Sequential Pattern Mining in StarCraft: Brood War for Short and Long-Term Goals

Michael Leece and **Arnav Jhala**
Computational Cinematics Studio
UC Santa Cruz
{mleece, jhala} at soe.ucsc.edu

Abstract

A wide variety of strategies have been used to create agents in the growing field of real-time strategy AI. However, a frequent problem is the necessity of hand-crafting competencies, which becomes prohibitively difficult in a large space with many corner cases. A preferable approach would be to learn these competencies from the wealth of expert play available. We present a system that uses the Generalized Sequential Pattern (GSP) algorithm from data mining to find common patterns in StarCraft:Brood War replays at both the micro- and macro-level, and verify that these correspond to human understandings of expert play. In the future, we hope to use these patterns to learn tasks and goals in an unsupervised manner for an HTN planner.

Real-time strategy (RTS) games have, in recent years, become a popular new domain for AI researchers. The reasons for this are many, but at the core it is due to the inherent difficulty of creating intelligent autonomous agents within them. This in turn is caused by the imperfect information, real-time, adversarial nature of the games. Additionally, the requirement to reason at multiple levels of abstraction, with interaction between the decisions made at each of the different levels, poses another challenge. On top of all of this, the enormous sizes of the state and action spaces mean that straightforward applications of traditionally successful techniques such as search and MCTS run into difficulties.

In light of these challenges, many different approaches to creating intelligent agents in RTS games have been tested. Some of these include hand-crafted state machines, search-based approaches, goal-driven autonomy, or, more commonly, some combination of techniques. We are interested in planning approaches to the problem, and are particularly looking at Hierarchical Task Networks (HTNs). An HTN consists of a dictionary of primitive tasks (basic domain competencies), complex tasks (compositions of primitive and complex tasks), and goals to be achieved. These tasks can have preconditions and postconditions, with many extensions such as durative actions and external preconditions.

While many papers cite HTNs as a successful AI technique, it is nearly always followed with the caveat that they require an immense amount of programmer/expert curation, as they need to be defined and refined by hand. However, with enough data the possibility to learn this structure in an unsupervised manner exists, and has been the study of recent research.

This paper presents a system that uses data mining techniques to search for action patterns in RTS replays. There are two main goals for the results. The first is that they give us insight into the universal aspects of gameplay from human players, which may be useful when designing hand-crafted agents. The second, and more ambitious, is to find common sequences of actions that may translate to meaningful task/goal pairings in an HTN model for an RTS agent.

Background and Related Work

StarCraft:Brood War

StarCraft: Brood War (SC:BW) is an RTS game produced by Blizzard Entertainment. As it has been the focus of much recent work, we will give a high-level overview while highlighting the relevant aspects to this work.

Play in SC:BW proceeds in the manner of traditional RTS games: players build up an economy in order to train an army that lets them defeat their opponent. The economy is built by training worker units and expanding to new resource locations. Army development requires construction of training buildings, from which military units can be trained, and also tech buildings, which are required to construct higher tech units or unlock upgrades to make current units stronger. Players must balance resources between economy and military to both not fall behind on production capability while also not becoming vulnerable to attack from the opponent.

One of the attractive aspects of RTS games in general is the requirement for planning at multiple levels of abstraction, from the individual unit movement level up to the high-level resource allocation and tech advancement problem. In addition, these plans must be coordinated with each other, in that if the resource allocation plan is an aggressive military one, this greatly affects how units must be moved at the mid-level positioning problem and even the low-level micro problem.

Another feature that has elevated SC:BW as an AI domain is the existence of expert human play. As one of the first games to become an 'eSport', SC:BW has professional

Figure 1: An example screenshot from SC:BW. Shown is a Terran base with mining worker units, and a portion of the Terran player's army.

leagues and tournaments, and large numbers of replays from professional players can be acquired online. This gives researchers high-quality demonstrations of play from which to train agents. To us, who are interested in unlabeled learning from demonstration in this domain, this is a critical aspect.

Related Work

The sequential pattern mining problem was brought to the forefront in (Agrawal, Imieliński, and Swami 1993), who set forth many of the challenges and tradeoffs that must be considered when approaching the problem. Later, Agrawal et al. summarized the main algorithms for the problem in (Agrawal and Srikant 1995). Since then, many extensions and optimizations have been developed for these algorithms, but the core set is sufficient for our purposes.

The work most similar to ours algorithmically was presented in (Bosc et al. 2013), which also used sequential data mining to analyze RTS games, in this case StarCraft 2. However, their work focuses on the extraction itself, with some additional analysis of high-level build order success/failure rates, with an eye towards game balance. We feel that the approach has much more potential than this.

One of the inspirations for this work is HTN-MAKER (Hogg, Munoz-Avila, and Kuter 2008). This system learns HTN methods from observation of expert demonstrations, which is our stated end goal. However, it has issues with generating large method databases even in simple domains, something that will explode when transposed to the complexity of SC:BW. Additionally, it uses a more logical than statistical approach, which we feel is less appropriate when working with human demonstrations that are likely to contain errors. (Yang, Pan, and Pan 2007) use an EM clustering of primitive actions to abstract tasks to incrementally build up a hierarchy of methods, which is more likely to filter out infrequent errors, but must use a total-order assumption in their assignment of actions to tasks that does not hold in human SC:BW gameplay.

More generally related to our motivation, there has been some amount of work on both HTNs in real-time games and also learning from unlabeled demonstrations. Hoang et. al used HTN representations in the first person shooter game Unreal Tournament to good success, merging event-driven agents with the higher-level planning to achieve both reactiveness and strategy (Hoang, Lee-Urban, and Muñoz-Avila 2005). Another great success of HTNs in games was Bridge Baron 8, which won the 1997 computer bridge championship (Smith, Nau, and Throop 1998). While not real-time, its management of the imperfect information aspect of the game is highly relevant to the RTS genre.

While our end goal is to learn an HTN model of expert play, prior work on learning from demonstration in the RTS domain has mostly focused on working from case libraries. Weber et al. implemented a goal-driven autonomy system and extended it to use a case library from expert replays for detecting discrepancies and creating goals in (Weber, Mateas, and Jhala 2012). Additionally, while more supervised in that the demonstrations provided had partial labeling, Ontañón et al. used case-based planning to implement a successful Wargus agent based on demonstrations of a human player executing various strategies (Ontañón et al. 2010). This system has been extended in a number of ways, including towards automated plan extraction from human demonstrations in (Ontanón et al. 2009), in which the authors use plan dependency graphs to correspond actions to goals, but it still requires some amount of goal encoding from the human moderator.

Many other approaches to strategy and planning have been taken for SC:BW. A useful survey of these can be found in (Ontanón et al. 2013).

Generalized Sequential Patterns

Generalized Sequential Patterns (GSP) is a sequential pattern mining algorithm developed by (Srikant and Agrawal 1996). Its greatest appeal is in the flexibility that it affords the searcher in placing restrictions on the types of patterns to be searched for. In particular, it introduced the notion of a maximum or minimum gap between elements in the pattern, which places a hard limit on how separated consecutive elements in a pattern are allowed to (or must) be. This is useful for us, as we intend to search for short-term patterns to identify actions that are linked together in expert play. Without this gap limitation, we might identify ”Build Barracks, Train SCV, Train Tank” as a common pattern, since it would appear in nearly every Terran game (with other actions in between), while the actions themselves are not necessarily directly linked in the player's mind. Another capability offered by GSP is user-defined taxonomies, with support for patterns that include items from different levels of the tree. While we have not yet included this aspect, we do feel it will be valuable in the future.

GSP works by performing a series of scans over the data-sequences, each time searching for frequent patterns one element longer than the scan before. Given a set of frequent n-length patterns, we construct a *candidate* set of $n+1$-length patterns by searching for overlapping patterns within our frequent set (that is, a pair of patterns where the last $n-1$ elements in one matches the first $n-1$ elements in the other). We stitch these together to create a $n+1$-length pattern for the candidate set. We then search for each candidate in each sequence, to determine the amount of support and whether to add it as a frequent pattern. This approach is guaranteed to generate all frequent patterns (due to the fact that frequent patterns must be made up of frequent patterns), and in practice greatly reduces extraneous searching.

A replay of SC:BW can be seen as two sequences of actions, performed by each player. However, if we look at the full actions, we will find no overlapping patterns between games, due to the ever-present RTS problem of action-space size. Two players may move two units to minutely different locations, and these actions will not match up in a pure pattern match. As a result, we must blur our vision to some degree to find meaningful patterns. For this work, we zoomed far out, removing location information entirely from actions. Some example commands from our resultant sequences would be 'Train(Marine)', 'Build(Barracks)', or 'Move(Dragoon)'. The last is the main weakness of our abstraction, and our highest priority moving forward is to reintroduce locality information via high-level regions. Even so, the patterns that we extract are meaningful and starting points for learning goals and tasks.

To demonstrate both the GSP algorithm and our processing of replays, consider Fig. 2. Imagine that the maximum acceptable gap between pattern elements has been set at 4 seconds, and we require support from every trace to consider a pattern frequent. The initial pass will mark "Move(Probe)", "Train(Probe)", and "AttackMove(Zealot)" as frequent 1-element patterns, as they all appear in each trace. Then, every combination of these patterns will be generated as a candidate 2-element pattern, of which only "Move(Probe), Move(Probe)", "Move(Probe), Train(Probe)", and "Train(Probe), AttackMove(Zealot)" will be supported by all 3 traces. The only 3-element candidates generated are then "Move(Probe), Move(Probe), Train(Probe)" and "Move(Probe), Train(Probe), AttackMove(Zealot)", as any other 3-element pattern would have a non-frequent sub-pattern, and thus can be guaranteed to be non-frequent itself.

Of the candidates, "Move(Probe), Train(Probe), AttackMove(Zealot)" does not find support, as it cannot be satisfied in Trace 3 without using a pattern with elements more than 4 seconds apart. Therefore, we add "Move(Probe), Move(Probe), Train(Probe)" to our frequent list and terminate, as we cannot generate any 4-element candidates.

We extracted the action sequences using the fan-developed program BWChart[1]. Once the sequences were extracted from replays and preprocessed to the level of abstraction described above, we then ran the actual GSP algorithm on them. For our system, we used the open-source data mining library SPMF[2], which includes an implementation of GSP. Some small code adjustments to the SPMF implementation were required to accomodate longer sequences.

[1]available at http://bwchart.teamliquid.net/

[2]available at http://www.philippe-fournier-viger.com/spmf/

Second	**Action**	
161	Move(Probe)	**Trace 1**
162	Move(Probe)	
164	Move(Probe)	
166	Train(Probe)	
167	AttackMove(Zealot)	
168	AttackMove(Dragoon)	
388	Move(Probe)	**Trace 2**
389	Build(Gateway)	
391	Train(Probe)	
394	AttackMove(Zealot)	
402	Move(Probe)	
403	Move(Probe)	
407	Train(Probe)	
222	Move(Probe)	**Trace 3**
223	Move(Probe)	
224	Move(Probe)	
225	Move(Probe)	
226	Train(Probe)	
239	Train(Probe)	
240	AttackMove(Zealot)	
243	AttackMove(Dragoon)	
244	AttackMove(Zealot)	

Figure 2: Snippets from three replay traces that have been preprocessed into our system's format

Experiments and Results

For our experiments, we used 500 professional replays downloaded from the website TeamLiquid[3]. We focused on the Terran vs. Protoss matchup for our analysis, though it can be extended to the other 5 matchups as well. Our tests ended up splitting themselves into two categories: micro- and macro-level patterns. In the former, we ran our system as described above, with maximum gaps of 1-4 seconds, to search for actions that human players tend to chain together one immediately after the other. In the latter, we attempted to look for higher level goals and plans by removing the unit training and movement actions, leaving only the higher level strategy-directing actions: building construction and researches.

One thing to note is that we would prefer to use a larger number of replays to attain even more confidence in the mined patterns, but were restricted by system limitations. Because the GSP algorithm needs to loop through every sequence for each pattern to see if support exists, it ends up storing all sequences in memory. For StarCraft:Brood War traces, with thousands of actions, this fills up memory rather quickly. The most prevalent sequence mining application is purchase histories, which are much shorter, and therefore the algorithm implementations are generally more geared towards that problem type. That being said, a possible extension to this work would be to use a batch approach, where candidate patterns are generated per batch, then tested over the whole

[3]www.teamliquid.net

suite to determine if they are truly supported or not.

Micro-level Patterns

One type of pattern that we investigated was sequences of actions separated by small amounts of time, which we term 'micro-level patterns'. These are actions that occur frequently and immediately after one another, thereby indicating that they are linked to each other and in pursuit of the same goal.

In order to find these patterns, we ran our system allowing gaps between actions of 1, 2, and 4 seconds. In the end, there was not a qualitative difference between the results for any of these gaps, so all results shown here are using a 1 second maximum gap.

Upon examination, the mined patterns fell into three main classes: action spamming, army movement, and production cycles, examples of which are shown in Figure 3.

Action Spamming Action spamming is the habit of performing unnecessary and null operator actions purely for the sake of performing them. It is a technique often used by professional players at the beginning of a game when there are not enough units to tax their abilities, in order to warm up for the later stages of the game when they will need to be acting quickly. For the most part, these commands consist of issuing move orders to worker units that simply reinforce their current order. Since the habit is so prevalent, it is unsurprising that we find these patterns, although they are not particularly useful. If in the future their existence becomes problematic, we should be able to address the problem by eliminating null-operation actions.

Army Movement Another category of extended pattern that is frequent in the data set is that of army movement. This type of pattern is more in line with what we hope to find, as the movement of one military unit followed by another is very likely to be two primitive actions in pursuit of the same goal. Unfortunately, actually identifying the goals pursued would require more processing of the data, due to the loss of location information from our abstraction. However, we are confident that once we reintroduce this information, meaningful army movement patterns will be apparent.

Production Cycles The final micro-level pattern that shows up in our data is what we term 'production cycles'. Professional players tend to sync up their production buildings in order to reissue training commands at the same time. For example, if a Protoss player has 4 Gateways, he will likely time their training to finish at roughly the same time, so that he can queue up 4 more units at once, requiring less time for mentally switching between his base and his army. This is reflected in the patterns we find, as these Train commands tend to follow immediately after one another. This is another example of a promising grouping of primitive actions that could be translated into a complex action in the HTN space, after preconditions and postconditions had been learned.

Action Spamming
1: Move(Probe)
2: Move(Probe)
3: Move(Probe)
4: Move(Probe)
5: Train(Probe)
6: Move(Probe)
7: Move(Probe)

Army Movement
1: AttackMove(Zealot)
2: AttackMove(Zealot)
3: AttackMove(Zealot)
4: AttackMove(Dragoon)
5: AttackMove(Dragoon)
6: AttackMove(Dragoon)
7: AttackMove(Dragoon)

Production Cycle
1: Train(Dragoon)
2: Train(Dragoon)
3: Train(Dragoon)
4: Train(Dragoon)

Figure 3: A sample of frequent patterns generated by the system. The maximum gap between subsequent actions is 1 in-game second.

Macro-level Patterns

In the opening stages of SC:BW, there is very little interaction and information flow between players. As a result, a relatively small number of fixed strategies have been settled upon as accepted for the first few minutes of play. These are commonly referred to as 'build orders', and they are generally an ordained order of constructing tech buildings and researches. How long players remain in these build orders, similar to chess, is dependent upon the choice of each, and whether either player manages to disrupt the other's build with military aggression.

In order to search for high-level goals, of which build orders are the most stable example, we removed unit training and movement actions from our traces and expanded the amount of time allowed between actions to 60 seconds. With these modifications, we ended up with two main types of patterns.

The first was simple chains of production structures and supply structures. Players in SC:BW must construct supply structures in order to support new units. As a result, once the economy of a player is up and running, construction comes down to increasing training capacity and building supply structures to support additional military units. These patterns would translate well to long-term high-level goals in an HTN formulation of building up infrastructure.

The second type of pattern was what we had hoped to see, build order patterns. These were long chains of specific training, tech, and supply structures in a particular order. In order to verify these results, we compared them with the fan-moderated wikipedia at TeamLiquid, and found that each of

Build Orders

```
1: Build(SupplyDepot)
2: Build(Barracks)
3: Build(Refinery)
4: Build(SupplyDepot)
5: Build(Factory)
6: AddOn(MachineShop)

1: Build(Pylon)
2: Build(Gateway)
3: Build(Assimilator)
4: Build(CyberneticsCore)
5: Build(Pylon)
6: Upgrade(DragoonRange)
7: Build(Pylon)
```

Figure 4: Two build orders generated by our system. According to TeamLiquid, the first is a Siege Expand, "one of the oldest and most reliable openings for Terran against Protoss", while the second is a One Gate Cybernetics Core, which "can be used to transition into any kind of mid game style".

the early game patterns generated by our system was posted as a well-known and feasible build order. We feel that these patterns are the strongest of the ones found, and the most easily translated into high-level goals.

Discussion and Future work

The final goal of this work is to use the patterns found in the data to generate complex tasks for an HTN model. Given these complex tasks, we can use existing unsupervised techniques to learn preconditions and postconditions in order to create a fully functioning HTN planner for SC:BW.

Realistically, it is unlikely that a pure HTN planner learned in a completely unsupervised manner will be a highly competitive agent. In particular, it is probable that the agent will require some amount of reactive agency for the lowest level management of units. While it is certainly possible to author tasks that dictate how to plan out an engagement, we do not currently have a solution as to how to learn these sorts of tasks in an unsupervised setup. That being said, we do believe that higher level strategy and more mid level army positioning can absolutely be learned, and feel that these results back up that claim. While it is true that the build order knowledge discovered by our system has been hand-curated and already exists, the fact that it lines up so well gives us confidence in the approach.

One example of an agent that has combined reactivity and planning can be found in Ben Weber's work (Weber, Mateas, and Jhala 2012) (Weber 2012), which used a reactive planner to achieve goals generated by a GDA system. It may be the case that we learn methods for this sort of reactive planner, and match them with goals using differential state analysis across our database of replays.

There are three main directions that we hope to extend this work. The first is to reduce the amount of location abstraction that we are performing. The reasoning behind removing it for this project was the fact that different regions on different maps can be difficult to identify as performing similar roles. The starting region for each player is easily translated from map to map, and perhaps the first expansion location, but beyond that it can become difficult to say: "Region A on Map X plays a similar role as Region B on Map Y". However, we are currently working on a system to do a data-driven mapping of maps, and hope to alleviate this issue soon.

A second area of extension is to utilize the taxonomy competency given by the GSP algorithm to see if this generates even more useful patterns. Taxonomies are natural to SC:BW, a simple example would be to classify any Terran unit produced from the Barracks as 'Infantry', or to have an umbrella classification of 'Military Unit' for all non-worker units. The added structure may result in longer and/or more meaningful patterns generated.

A last goal would be to use these patterns to learn meaningful predicates for HTN methods. For example, if post-processing determined that a frequent pattern was to move 5, 6 or 7 Dragoons toward the enemy base at a time when the player owned 5, 6 or 7 Dragoons respectively, it may be the case that we can more accurately define the task being performed as Move 'all' Dragoons.

Conclusion

In conclusion, we have presented a data mining system that searches for patterns within SC:BW replays, and shown that the patterns generated are meaningful on both a micro and macro level. With this success, we intend to continue toward the motivation for the work, which is an unsupervised method for learning HTN tasks and goals from expert demonstrations.

References

Agrawal, R., and Srikant, R. 1995. Mining sequential patterns. In *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, 3–14. IEEE.

Agrawal, R.; Imieliński, T.; and Swami, A. 1993. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, 207–216. ACM.

Bosc, G.; Kaytoue, M.; Raıssi, C.; and Boulicaut, J.-F. 2013. Strategic pattern discovery in rts-games for e-sport with sequential pattern mining.

Hoang, H.; Lee-Urban, S.; and Muñoz-Avila, H. 2005. Hierarchical plan representations for encoding strategic game ai. In *AIIDE*, 63–68.

Hogg, C.; Munoz-Avila, H.; and Kuter, U. 2008. HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. In *AAAI*, 950–956.

Ontanón, S.; Bonnette, K.; Mahindrakar, P.; Gómez-Martín, M. A.; Long, K.; Radhakrishnan, J.; Shah, R.; and Ram, A. 2009. Learning from human demonstrations for real-time case-based planning.

Ontañón, S.; Mishra, K.; Sugandh, N.; and Ram, A. 2010. On-line case-based planning. *Computational Intelligence* 26(1):84–119.

Ontanón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; and Preuss, M. 2013. A survey of real-time strategy game ai research and competition in starcraft.

Smith, S. J.; Nau, D.; and Throop, T. 1998. Computer bridge: A big win for ai planning. *Ai magazine* 19(2):93.

Srikant, R., and Agrawal, R. 1996. *Mining sequential patterns: Generalizations and performance improvements*. Springer.

Weber, B. G.; Mateas, M.; and Jhala, A. 2012. Learning from demonstration for goal-driven autonomy. In *AAAI*.

Weber, B. 2012. *Integrating learning in a multi-scale agent*. Ph.D. Dissertation, UC Santa Cruz.

Yang, Q.; Pan, R.; and Pan, S. J. 2007. Learning recursive HTN-method structures for planning. In *Proceedings of the ICAPS-07 Workshop on AI Planning and Learning*.