Optimizing R* Search through Parallel Subsampling

By: Chris Bayley, Harsh Yallapantula, Kimberly Lim
Introduction

- A* search is widely used search algorithm
- R* search is an optimized version of A* which uses gateways to optimize the search time at the cost of reduced accuracy
- We propose a method to significantly reduce run-time by using a parallel algorithm to subsample a complex graph
- This is used to create gateways and provide an approximate path for A*
Motivation for Project

- Many applications require fast search, even if least costly path is not found
- Previous work has been done before to find optimal paths quickly
- Searching in parallel increases the speed significantly
- Few drawbacks, including chance of failure (Neighbors not connected and start and end goals happen to be in different islands)
Overview of A*

- Enter start node into a priority queue
- While queue not empty:
  - Pop from queue
  - If popped node is target, done
  - Add all neighbors of popped node with cost equal to current cost + heuristic
Overview of R*

- Enter start node into priority queue
- While queue not empty:
  - Pop from queue
  - Choose K number of random nodes distance \( \Delta \) from current node
  - Use A* to find path to each of K nodes
  - If successful, add into priority queue with cost equal to current cost + cost of found path
  - If unsuccessful, mark as an Avoid node and enter into priority queue with first bit set
  - Note: Always add end state if distance is less than delta from current node
Overview of our Approach

- R* operates by selecting gates randomly and finding paths to these
  - This operation could be made parallel for all K selected for current node, but why not parallel over entire graph?
- Select K = S_r*N random nodes from entire state space of graph, label as marked, where S_r is subsampling ratio
Overview of our Approach

- Search distance $\Delta$ around each node, if a node encounters marked node, add edge to this node
- End result is a graph with $K$ nodes and some edges between
- This process can be repeated until graph is sufficiently small to perform search over
- All nodes found in path over subsampled graph can be used as gates for $R^*$
Details

- Must maintain branching factor of original graph, otherwise graph explodes or withers away
  - $Pr[\text{expanded node is marked}] = S_r$
  - $Ex[\text{encountered marked nodes}] = \Delta * S_r = B; \; \Delta = B/S_r$
- Expanding $N*S_r$ nodes is expensive, but embarrassingly parallel
  - Total runtime of this step is maximum cost of expanding a single node ($\Delta$) + overhead
- Start and end nodes are guaranteed to be in subsampled graph
- A single connected graph may end up having islands after size reduction
  - Problem, no guarantee that path from start to end exists anymore
Path Reconstruction

- After reducing the size of the graph, a search can be performed to find a path using states in the subsampled graph
- The nodes in this path are used as gates for $R^*$, and an $A^*$ search is performed to find the path from one node to the next
  - This can happen entirely in parallel, meaning the total sequential work at this step is the max exploration cost over all the $A^*$ searches performed
Addressing the Issue of Disconnected Graphs

- Problem can be avoided naively by enforcing that every node must reach a specific node
  - Ensures that all nodes are connected by that node
  - Means every node must find this node, just as hard as original task
- Better approach, small group around one node are sampled and must find path to special node
  - All of these nodes are now guaranteed connected
- Larger group around previous group must expand until they find a node in previous group
- Repeat log n times, guarantees a connected graph
Addressing the Issue of Disconnected Graphs

- Tradeoff is changing embarrassingly parallel simple strategy to $\log n$ with parallel steps in between.
- Is this worth it? Turns out no, it’s cheaper to just start over in event of failure.
  - Assumes that parallel resources are enough to do entire previous step in parallel.
- Some workloads and environments may still benefit from this, this idea could be fleshed out in the future.
Results

- Tested on path-finding in grid with obstacles, with each cell being able to move to its 8 neighbors

```
Size: 7775
S_r = 0.1
Size: 779
S_r = 0.1
Size: 79
```
Results on Sampling Ratio and Success Rate

- Smaller $S_r$ values lead to more consistent graph size reduction
- Fewer iterations achieve greater work

![Average iterations before disconnected graph](image1)

![$S_r$ vs Graph Size Reduction](image2)
Results on Sampling Ratio and Work Reduction/Cost

- Work reduction is greater for smaller $S_r$ values
- No compromise on optimality vs different $S_r$ values
Stopping Criterion

- Small $S_r$ values generally succeed in reducing the size of the original graph
- Allows introduction of stopping criterion; end iterations when the resulting graph size is less than 5% of the original size
Results on Exploration vs iteration number

Search Cost $R = 0.1$

- Expanded Nodes in Search
- Expanded Nodes in Subsample
- Expanded Nodes in Reconstruction

Search Cost $R = 0.3$

- Expanded Nodes in Search
- Expanded Nodes in Subsample
- Expanded Nodes in Reconstruction
Results on Cost increase vs iteration number

Found Path Cost $R = 0.1$

Found Path Cost $R = 0.3$
Conclusions

- Our method is successful at greatly reducing the search task, with a small hit to the optimality of the solution.
- This method requires great parallel resources to get the exploration costs we described.
  - Overhead of parallel resources was not accounted for in our calculations.
- Smaller S_r values are generally preferred, which require less parallel resources as well.
- Failures will require a restart of the algorithm, but the overall performance gain is still significant in our test cases.
Future Work

- Find a way to either guarantee success of a run, or at least increase the chances of the resulting graphs remaining connected
- Implement on GPU hardware and measure cost of overhead for each iteration
- Test on data from various applications and determine feasibility
  - Different branching factors may lead to very different results
References

https://www.cs.cmu.edu/~maxim/files/rstar_aaa08.pdf