“Flexi-Trees and FSM Generators”

A Machine Learning Algorithm for a Real Time Strategy Game

An example of dependencies:

Requires B Requires C

For example, Unit A may be a barracks, and unit B 100 trees and unit C 500 gold or unit B 200 vespine gas and unit C a meeting hall. This graph purely represents the dependencies that are necessary to win the game.

Assume that we start at the goal node, “Enemy Defeat”, and a tree of decisions that lead up to the defeat of the enemy:

Goal

Produced 100 knights

Produced 100

soldiers

Built a barracks

This graph clearly shows memory of state, past situations where a certain series of actions led to victory.

If decision-making graph is traversed more than once, probability can be induced for decision making. For example,

Goal

50% 50%

Produced 100 knights

Produced 100

soldiers

20%

Built two barracks

Probability weights are used here to record success/failure ratio (0-1).

These probability weights influence traversal of the tree, making it when certain strategies are successful, they are used more often.

The initial requirements graphs can be embedded within the decision making graph, so that prerequisites for armies can be fulfilled. For example:

Goal

50%

Produced 100

soldiers

Trained 100 soldiers

The node “produced 100 soldiers” can be expanded into:

Built a barracks

Mined 1000 gold

Farmed 100 Trees

**Pruning Process**

During shutdown, a cleanup process is incurred that prunes off the edges of the screen with a low success rate. During the game selected nodes are put enqueue for this process.

The overall tree has the following structure:

Root – represents GAME\_BEGIN

Branch – represents a decision made to move from parent to child node

Leaf – represents GAME\_END for either player’s favor.

The pruning process attempts to remove the paths to the deepest leaf GAME\_END nodes. If a node is a GAME\_END but in the enemy’s favor, the branch is kept, but scored to a 0 weight so that it will never be traversed. This way if an identical branch or subset of the branch is encountered, time is not wasted on expanding it and recording it.

**Tree Auto-Growth**

The tree is automatically built via a random finite state machine called the generator which attaches to the flexi-tree through an interface. The generator will forward different game actions into the tree through the interface, and the tree will then execute these actions. The generator is only used when the tree traversal is at a leaf node (that without children) and the state of the game is not END\_GAME.

When a successful path through the tree is found and a game is won, the probability weights in the tree are “back-flushed”, strengthening that branch of the tree.

The probabilistic weights along with the generator give the tree an almost realistic nature to them with the branches growing at the stems dynamically and the tree shifting from left to right because of the weights, like a sapling growing and swaying in the wind.

There are two phases to the tree that involve the auto-generating FSM: the initial tree construction phase and its maintenance phase.

Tree Generator:

Uses a finite state machine to generate tree elements. In the tree, for any given parent node to child node, there should exist a path with a weight of 1/n by default, where n is the total number of children in the tree.

**Example Generator:**

Not enough Peasants,

Not enough Gold,

Barracks not built,

Not enough soldiers

Train soldiers

Train Peasant

Build Barracks

Mine Gold

FSMA (Finite State Generator)

Decision Des

Flex-Tree

Decision Node

Insert

Game

Over

**Finite State Machine Generator (FSMA) Dependency Sets:**

(for example deployment in S3):

Footmen : {Town Hall, Barracks}

Archer : { Town Hall, Barracks, Lumber Mill }

Knight: { Town Hall, Barracks, Lumber Mill, Blacksmith, Fortress, Stables }

Catapult: { Town Hall, Barracks, Lumber Mill, Blacksmith }

Tower: { Town Hall, Barracks, Lumber Mill, Blacksmith }

Wall : { 0 }

The > | S | (overall magnitude of requirements),

The > p (power or strength of unit)

AND The > t (time to create unit)

**Dependency Diagram**

Barracks

Requires Gather Gold Action

Requires Train Peasants Action

Requires Town Hall Built Action

Requires Gather Lumber Action

Action Score (|S| = p/t) acts as a heuristic in determining cost of a course of action.

The requirements and actions can be organized into a linear finite state machine that will be used for our FSMA.

The level S of the FSMA construct reflects the depth of the FSM.

Example Linear FSMA:

Build Town Hall Action

Is Town Hall Built?

Are X Peasants Trained?

Train X Peasants Action

Build Barracks Action

Is Barracks Built?

Train X Footmen

Are X Footmen Trained

Is Lumber Mill Built

Build Lumber Mill

Above, the FSM is followed linearly downwards, as it does so the value of p increases. When an action (right hand side) is triggered, it is sent to the Flex Tree to be added to the barren leaf node.