PlanBot: A Replanning Agent for Starcraft

Michael Leece MLEECE@UCSC.EDU
Arnav Jhala AJHALA@UCSC.EDU

Abstract

State-of-the-art agents in the real-time strategy game domain currently rely on hand-crafted scripted strategies, and as a result are inflexible, responding poorly to unexpected events brought on by the actions of their adversary. A self-introspective agent with goal reasoning could overcome this challenge, allowing an agent to consider how its strategy has succeeded or failed and what must be modified as a result of this. We demonstrate such an agent in its initial development, and show that it performs well against weak to medium strength opponents, though still falls short against well-scripted play. In addition, we discuss possible extensions and research areas that can be explored with this base system.

1. Introduction

Real time strategy (RTS) games have been growing in popularity as a challenge domain for artificial intelligence research. They are abstract economic and military simulations in which players allocate resources to various infrastructure development, and attempt to steer the game toward the strengths of their choices and away from the weaknesses. As simulations, they contain many problems that arise when working in the real-world in adversarial environments, such as the need for flexible and reactive decision-making, in addition to the requirement to plan ahead and not simply always react to an opponent's actions.

Unfortunately, many of the current state-of-the-art agents use scripted logics or small state machines for high-level strategic decisions, leaving them rigid and poorly able to adapt to new situations and opponents. The progression from inflexible scripted strategies to systems that reflect human intelligence and rational decision-making has been a common sight in past challenge domains, and is where we hope to go with this work.

We have created a planning agent for the RTS game Starcraft:Brood War (see section 2), which uses replanning as a form of goal reasoning in order to adapt its plan to unexpected changes in the environment due to inaccurate beliefs about the world state or active opposition from an adversary. In the future, we hope to use this system as a base from which to explore more complex goal reasoning approaches, adversarial planning, and learning hierarchical models from demonstration.

2. StarCraft:Brood War

Starcraft:Brood War (SC:BW) is a well-known RTS game produced by Blizzard Entertainment. At a very high level, gameplay consists of allocating resources to grow an economy and military infrastructure, then using this infrastructure to train an army to destroy your opponent's troops and buildings. It can be played with up to eight players, but competitive games traditionally feature

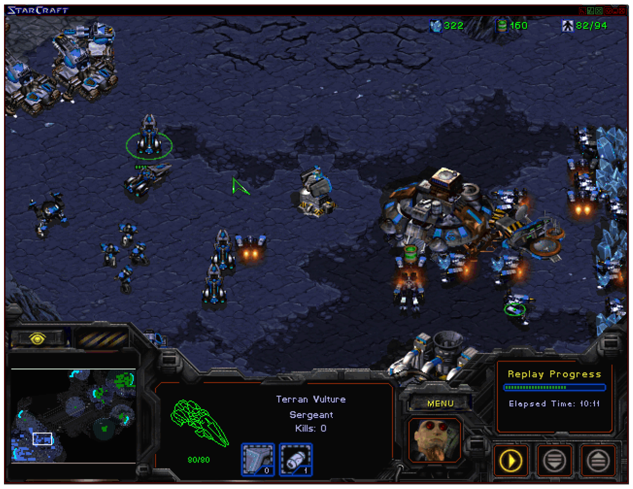

Figure 1. An example screenshot from SC:BW. Shown is a base with mining worker units, and a portion of the player's army.

two players in a head-to-head match. While seemingly simple at first, the interaction of resource management, terrain considerations, and asymmetric army compositions results in a rich space of strategies and tactics for players.

In recent years SC:BW has grown in popularity as a research testbed, due to a number of desirable properties. It is a real-time environment (or near-real-time, at 60 possible decision points per second), which is a departure from previous AI challenge games such as Chess or Go. In addition, it is a game of imperfect information. Only areas of the map that are visible to a player's units will be revealed to that player, which means that one is never completely aware of how an opponent is allocating their resources. The state and action spaces of the game are very large, a rough calculation of the state space estimating it to be $10^{11,500}$ (Weber, 2012). This calculation doesn't take into account information states, which would increase it even more.

The conjunction of these properties means that any agent that wishes to play the game well must be able to make on-the-fly decisions adapting to new information about an opponent's strategy and the world state, since it is clearly infeasible to precompute a full state transition strategy. These requirements are common to real-world problems and still challenging to artificial agents, which is exactly what we desire in a testbed.

An added benefit of this specific game is its popularity. At its peak, professional leagues existed with players competing regularly in tournaments with tens of thousands of dollars in cash rewards. In combination with the ability to save and post replays, this has resulted in large archives of expert-level play, which can be used as demonstrations for learning systems.

For possible evaluations, multiple tournaments for SC:BW agents are held each year (Churchill, 2013). Additionally, while much of the professional scene has moved on to more modern RTS games, the game remains popular enough online to have a sustainable test set of human players, with a competitive online ladder system against which an agent's performance can be measured quantitatively.

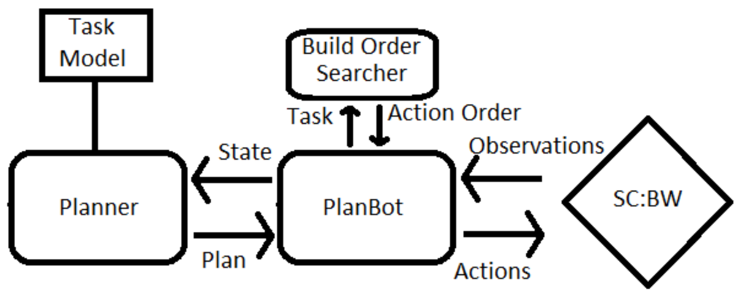

Figure 2. Information flow for PlanBot and associated components

3. System

Our system (called PlanBot) is built off of a base platform provided by UAlbertaBot (Churchill & Buro, 2012) (Churchill & Buro, 2013). We will give a brief overview of UAlbertaBot and how it operates, then explain the changes that have been made to create our agent.

3.1 UAlbertaBot

UAlbertaBot is a SC:BW agent developed at the University of Alberta. One of the main goals of the system has been to provide an entry point for developing SC:BW agents that is modular and easily modifiable. Control has been broken into a number of 'managers', which handle internal state for the agent, and 'commanders', which interact with the game itself. For example, the main actor is the 'GameCommander', which queries the 'StrategyManager' for the next set of units and buildings that the agent should train/construct. Given that set, it passes it on to a build order searcher, which calculates an order in which the set of units should be constructed, optimizing for time. Other components manage building placement, positioning for fights, and more. Each component uses its own decision-making process, including scripts, simulation, and game-tree searches. Futher information on UAlbertaBot, and the code itself can be found at (Churchill, 2015).

This modular structure, while enforcing certain relations between components, is attractive from an engineering standpoint. For example, a researcher can swap in a new high-level Strategy manager without needing to interfere with the existing decision process for building placement, if it isn't necessary. This has resulted in removing much of the software engineering burden for creating a fully-functioning SC:BW agent, which can be intimidating for researchers new to the domain, or those only interested in specific challenge areas.

3.2 PlanBot

The primary change to create PlanBot is a full replacement of the strategy module in UAlbertaBot, responsible for high-level resource allocation decisions. As mentioned in the example earlier, this part of the agent is queried by the commander module for a list of units and/or buildings that should be constructed next, and then search is performed to determine the optimal order to construct the

	Built-in AI	Bonjwa	UAlbertaBot
PlanBot	98%	72%	12%

Table 1. Winning percentages of PlanBot against a range of opponents.

	Built-in AI	Bonjwa	UAlbertaBot
PlanBot	15:54 ± 1:02	18:31 ± 0:45	10:02 ± 0:36

Table 2. Average game lengths for each opponent.

full set, taking into account prerequisites and resource collection rates. This results in a hybrid of planning and search approaches, where primitive operators in the planning model consist of sets of units to train/construct, the order of which is optimized by the search component. This boundary between search and refining the current plan is a potential point of future investigation. Originally, the strategy module returned a static choice from a set of fixed if-then evaluations of the current state. PlanBot instead uses a modified version of a Python implentation of SHOP[1] (Nau et al., 1999) and slightly modified to handle some of the extra domain complexities. The task model used for planning is handwritten by the authors, and is a candidate for future improvement of the system.

We also modify the commander module, which is responsible for coordinating the other modules, to reflect the changes in the strategic decision-making of the agent. The original agent simply queried the strategy module for a new set of buildings/units as soon as it finishes the last set it was given. Two changes were made to this interaction: first, when the agent is out of things to do and needs direction, it also passes a flag to the strategy manager indicating one of three things. If the last task was executed successfully (all units and buildings were constructed), it requests the next step in the current plan. If the last task was not executed successfully (some units and/or buildings were destroyed while the task was open), we note that this goal has not been satisfied and request for a new task that will achieve it in the current state, then continue with the plan. If the state has changed dramatically from the plan's expectations, the flag indicates that complete replanning should occur. Figure 2 shows this information flow, with the PlanBot node representing the high-level commander module and encapsulating the unchanged modules of UAlbertaBot not discussed here.

As one can see from the description above, the system's goal reasoning capabilities are currently limited to evaluation and replanning if required, while not taking advantage of the full goal lifecycle and opportunities to cycle into different parts of it based on the evaluation of a task's execution. We intend to extend the capabilities of PlanBot to include more complex goal reasoning in the future, as it will likely be necessary in such a complex domain.

4. Evaluation

Our evaluation was based on simple strength of play in head-to-head matches against other agents, and is shown in Table 1. Each opponent was played 100 times over 10 different maps. The built-in AI is, as expected, the AI that is shipped with the game. It generally uses a fairly unoptimized rushing strategy and does not attempt to grow its economy very much. There are three asymmetric races to choose from in the game, and most agents select a specific race to build for, but for the built-in AI we randomized the race, as it is roughly equivalent with each of the three.

Bonjwa is an open-source agent from the 2014 AIIDE tournament, developed by Dustin Dannenhauer. We chose it because it plays the Terran race, while UAlbertaBot (and consequently PlanBot) plays the Protoss race, and we wanted to include an asymmetric matchup with a developed agent. Furthermore, UAlbertaBot is a very strong player, and we wanted a mid-level challenge to measure against as well (according to the rankings from the last open tournament). Bonjwa was forked off an earlier version of UAlbertaBot, and therefore uses the same decision mechanisms, though it has different hand-coded strategies, as it plays a different race. The final opponent, UAlbertaBot, is the unmodified script-based strategy version.

While the results clearly indicate that PlanBot is a capable agent, scoring well against both the built-in AI and Bonjwa, it is also clear that it performs poorly against its own predecessor, winning only 12 games out of 100. In order to gain more insight into what these results actually meant, we watched a number of games from each of the matchups. In general, it seemed that PlanBot did better the longer the game progressed. UAlbertaBot generally executes an optimized rushing strategy that PlanBot was unable to hold off, while as long as it could hold off the built-in AI's initial push, it was eventually able to outmaneuver it. In light of that, we measured the average game times for each opponent, and the results, shown in Table 2, seem to support the observations.

On the bright side, this indicates that our goal of being able to adapt as the situation changes and plans need to be altered is being reflected, but it also means that we are not responding well to early pressure. In all likelihood, this is the result of an incomplete task model due to the authors' incomplete knowledge of the domain, and could be improved with iterated testing and expanding/refining. However, rather than obsessing over improving the win rate, we would like to focus on improving the general intelligence of the system, which we hope to do in the following ways.

5. Future Work

This system is intended to be a base for future research in a number of areas. As mentioned before, the goal reasoning performed by the system currently is of a fairly basic form. We would like to deepen the task evaluation functionality in the commander module to support more fine-grained responses to successful or failed tasks, and the gradations between the two. We believe there are also interesting areas to explore regarding the adversarial nature of the domain, and being able to differentiate between a task failing due to a mistaken belief about the world state or active interference from the opponent, and if or how that should affect the agent's response.

1. Code found at `https://bitbucket.org/dananau/pyhop`

We also hope to integrate some of our prior research to improve the general intelligence of the system. Human players have capabilities that are not reflected in the current agent, such as general reasoning about what one's opponent is doing and how that should affect one's own strategic choices, or the interaction between strategic choices and tactical choices. Some amount of opponent modeling (Leece & Jhala, 2014a) would give the agent a greater idea of when things are straying away from what was expected when the original plan was generated, and also increase the information available to the planner. This would also help in task evaluation, as it could be performed more accurately mid-task. In addition, we would like to make use of the archives of expert play available online to attempt to learn the task models directly, rather than needing to hand code them, which we have begun work on in (Leece & Jhala, 2014b).

Lastly, we would like to implement some consideration of adversarial planning, similar to the work found in (Meijer & Koppelaar, 2001).

6. Related Work

The strongest predecessor of this project is the work done by Weber et al. on EISBot, a SC:BW agent that used goal-driven autonomy to make its strategic decisions (Weber, 2012) (Weber, Mateas, & Jhala, 2011). Written in the reactive programming language ABL, it used a library of states from human games and case-based reasoning to identify discrepancies and set goals (Weber, Mateas, & Jhala, 2012). Alternatively, Dannenhauer and Muñoz-Avila presented a Goal-Driven Autonomy agent that uses ontologies to identify and explain discrepancies, resulting in a more human-like decision process (Dannenhauer & Muñoz-Avila, 2013). As other projects related to goal reasoning and in the same domain, we hope to compare and contrast our work with these projects moving forward.

Some other work with applying planning to real-time games can be found in (Hoang, Lee-Urban, & Muñoz-Avila, 2005). They combine a strategic planner to direct motion and positioning with a reactive state machine that controls low-level behavior when handed control from the planner. In a more real-world domain, Roberts et al. implemented a planning system that automatically generates finite state machines for heterogeneous teammates based on assigned goals from a coordinating planner (Roberts et al., 2014). This work could be viewed as similar to our system, in breaking plan components into things that are tractable to compute more directly, and will be particularly useful to keep in mind in that it has a better treatment of durative reasoning than our current system.

With regards to our other planned goals for the system, a theoretical basis for learning hierarchical models from demonstrations can be found in (Garland, Ryall, & Rich, 2001) (Garland & Lesh, 2003), and learning the CaMeL system for learning method preconditions (Ilghami et al., 2002) (Ilghami et al., 2005), although these both assumes that some of the structure in the training examples provided has been annotated. Some practical work with entirely unannotated examples, as is available to us, has been done by Hogg et al. in their work on HTN-Maker (Hogg, Munoz-Avila, & Kuter, 2008) (Hogg, Kuter, & Munoz-Avila, 2010). In addition, work has been done within the cognitive systems community itself, due to the fact that most cognitive architectures use some form of hierarchical reasoning or data storage. An example of this can be found in (Ichise, Shapiro, & Langley, 2002).

7. Conclusion

We have created a goal reasoning system for Starcraft:Brood War, a complex adversarial domain which requires flexible and adaptable reasoning. It performs well against the built-in AI, though it is still defeated by well-scripted agents. We hope to extend this agent to use more and more self-introspection regarding its task evaluation process in the future, in addition to using it for other interesting research areas.

References

Churchill, D. (2013). 2013 AIIDE starcraft ai competition report. http://webdocs.cs.ualberta.ca/∼cdavid/starcraftaicomp/ report2013.shtml. Accessed: 2015-04-14.

Churchill, D. (2015). Ualbertabot - starcraft ai competition bot. https://github.com/davechurchill/ualbertabot. Accessed: 2015-03-16.

Churchill, D., & Buro, M. (2012). Incorporating search algorithms into rts game agents. *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*.

Churchill, D., & Buro, M. (2013). Portfolio greedy search and simulation for large-scale combat in starcraft. *Computational Intelligence in Games (CIG), 2013 IEEE Conference on* (pp. 1–8).

Dannenhauer, D., & Muñoz-Avila, H. (2013). Luigi: A goal-driven autonomy agent reasoning with ontologies. *ACS: Workshop on Goal Reasoning*.

Garland, A., & Lesh, N. (2003). Learning hierarchical task models by demonstration. *Mitsubishi Electric Research Laboratory (MERL), USA–(January 2002)*.

Garland, A., Ryall, K., & Rich, C. (2001). Learning hierarchical task models by defining and refining examples. *Proceedings of the 1st international conference on Knowledge capture* (pp. 44–51).

Hoang, H., Lee-Urban, S., & Muñoz-Avila, H. (2005). Hierarchical plan representations for encoding strategic game ai. *AIIDE* (pp. 63–68).

Hogg, C., Kuter, U., & Munoz-Avila, H. (2010). Learning methods to generate good plans: Integrating htn learning and reinforcement learning. *AAAI*.

Hogg, C., Munoz-Avila, H., & Kuter, U. (2008). Htn-maker: Learning htns with minimal additional knowledge engineering required. *AAAI* (pp. 950–956).

Ichise, R., Shapiro, D., & Langley, P. (2002). Learning hierarchical skills from observation. *Discovery Science* (pp. 247–258).

Ilghami, O., Munoz-Avila, H., Nau, D. S., & Aha, D. W. (2005). Learning approximate preconditions for methods in hierarchical plans. *Proceedings of the 22nd international conference on Machine learning* (pp. 337–344).

Ilghami, O., Nau, D. S., Munoz-Avila, H., & Aha, D. W. (2002). Camel: Learning method preconditions for htn planning. *AIPS* (pp. 131–142).

Leece, M., & Jhala, A. (2014a). Opponent state modeling in rts games with limited information using markov random fields. *Computational Intelligence and Games (CIG), 2014 IEEE Conference on* (pp. 1–7).

Leece, M. A., & Jhala, A. (2014b). Sequential pattern mining in starcraft: Brood war for short and long-term goals. *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Meijer, A., & Koppelaar, H. (2001). Pursuing abstract goals in the game of go. *BNAICâĂŹ01 Sponsors*.

Nau, D., Cao, Y., Lotem, A., & Muñoz-Avila, H. (1999). Shop: Simple hierarchical ordered planner. *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2* (pp. 968–973).

Roberts, M., Vattam, S., Alford, R., Auslander, B., Karneeb, J., Molineaux, M., Apker, T., Wilson, M., McMahon, J., & Aha, D. W. (2014). Iterative goal refinement for robotics. *ICAPS Workshop on Planning and Robotics*.

Weber, B. (2012). *Integrating learning in a multi-scale agent*. Doctoral dissertation, UC Santa Cruz.

Weber, B. G., Mateas, M., & Jhala, A. (2011). Building human-level ai for real-time strategy games. *AAAI Fall Symposium: Advances in Cognitive Systems*.

Weber, B. G., Mateas, M., & Jhala, A. (2012). Learning from demonstration for goal-driven autonomy. *AAAI*.