Brendan Anderson, Anthony Ou

Strategy Clustering

Summary

For our CMPUT350 project, we designed and implemented a strategy clustering method to be used in a wide array of applications. It's capable of being utilized in any of the RTS games discussed in the course and with minor variations it could be integrated into the ANTS, ORTS and Starcraft. We also modified an architecture for parsing the game state of a Starcraft game using the broodwar API to integrate and test our strategy clustering using professional replays. With minimal effort, our work can be added into the existing ualberta bot to provide the groundwork for strategy identification and prediction in the bot.

The Problem

When the UAlbertaBot was discussed to us in class, we noticed an omission in the design. As described by Michael Buro, the bot's strategy management worked like thus: When first facing a new opponent, randomly select one of the bot's pre-programmed strategies. Play out the game using that strategy. If the bot won with that strategy, make it more likely to be randomly selected again against this opponent. If it lost, make it less likely. In this way, the bot was developing a "Strategy-to-Player" mapping.

There are several fundamental problems with this approach. When you talk about strategies and their relative effectiveness, it is within the context of the opponents strategy due to the game's adversarial nature. Whether one move or another is a good move is based off the moves of the opponent. The "Strategy-to-Player" mapping carries with it the assumption that the opponent player is a stable predictor or proxy measure of the opponent strategy employed. This is not a particularly safe assumption however, since players are able to enact many different strategies in the game. The bot itself attempts to tune it's strategies as it plays multiple games against the same opponent yet carries with it the assumption that the enemy bot isn't doing the same. Furthermore, this approach cannot generalize learning across different players. All learning done is against the current opponent only, the bot has to rely on the starting probabilities and pre-canned strategies when faced against a new opponent.

When you look at professional players and their analysis of strategies, the framework of the discussion is based on a perception of the relative 'strength' of the strategy and it's counter strategies. The perception of relative strength is a subjective judgement of the strategies likelihood of winning against other strategies and it's resistance to being countered. In this way, strategies are mapped "Strategy-to-Strategy". In this way, knowledge generalizes across players since players are seen as implementers of strategy rather than the targets of learning themselves. Additional learning can be applied on top as you learn about a given players bias for strategy implementation, but it is most useful to think of these as separate layers of learning.

Our Approach

To solve the problem of generating a “Strategy-to-Strategy” mapping framework, we identified two sub problems that we had to solve. We had to develop a system for reading out the state of a starcraft game. To address this we created our “Replay Parser” system, which was designed to watch replays of professional starcraft games and provide game state information for our strategy clustering methods. This system is not restricted to being run with replays however, it runs in the live game environment as well! With minimal effort an existing starcraft bot could be modified to include our game state parsing system and be given the tools needed to gather the information our clustering system needs.

The second sub problem is the strategy clustering problem. To do meaningful comparisons, we need strategies to generalize into ‘similar’ looking groups of strategies. A “Rush” strategy should look like and be treated like other “Rush” strategies despite minor inconsistencies. The issue present here however is addressing what it means for strategies to be similar, and following from that what is a minor inconsistency and what is a major one. We developed a K-Means Clustering algorithm to generalize strategies into K clumps of like strategies and a similarity rating algorithm to drive the comparison between strategies.

The Parser

This feature began by using the BWAPI to dump the game state from a replay into a text format and integrated into the UAlbertaBot. The UAlbertaBot was used as a base for replay dumping because this is the intended platform for it’s future deployment; in the end replay dumping and AI modules can work together without adding additional complexity. Code from parsing text could be reused in parsing the live game state, but this is still specific enough to require refactoring for UAlbertaBot. The parser and replay dumper can take the entire game state and parse a vector of “units” into an interval we specify. Since we are only interested in early game data we parse only between the game frames 1-8000.

The replays in each file list an event, which includes created, destroyed, or discovered, the unit to which this event happened to. Some features left out were research events and supply used, because we decided not to include this data in our clusters. The Parser reads over the saved reports from previously parsed games then pipes the various game states into K-means clustering.

The Feature Vectors

We attempted to convert the game state into a feature vector in two ways. One way was representing the constructed units and buildings beyond the starting structures and units of the game, thus using the players construction as a proxy measure of the implemented strategy. The other way was representing the resources spent on new units and buildings, attempting to use resource investment as a proxy measure of the implemented strategy. Currently the transcription from raw game state to feature vector takes place within the parser.

K Means Cluster

The K means clustering algorithm is a relatively simple approach to clustering data, yet is flexible and robust enough to scale to the demands of our application. It begins by selecting K arbitrary data points and building a cluster around each of them. Then it iterates over all the data points and assigns each data to the centroid of each cluster, and then assigns the data point to the cluster it is closest to. (A centroid is the mean value of the cluster, used as the basis for identifying and comparing to that group of data.) Once it has assigned all data points to a cluster, the algorithm recalculates the centroids of each cluster and tracks the shift theses updates cause. If the centroids shifted by a value greater than the cutoff, all data points are re-assigned to clusters based on the new centroids and the process repeats until the shift is less than the cutoff.

In our implementation we assigned based on “most similar” instead of “closest”. To evaluate similarity we use the dot product of the subjects of comparison rather than linear distance. The motivation for this choice is the nature of the strategies. A strategy is like a vector in many ways. All strategies begin from the origin point of the game, branch out and invest limited time and resource in a direction. You cannot regress in the path of a strategy, and to change one strategy into another you must invest the difference in the context of the strategy you wish to change to if even possible.

After the K means clustering algorithm has reached approximate convergence, it returns an array of K centroids. These are the mean centers of the strategy clusters the algorithm developed. These centroids can be used by calling our similarity evaluation method against the game state feature vector you wish to evaluate. Whichever centroid the new strategy you wish to evaluate is closest to, the algorithm would assign it to that cluster. Thus given a new strategy, using our methods you could abstract down to the strategy cluster level allowing for generalization and future development of growth prediction and counter-play learning.

The Combined Implementation

To test the validity of our approach and our implementation we attempted to cluster the strategies professional players employed in the ‘early game’ phase of tournament matches. Tournament replays were available online from GoSu Gamers and ICCUP. Focusing on Protoss replays, we played a sampling of replays through our parsing system and extracted game states from these professional matches. Then we converted the raw state information into feature vectors for the games and piped this feature data into our clustering algorithm. The clustering algorithm is then run to convergence and returns a set of K centroids. Using the tools of our algorithm, you would then be able to compare new feature vectors to these generated centroids and answer the question “which strategy is most like what I am seeing?”

Results

We ran 120 replays through our parsing system, then ran multiple iterations of the clustering algorithm with varied settings. (See Fig1) Without encoding any of our own knowledge of the game, our algorithm has generated 3 strategy prototypes. Strategy 0 decodes to 1 assimilator, 4 dragoons, 21 additional probes, 2 zealots, 1 gateway, and 4 pylons. Strategy 1 has 0 assimilators, 2 dragoons, 16 probes, 4 zealots, 1 gateway, 1 photon cannon and 4 pylons. Strategy 2 is similar to strategy 0 with 1 assimilator, 3 dragoons, 26 additional probes, two gateways, and four pylons.

Some expert knowledge will tell you that there are problems with this clustering. It is impossible to have a photon cannon without a forge, or dragoons withou a cybernetics core. Additionally in centroid 1, we’ve got both a larger number of zealots (offensive rush) and forges (defensive turtle). So with k = 3 we’re clustering very different strategies together, which is less than ideal.

Next we tried clustering again with feature vectors based on minerals spent instead of raw unit count. This data is more interesting because of how it makes predictions based on less common features like the templar archives. Additionally the clusters are different than when we ran with just unit counts. Centroid 2 includes 201 resources spent on cannons visible now, instead of the 0 cannons we saw before. This is a better match to the high economy play seen in the other elements of this feature vector. We argue that this is good initial evidence for the advantage of our feature vector coding instead of using raw unit counts.

The next modification we attempted was scaling up the number of clusters. With increasing k (See Fig3) our algorithm performed well. The pattern that emerged was as we increased K, one of the centroids would split in two while the other strategies stay relatively constant. To make the full argument for an optimal K or an optimal feature vector encoding we would need to be using these centroids to make predictions or estimations in game, but this implementation is beyond the scope of our project. Increasing K will reduce error, but hurt generalization and carries run-time costs. Whether a larger K makes for better learning is something that remains to be tested by a future team.

Conclusion

We’ve clearly succeeded in the successful implementation of game parsing and k-means clustering. We have shown that with our approach, replays of professional games and live games can be parsed to extract game state features, and that those features can be used to generate strategy clusters. Using our tools, a bot could likely be improved by integrating in a system to predict the opponents strategy and perform learning based off strategy clusters. These strategy clusters would be general to the game being played not a specific player so this could be the foundation of developing bot expertise.

Future Work

There is much that remains to be done with further implementation of the system we created and increasing the accuracy or fidelity of the system. A primary site for future work is successfully incorporating this functionality into a running brood war bot. This contains multiple sub-problems to be worked on, including using the strategy clustering to drive prediction of future strategies. This will require running the clustering using mid game information in addition as well as using machine learning to train the association between early game centroids and mid game centroids. There is the additional work to be done in using the base early game predictions to drive early game strategy selection and learning.

Another problem is going from the centroids built on perfect information to a scouting report built on an imperfect snapshot of the opponents base. How do you go about predicting the opponents strategy when you can only see a portion of the base?

Additionally, the validity of our feature vector and by extension our similarity measure is open to question and further development. Are there better features to extract from the game state? What about combinations of features, or units destroyed (much more important for mid and late game states)?

Current limitations to the feature vector conversion in the parser parser is the fact that Protoss units are the only units considered. Other races can be quickly added but this can only be done manually. Generalized clustering with multiple different kinds of races in the same data set is untested.

In terms of what was gained from gathering texted based replays the data we used was narrowed to what was created. The replays include data like where events are happening and at what time. Hard data, like units made can be easily be utilized by K means. However, higher level tactical abstractions can be looked at, but not by K means. In StarCraft unit positioning and timing of battles is a strategy on its own that we do not look at.

FEATURE VECTOR DECODE KEY:
each index directly corresponds to a protoss unit as follows

Arbiter,Archon,ArbiterTribunal,Assimilator,
Carrier,Corsair,CitadelofAdun,CyberneticsCore,
Darkarchon,DarkTemplar,Dragoon,HighTemplar,
Interceptor,Observer,Probe,Reaver,
Scout,Shuttle,Zealot,Gateway,Nexus,
Observatory,FleetBeacon,Forge,Photoncannon,
Pylon,RoboticsFacility,RoboticsSupportbay,Shieldbattery,
Stargate,TemplarArchives

=======
Fig1. Example games clustered by unit count with k=3
centroid 0: 0,0,0,1,0,0,0,0,0,0,4,0,0,0,21,0,0,0,2,1,0,0,0,0,0,4,0,0,0,0,0,
centroid 1: 0,0,0,0,0,0,0,0,0,0,2,0,0,0,16,0,0,0,4,1,0,0,0,0,1,4,0,0,0,0,0,
centroid 2: 0,0,0,1,0,0,0,0,0,0,3,0,0,0,26,0,0,0,2,2,0,0,0,0,0,4,0,0,0,0,0,

======
Fig2. Example games clustered with k=3, using mineral spent feature vectors
centroid 0: 0,0,0,104,0,6,13,200,0,0,761,0,0,4,1198,0,0,17,133,336,222,21,0,10,10,486,146,6,0,6,3,
centroid 1: 0,0,0,113,0,17,45,179,0,16,349,0,0,2,1072,35,0,28,418,278,173,9,0,45,58,447,81,21,2,26,19,
centroid 2: 0,0,0,146,0,28,60,150,0,31,218,0,0,0,1268,0,0,0,281,229,300,1,0,93,201,453,37,0,0,37,23,

=====
Fig3. Example games clusters with k=, k=5 and k=6
centroid 0: 0,0,0,110,0,15,15,200,0,0,677,0,0,3,1186,0,0,10,163,315,252,23,0,15,23,468,147,7,0,15,0,
centroid 1: 0,0,0,113,0,17,45,179,0,16,349,0,0,2,1072,35,0,28,418,278,173,9,0,45,58,447,81,21,2,26,19,
centroid 2: 0,0,0,146,0,30,65,146,0,33,200,0,0,0,1273,0,0,0,266,225,320,1,0,100,215,430,33,0,0,40,25,
centroid 3: 0,0,0,103,0,0,10,200,0,0,799,0,0,4,1207,0,0,21,139,348,185,17,0,5,0,521,142,5,0,0,5,

centroid 0: 0,0,0,110,0,15,15,200,0,0,677,0,0,3,1186,0,0,10,163,315,252,23,0,15,23,468,147,7,0,15,0,
centroid 1:
0,0,0,104,0,14,36,177,0,18,375,0,0,2,1025,39,0,32,443,280,148,10,0,33,33,458,90,24,3,16,21,
centroid 2: 0,0,0,129,0,44,88,164,0,58,264,0,0,0,1279,0,0,0,323,273,282,2,0,61,61,452,35,0,0,44,35,
centroid 3: 0,0,0,103,0,0,10,200,0,0,799,0,0,4,1207,0,0,21,139,348,185,17,0,5,0,521,142,5,0,0,5,
centroid 4: 0,0,0,175,0,22,67,150,0,0,118,0,0,0,1342,0,0,0,195,195,380,0,0,150,367,385,20,0,0,60,7,

centroid 0: 0,0,0,110,0,15,15,200,0,0,677,0,0,3,1186,0,0,10,163,315,252,23,0,15,23,468,147,7,0,15,0,
centroid 1:
0,0,0,103,0,15,39,182,0,19,405,0,0,3,1038,43,0,35,380,271,154,11,0,31,26,456,98,26,3,18,23,
centroid 2: 0,0,0,129,0,44,88,164,0,58,264,0,0,0,1279,0,0,0,323,273,282,2,0,61,61,452,35,0,0,44,35,
centroid 3: 0,0,0,103,0,0,10,200,0,0,799,0,0,4,1207,0,0,21,139,348,185,17,0,5,0,521,142,5,0,0,5,
centroid 4: 0,0,0,175,0,22,67,150,0,0,118,0,0,0,1342,0,0,0,195,195,380,0,0,150,367,385,20,0,0,60,7,
centroid 5: 0,0,0,120,0,0,0,120,0,0,25,0,0,0,880,0,0,0,1160,390,80,0,0,60,120,480,0,0,0,0,0,