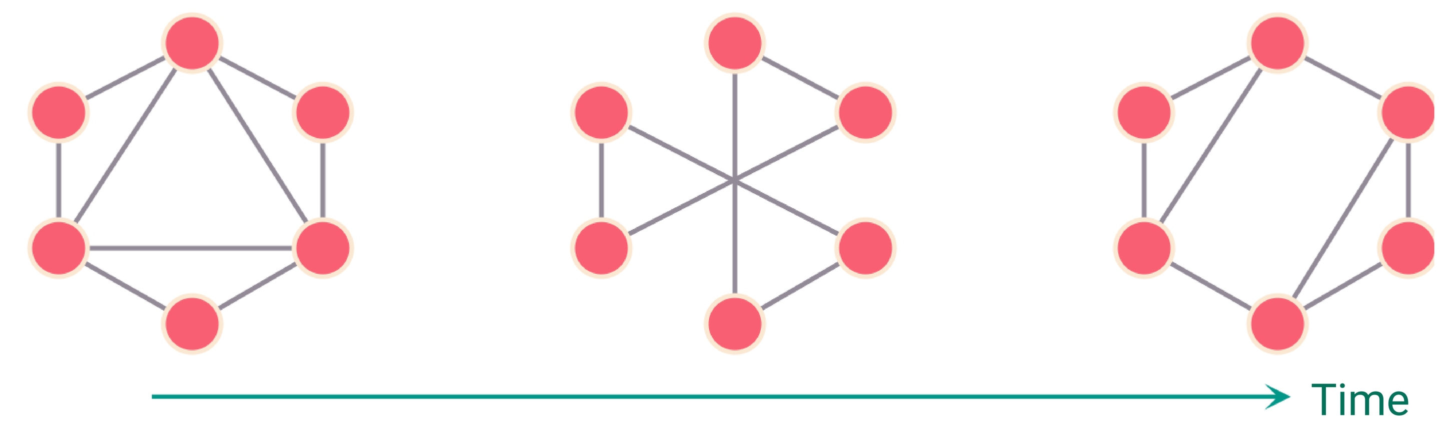
Path Planning for Dynamic Graphs using A* on GPU

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.

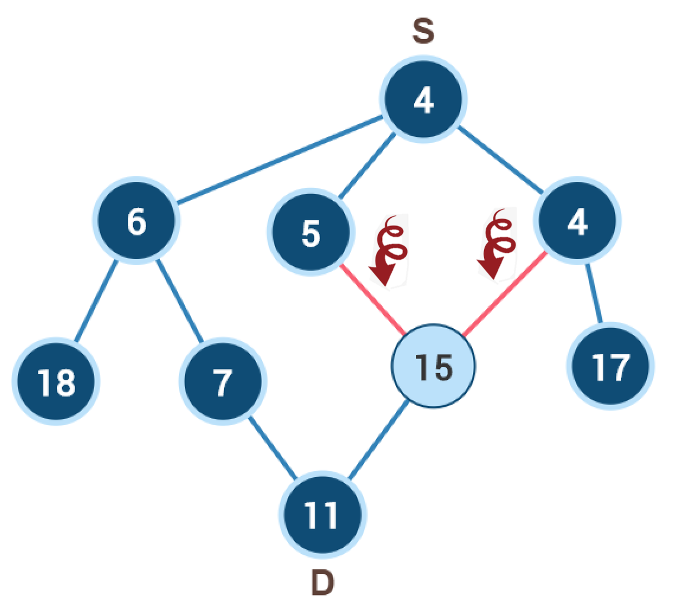


Fig 1: Two threads processing newly added edge (shown in red)

Pseudocode!

1. For edges(u, v) inserted, add node v to update_list, if $f(v)_{new} < f(v)_{old}$.
2. While update_list not empty:
   a. Extract node n from update_list.
   b. For each child of n:
      i. **lock(child)**
      ii. if $f(child)_{new} < f(child)_{old}$, add child to update_list.
      iii. **unlock(child)**

$f(v)$: cost of node v = $g(v) + h(v)$

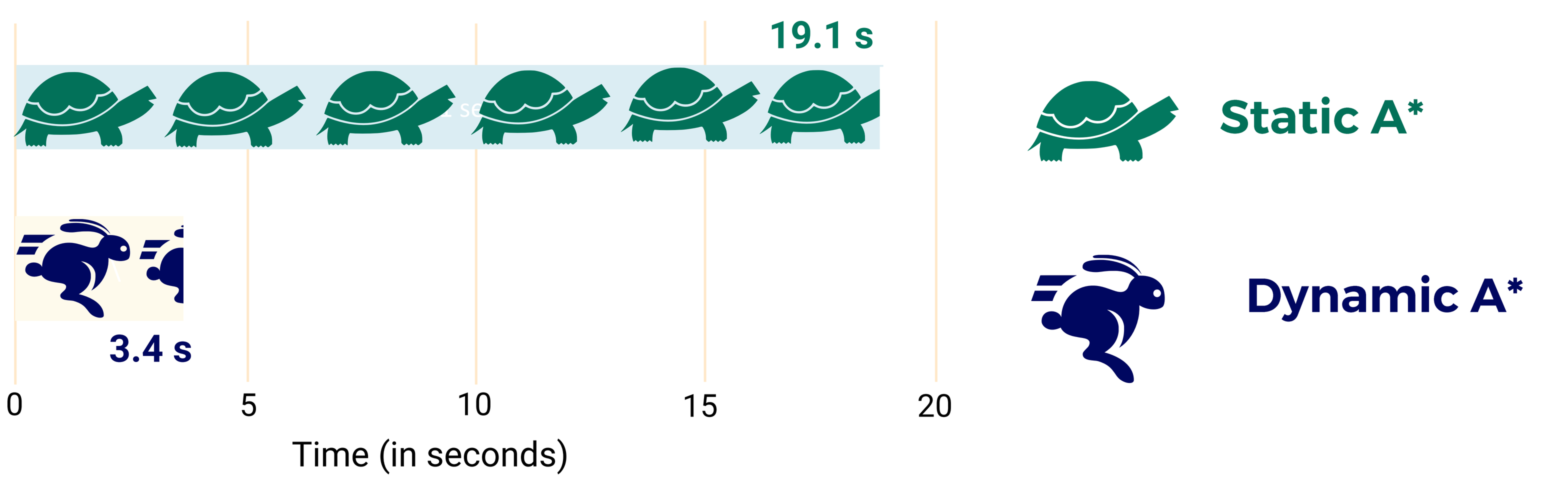
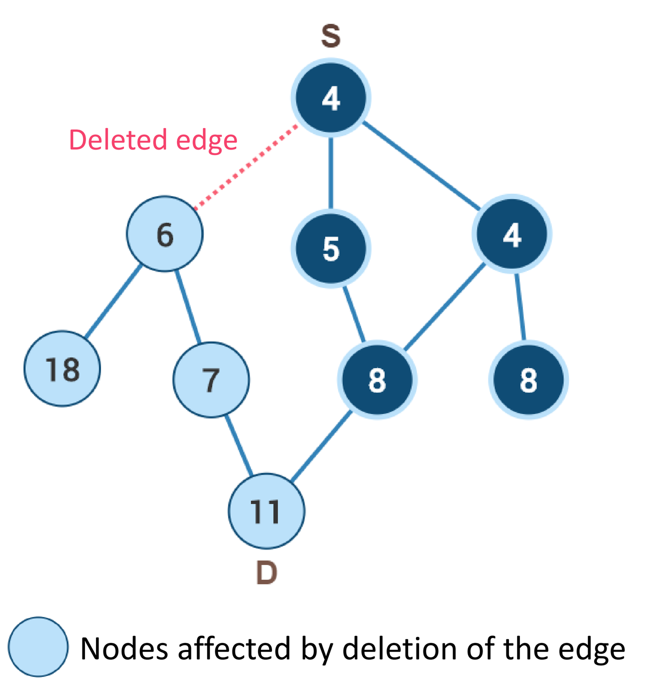


Fig 2: Execution time of static A* and dynamic A*(only insertions) on graph Wiki-Talk

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.



Pseudocode!

1. For each deleted edge u → v:
   a. Compute f(v) from the neighbours of v.
   b. Add v to update_list.
2. While update_list is not empty:
   a. Extract node **n** from update_list.
   b. For each child of **n** such that optimal_parent(child) = **n**:
      i. If $f(child) > f(child)_{new}$ then compute cost of child from each of its neighbour and add it to update_list.

optimal_parent(v): neighbour of node v with least f(v)

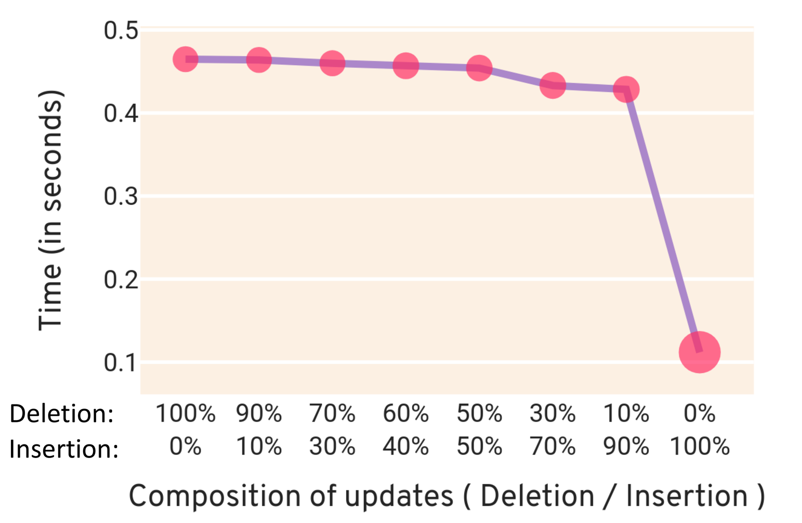
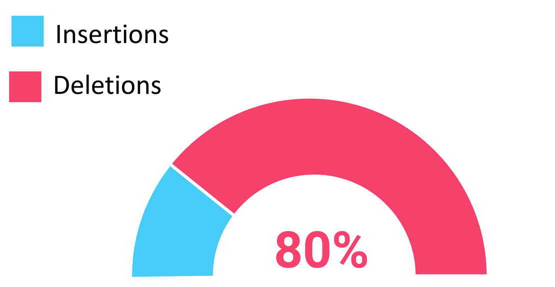


Fig 3: Execution time vs composition of updates



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.

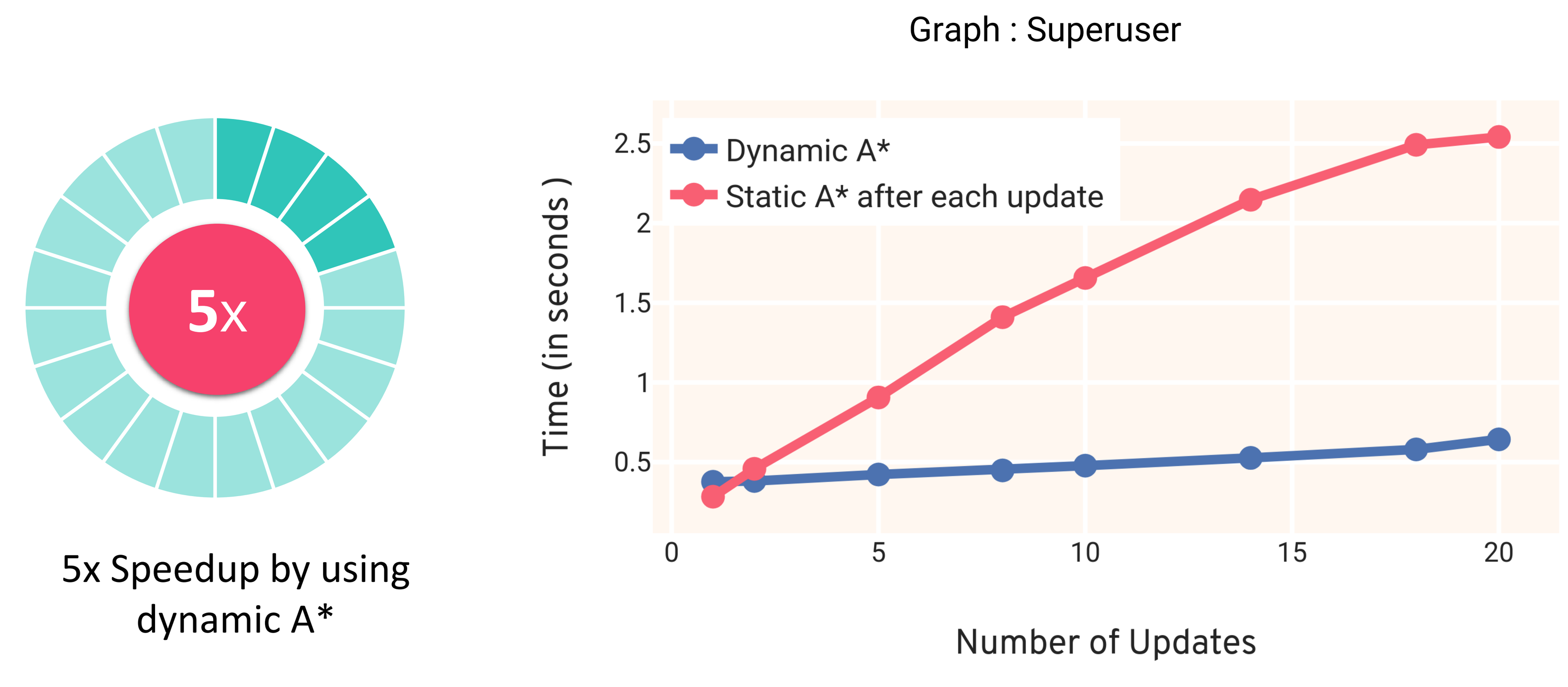


Fig 4 : Execution time vs number of updates In the graph

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

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

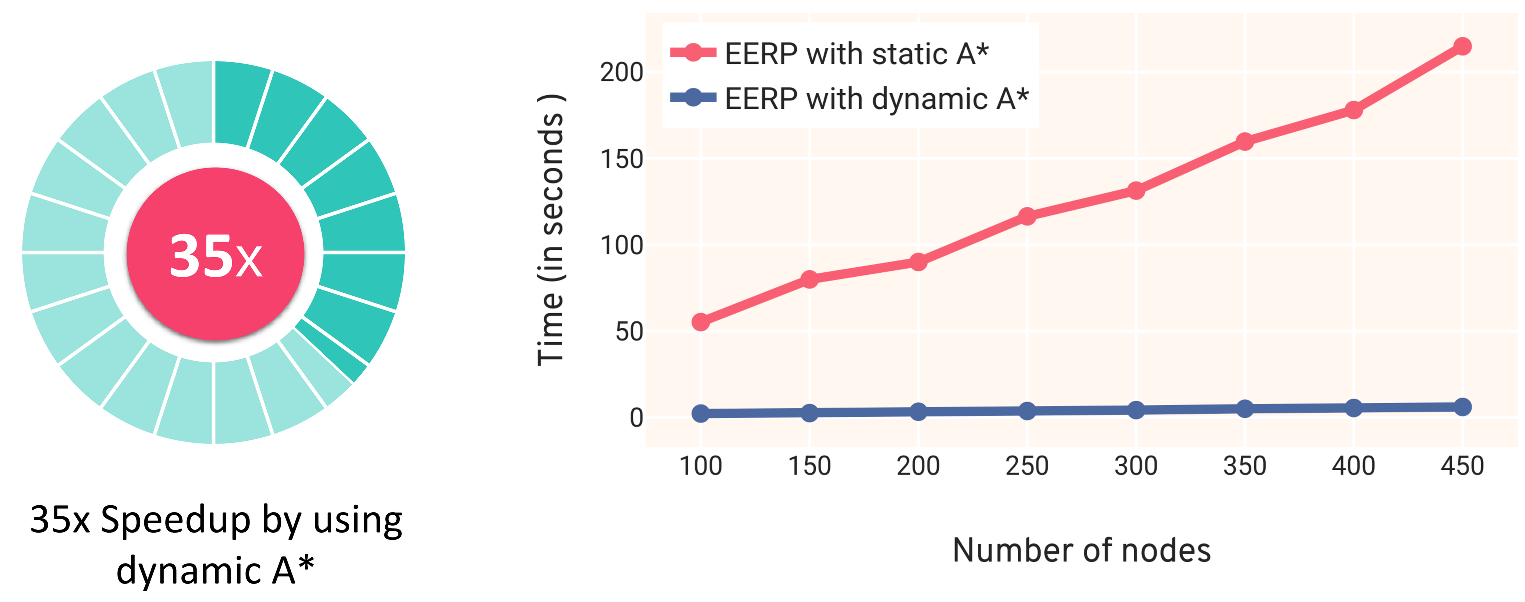


Fig 5: Comparison b/w static A* and dynamic A* on EERP algorithm

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

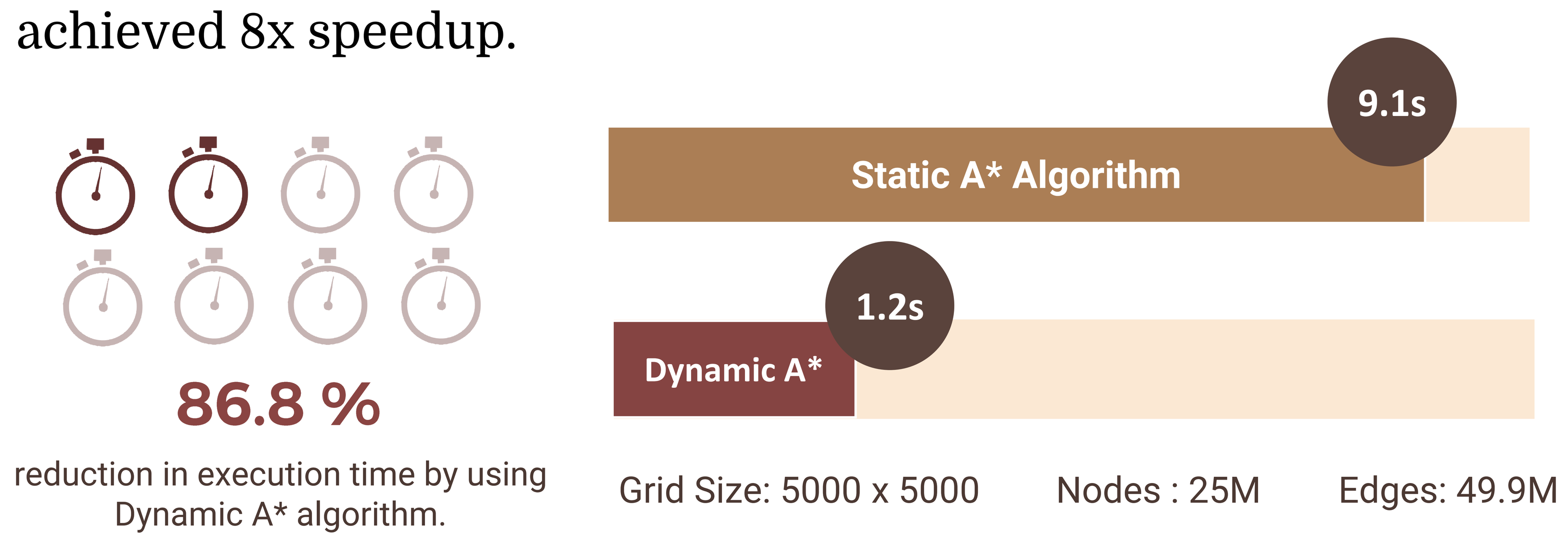


Fig 6: Comparison b/w static A* and dynamic A* on solving 5000 x 5000 maze

References

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

! 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.