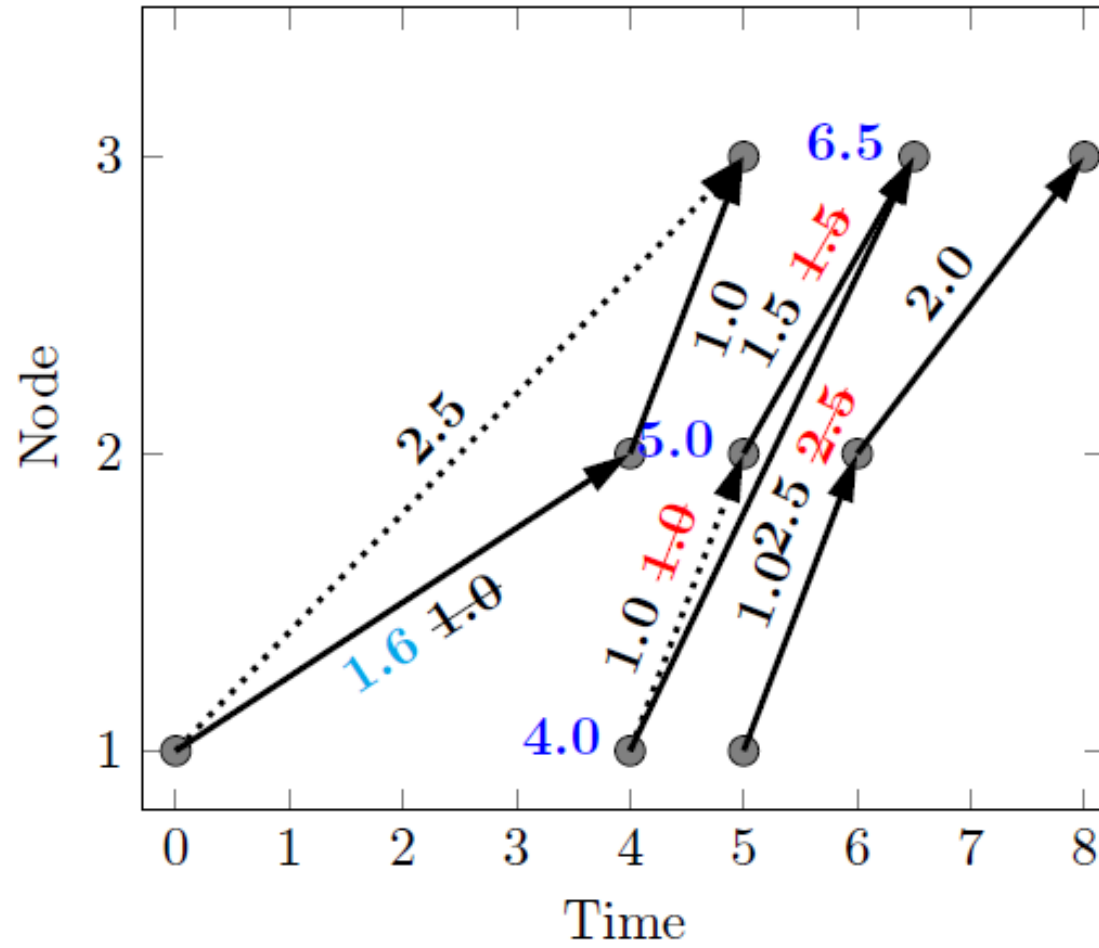
UB = 2.5





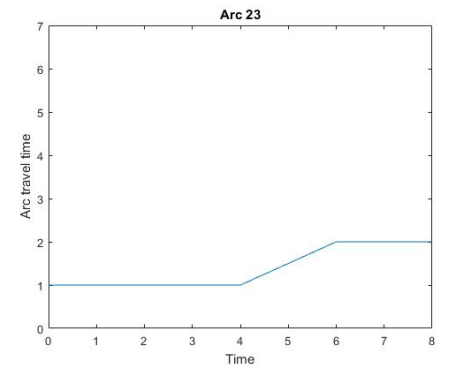
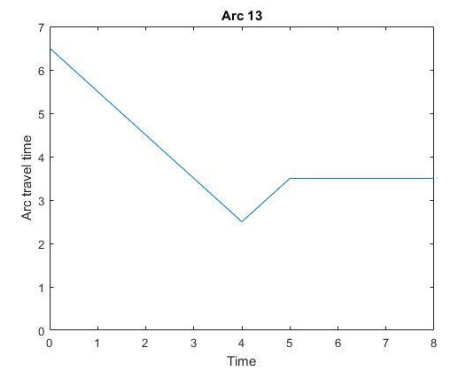
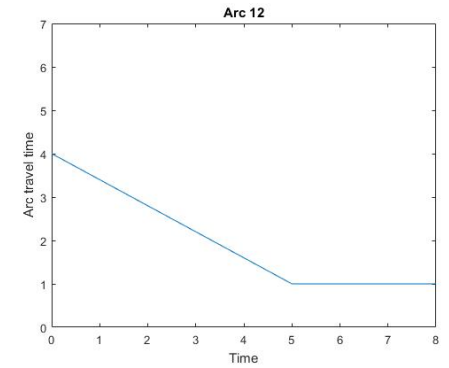
UB = 2.5





UB = 2.5

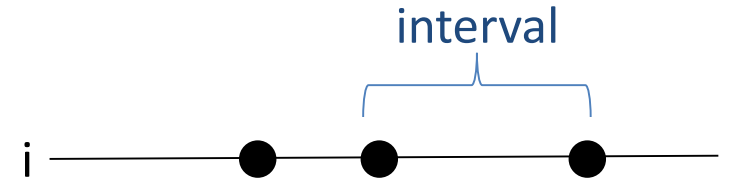
LB = 2.5



Concept for a DDD Approach

- Dynamic discretization discovery (DDD) idea:

- Each node has a time discretization
- The time between consecutive time points is an interval
- Time points are created using **Arc-Completed Backward Shortest Path Trees** (ABSPTs)
- Given a time, t , at the end node
 - Find the backward shortest path tree
 - Create a copy of each arc, using the time points created (some may be “too short”)
- Give each arc a **cost**: the **minimum** travel time for that arc over the subsequent interval
- The least cost path in any ABSPT must be a lower bound on the minimum duration
- If there is a gap between lower and upper bound, look for an arc in the ABSPT that gave the lower bound whose next breakpoint is strictly within the next interval
 - Find the path from that breakpoint arc’s origin node at the breakpoint time to the end
 - This gives a time on the end node: construct the ABSPT
 - Update the costs on arcs in the preceding ABSPT
- *If there is no such breakpoint, then the ABSPT can safely be deleted but its time points kept*



The DDD Algorithm

- The problem of finding the least cost path decouples by ABSPTs; these do not interact with each other except for providing time points for arc cost calculations
- This allows UB and LB for each ABSPT to be computed independently (cheaply)

Algorithm 1: Dynamic Discretization Discovery (DDD) Algorithm.

input : $G = (N, A)$, $c_{i,j}(t)$, T

output: minimum duration shortest path

$L \leftarrow (\mathcal{D}^{(1,0)}, \mathcal{D}^{(n,T)})$;

$UB \leftarrow \min\{\text{computeUB}(\mathcal{D}^{(1,0)}), \text{computeUB}(\mathcal{D}^{(n,T)})\}$;

$LB \leftarrow \text{computeLB}(\mathcal{D}^{(1,0)})$;

$\mathcal{D}^{(1,t_1^k)} \leftarrow \mathcal{D}^{(1,0)}$;

while ($LB < UB$) **do**

if *there is a breakpoint* (j, τ) *between* $\mathcal{D}^{(1,t_1^k)}$ *and* $\mathcal{D}^{(1,t_1^{k+1})}$ **then**

$L \leftarrow \text{addtolist}(L, \mathcal{D}^{(j,\tau)})$;

if $\text{computeUB}(\mathcal{D}^{(j,\tau)}) < UB$ **then**

$UB \leftarrow UB(\mathcal{D}^{(j,\tau)})$

end

$\text{recomputeLB}(\mathcal{D}^{(1,t_1^k)})$;

$\text{computeLB}(\mathcal{D}^{(j,\tau)})$; $\text{insert}(L, \mathcal{D}^{(j,\tau)})$;

else

$LB(\mathcal{D}^{(1,t_1^k)}) = UB(\mathcal{D}^{(1,t_1^k)})$;

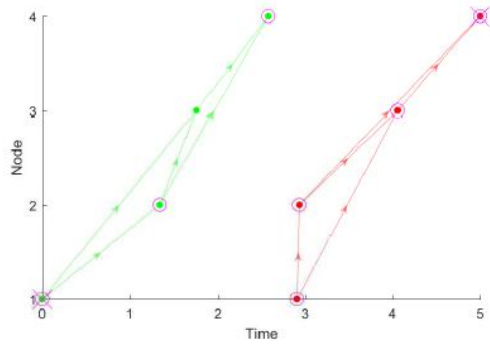
end

$LB \leftarrow \text{updateLB}(L)$;

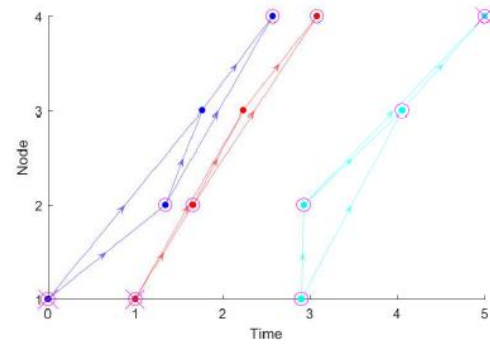
$\mathcal{D}^{(1,t_1^k)} \leftarrow \text{getBestLB}(L)$;

end

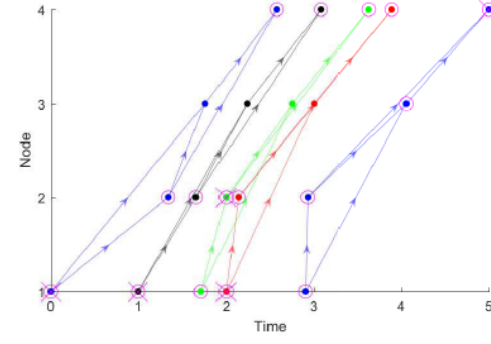
A Bigger Example



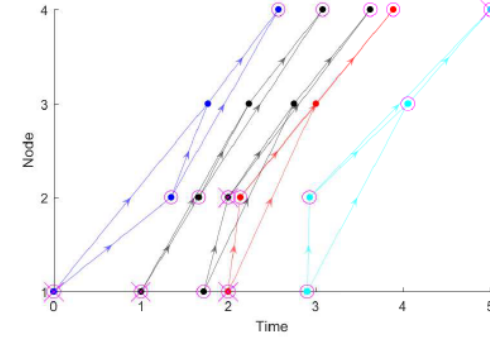
(a) Iteration 1



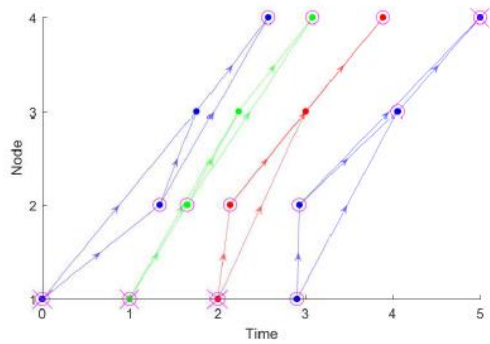
(b) Iteration 2



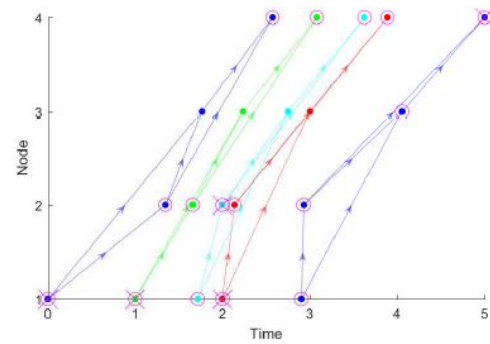
(e) Iteration 5



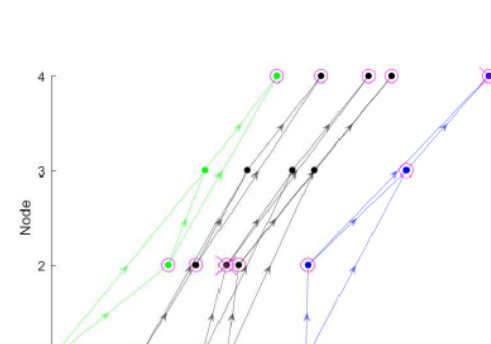
(f) Iteration 6



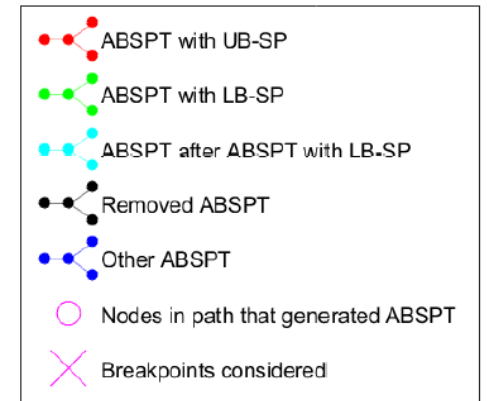
(c) Iteration 3



(d) Iteration 4



(g) Iteration 7



(h) Legend

Numerical Experiments

- Randomly generated test instances
- Graphs are dense: there is an arc between every pair of nodes
- Travel time functions are linear interpolants of sine functions (which ensures FIFO property) with random period
- Comparison of DDD with Foschini & Suri
- DDD investigates significantly fewer breakpoints and scales better when the number of breakpoints are increased

n	S	BP			Total #BP	%BP	Avg. Time DDD	Avg. Time Enum.	Ratio
		Min.	Avg.	Max.					
20	1	112	169.7	224	3800	4.47	103.4	55.8	1.86
	2.5	127	175.2	229	9500	1.84	118.3	218.9	0.54
	5	139	185.0	242	19000	0.97	136.2	690.0	0.20
30	1	182	229.4	268	5800	3.96	290.6	170.4	1.71
	2.5	165	243.9	303	14500	1.68	376.7	774.9	0.49
	5	179	261.8	328	29000	0.90	439.5	2493.0	0.18

n = number of nodes in network
 S = number of breakpoints per integer interval
 $T = 200$

Extension: Minimizing Total Travel Time

- Theorem: There exists an optimal minimum travel time path that contains travel subpaths (maximal subpath that does not contain any waiting) which are optimal minimum duration paths

Waiting arcs need to be added to the time-expanded network

Extension: Minimizing Total Travel Time

n	ntype	ttype	BP	Total # BP	% BP	Avg. Time DDD (s)	Avg. Time Enum (s)	% Time	Subpaths	Opt. Arcs
20	1	1	67.7	933	7.3	118.7	1505.2	7.9	2.0	3.7
20	1	2	51.9	933	5.6	77.3	1503.1	5.1	2.6	4.1
20	2	1	49.8	933	5.3	55.2	1534.7	3.6	1.4	3.9
20	2	2	45.0	933	4.8	45.3	1527.6	3.0	1.6	4.7
20	3	1	155.5	933	16.7	286.4	1570.4	18.2	3.1	10.6
20	3	2	128.4	933	13.8	244.8	1599.2	15.3	3.7	10.5
20	4	1	83.6	933	9.0	110.5	1654.2	6.7	2.2	7.4
20	4	2	51.0	933	5.5	48.5	1639.6	3.0	2.2	6.8
30	1	1	66.9	1423	4.7	249.1	9463.8	2.6	1.3	3.9
30	1	2	62.0	1423	4.4	196.8	9618.7	2.0	1.4	3.9
30	2	1	83.8	1423	5.9	285.8	9423.0	3.0	1.6	4.4
30	2	2	72.4	1423	5.1	215.5	10132.5	2.1	1.9	4.4
30	3	1	299.7	1423	21.1	2462.4	9506.8	25.9	3.7	16.3
30	3	2	341.5	1423	24.0	2886.1	9530.9	30.3	4.1	16.1
30	4	1	144.7	1423	10.2	671.0	9860.0	6.8	2.6	7.9
30	4	2	131.9	1423	9.3	631.5	9677.6	6.5	3.1	7.9

Generalizations

- Given
 - directed graph $D = (V, A)$
 - origin node 1, destination node n ,
 - a time horizon $[0, T]$, and
 - time-dependent travel time $c_{i,j}(t)$ for each arc $(i, j) \in A$ and time t , giving the time to traverse the arc if starting at time t , which
 - satisfy the FIFO property
 - tally set M
 - CSPP: penalty for waiting $w \geq 0$
 - CWMTTP: total waiting allowed $W \geq 0$
 - Find
 - start time $t_1^* \geq 0$ and
 - a time-dependent path $P(t_1^*)$ that
 - arrives at node n by time T
- Tally set: nodes where waiting contributes to objective or constraint
- CSPP: Minimize combination of travel time and waiting
- CWMTTP: Minimize travel time subject to constraint on waiting

Generalizations

Table: Complexity results for variants of CSPP.

	$w = 0$	$0 < w \leq 1$	$w > 1$
$M = N$	MTTP	MTTP*	NP-hard
$M = N \setminus \{1, n\}$	MTTP	MTTP	MDP
$M = N \setminus \{1\}$	MTTP	MTTP	MATP*
$M = N \setminus \{n\}$	MTTP	MTTP	MATP
$\{1, n\} \subset M \subseteq N$	MTTP	MTTP	MTTP
$\{1, n\} \not\subset M \subseteq N$	MTTP	MTTP	MTTP

Idea: Add appropriate costs to time-expanded network.

Table: Complexity results for variants of CWMTTP.

	$W = 0$	$W > 0$
$M = N$	NP-hard	NP-hard
$M = N \setminus \{1, n\}$	MDP	$\mathcal{O}(nK^2) \times$ MTTP
$M = N \setminus \{1\}$	MATP	$\mathcal{O}(nK^2) \times$ MTTP
$M = N \setminus \{n\}$	MATP [†]	$\mathcal{O}(nK^2) \times$ MTTP
$\{1, n\} \subset M \subseteq N$	MTTP	NP-hard
$\{1, n\} \not\subset M \subseteq N$	MTTP	NP-hard

Idea: If waiting constraint not tight, solution is an MTTP for some time horizon. Else if waiting constraint is tight, how long to wait is known.

Idea: Reduction from PARTITION.

Example

Table: Complexity results for variants of CWMTTP.

	$W = 0$	$W > 0$
$M = N$	NP-hard	NP-hard
$M = N \setminus \{1, n\}$	MDP	$\mathcal{O}(nK^2) \times$ MTTP
$M = N \setminus \{1\}$	MATP	$\mathcal{O}(nK^2) \times$ MTTP
$M = N \setminus \{n\}$	MATP [†]	$\mathcal{O}(nK^2) \times$ MTTP
$\{1, n\} \subset M \subseteq N$	MTTP	NP-hard
$\{1, n\} \not\subseteq M \subseteq N$	MTTP	NP-hard

- If waiting time constraint is not tight:
 - There exists an optimal CWMTTP solution that is an MTTP solution for some time interval
 - By perturbation arguments that MTTP solution must start at one of the timed copies of node 1 (K) and end at one of the timed copies of node n (K)
 - Examine all possible time intervals (K^2)

Example

Table: Complexity results for variants of CWMTTP.

	$W = 0$	$W > 0$
$M = N$	NP-hard	NP-hard
$M = N \setminus \{1, n\}$	MDP	$\mathcal{O}(nK^2) \times \text{MTTP}$
$M = N \setminus \{1\}$	MATP	$\mathcal{O}(nK^2) \times \text{MTTP}$
$M = N \setminus \{n\}$	MATP [†]	$\mathcal{O}(nK^2) \times \text{MTTP}$
$\{1, n\} \subset M \subseteq N$	MTTP	NP-hard
$\{1, n\} \not\subset M \subseteq N$	MTTP	NP-hard

- If waiting time constraint is tight:
 - Either there exists an optimal CWMTTP solution whose first to second last travel subpath is an MTTP solution from node 1 to some node j , or there exists an optimal CWMTTP solution whose second to last travel subpath is an MTTP solution from some node j to node n
 - Remaining travel time subpath can be found by waiting until the waiting time constraint is tight and finding a minimum arrival time path
 - Examine all possible time intervals (K^2)

Time-Dependent Traveling Salesman Problem with Time Windows

Traveling Salesman Problem with Time Windows

- The Traveling Salesman Problem (TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"
- The Traveling Salesman Problem with Time Windows (TSPTW) is similar to the TSP except that the cities must be visited within a given time window. This added time constraint renders the problem even more difficult in practice. In fact, even finding a feasible solution is difficult.

Traveling Salesman Problem with Time Windows

Compact formulation

$$\min \sum_{(ij)} c_{ij} x_{ij}$$

$$\sum_j x_{ji} = 1 \quad \forall i$$

arrive exactly once

$$\sum_j x_{ij} = 1 \quad \forall i$$

depart exactly once

$$t_j \geq (t_i + \tau_{ij})x_{ij} \quad \forall (ij)$$

dispatch time consistency

$$e_i \leq t_i \leq l_i \quad \forall i$$

respect time windows

$$x_{ij} \in \{0, 1\} \quad \forall (ij)$$

do we travel from i to j?

$$t_i \geq 0 \quad \forall i$$

time we depart from i

$$\min \sum_{(ij)} c_{ij} x_{ij}$$

$$\sum_j x_{ji} = 1 \quad \forall i$$

$$\sum_j x_{ij} = 1 \quad \forall i$$

$$t_j \geq t_i + \tau_{ij} - M(1 - x_{ij}) \quad \forall (ij)$$

$$e_i \leq t_i \leq l_i \quad \forall i$$

$$x_{ij} \in \{0, 1\} \quad \forall (ij)$$

$$t_i \geq 0 \quad \forall i$$

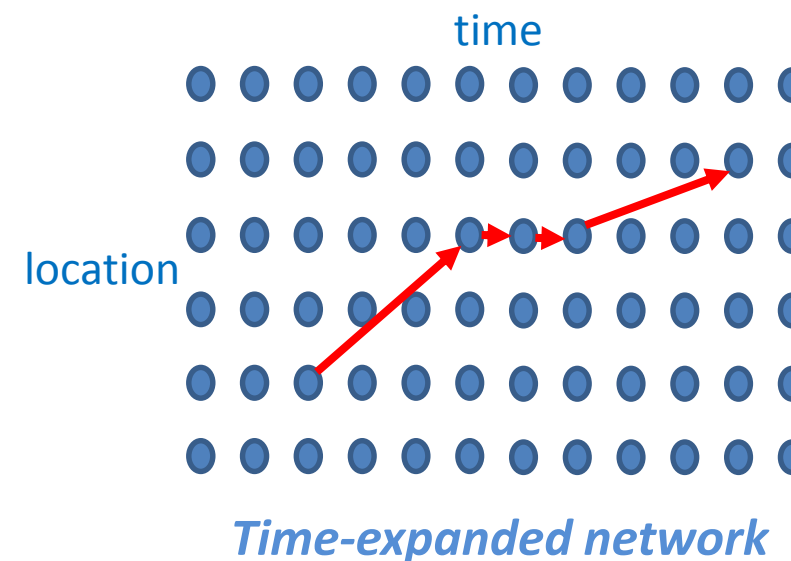
Traveling Salesman Problem with Time Windows

Extended formulation

$$\begin{aligned} \min \quad & \sum_t \sum_{(ij)} c_{ij} x_{ij}^t \\ \sum_j x_{ji}^{t-\tau_{ji}} - \sum_j x_{ij}^t &= 0 \quad \forall i, t \\ \sum_{e_i \leq t \leq l_i} \sum_j x_{ij}^t &= 1 \quad \forall i \end{aligned}$$

$$x_{ij}^t \in \{0, 1\} \quad \forall (ij), t$$

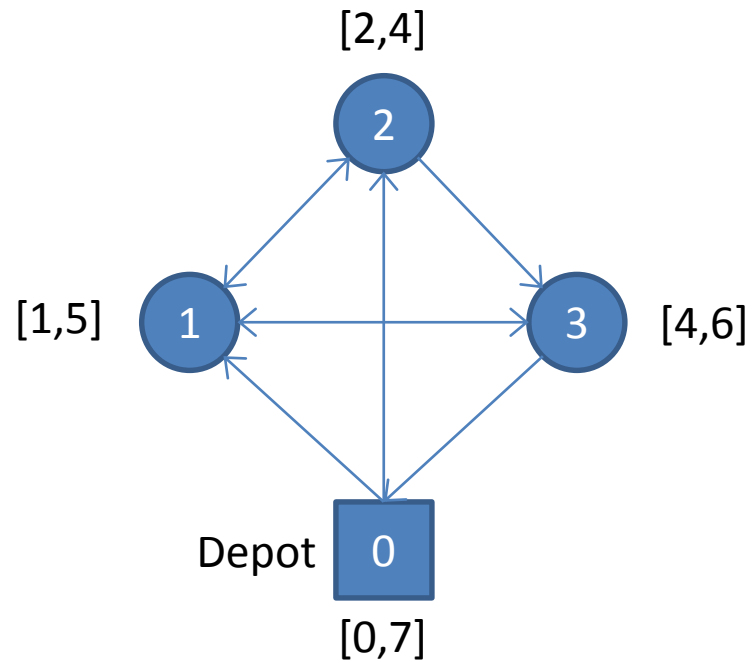
do we depart from i to j at time t ?



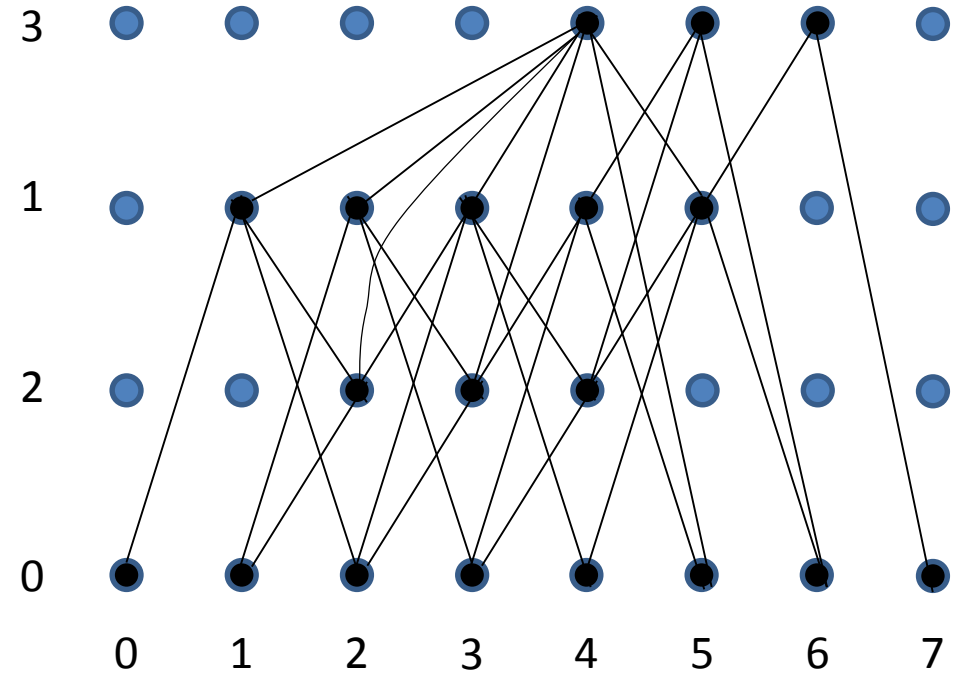
Observations

- Compact models that use continuous variables to model time have weak linear programming relaxations.
- Extended formulations with binary variables indexed by time have strong linear programming relaxations, but (tend to) have a huge number of variables.

Time-expanded networks



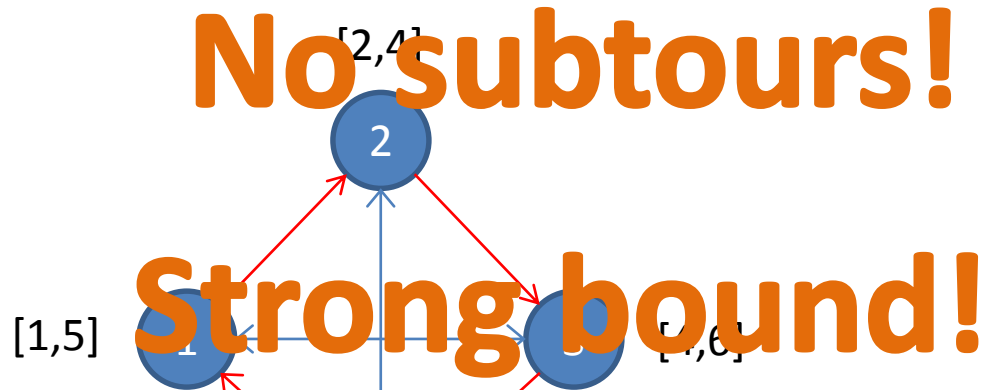
All arcs have travel time 1



All arcs are directed forward in time

Time-expanded networks

No subtours!

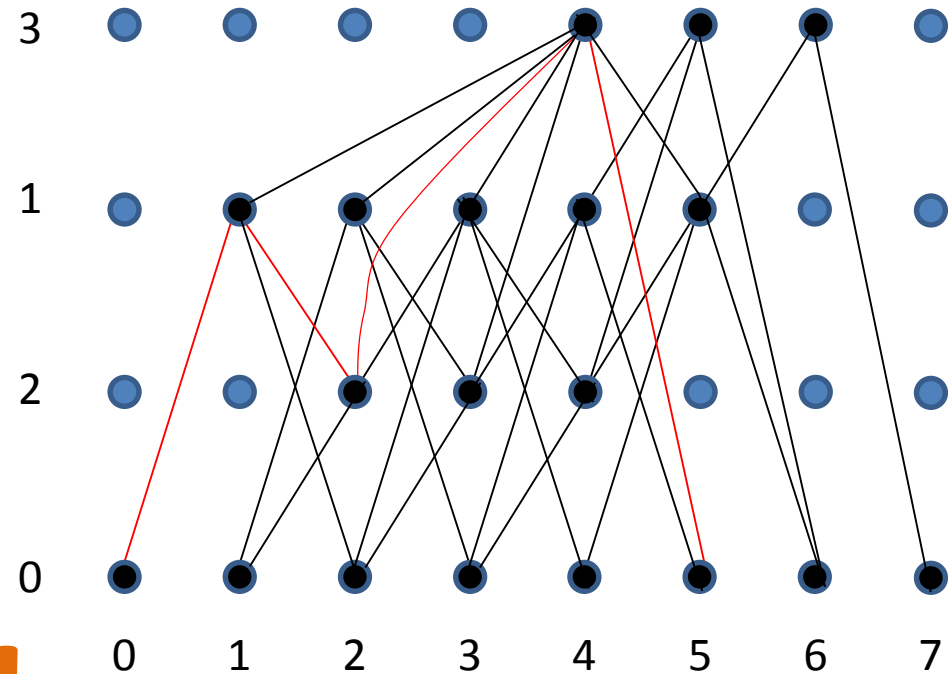


Strong bound!

Readily extends to time-dependent case!

All arcs have travel time 1

...but HUGE network...



All arcs are directed forward in time

1 unit flows into every node

1 unit flows out of every node

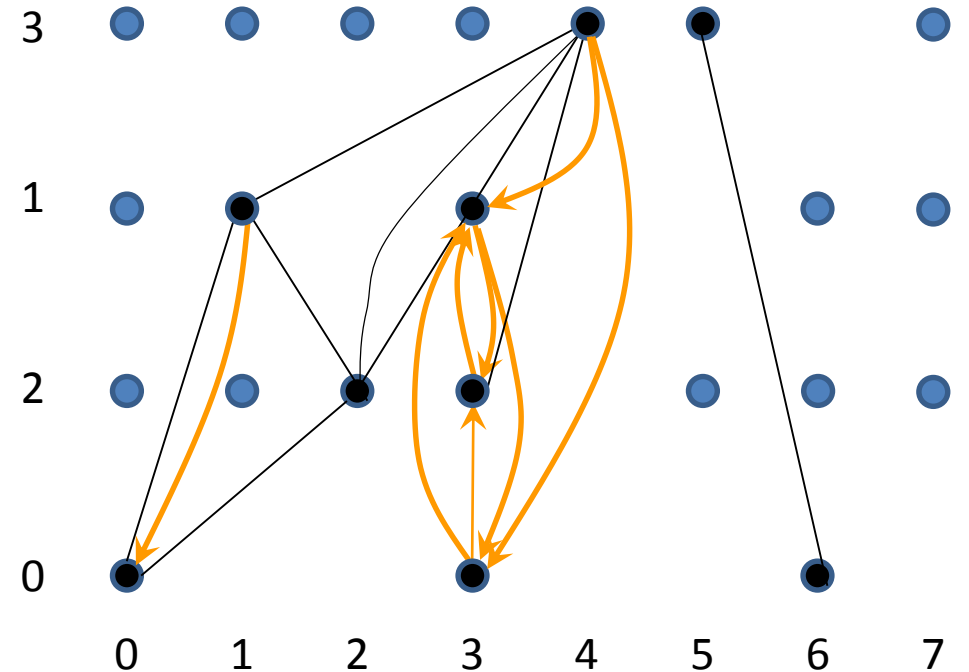
Flow in = flow out at each time-space node

Dynamic Discretization Discovery

- Partially time-expanded network
 - Include TW start nodes
 - Early arrival property
- Dual bound formulation
 - Flow in = flow out at each TS node
 - Enter each node exactly once
 - Guaranteed lower bound

- *Issues*

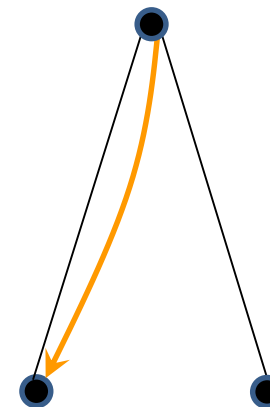
- *May have subtours*
- *How to get a primal feasible solution?*
- *How to improve the dual bound/refine the discretization?*



All arcs are directed forward in time...
...except those that are **too short**

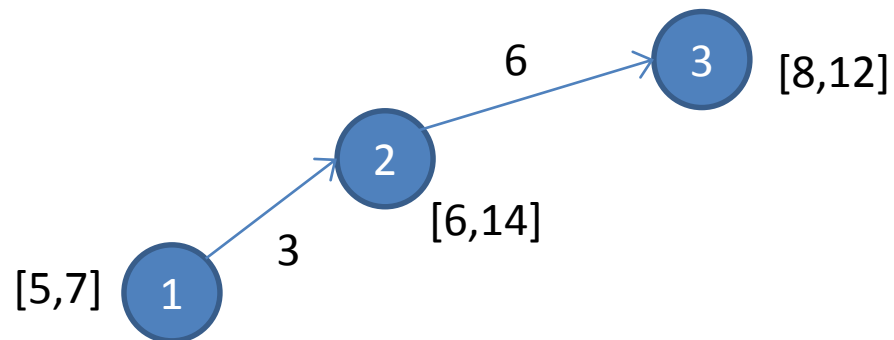
Addressing subtours

- Options
 - Use formulations that prevent subtours with continuous flow variables
 - Single commodity
 - Multicommodity (one for each customer)
 - Flowing in original network or in time-expanded network
 - Refine time-expanded network until none are possible
 - Add a new time point
 - Lengthen a too-short arc
 - Now the subtour cannot be used

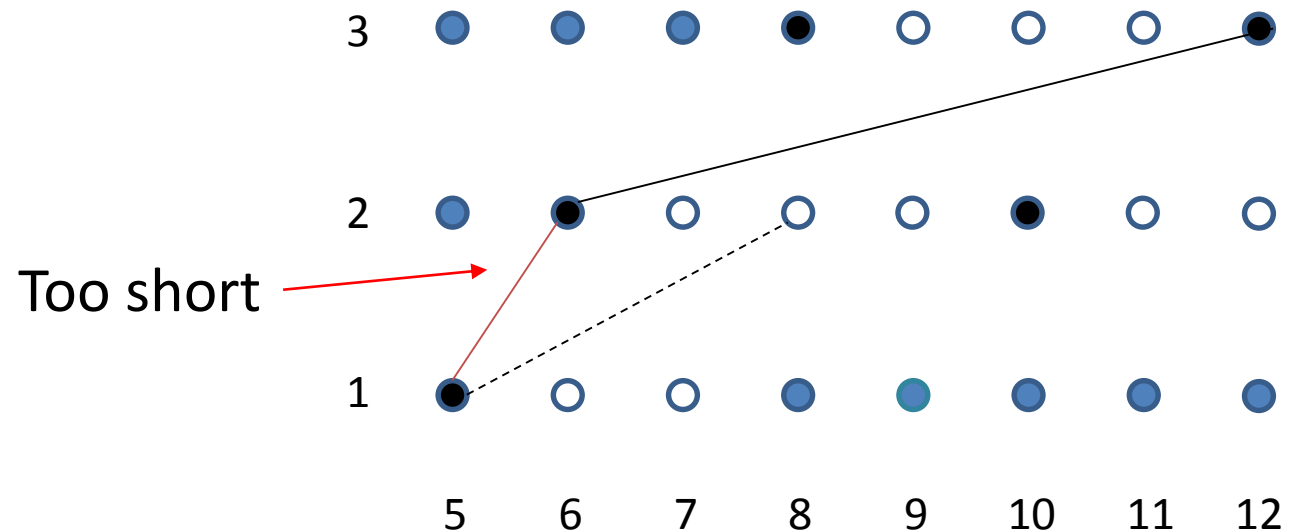


Getting a primal feasible solution

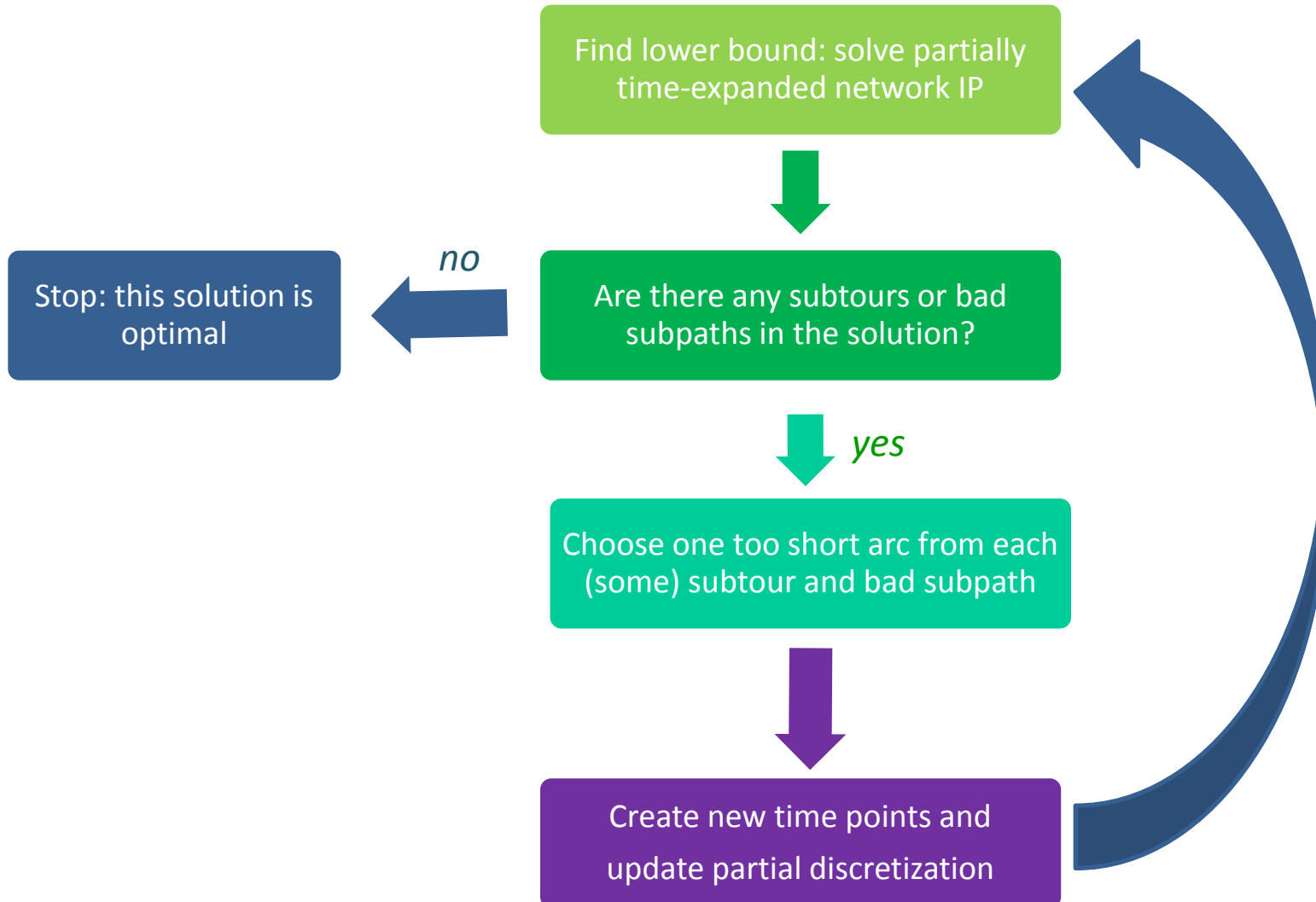
- Once subtours are eliminated, any feasible solution to the IP formulation on the partially time-expanded network induces a tour
- However the tour may violate time windows
 - Any TW-violating subpath in the tour must contain an arc that is too short



Bad path: $5 + 3 + 6 > 12$



Dynamic Discretization: base algorithm



Algorithm Enhancements

- Preprocessing

- On original network

- Derive precedence relations
 - Tighten time windows
 - Eliminate arcs

- On partially time-expanded network

- Maintain latest time tour can depart customer i to go to customer j without making it impossible to visit some customer k
 - Eliminate/do not create time-space arcs starting at later times

- Cutting

- Textbook subtour elimination

- Textbook bad path elimination

$$\sum_{(i,j) \in S^2} \sum_{((i,t),(j,t')) \in \mathcal{A}_{\mathcal{T}}} x_{((i,t),(j,t'))} \leq |S| - 1$$

Algorithm Enhancements

- Primal heuristics
 - Create a partially time-expanded network that yields an upper bound formulation
 - for each time-space node (i,t) and each original arc (i,j)
 - find smallest s such that $s - t$ is at least the travel time from i to j
 - create time-space arc $((i,t),(j,s))$
 - Any feasible solution found is feasible for the TSPTW
 - Run with a time limit

Algorithm Enhancements

- A second primal heuristic that also accelerates dual convergence
 - Start with the current lower bound network
 - for any arc that is currently going backwards in time
 - lengthen it as little as possible, but enough to ensure it goes forward in time
 - Feasible solutions may or may not be feasible for the TSPTW
 - Run with a time limit
 - Harvest all feasible solutions found
 - if feasible for the TSPTW, may replace incumbent
 - if not, collect bad subpath information and add the corresponding cutting planes to the dual, lower bound, IP

Computational Results

Instance set	Traffic pattern	TTBF-CB			DDD-TD-TSPTW		
		Inst	Slv	Tme	Inst	Slv	Tme
Set 1	A	478	470	31.99	478	478	12.91
	B	474	470	20.44	474	474	8.79
Set <i>w100</i>	A	108	107	100.13	480	480	1.31
	B	108	107	93.10	480	480	1.14

Table 1 Performance on Arigliano et al. (2015) instances

Dynamic Discretization Discovery

- N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh. “The Continuous-Time Service Network Design Problem”, *Operations Research* 65, 1303-1321, 2017
- F. Lagos, N. Boland and M. Savelsbergh, “The continuous time inventory routing problem”, *Transportation Science*, to appear.
- D.M. Vu, N. Boland, M. Hewitt, and M. Savelsbergh, “Solving Time Dependent Traveling Salesman Problems with Time Windows”, *Transportation Science*, to appear.
- N. Boland and M. Savelsbergh, “Perspectives on Integer Programming for Time Dependent Models”, *TOP*, available online.
- L. Marshall, N. Boland, M. Hewitt, and M. Savelsbergh. “Interval-based Dynamic Discretization Discovery for Solving the Continuous-Time Service Network Design Problem”, *Optimization Online* 6883, 2018
- E. He, N. Boland, G. Nemhauser and M. Savelsbergh, “Dynamic Discretization Discovery Algorithms for Time-Dependent Shortest Path Problems”, *Optimization Online* 7082, 2019

Dynamic Discretization Discovery: Solving Time-Extended Network



Formulations Effectively

Questions?

