TRAVELING SALESMAN PROBLEM WITH TIME WINDOWS (TSPTW)

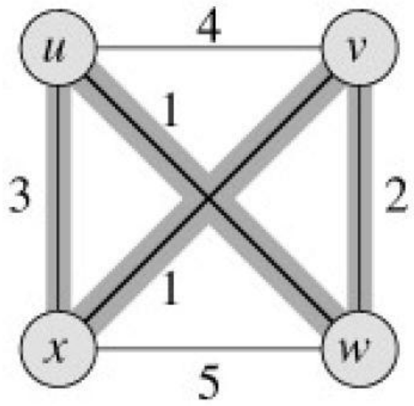
Aakash Anuj	10CS30043
Surya Prakash Verma	10AE30026
Yetesh Chaudhary	10CS30044

Supervisor: Prof. Jitesh Thakkar

TSP

- 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 TSP has several applications even in its purest formulation, such as **planning**, **logistics**, and the **manufacture of microchips**.
- It is an **NP-hard** problem in combinatorial optimization

TSP



An instance of the traveling-salesman problem.
Shaded edges represent a minimum-cost tour, with cost 7.

TSPTW - Definition

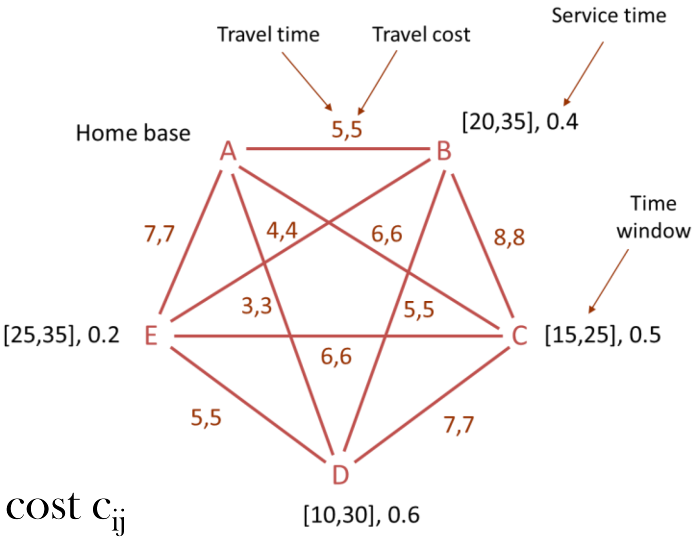
- A more difficult problem than TSP
- Involves the design of a minimum cost tour such that
  - Every node is visited exactly once
  - Service at a node must begin and end within the time interval specified at each node
- Incorporates service time at each node and travel time to visit from one node to other

TSPTW - Motivation

- Practical applications in :
  - Postal or office deliveries within specified timings for each
  - School bus routing and scheduling
  - Automated manufacturing environments
  - Automated guided vehicles

TSPTW – Formulation

- Consider a network G=(N,A)
  - N={1,2,3, ..., n} is the set of nodes and A is the set of arcs
- Each node i∈N is associated with
  - A time window $[a_i,b_i]$
  - A service time s_i
- Each arc is associated with a travel time t_{ij} and a travel cost c_{ij}



Related Work

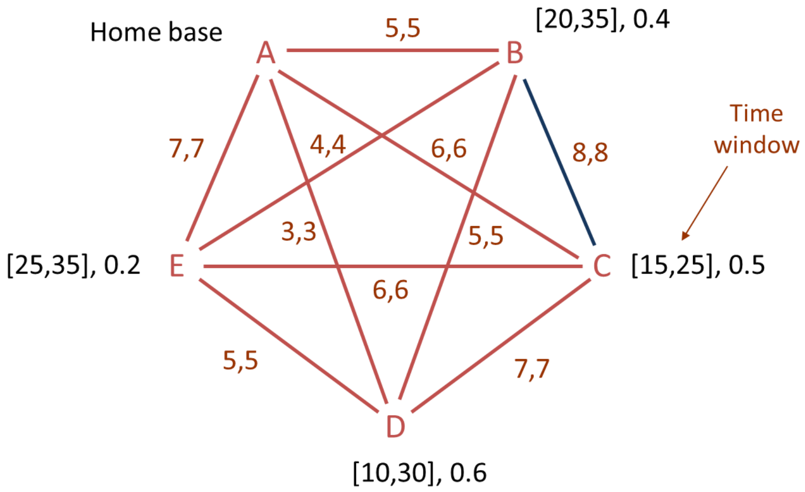
- Savelsbergh (1985) showed that even finding a feasible solution to TSPTW is an NP complete problem
- Bakers (1983) proposed an approach which performed well on problems with upto 50 nodes
- This work by Dumas et al. (1995) is successful in solving problems with upto 200 nodes and fairly wide time windows

Approach

- **Preprocessing to remove infeasible arcs**
- Dynamic Programming
- Reduction of the state space via infeasibility tests
  - Takes advantage of the time window constraints
  - Performed during the execution of the algorithm

Preprocessing

- An arc (i,j)∈ A is feasible if $a_i + s_i + t_{ij} \le b_j$



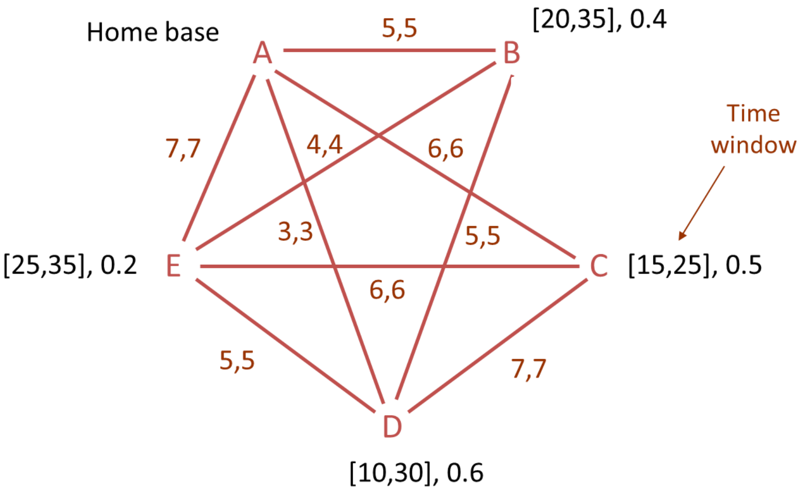
For the edge BC:

$a_B + s_B + t_{BC} = 20+0.4+8$
$=28.4 > b_C = 25$

Edge BC can be pruned!

Preprocessing

- An arc $(i,j) \in A$ is feasible if $a_i + s_i + t_{ij} \leq b_j$



For the edge BC:

$a_B + s_B + t_{BC} = 20+0.4+8$

$=28.4 > b_C = 25$

Edge BC can be pruned!

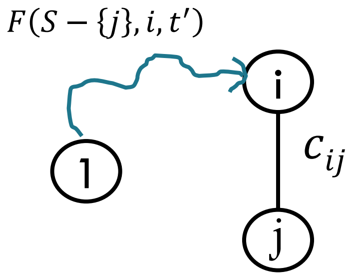
Approach

- Preprocessing to remove infeasible arcs
- **Dynamic Programming**
- Reduction of the state space via infeasibility tests
  - Takes advantage of the time window constraints
  - Performed during the execution of the algorithm

Dynamic Programming

- Define $\boldsymbol{F(S, i, t)}$ as the least cost of a path **starting at node 1** passing through **every node of S exactly once** and **ending at node i∈S** and ready to **serve node i at time t or later.**

- The function $F(S, j, t)$ can be computed by the recurrence
  - $F(S, j, t) = \min_{(i,j) \in A} \{F(S - \{j\}, i, t') + c_{ij}\}$ where $\mathrm{t}' + \mathrm{s_i} + \mathrm{t_{ij}} \leq t$ and $a_i \leq t' \leq b_i$
  - Base condition: $F(\{1\}, 1,0) = 0$, since we start at node 1



Dynamic Programming

- Define $\boldsymbol{F(S,i,t)}$ as the least cost of a path **starting at node 1** passing through **every node of S exactly once** and **ending at node i∈S** and ready to **serve node i at time t or later.**

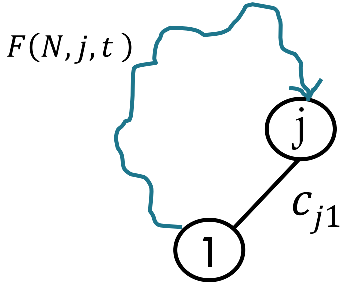
- The function $F(S,j,t)$ can be computed by the recurrence

  - $F(S,j,t) = \min_{(i,j)\in A}\{F(S-\{j\},i,t') + c_{ij}\}$ where $\mathrm{t}' + \mathrm{s_i} + \mathrm{t_{ij}} \leq t \; \; and \; a_i \leq t' \leq b_i$

  - Base condition: $F(\{1\},1,0) = 0$, since we start at node 1

- The optimal TSPTW solution is given by:

  - $\min_{(j,1)\in A}\{F(N,j,t\,) + c_{j1}\} \; \; s.t. \quad a_j \leq t \leq b_j$



Dynamic Programming

- Let ξ_k be a state set which contains all feasible states (S, i, t) s.t **|S|= k**
  - S i t
    ξ_1={({1},1,0)}
- For computing ξ_k **from** ξ_{k-1}, do the following steps:
  - For each $(S, i, t) \in \xi_{k-1}$, add the state (S', j, t') *to* ξ_k *where*
  - $S' = S \cup \{j\}, (i,j) \in A$ *and*
  - $t' = \max(a_j, t + s_i + t_{ij})$ *and*
  - (S', j, t') *is a* ***feasible expansion*** *of* (S, i, t)

Algorithm

- Initialize $\xi_1=\{(\{1\},1,0)\}$ and $F(\{1\},1,0)=0$
- for(k=2,3,.....n) do
  - for $(S, i, t) \in \xi_{k-1}$ do
    - add the state (S', j, t') to ξ_k only if (S', j, t') passes elimination tests
    - update $F(S', j, t') = F(S, i, t) + c_{ij}$
- The optimal solution is $\min_{(j,1)\in A} \{F(N, j, t) + c_{j1}\}$ $s.t.\ a_j \le t \le b_j$

Elimination/Infeasibility tests

- Test 1
- Test 2
- Test 3
- Dominance Tests

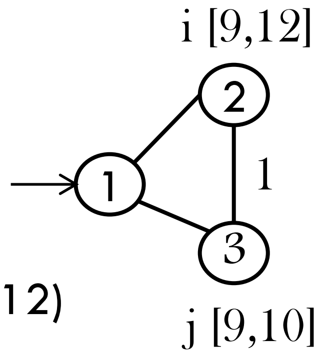
Test 1

- Let FIRST(S, i) denotes the smallest time when a service can begin at node i
- Let LDT(i, j) denote the latest departure time at i to begin service at node j s.t. time of service at j is feasible
- Reject (S, i, t), $a_i \leq t \leq b_i$ if
  - FIRST(S, i)> $\min_{j \notin S} LDT(i,j)$

Let ξ_k = {({1,2},2,11), ({1,2},2,12)}

FIRST({1,2},2)=11 > LDT(2,3)=9

Can remove ({1,2},2,11) and ({1,2},2,12)

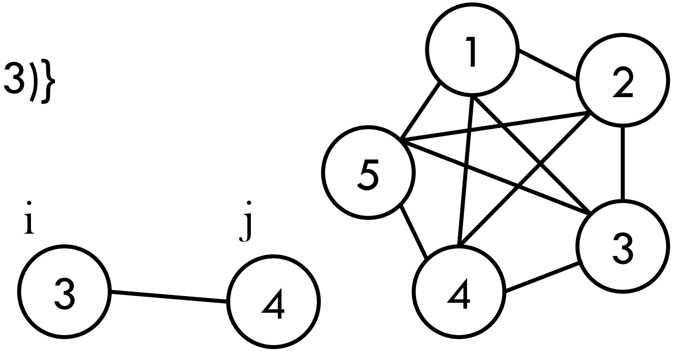


Test 2

- Let BEFORE(j) denote the set of nodes which must be visited before visiting the node j
- Reject (S, i, t), $a_i \leq t \leq b_i$ if
  - $\exists j \notin S$ and $(i,j) \in A$ and BEFORE(j)$\not\subset$S

Let $\xi_k = \{(\{1,2,3\},3,12),(\{1,5,3\},3,13)\}$

BEFORE(4)={1, 2, 5}



Test 3

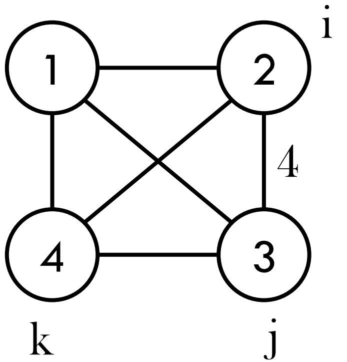
- Reject (S', j ,t'), $a_j \leq t' \leq b_j$ if
  - $\exists (S, i, t)$ $s.t$ j ∉ S , (i,j) ∈ A , t ≤ LDT(i,j) and
  - $\forall k$ $s.t$ k ∉ S and t + t_{ij} > LDT(j,k)

Let (S, i, t) = ({1,2},2,18) and (S′, j, t′)=({1,2,3},3,22)

LDT(2,3)=20 and LDT(3,4)=12 and t_{23}=4

(2,3) can be feasible since t=18≤ LDT(2,3) = 20 but

(3,4) is infeasible since t+t_{23}=18+4=22>LDT(3,4)=12



Dominance Test

- Reject (S, i, t') ∈ ξ_k if ∃ (S, i, t) ∈ ξ_k s.t
  - $F(S, i, t) \leq F(S, i, t')$
  - $t \leq t'$

Experimental results

- Variation of CPU time with number of nodes

n	|A'|	|(S,i,t)|	CPU time
20	34.2	4.4	0.02
40	121.8	16.0	0.08
60	226.0	21.8	0.15
80	362.6	49.8	0.35
100	510.0	45.0	0.62
150	975.8	326.8	2.44

w=20

n	|A'|	|(S,i,t)|	CPU time
20	115.2	35.6	0.14
40	404.2	554.0	4.37
60	670.6	1344.4	6.84
80	1149.6	7716.8	55.32
100	1731.4	6804.8	107.95
150	2953.6	26351.0	462.97

w=60

Experimental results

- Comparison of number of labels removed by each test

	w	Test 1	Test 2	Test 3
(S,i,t)	20	21.8	3716.4	37.4
CPU time		0.15	6.20	0.17
(S,i,t)	40	145.8	18520.5	229.0
CPU time		0.91	49.23	0.98

n=60

Conclusion

- For a given problem size, problem difficulty increases with the time windows
- For narrow widths, its behavior is less than exponential allowing large size problems to be solved.
  - E.g. a 250 node problem with w=20 is solved in less than 10 sec.
- The algorithm can accommodate additional costs for waiting at a node such as total schedule time.
- The algorithm works for larger problem sizes and time windows than any previous work.

References

- **Research papers**:
  - http://pubsonline.informs.org/doi/pdf/10.1287/opre.43.2.367
  - http://www.jstor.org/stable/pdfplus/172015.pdf?&acceptTC=true&jpdConfirm=true

Thank you

Questions / Comments?