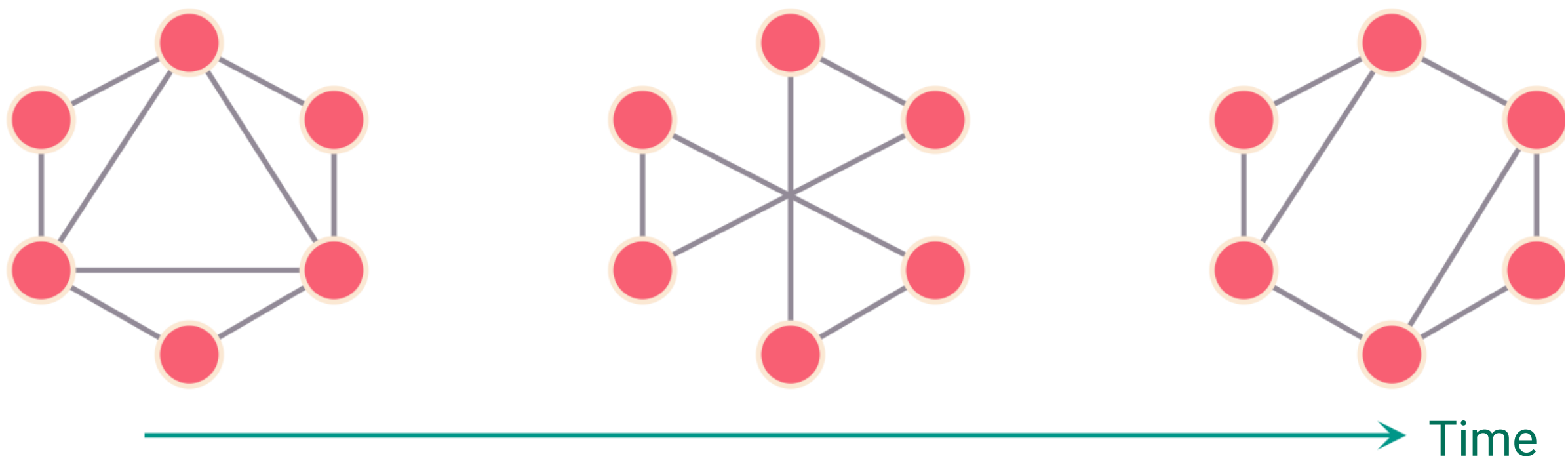


# Path Planning for Dynamic Graphs using A\* on GPU

## Abstract

- A\* is one of the widely used path planning algorithms applied in a diverse set of problems in robotics and video games.
- Zhou and Zeng [1] proposed a parallel variant of A\* for GPU, which keeps multiple priority queues to find the optimal path in a static graph ( static A\*).
- Here we present A\* for dynamic graphs (dynamic A\*) on GP-GPUs which achieves 2x-7x speedup than static A\* on the SNAP dataset [2]

## Dynamic Graphs



## Dynamic A\*: Insertion of edges

- Newly added edges can alter the optimal path.
- To find the new optimal path instead of executing A\* from scratch, we propagate the change to the affected nodes of the graph.

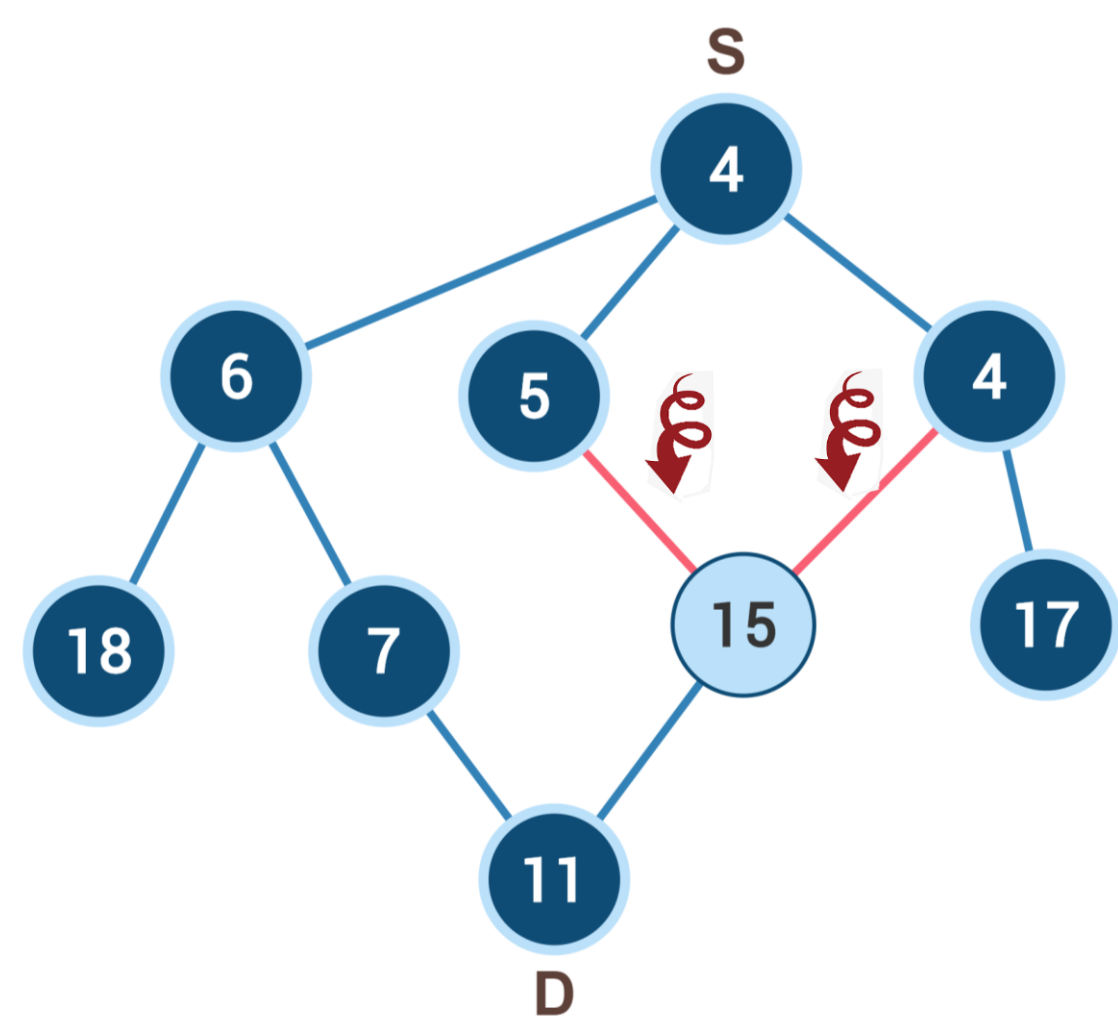


Fig 1: Two threads processing newly added edge ( shown in red )

### Pseudocode<sup>1</sup>

- For edges(u, v) inserted, add node v to update\_list, if  $f(v)_{new} < f(v)_{old}$ .
- While update\_list not empty:
  - Extract node n from update\_list.
  - For each child of n:
    - lock(child)**
    - if  $f(child)_{new} < f(child)_{old}$ , add child to update\_list.
    - unlock(child)**

$f(v)$ : cost from source to v + heuristics value of v

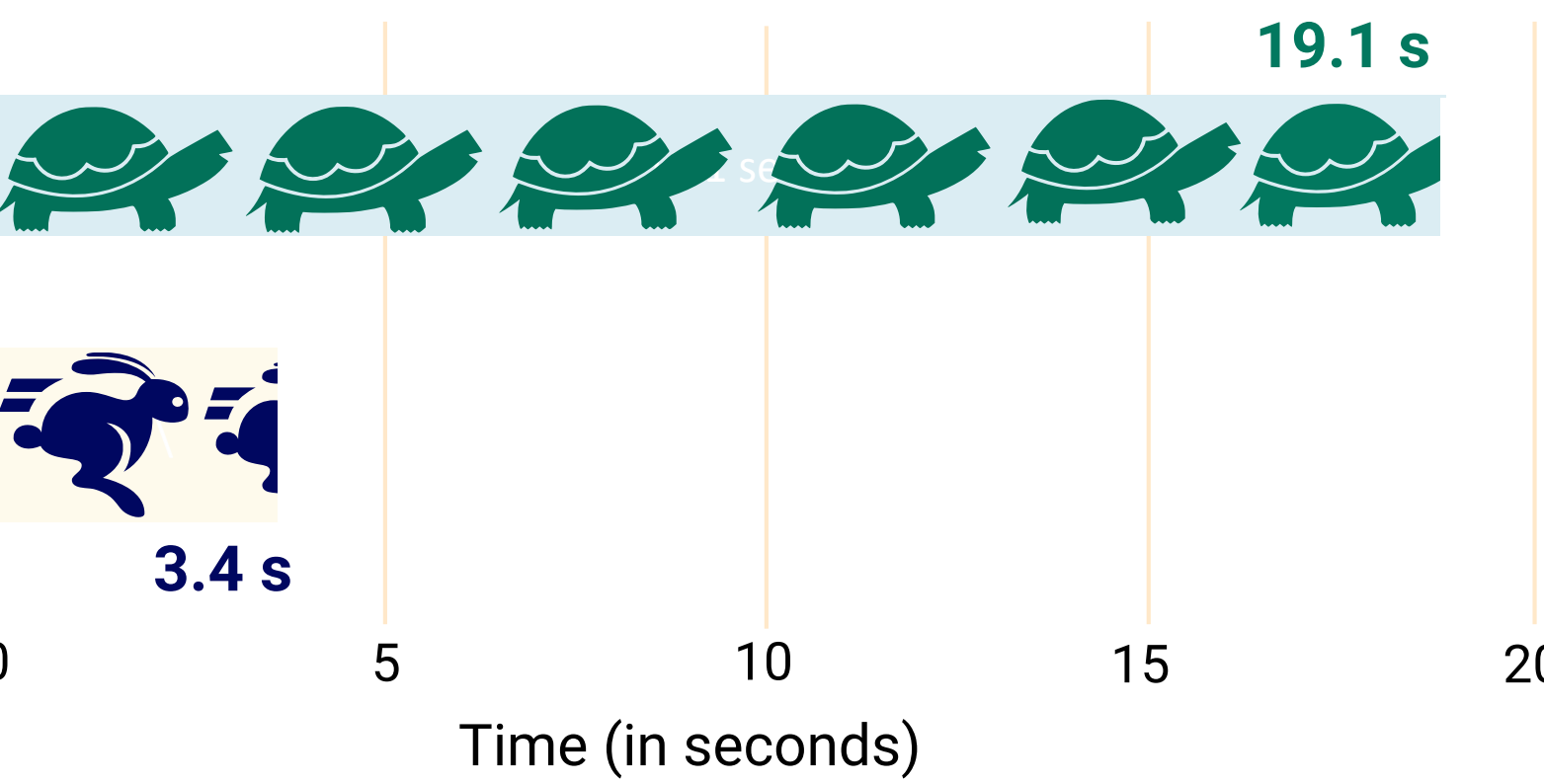
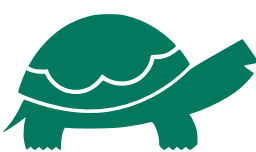


Fig 2: Execution time of static A\* and dynamic A\* (for only insertions) on graph Wiki-Talk



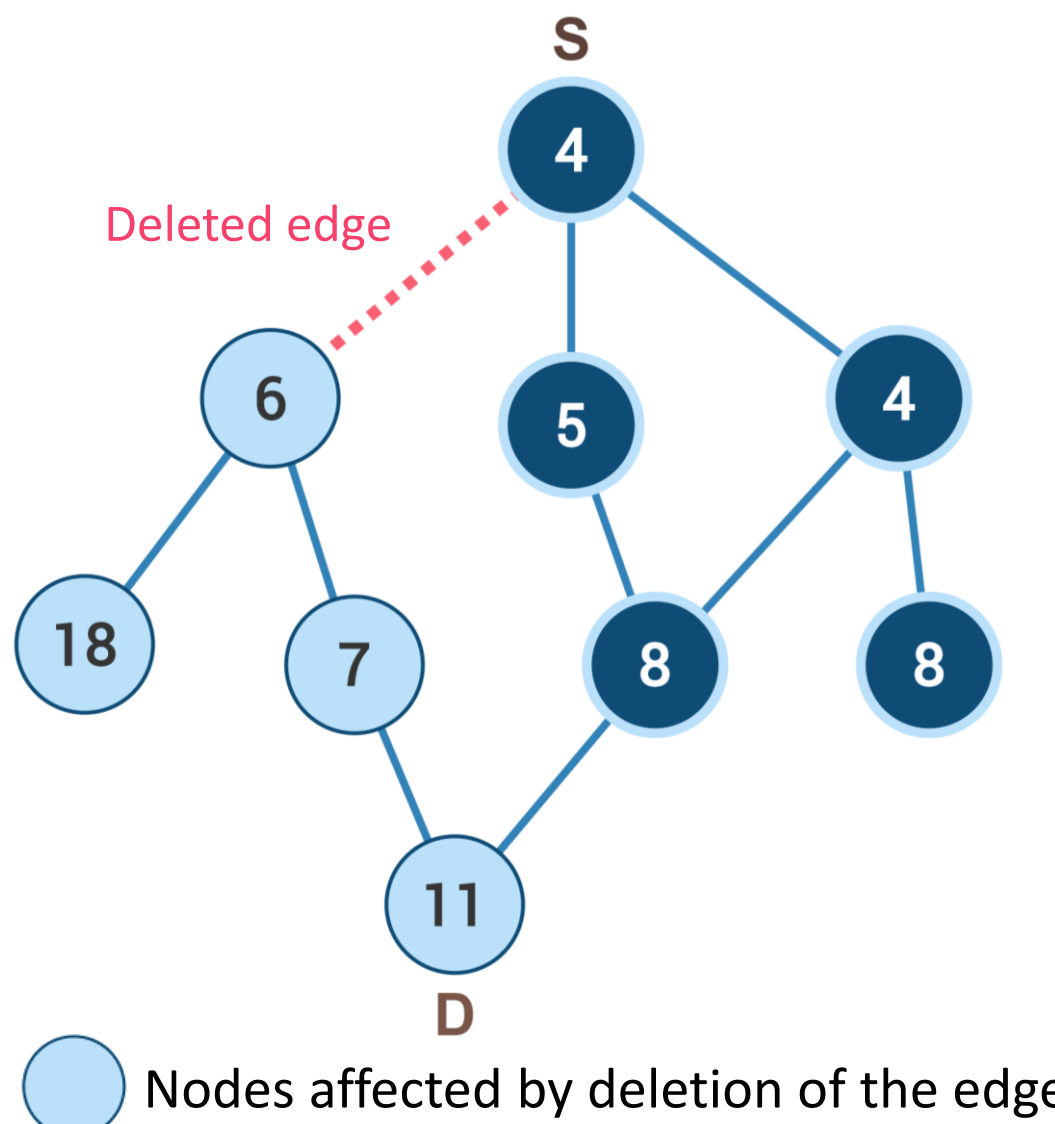
Static A\*



Dynamic A\*  
( Insertions )

## Dynamic A\*: Deletion of edges

- Deleting only that edge which belongs to the optimal path can create a new optimal path.
- For all such affected nodes recompute the cost and select the neighbour with the least cost.
- Propagate the updated cost to all the affected nodes.



Nodes affected by deletion of the edge

### Pseudocode<sup>1</sup>

- For each deleted edge  $u \rightarrow v$ :
  - Compute  $f(v)$  from the neighbours of v.
  - Add v to update\_list.
- While update\_list is not empty:
  - Extract node n from update\_list.
  - For each child of n such that  $\text{optimal\_parent}(\text{child}) = n$ :
    - If  $f(\text{child}) > f(\text{child})_{new}$  then compute cost of child from each of its neighbour and add it to update\_list.

$\text{optimal\_parent}(v)$ : neighbour of node v with least  $f(v)$

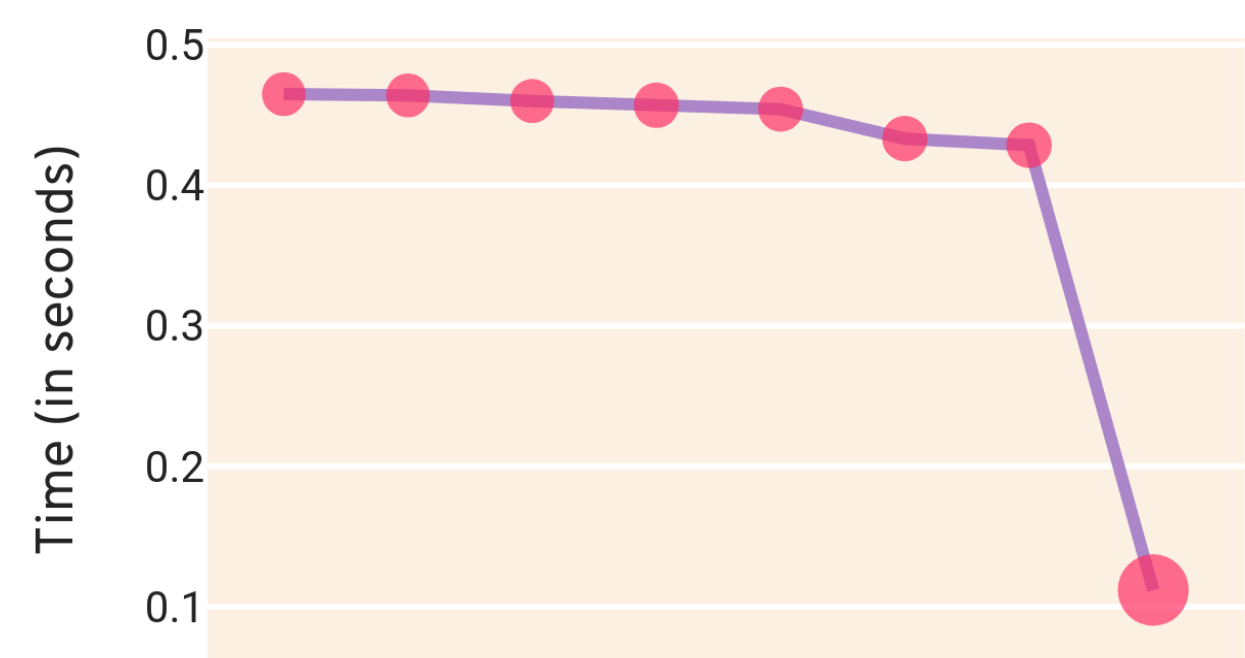
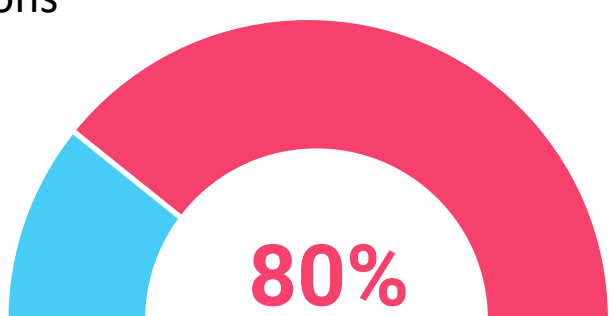


Fig 3: Execution time vs composition of updates

Insertions  
Deletions



80% of execution time used in processing deletions

## Dynamic A\*: Fully Dynamic

- The update contains both insertion and deletion of edges.
- Propagate insertions and deletions of edges separately.
- Performs better than re-executing static A\* algorithm after each update.



5x Speedup by using dynamic A\*

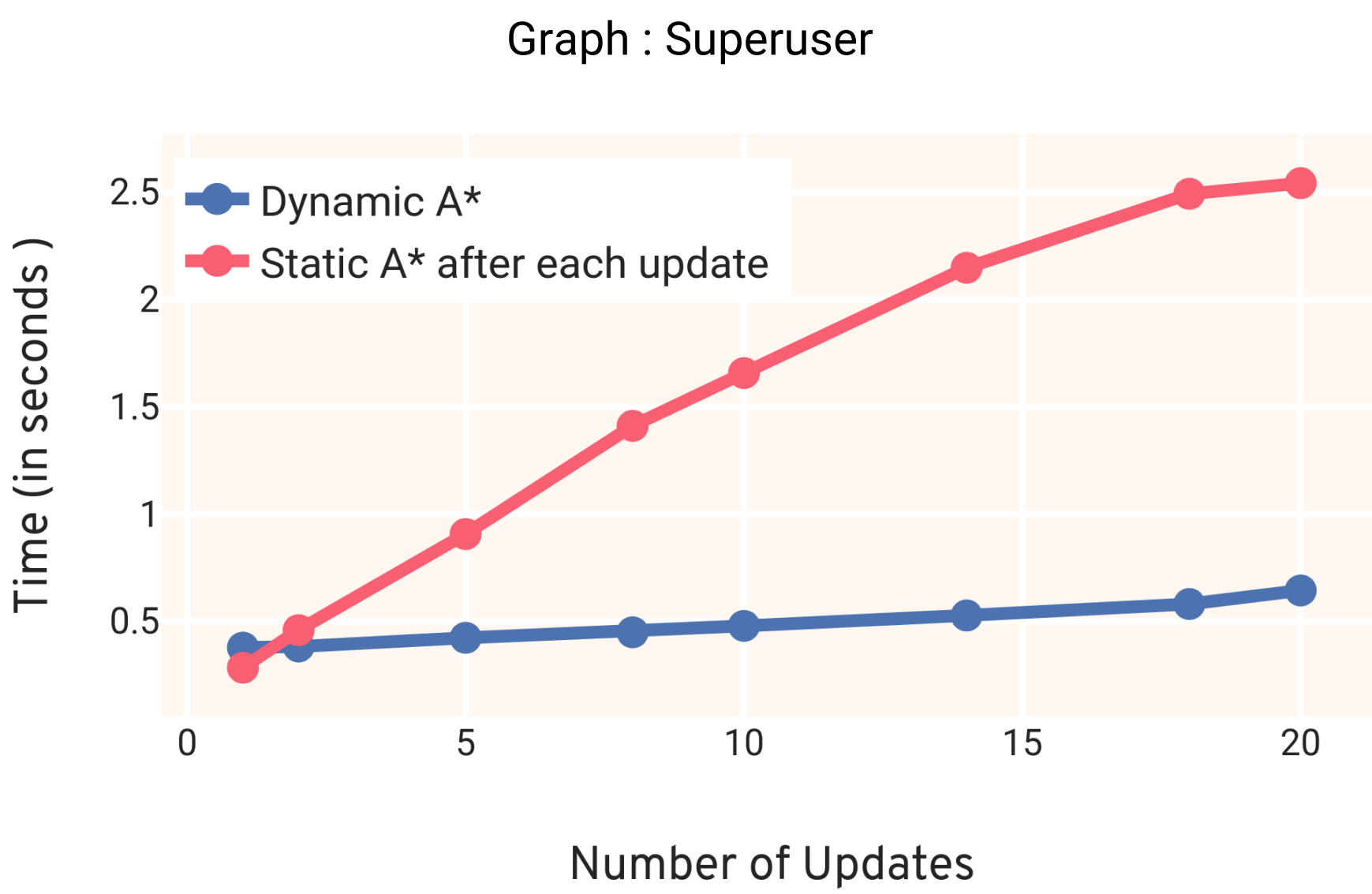


Fig 4 : Execution time vs number of updates In the graph

## Results

The below table shows execution time (in seconds) and speedup of dynamic A\* compared to re-executing static A\* every time.

No.	Graph	Edges	Queries	Dynamic A*	Static A*	Speedup
1	Live Journal	34,681,189	10	6.01	33.93	5x
2	Wiki Talk	7,833,140	10	12.24	24.84	2x
3	Ask Ubuntu	964,437	10	0.25	1.31	5x
4	YouTube	2,987,624	10	0.81	5.78	7x
5	Math Overflow	506,550	10	0.09	0.67	7x
6	Live Journal	34,681,189	100	11.41	424.06	37x

## Applications

- We have applied dynamic A\* on energy efficient routing protocol (EERP) and achieved 35x speedup from static A\*.



35x Speedup by using dynamic A\*

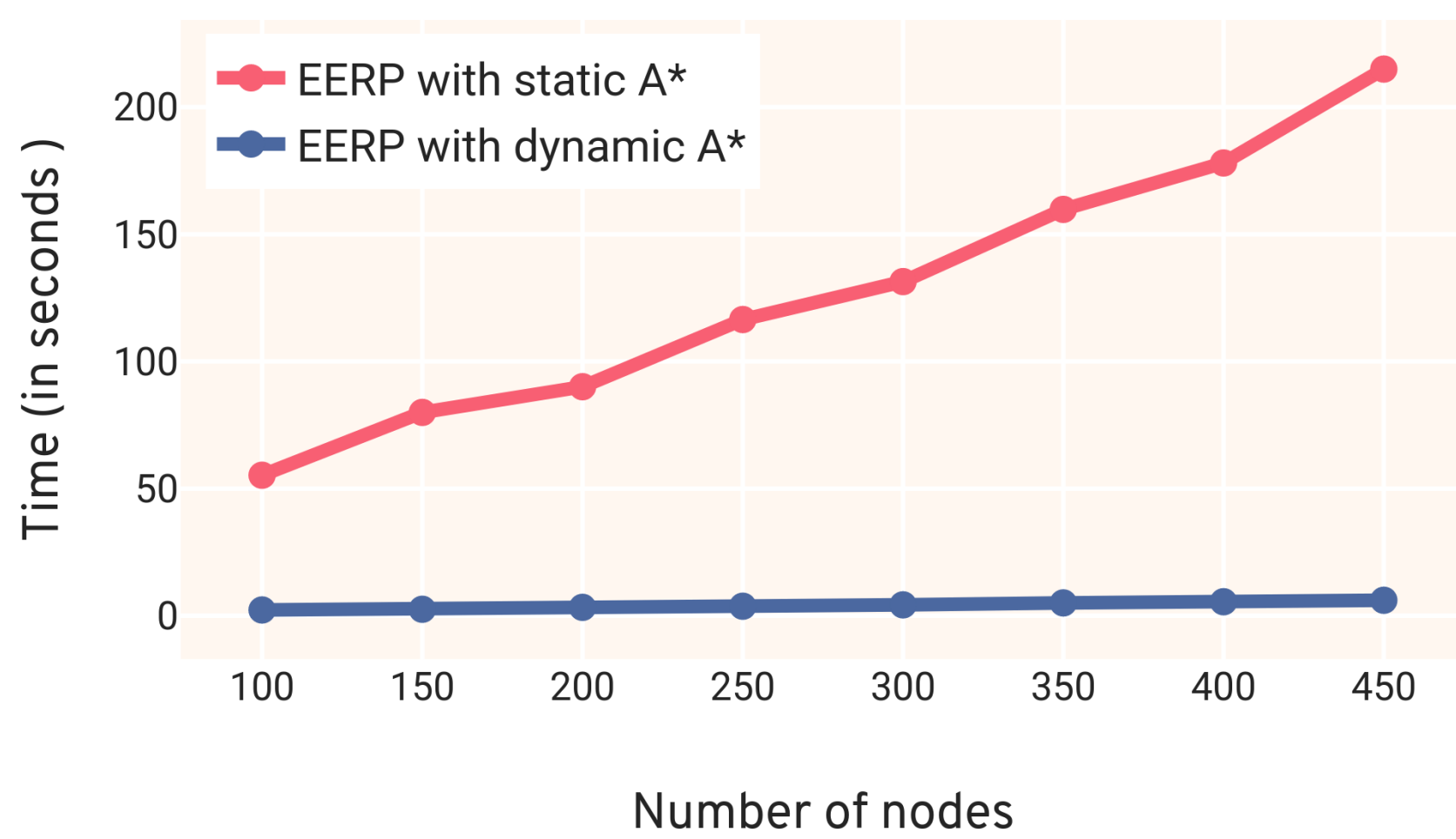
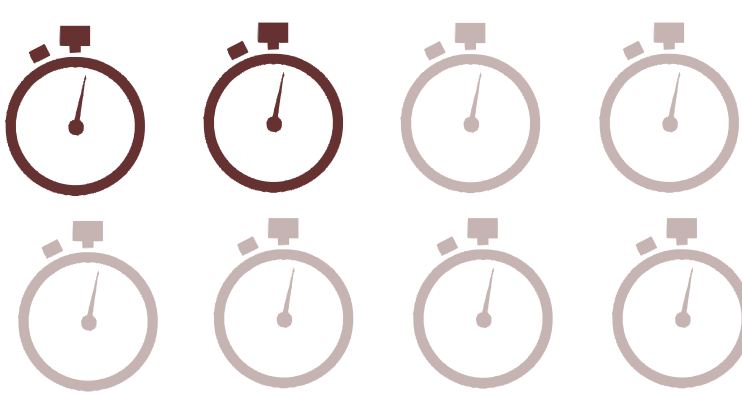


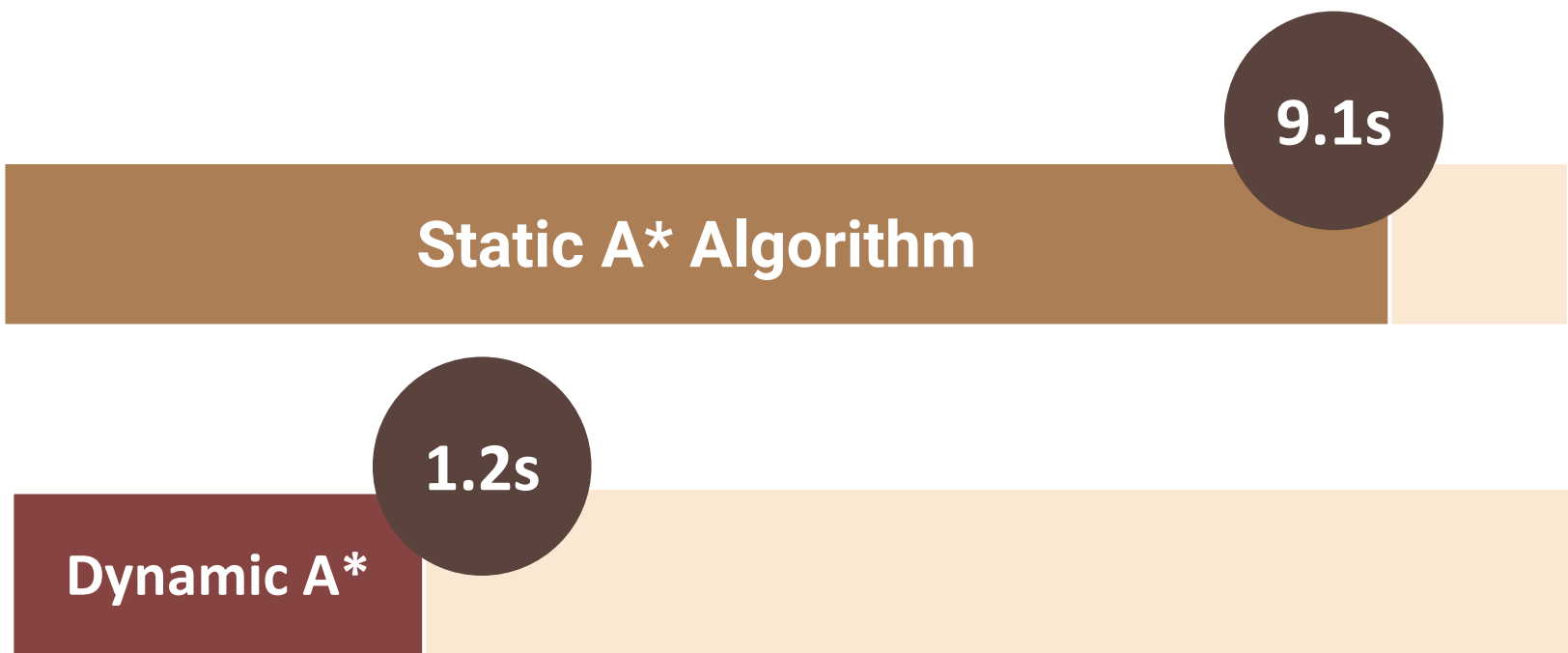
Fig 5: Comparison b/w static A\* and dynamic A\* on EERP algorithm

- On applying dynamic A\* for pathfinding in the maze, we achieved 8x speedup.



86.8 %

reduction in execution time by using Dynamic A\* algorithm.



Grid Size: 5000 x 5000 Nodes : 25M Edges: 49.9M

Fig 6: Comparison b/w static A\* and dynamic A\* on solving 5000 x 5000 maze

Time in seconds

## References

- Yichao Zhou and Jianyang Zeng. "Massively Parallel A\* Search on GPU". In: Twenty-Ninth AAAI Conference on Artificial Intelligence (2015).
- Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. <http://snap.stanford.edu/data>. June 2014.

<sup>1</sup> Pseudocode described above is to give a basic idea of the algorithm. It does not cover all the cases, for more information please refer to GitHub.