Graph-Based Modeling, Scheduling, and Verification for Intersection Management of Intelligent Vehicles

CS637A: Embedded and Cyber Physical Systems

Fall 2020: Project Presentation

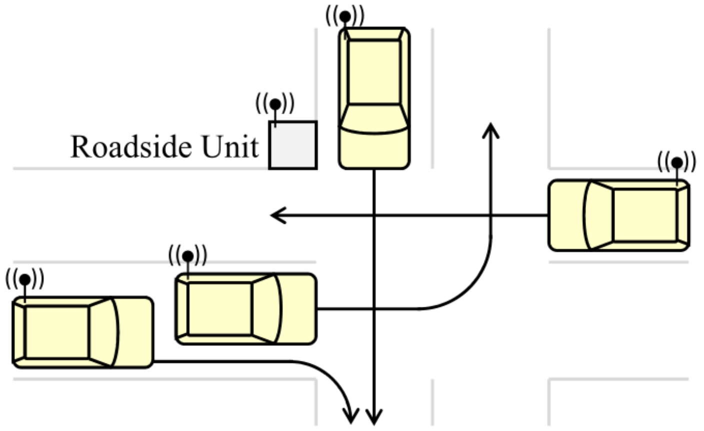
Paper Authors:
YI-TING LIN, HSIANG HSU, SHANG-CHIEN LIN, CHUNG-WEI LIN,
IRIS HUI-RU JIANG, National Taiwan University, Taiwan
CHANGLIU LIU, Carnegie Mellon University, USA

Conference: EMSOFT 2019

Presented By:
Ashwin Shenai (180156)
Kshitij Kabeer (180366)

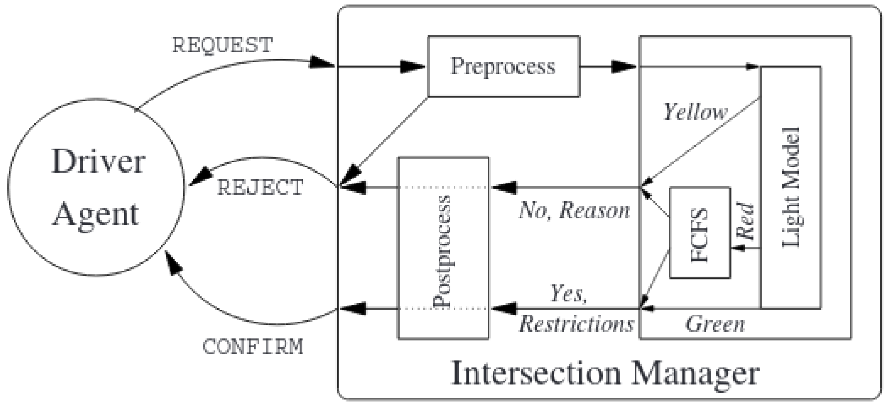
Intersection Management

- Management of vehicles and their passing order, at intersections
- Crucial for efficient traffic management and safety, especially with the advent of autonomous vehicles
- Optimizing passing time, preventing deadlock and ensuring no collisions – some of the prime objectives
- Position of each vehicle and commands communicated amongst themselves, or to a roadside unit – the intersection manager

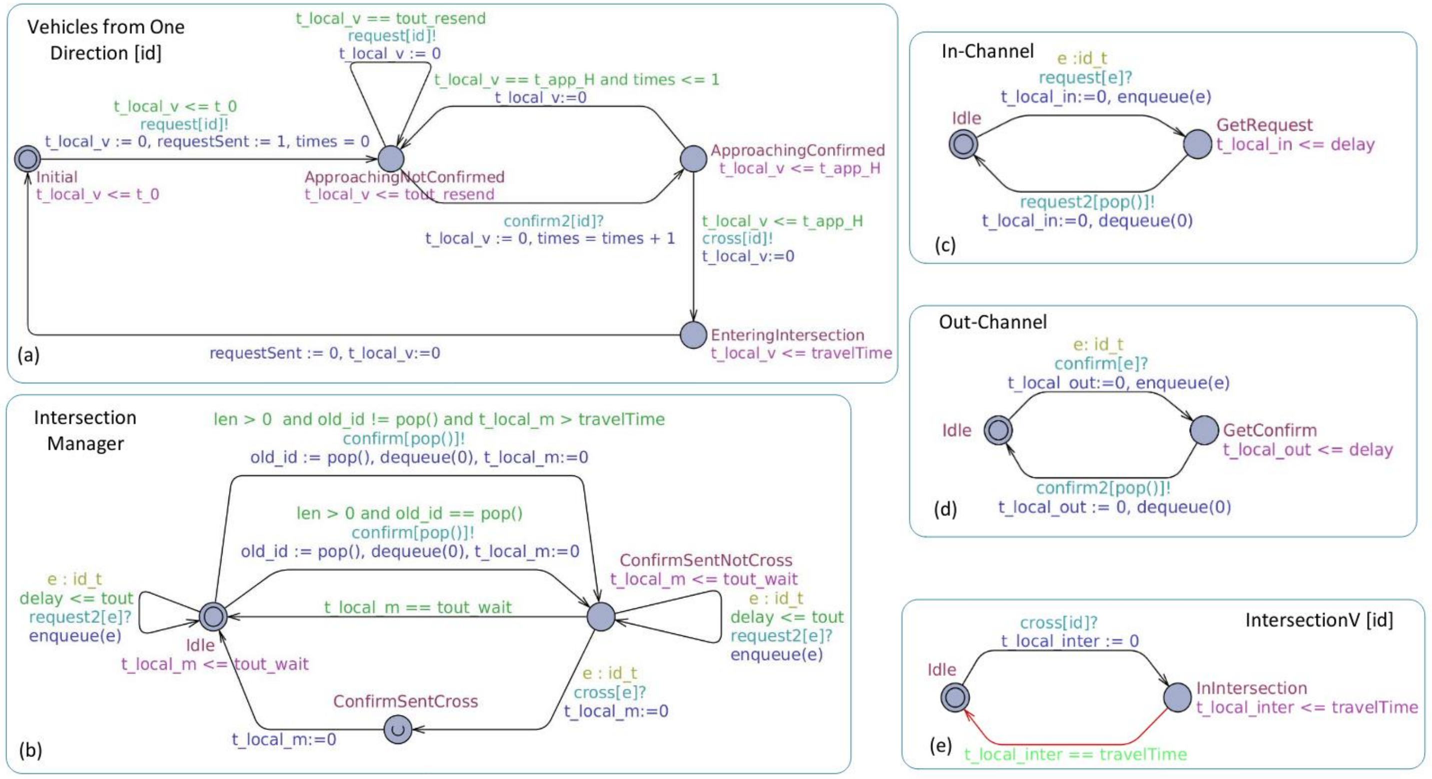


Related Work

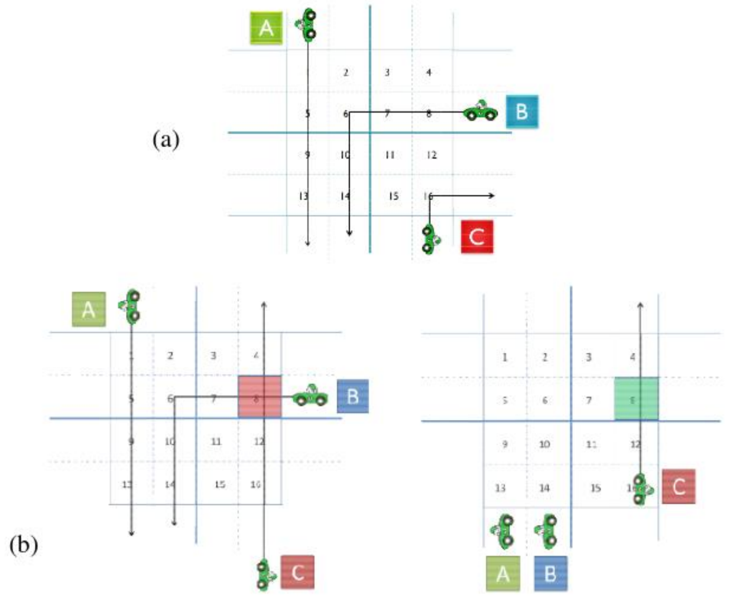
- Protocols between vehicles and a centralized intersection manager



Multi-Agent Reservation-based Scheduler [12]



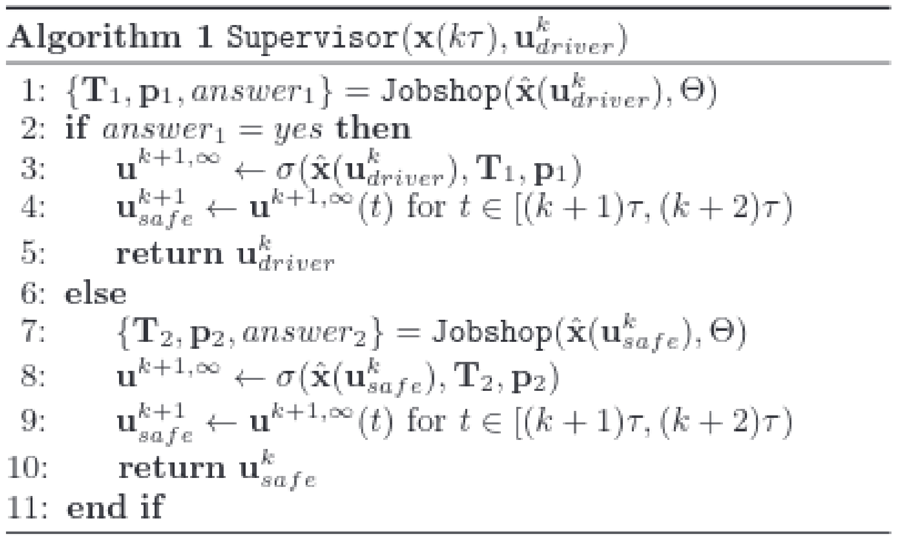
Delay-aware centralized intersection manager [26]



STIP: V2V Intersection Protocols [4]

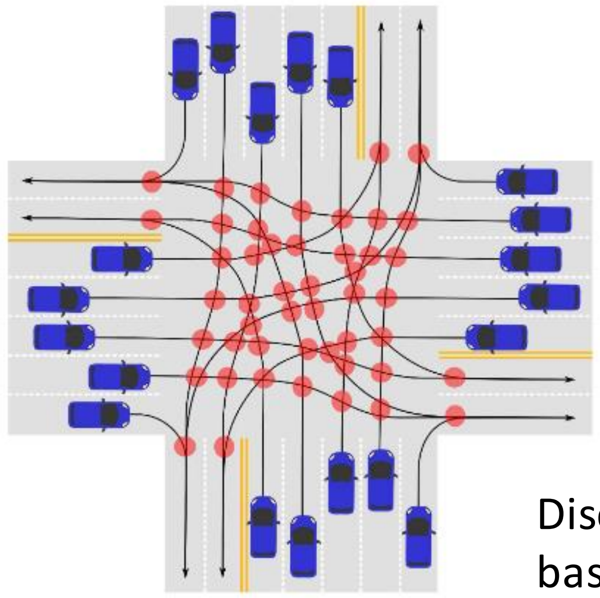
Related Work

- Discrete-event control and conflict resolution in a centralized setting

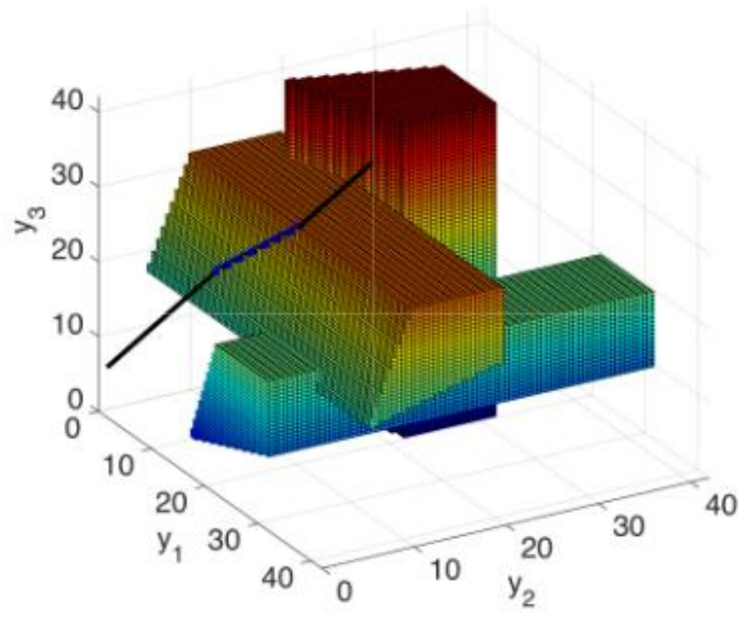


```
Algorithm 1 Supervisor($\mathbf{x}(k\tau), \mathbf{u}^k_{driver}$)
{$\mathbf{T}_1, \mathbf{p}_1, answer_1$} = Jobshop($\hat{\mathbf{x}}(\mathbf{u}^k_{driver}), \Theta$)
if $answer_1 = yes$ then
    $\mathbf{u}^{k+1,\infty} \leftarrow \sigma(\hat{\mathbf{x}}(\mathbf{u}^k_{driver}), \mathbf{T}_1, \mathbf{p}_1)$
    $\mathbf{u}^{k+1}_{safe} \leftarrow \mathbf{u}^{k+1,\infty}(t)$ for $t \in [(k+1)\tau, (k+2)\tau)$
    return $\mathbf{u}^k_{driver}$
else
    {$\mathbf{T}_2, \mathbf{p}_2, answer_2$} = Jobshop($\hat{\mathbf{x}}(\mathbf{u}^k_{safe}), \Theta$)
    $\mathbf{u}^{k+1,\infty} \leftarrow \sigma(\hat{\mathbf{x}}(\mathbf{u}^k_{safe}), \mathbf{T}_2, \mathbf{p}_2)$
    $\mathbf{u}^{k+1}_{safe} \leftarrow \mathbf{u}^{k+1,\infty}(t)$ for $t \in [(k+1)\tau, (k+2)\tau)$
    return $\mathbf{u}^k_{safe}$
end if
```

Job scheduling-based semi-autonomous supervisory control



Discrete-event conflict-based modelling

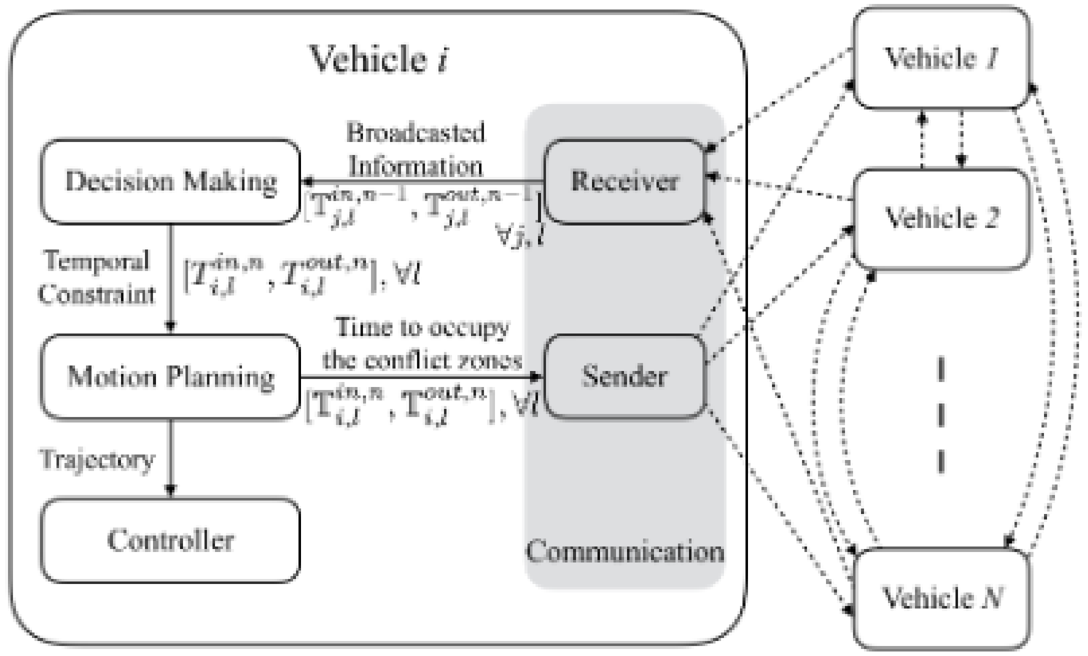


Reactive supervisory control

Refs: [2,7,9,22]

Related Work

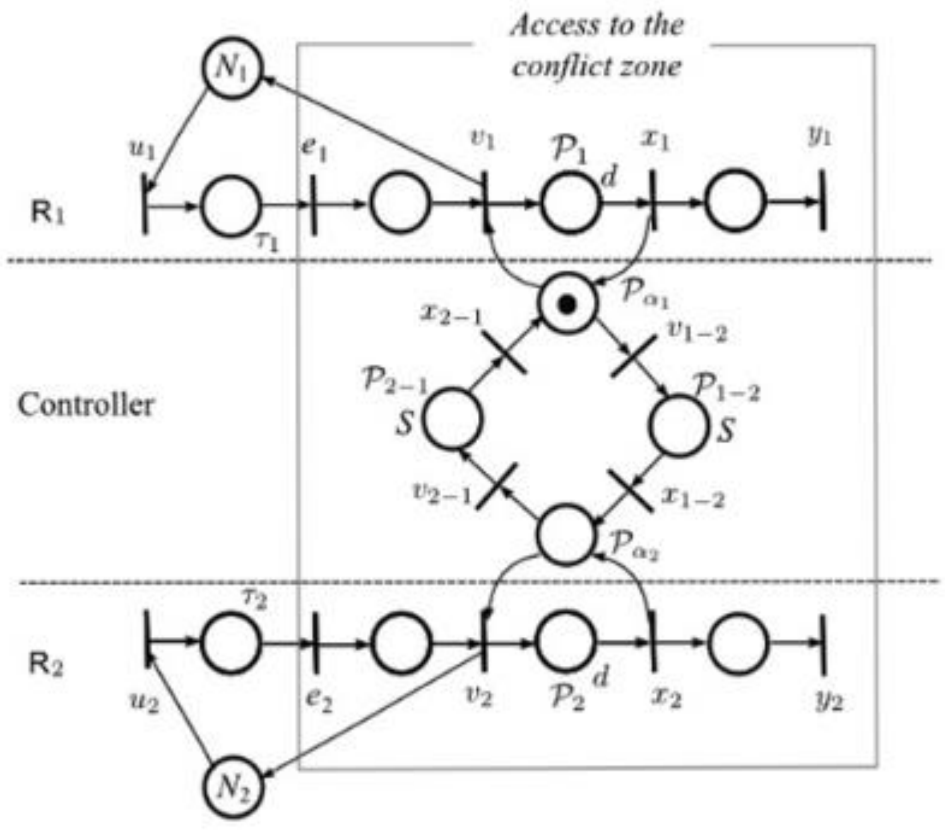
- Distributed inter-vehicle communication-based scheduling



Vehicle i
Decision Making
Broadcasted Information
$[T_{j,l}^{in,n-1}, T_{j,l}^{out,n-1}]$ $\forall j, l$
Receiver
Temporal Constraint
$[T_{i,l}^{in,n}, T_{i,l}^{out,n}], \forall l$
Motion Planning
Time to occupy the conflict zones
$[T_{i,l}^{in,n}, T_{i,l}^{out,n}], \forall l$
Sender
Trajectory
Controller
Communication
Vehicle 1
Vehicle 2
Vehicle N

Vehicle model for distributed scheduling [18]

- Petri net-based modelling for cooperative vehicles

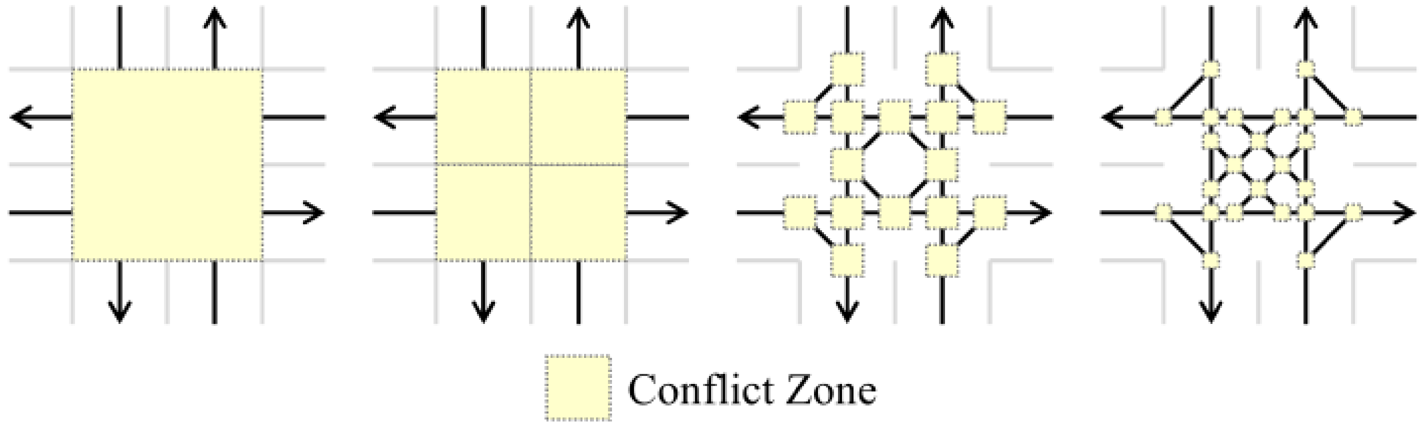


Timed petri-net model for two-lane intersection [24]

Paper Contributions

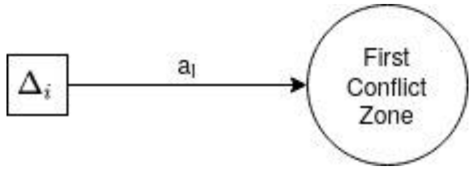
- Graph based model – can deal with various granularities of intersections, highly expressive
- Centralized cycle removal for efficient, safe and deadlock free crossing of vehicle
- Efficiently scalable in response to increasing number of vehicles and conflict zone complexity
- Formal verification techniques to guarantee deadlock-freeness in all scenarios

Terminology

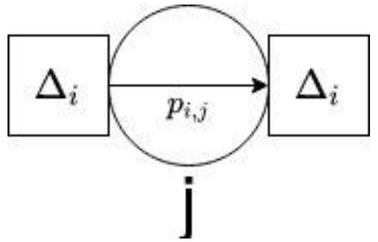


- Intersection
- Conflict Zone (j)
- Vehicle (Δ_i)
- Intersection Manager

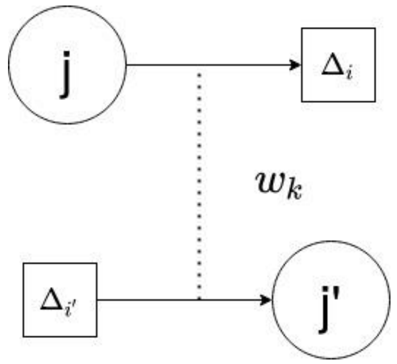
- Earliest Arrival Time(a_i)



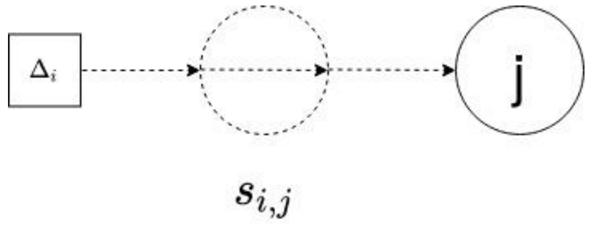
- Vertex Passing Time ($p_{i,j}$)



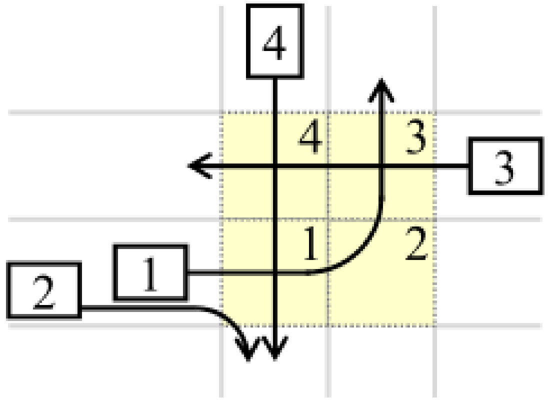
- Edge Waiting Time(w_k)



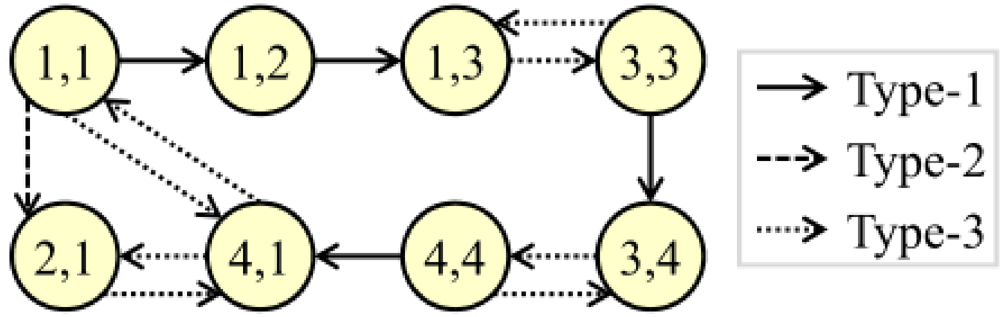
- Vertex Entering Time($s_{i,j}$)



Timing Conflict Graph (TCG)



(a)



(b)

- Type-1: Vehicle Δ_i goes from j to j'
- Type-2: Vehicles Δ_i and $\Delta_{i'}$ (in the same starting lane) go through j
- Type-3: Vehicles Δ_i and $\Delta_{i'}$ (in different starting lanes) go through j
  - Always in pairs

Problem Modelling

- Given a TCG G, earliest arrival times, edge waiting and passing times

1. Compute an acyclic subgraph G'
   - With all the vertices, Type-1 and Type-2 edges
   - Only one out of each pair of Type-3 edges
2. Guarantee no deadlock in G'
3. Assign an entering time to each vertex in G'
4. Minimize the maximum leaving time $t_{max} = \max_{G'}(s_{i,j} + p_{i,j})$

1. Collision Freeness
2. Liveness/Feasibility
3. Scheduling
4. Optimality of Schedule

Assumptions

- Perfect, no-delay communication among vehicles and intersection managers.
  - Can model delay by increasing edge wait times, or adding noise in inputs.
- Problem solved in discrete chunks, no dynamic addition of vehicles
  - Vehicles coming in before the current graph is processed will be scheduled in the next chunk
- Dynamics of the vehicles aren't modelled – speed is constant or zero
  - No overtaking allowed

Verification

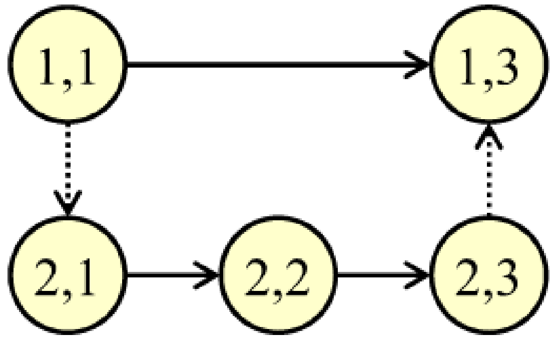
- Collision-freeness is guaranteed by the scheduler
- Need to ensure deadlock-freeness through verification
  - Graph-based verification
  - Petri net-based verification
- Either method can be used as a sub-routine to verify liveness of candidate schedules during scheduling

Graph-based Verification

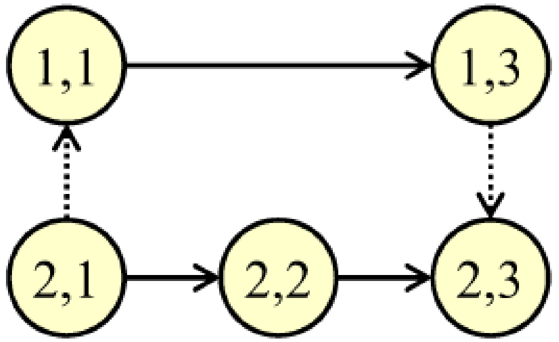
- One would expect deadlock to occur when there is a cycle in the timing conflict graph. But:

  <u>Having no cycle in G' or G does not guarantee deadlock-freeness</u>

- Deadlock can occur due to two parallel paths between same start and end vertices.
- Create an alternative graph to model deadlocks as cycles based on the timing conflict graph.



Ex. 1: V2 waits for V1 to pass Z1
V1 waits for V2 to pass Z3
Deadlock, no cycle



Ex. 2: V1 waits for V2 to pass Z1
V2 waits for V1 to pass Z3
No deadlock, V2 can move to Z2

Resource Conflict Graph (RCG)

- The basic idea is to combine edges of the conflict graph into vertices
- All Type-1 and Type-2 edges absorbed into vertices
- Each edge in the resource conflict graph is a Type-3 edge in the timing conflict graph
- At least one of the j-indices are equal across an edge

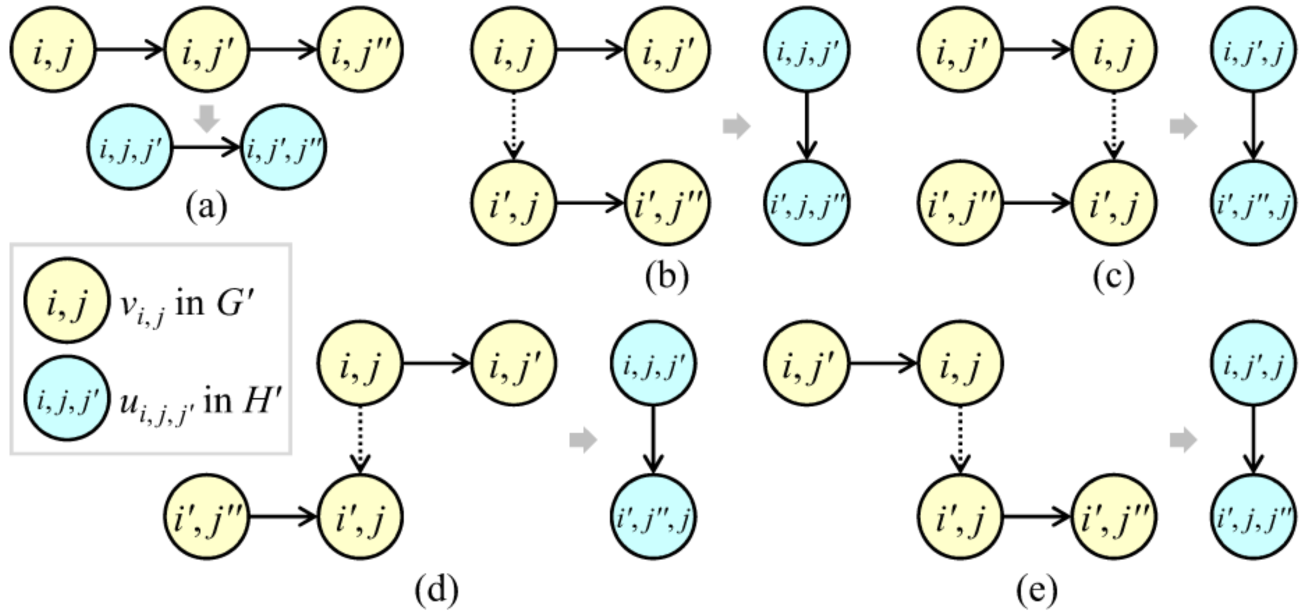
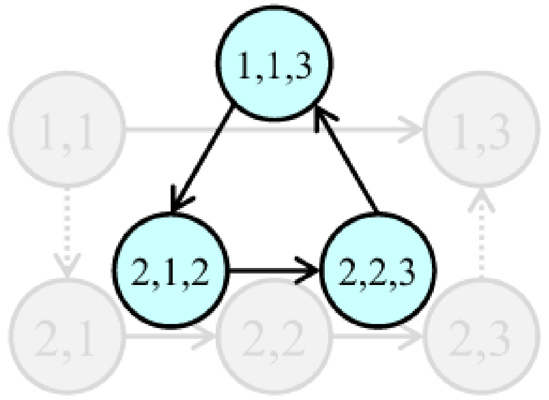


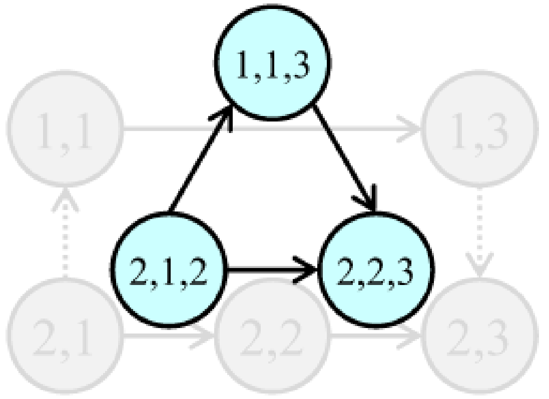
Fig. 6. The construction rules of resource conflict graphs.

Verifying Liveness

- An edge $(i_k, j_k, j'_k) \rightarrow (i_{k+1}, j_{k+1}, j'_{k+1})$ in RCG implies i_k must free up the common conflict zone before i_{k+1} arrives.
  - If there is a cycle in RCG, then there is a deadlock.
- If there is a deadlock, say i can't move from j to j', then there must be an edge to (i, j, j') in RCG
  - Can show using one of the construction rules
  - Repeatedly apply above statement to all vehicles in deadlock
  - Ultimately forms a cycle in RCG, since vehicles and zones are finite.
- So, to verify that acyclic subgraph has no deadlock – construct its resource conflict graph and check for cycles in it.



Ex. 1: Deadlock, cycle exists



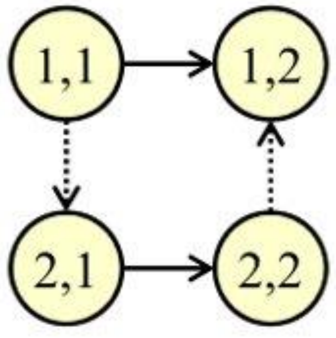
Ex. 2: No deadlock, no cycle

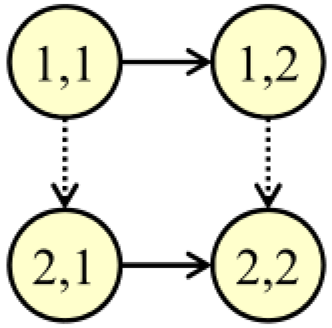
Petri Net Construction

With Deadlock

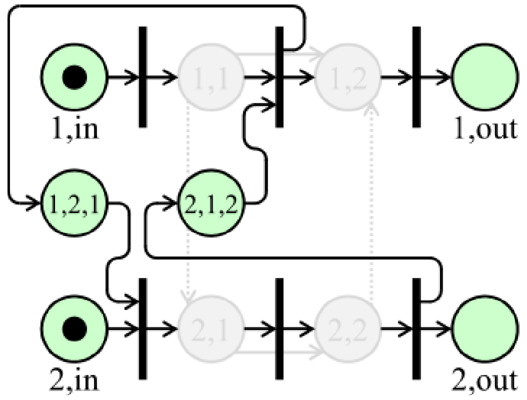
Without Deadlock

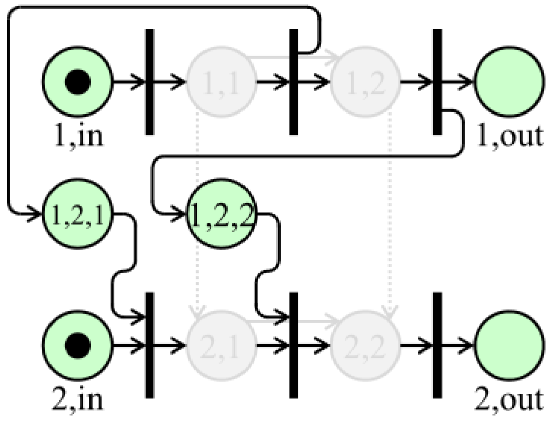
Acyclic TCG G'





Equivalent Petri-Net Π





Petri Net Verification

<u>The Petri net Π has a deadlock if and only if G' has a deadlock</u>

- If Π has a deadlock, at least one place $q_{i,i',j}$ never receives a token, which implies that Δ_i cannot leave j before $\Delta_{i'}$ enters j (so deadlock in G')
- If deadlock occurs in G' (suppose that some Δ_i can't go from j to j') it implies that $q_{i',i,j}$ will never receive a token (so deadlock in Π)
- So, to verify that acyclic subgraph has no deadlock – construct equivalent Petri Net and check it for deadlocks

Scheduling

- Naïve approach: first-come first-serve schedule
  - Ignores key interactions between vehicles and conflict zones
  - Introduces extra delays in many cases
- Generate a passing order for vehicles by constructing acyclic subgraph G' from conflict graph G with minimum total passing time.
  - Subgraph generated through cycle removal
- Naïve cycle removal: DFS traversal of the graph
  - May not always remove "good" edges to optimize objective
  - Can't remove some types of edges due to safety constraints

Cycle Removal-Based Scheduling

- We need to remove cycles while minimizing total passing time
  - Min. Spanning Tree – acyclic subgraph with minimum sum of edge weights
  - Iteratively remove max-cost edge whose removal doesn't disconnect graph
- Proposed algorithm is based on the above idea
  - Iteratively remove max-cost Type-3 edges without violating constraints
  - Ensuring liveness complicates the problem – deadlocks exist even in acyclic graphs, as shown earlier
  - Need to efficiently handle cases where max-cost edge cannot be removed
    - Backtracking and redoing is computationally expensive – equivalent to brute force

Edge and Vertex States

Edge State: For an edge e,

ON - e is included in G'

OFF – e has been removed from G'

UNDECIDED – Will decide ON/OFF in current subproblem

DONTCARE – e not included in current subproblem

- All Type-1 and Type-2 edges always ON

Vertex State: For a vertex v,

BLACK – Entering time scheduled

GRAY – Entering time depends on Type-3 edges only

WHITE – Entering time can depend on any type of edges

- If any outgoing edge is ON, v is BLACK
- If v is BLACK, all edges through it must be ON/OFF
- If v is GRAY, v' must be BLACK if (v', v) is not a Type-3 edge

Vertex Entering Time

- Δ_i can't enter j before all earlier vehicles $\Delta_{i'}$ have passed

$$\max\{s_{i',j} + p_{i',j} + w_k\}$$

- Additionally, need to wait for $\Delta_{i'}$ to move to next zone j'

$$\max\{s_{i',j'} - w_{k'} + w_k\}$$

- Entering time is max of above two quantities – need to fulfill both
- For the first conflict zone on Δ_i's path, also depends on arrival time a_i
- For v, depends on the earlier vertices u where (u,v) is an edge of G'
  - Since G' is acyclic, compute in topological order

Vertex Slack

- Maximum delay that can be added at vertex without changing the maximum leaving time t_{max} (i.e. the optimization objective)
- For the last vertex on the path of the last vehicle $v_{i,j'}$

$$t_{max} – (s_{i,j'} + p_{i,j'})$$

- For other vertices u, it is minimum of slack of all reachable vertices v where (u,v) edge in G'
- Compute reverse topologically for acyclic graph

Defining the Cost of an Edge

Edge Cost: Delay incurred in t_{max} due to adding this edge in G'

Only need to look at cost of Type-3 edges, $e_k = (v_{i,j}, v_{i',j})$

$$\text{cost}[e_k] = (s_{i,j} + p_{i,j} + w_k) - s_{i',j} - \text{slack}[v_{i',j}]$$

$(s_{i,j} + p_{i,j} + w_k)$ and $s_{i',j}$ are start times for $v_{i',j}$ with & without e_k

Compare with slack at $v_{i',j}$ to determine effect of e_k on t_{max}

If the cost is positive, t_{max} will increase.

But if cost is negative, t_{max} won't change.

Removal of Type-3 Edges

- Initialization
  - Include Type-1 and Type-2 edges in G', set their states to ON
  - Compute vertex entering times on G', leaving time of last vehicle as t_{max}
  - Set Type-3 states to UNDECIDED, compute vertex slacks.

- Identify candidate edges for removal
  - Leader vertex – $v_{i,j}$ where i is first vehicle on source lane, j is first conflict zone
  - Candidate edges – UNDECIDED Type-3 edges with one vertex as a leader vertex
  - Compute cost of these edges, and try to remove in decreasing order of cost

Ensuring Deadlock-Freeness

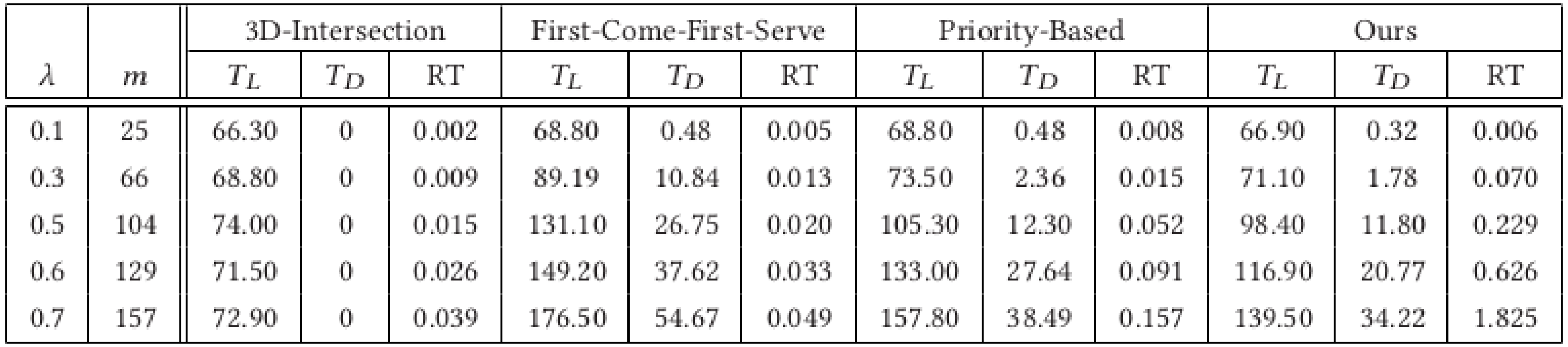
- Type-3 edges always in pairs – exactly one of two must be included
- Remove one and verify deadlock-freeness - if it fails, swap the edges
  - Use edge state variables to temporarily remove an edge
- If G' is deadlock-free, recompute vertex entering times and slacks
  - Identify newly set GRAY vertices as leader vertices, and repeat
- If G' is not deadlock-free, need to re-evaluate entire assignment till e_k
  - Backtracking is expensive – divide into subproblems
  - Schedule the first half of the vehicles arranged in increasing arrival time
    - For Type-3 edges between the two halves, assume first half passes before second half
    - Use the schedule of the first half while solving the second subproblem

Proof of Correctness and Time Complexity

- Type-1 and Type-2 edges included in G' by default.
- Exactly one Type-3 edge is selected out of every pair
- Deadlock-freeness is verified on removing each Type-3 edge
- For the solution obtained by recursively dividing into subproblems
  - No deadlock while merging both halves – we assume first passes before second
  - Each subproblem is essentially applying the same algorithm on a smaller set
  - Base case – only one vehicle: no Type-3 edges, so G' is feasible here
- Hence algorithm provably generates acyclic and deadlock-free G' <u>always</u>
- Time complexity of scheduling algorithm: $O(E^2 \log V)$

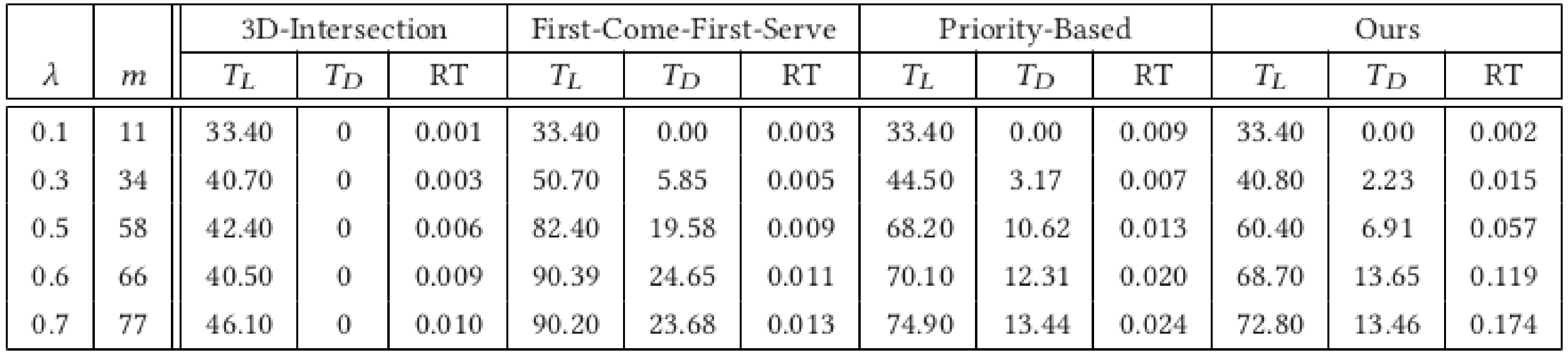
Results

For maximum a_i equal to 60 seconds



λ	m	3D-Intersection			First-Come-First-Serve			Priority-Based			Ours		
		T_L	T_D	RT	T_L	T_D	RT	T_L	T_D	RT	T_L	T_D	RT
0.1	25	66.30	0	0.002	68.80	0.48	0.005	68.80	0.48	0.008	66.90	0.32	0.006
0.3	66	68.80	0	0.009	89.19	10.84	0.013	73.50	2.36	0.015	71.10	1.78	0.070
0.5	104	74.00	0	0.015	131.10	26.75	0.020	105.30	12.30	0.052	98.40	11.80	0.229
0.6	129	71.50	0	0.026	149.20	37.62	0.033	133.00	27.64	0.091	116.90	20.77	0.626
0.7	157	72.90	0	0.039	176.50	54.67	0.049	157.80	38.49	0.157	139.50	34.22	1.825

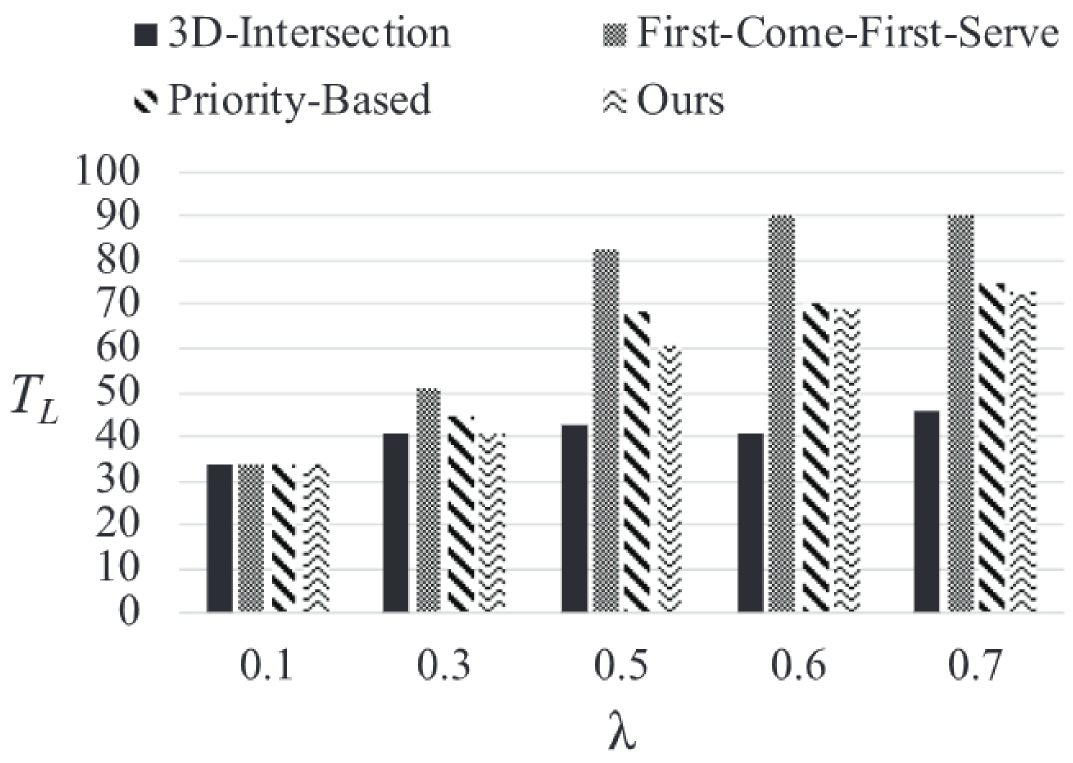
For maximum a_i equal to 30 seconds

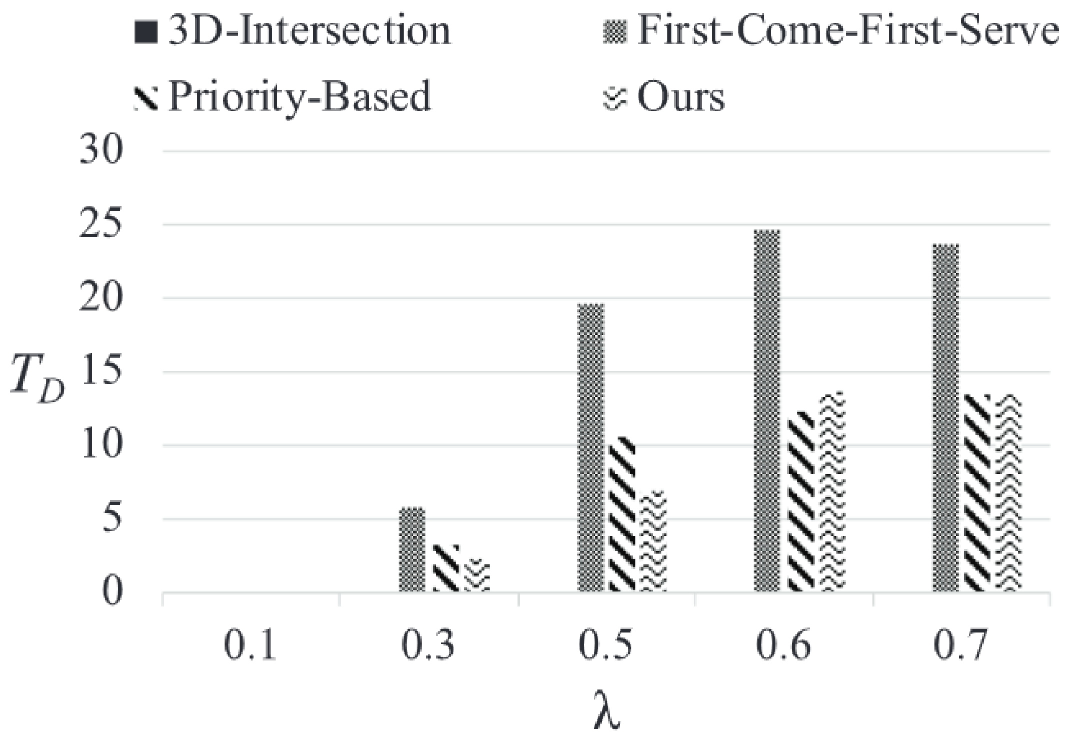


λ	m	3D-Intersection			First-Come-First-Serve			Priority-Based			Ours		
		T_L	T_D	RT	T_L	T_D	RT	T_L	T_D	RT	T_L	T_D	RT
0.1	11	33.40	0	0.001	33.40	0.00	0.003	33.40	0.00	0.009	33.40	0.00	0.002
0.3	34	40.70	0	0.003	50.70	5.85	0.005	44.50	3.17	0.007	40.80	2.23	0.015
0.5	58	42.40	0	0.006	82.40	19.58	0.009	68.20	10.62	0.013	60.40	6.91	0.057
0.6	66	40.50	0	0.009	90.39	24.65	0.011	70.10	12.31	0.020	68.70	13.65	0.119
0.7	77	46.10	0	0.010	90.20	23.68	0.013	74.90	13.44	0.024	72.80	13.46	0.174

Results

Graph of T_D and T_L for various algorithms, with maximum a_i = 30 seconds

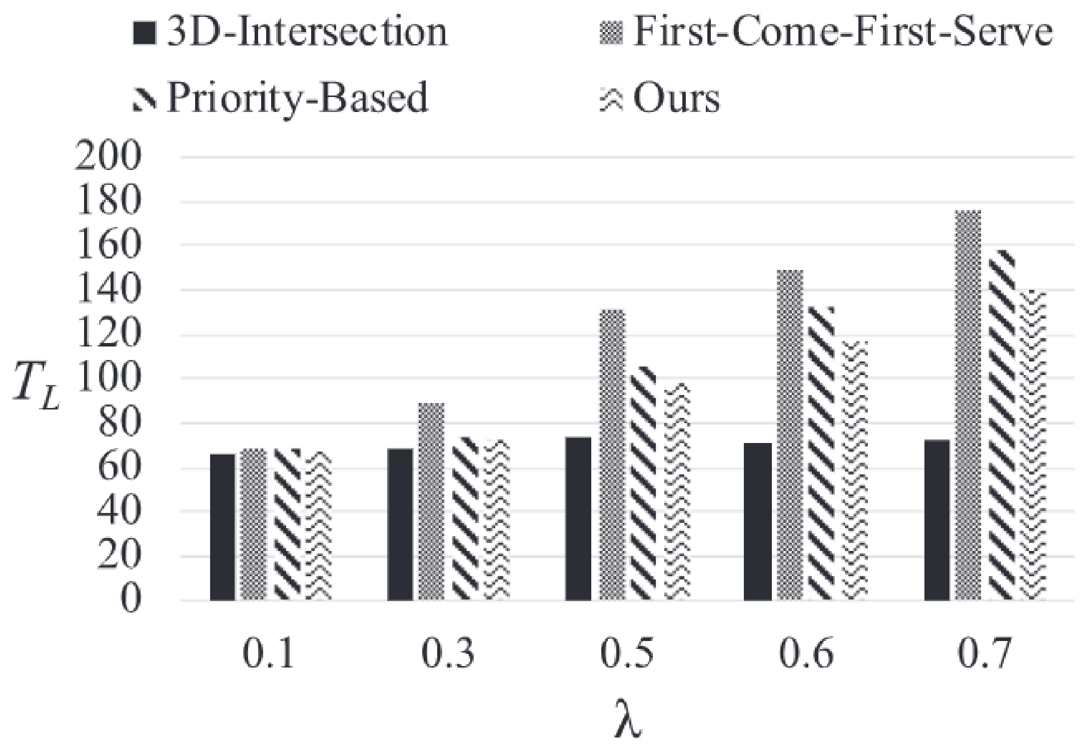


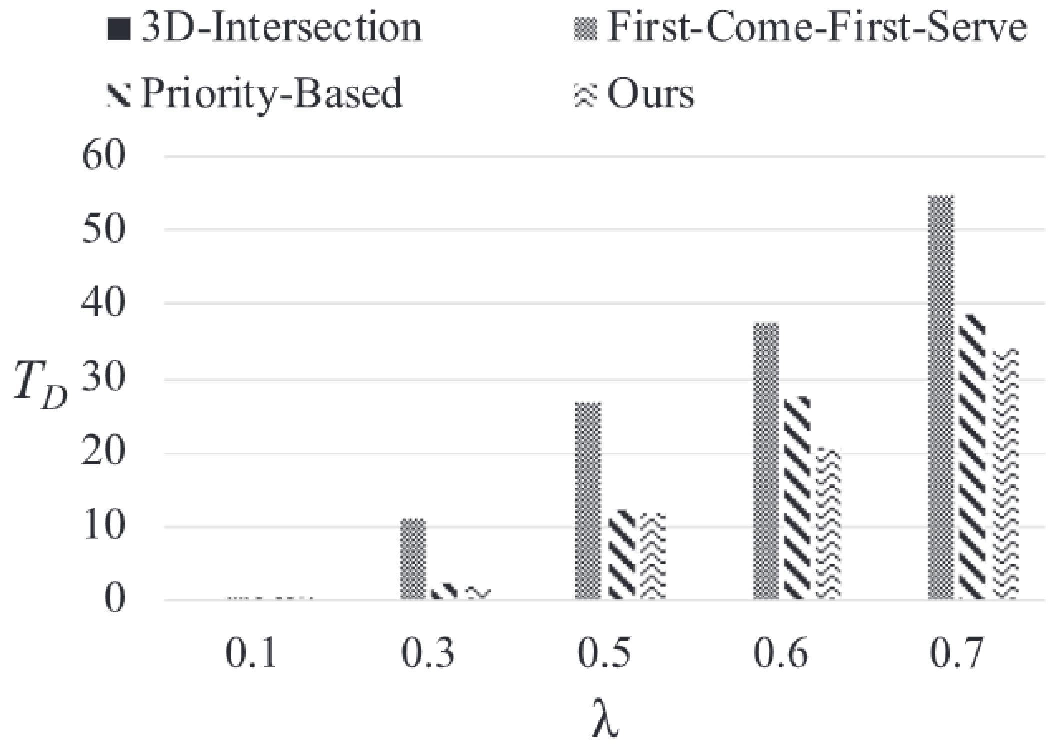


Experiments were run by the authors on a macOS Mojave notebook with 2.3 GHz Intel CPU and 8 GB memory.

Results

Graph of T_D and T_L for various algorithms, with maximum a_i = 60 seconds

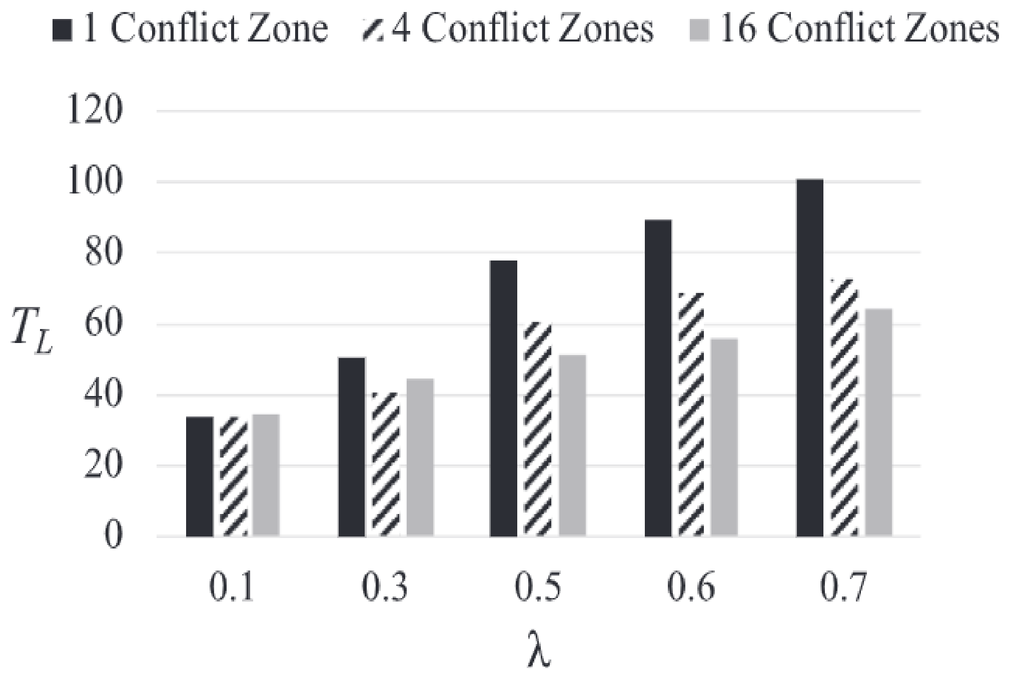


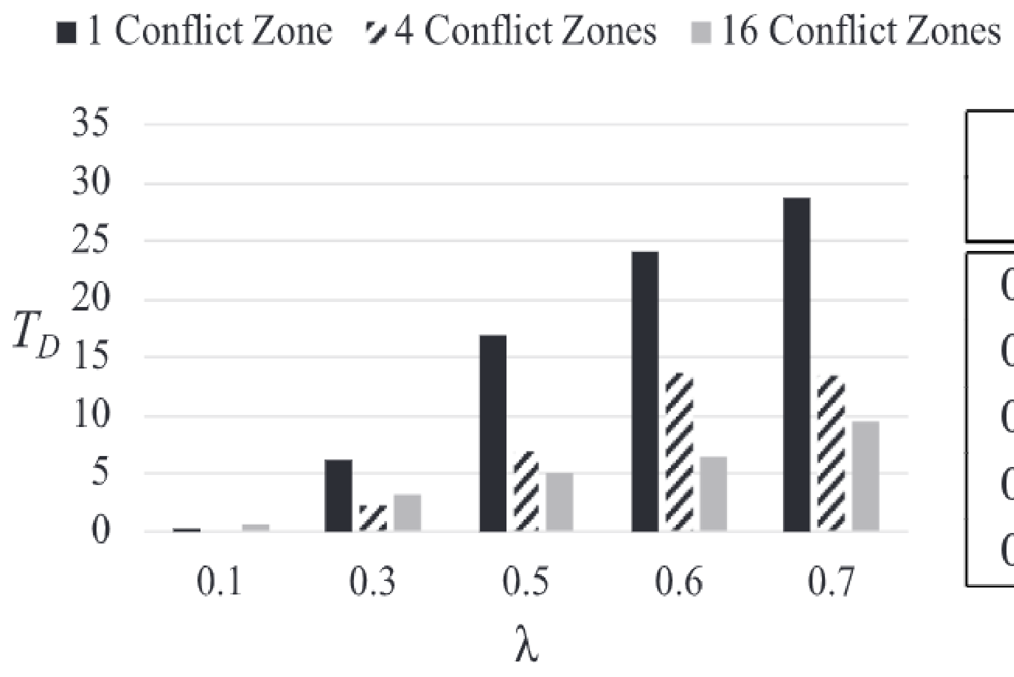


Experiments were run by the authors on a macOS Mojave notebook with 2.3 GHz Intel CPU and 8 GB memory.

Results

Graph of T_D and T_L for different number of conflict zones





λ	m
0.1	11
0.3	34
0.5	58
0.6	66
0.7	77

Experiments were run by the authors on a macOS Mojave notebook with 2.3 GHz Intel CPU and 8 GB memory.

Our Implementation

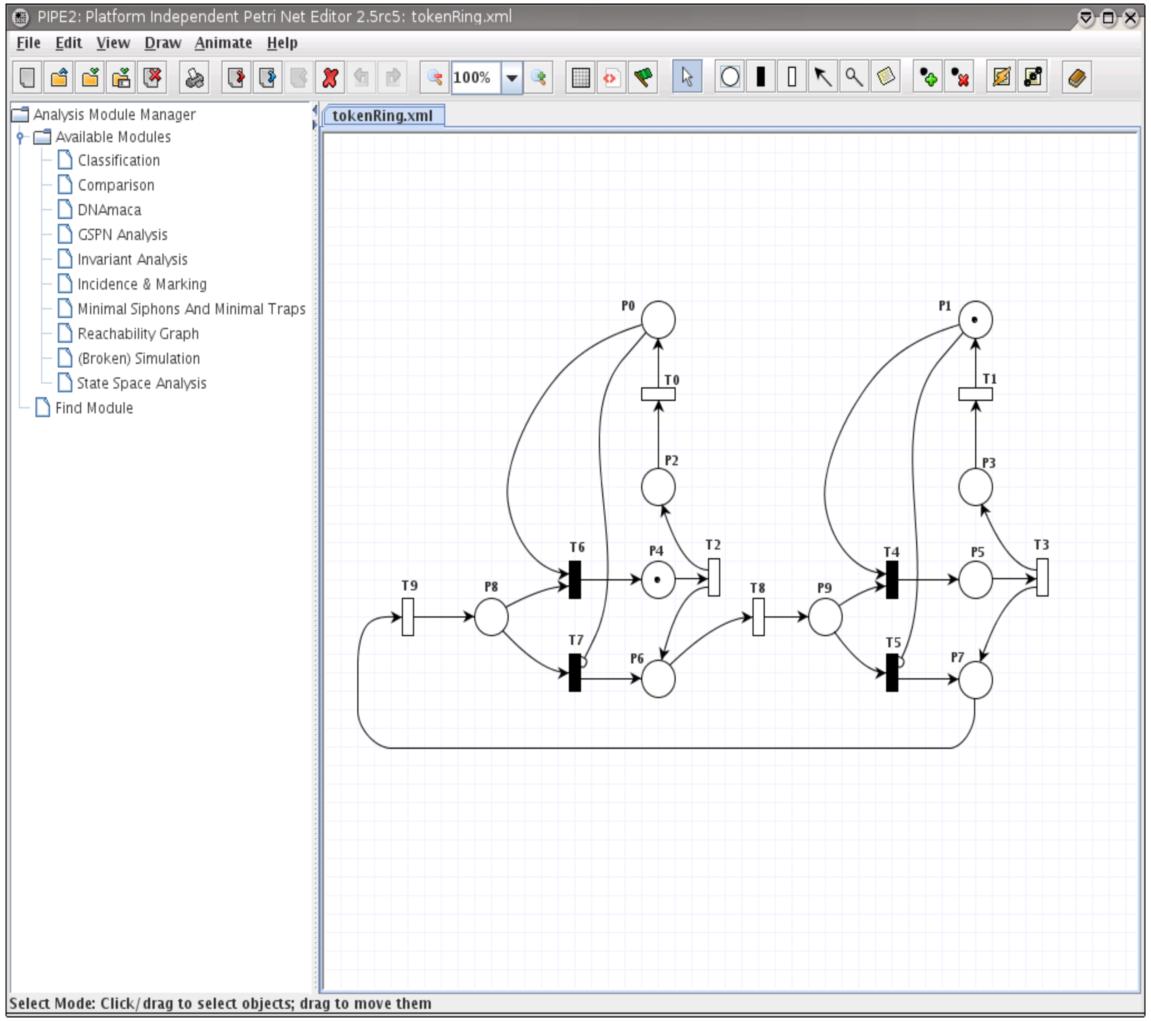
https://github.com/ashwin2802/CS637

- Have implemented the algorithm as well as the simulation aspect of it, using C/C++
- Random traffic generator, intersections with 1-16 conflict zones, TCG graph generator, scheduler and deadlock checker (using RCG and Petri-Net).
- Traffic and Intersection generator, TCG graph generator and deadlock checking work
- Were unable to resolve some logical errors in the code – a certain step in the algorithm hasn’t been detailed in the paper
- Also can visualize the order of vehicle passing using SUMO simulator

PIPE

http://pipe2.sourceforge.net/

- Paper authors have suggested using Platform Independent Petri-net Editor (PIPE2)
- A GUI tool for easily modelling and visualizing Petri-nets and doing reachability analysis
- However, documentation is scarce and no way to interface it with code

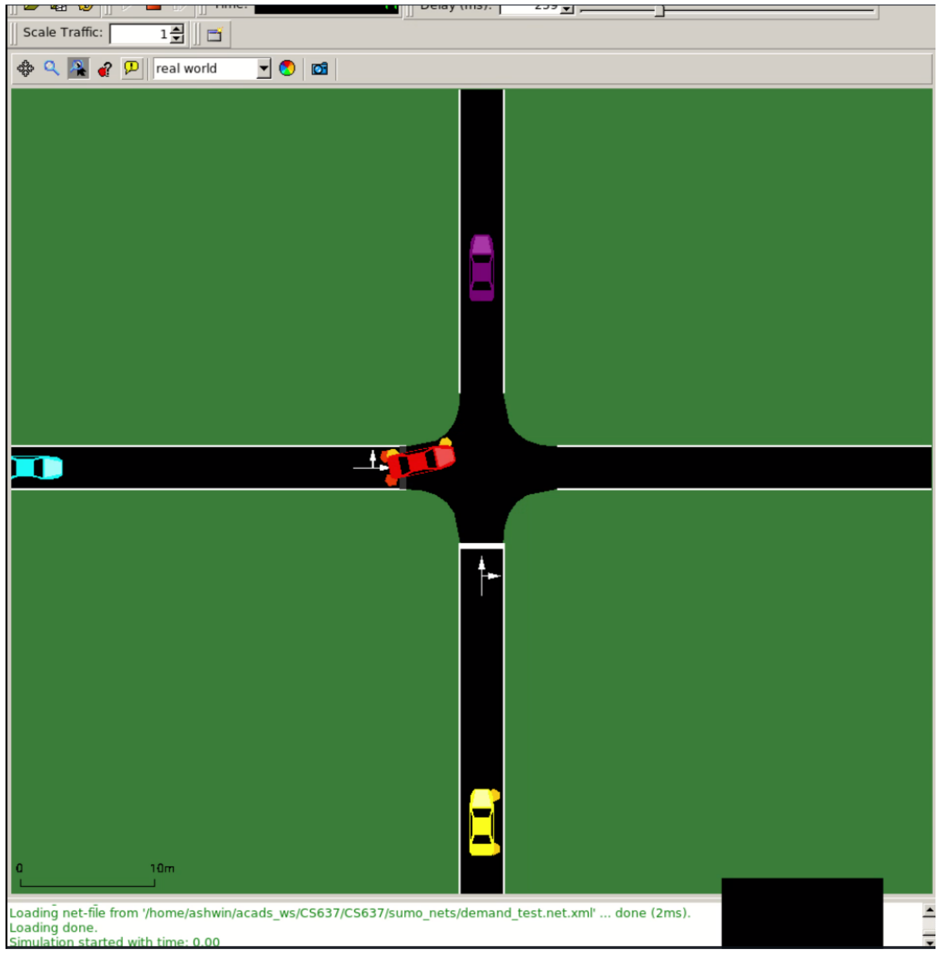

PIPE2: Platform Independent Petri Net Editor 2.5rc5: tokenRing.xml

Select Mode: Click/drag to select objects; drag to move them

SUMO Simulation of Urban Mobility

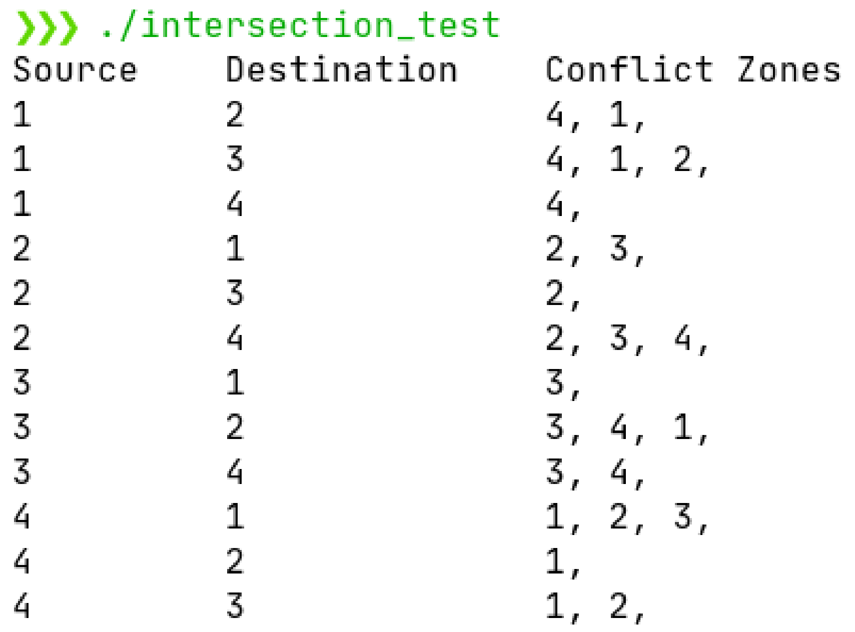
- Continuous traffic simulation package
- Completely open-source, highly portable
- Some features
  - Simulation of public transport
  - Simulation of logistics, individual people, trip chains
  - Optimal Path Routing, pedestrian traffic modeling
  - Bicycle, waterway and railway simulations

https://www.eclipse.org/sumo/



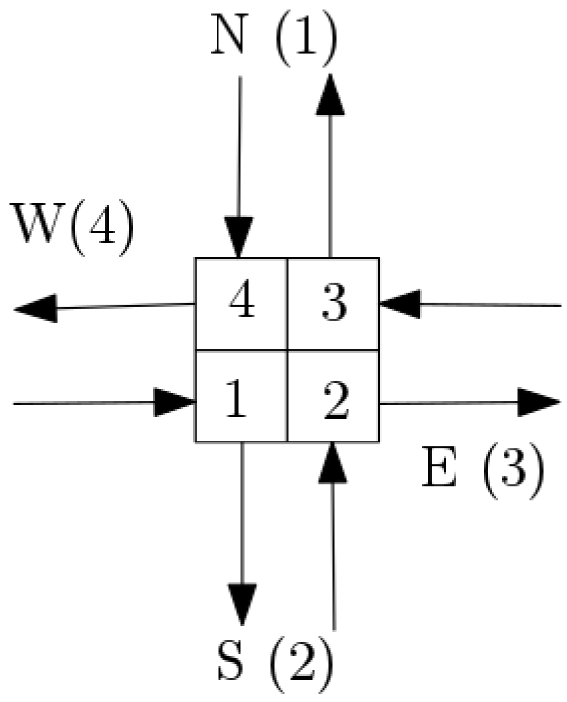
```
Loading net-file from '/home/ashwin/acads_ws/CS637/CS637/sumo_nets/demand_test.net.xml' ... done (2ms).
Loading done.
Simulation started with time: 0.00
```

Code Demos - Modelling

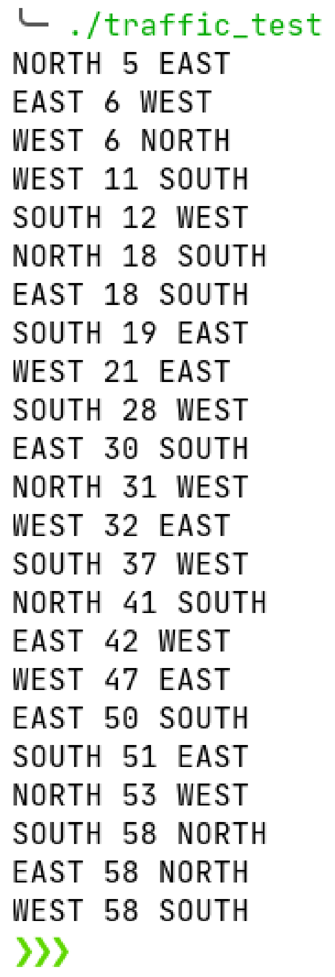


```
❯❯❯ ./intersection_test
Source      Destination     Conflict Zones
1           2               4, 1,
1           3               4, 1, 2,
1           4               4,
2           1               2, 3,
2           3               2,
2           4               2, 3, 4,
3           1               3,
3           2               3, 4, 1,
3           4               3, 4,
4           1               1, 2, 3,
4           2               1,
4           3               1, 2,
```

Intersection Model for 4 conflict zones



Lane Convention

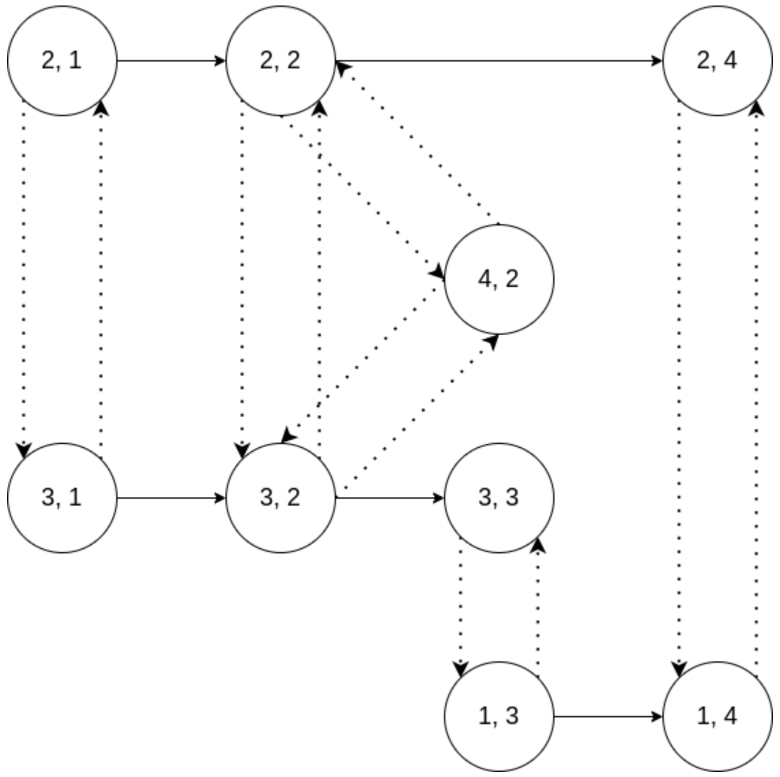


```
╰─ ./traffic_test
NORTH 5 EAST
EAST 6 WEST
WEST 6 NORTH
WEST 11 SOUTH
SOUTH 12 WEST
NORTH 18 SOUTH
EAST 18 SOUTH
SOUTH 19 EAST
WEST 21 EAST
SOUTH 28 WEST
EAST 30 SOUTH
NORTH 31 WEST
WEST 32 EAST
SOUTH 37 WEST
NORTH 41 SOUTH
EAST 42 WEST
WEST 47 EAST
EAST 50 SOUTH
SOUTH 51 EAST
NORTH 53 WEST
SOUTH 58 NORTH
EAST 58 NORTH
WEST 58 SOUTH
❯❯❯
```

Sample Generated Traffic Distribution

Code Demos - Modelling

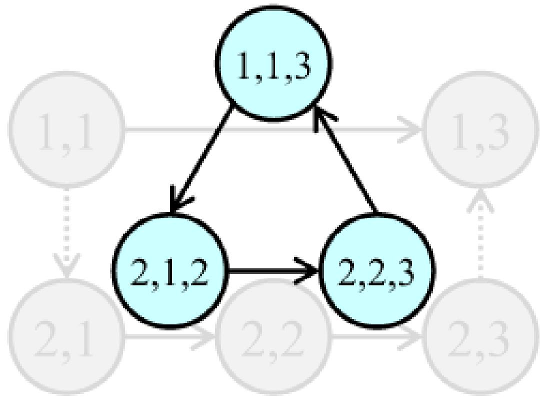
An example Timing Conflict Graph



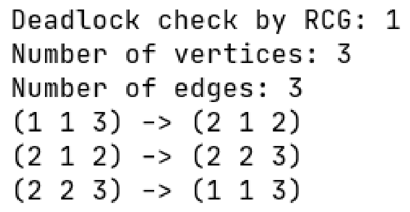
```
❯❯❯ ./tcg_test
EAST 8 WEST
NORTH 9 EAST
WEST 10 NORTH
SOUTH 11 EAST
Number of vertices: 9
Number of edges: 17
(1 3): First i? 1 First j? 1 Start time -: 8
(1 4): First i? 1 First j? 0 Start time -: 8
(2 1): First i? 1 First j? 0 Start time -: 9
(2 2): First i? 1 First j? 0 Start time -: 9
(2 4): First i? 1 First j? 1 Start time -: 9
(3 1): First i? 1 First j? 1 Start time -: 10
(3 2): First i? 1 First j? 0 Start time -: 10
(3 3): First i? 1 First j? 0 Start time -: 10
(4 2): First i? 1 First j? 1 Start time -: 11

(1 3) -> (1 4): 1 -> (3 3): 3
(1 4) -> (2 4): 3
(2 1) -> (2 2): 1 -> (3 1): 3
(2 2) -> (3 2): 3 -> (4 2): 3
(2 4) -> (2 1): 1 -> (1 4): 3
(3 1) -> (3 2): 1 -> (2 1): 3
(3 2) -> (3 3): 1 -> (2 2): 3 -> (4 2): 3
(3 3) -> (1 3): 3
(4 2) -> (2 2): 3 -> (3 2): 3
```

Code Demos - Verification

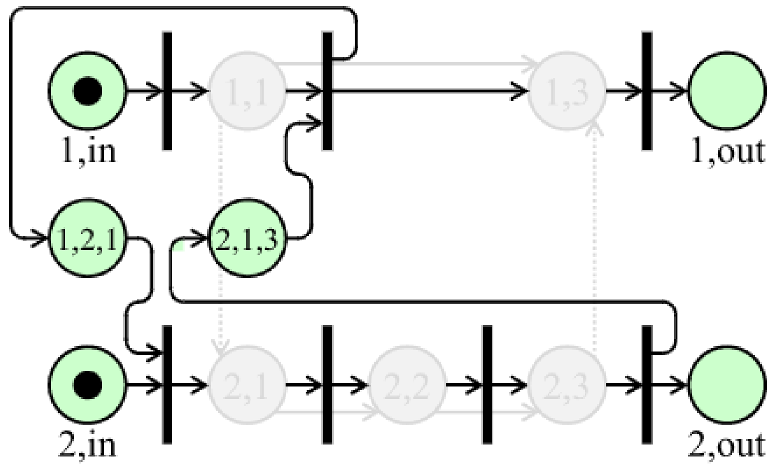


Test TCG and expected RCG

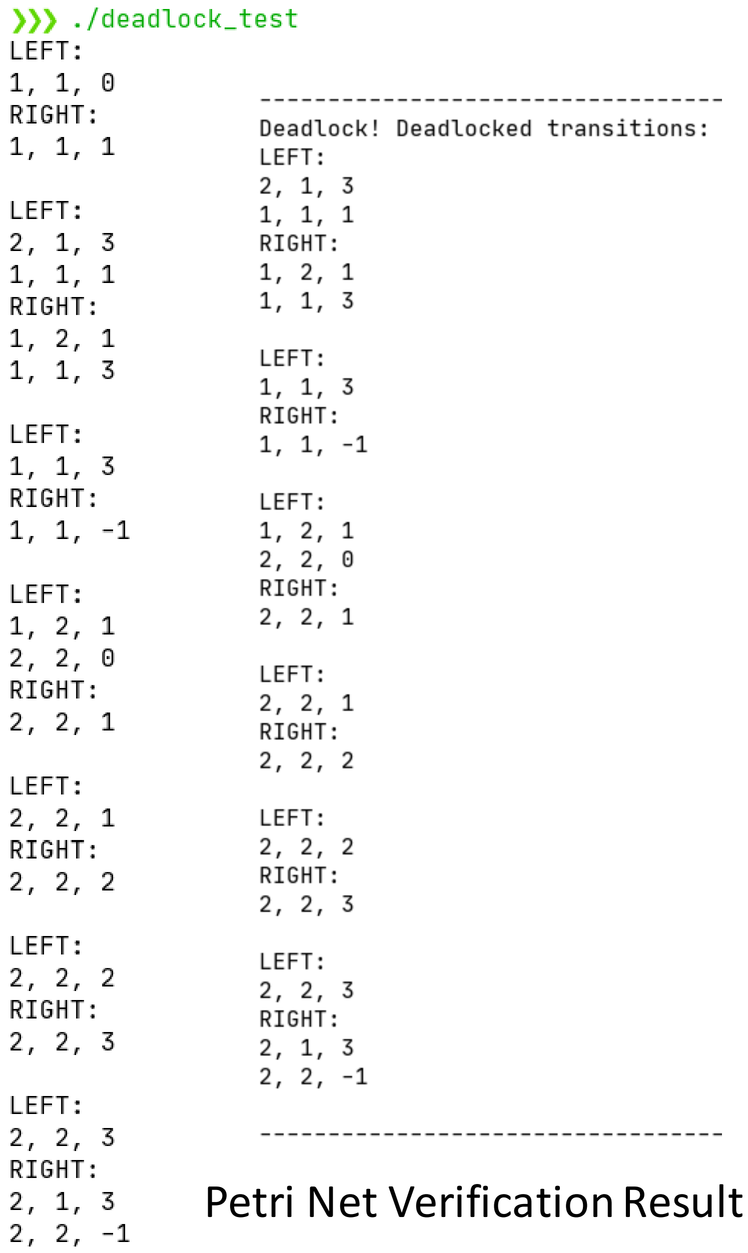


```
Deadlock check by RCG: 1
Number of vertices: 3
Number of edges: 3
(1 1 3) -> (2 1 2)
(2 1 2) -> (2 2 3)
(2 2 3) -> (1 1 3)
```

RCG Verification Result



Expected Petri Net Model



```
>>> ./deadlock_test
LEFT:
1, 1, 0
RIGHT:
1, 1, 1

LEFT:
2, 1, 3
1, 1, 1
RIGHT:
1, 2, 1
1, 1, 3

LEFT:
1, 1, 3
RIGHT:
1, 1, -1

LEFT:
1, 2, 1
2, 2, 0
RIGHT:
2, 2, 1

LEFT:
2, 2, 1
RIGHT:
2, 2, 2

LEFT:
2, 2, 2
RIGHT:
2, 2, 3

LEFT:
2, 2, 3
RIGHT:
2, 1, 3
2, 2, -1
```

```
--------------------------------
Deadlock! Deadlocked transitions:
LEFT:
2, 1, 3
1, 1, 1
RIGHT:
1, 2, 1
1, 1, 3

LEFT:
1, 1, 3
RIGHT:
1, 1, -1

LEFT:
1, 2, 1
2, 2, 0
RIGHT:
2, 2, 1

LEFT:
2, 2, 1
RIGHT:
2, 2, 2

LEFT:
2, 2, 2
RIGHT:
2, 2, 3

LEFT:
2, 2, 3
RIGHT:
2, 1, 3
2, 2, -1

--------------------------------
```

Petri Net Verification Result

SUMO Simulation

- Experimented with manually using SUMO
- Wrote code to control it through Traffic Control Interface(TraCI) using TCP.
- Currently doesn't work as intended, vehicles don't follow proper schedule
- Code is on the experimental_sumo branch of our Github repository
- First generates proper configuration files.
- Then runs a modified version of our original Petri-Net simulator in the background
- Updates the speed of the vehicles at each time step accordingly and communicates it to the GUI.

Conclusions

- Presents a very general graph-based model for intersection management
- Can be applied to other types of intersections as well – the core concept is modelling the conflict zones discretely
- Presents an effective realtime scheduling algorithm based on cycle removal
- Pretty lightweight once the requisite data is available to the code
- Presents formal verification approaches for deadlock-freeness
- Challenges faced in implementation on an embedded system aren't taken into account

Future Avenues

- Dynamic scheduling – scheduling vehicles as they arrive instead of in discrete intervals could reduce delays from chunk transitions
- Complex intersection topologies – designing algorithms to create conflict zone models out of any given intersection area
- Modelling dynamics – allowing vehicles to accelerate and decelerate instead of assuming constant speed will yield more realistic results
- Fault-proof scheduling – adversarial attacks on the intersection manager and issues arising from communication problems must be considered to ensure vehicles never receive an unsafe version of a verified schedule
- UAV Traffic Management – applying the same approach to manage a swarm of UAVs flying through a common airspace

Work Related to Future Avenues

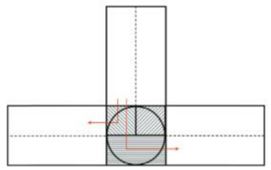
Some interesting papers we read related to the future avenues discussed below are briefly described here:

Multi-Agent Path Finding for UAV Traffic Management [1]

- Models intersection management for UAVs as a conflict resolution problem
- Finding multiple paths on a graph satisfying safety constraints – MAPF solving
- Uses a similar batch processing approach – vehicles are scheduled in chunks
- However MAPF solving is NP-hard – computationally expensive in comparison

Intersection Auctions and Reservation-Based Control in Dynamic Traffic Assignment [2]

- Discusses Dynamic Traffic Assignment as a flow problem on a graph
- Tile-based reservation – vehicles place bids for conflicting tiles when they reach the front of their lane, based on the delay incurred if they don't get that tile instantly
- Intersection zones are divided radially instead of linearly



Work Done Post Presentation

- Worked extensively on resolving the various issues in our implementation
- Experimented with visualizing solution schedules in SUMO
- Read some more papers related to future work avenues
- Added some more details to the presentation – demo screenshots and a few minor improvements

Individual Member Contributions

- We have closely collaborated on the entire project. Each aspect was thoroughly discussed, edited and finalized with matching effort.
- However, for the sake of distribution:

<u>Ashwin Shenai</u>

- Slides: 3-5, 11-14, 17-25, 28-29, 33-35, 37-40
- Code:
  - Calculation of Edge costs, Vertex Slacks
  - Cycle-removal based scheduler
  - Construction of and verification using Resource Conflict Graphs
  - SUMO simulation (before presentation)

<u>Kshitij Kabeer</u>

- Slides: 2, 6-10, 15-16, 26-27, 30-32, 36, 41
- Code:
  - Intersection and Random Traffic Generation
  - Construction of Timing Conflict Graph
  - Petri-net based Verification
  - SUMO simulation (after presentation)

References

<u>Project Paper:</u>

Yi-Ting Lin, Hsiang Hsu, Shang-Chien Lin, Chung-Wei Lin, Iris Hui-Ru Jiang, and Changliu Liu. 2019. Graph-Based Modeling, Scheduling, and Verification for Intersection Management of Intelligent Vehicles. *ACM Trans. Embed. Comput. Syst. 18, 5s, Article 95 (October 2019), 21 pages.*

DOI: https://doi.org/10.1145/3358221

<u>Related Work:</u>

References in Related Work section (Slides 3-5) of the presentation refer to accordingly numbered citations of the above paper.

References

Work Related to Future Avenues:

[1] Florence Ho, Ana Salta, Ruben Geraldes, Artur Goncalves, Marc Cavazza, and Helmut Prendinger. 2019. Multi-Agent Path Finding for UAV Traffic Management. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17,2019,* IFAAMAS, 9 pages

Link: http://www.ifaamas.org/Proceedings/aamas2019/pdfs/p131.pdf

[2] Levin MW, Boyles SD. Intersection Auctions and Reservation-Based Control in Dynamic Traffic Assignment. *Transportation Research Record*. 2015;2497(1):35-44.

DOI:10.3141/2497-04