# Sequential Decision Making: Proposal

Shauharda Khadka, Pavel Zaytsev, Francis James

January 23, 2017

## Proposal Option 1: Path Planning with Dynamic Obstacles using Multiple Heuristics

Robots can often be required to move about in areas with other agents that are constantly in motion. For instance, UAVs have to ensure that they don't crash into other UAVs while flying and warehouse robots must make sure that they don't bump into humans that may be moving about trying to perform their own tasks. While algorithms such as D*-lite can replan quickly when a new obstacle is suddenly encountered, it does not allow additional heuristics such as the distance to the new obstacle or the direction of the obstacle's motion to be used. In this project, we plan on modifying an existing path planning algorithm to account for more information such as the probabilistic expected states of the obstacle given its present position and velocity (or other relevant parameters).

### Approach

The robot predicts the motion of other agents (or moving obstacles) on the map based on the current measured velocities. While a simple heuristic such as Euclidean distance can be used assuming a static map, node costs are updated based on a second, dynamic heuristic which may be based upon distance to predicted states of the moving obstacle.

### Experimentation

To test the utility of heuristic-shaping agents, we will test our algorithm in a constricted 2D airspace with UAV traffic passing through.

#### Potential Experimentation:

UAV's will begin with a optimal solution based on static algorithm such as A*. This static algorithm comes at no initial cost for the UAV and the cost can be assumed to its parent (company or place of origin). Ideally the UAV could follow this solution, but instead, random variability be introduced and the UAV will ideally adjust for new node costs using a dynamic algorithm and following parent preferences (greedy versus low cost).

## Proposal Option 2: Path planning NTM

Neural Turing Machines (NTM) augment the capabilities of neural networks by coupling them with external memory resources, which they can interact with by attentional processes. NTMs have been recently shown to be able to infer simple algorithms such as copying, sorting, and associative recall from input and output examples. These operations are the elementary building block of algorithms, and NTMs success in successfully learning them paves the way forward for designing neural architectures that can learn algorithms. Once trained, the neural architectures promise incredible speedups in running performance, and memory requirements as compared to hard coded algorithms; and also pave the foundations for plastic and adaptive planning algorithms that can react dynamically to new observations. We propose to utilize a neuroevolutionary algorithms to train a NTM, to learn A-star planning algorithm.

### Approach

The computational graph (map) will be serialized as represented as tuples 1, where $n_i$ and $n_j$ represent two separate nodes i and j respectively, and $e_{ij}$ represents the edge cost between them.

$$(n_i, e_{ij}, n_j) \quad \forall \quad nodes \quad and \quad edges \tag{1}$$

The entire computationally graph will be stored in this format in a memory bank that can be though of a configuration file in a hard drive, where each configuration file (computational graph) represents a different problem. The NTM will have read access to this external memory bank. Additionally, the NTM will also have its own internal memory bank which it will interact with read and write heads modulated by attentional processes. This internal memory bank can be thought of as Random Access memory of our NTM. A neuroevolutionary algorithm will be utilized to train our NTM which will learn to read from the external memory bank, interact with its own internal memory bank, and learn to plan optimal* trajectories in the computational graph presented. To generate training examples, we plan to use randomly generated computational graphs as inputs and and trajectories generated by an actual A-star (coded), as targets. We realize the scale of this project, and thus will use simpler computational graphs limiting size to about 10 nodes, for the purpose of the 8-week deliverable.

### Experimentation

To test our path planning NTM, we will generate new computational graphs test sets using A-star. Added, we will test for pure generalizability, by testing on bigger computational graphs ( 15 nodes), than the one NTM was trained for ( 10 nodes).