Using Automated Replay Annotation for Case-Based Planning in Games

Ben G. Weber[1] and Santiago Ontañón[2]

[1] Expressive Intelligence Studio
University of California, Santa Cruz
`bweber@soe.ucsc.edu`
[2] IIIA, Artificial Intelligence Research Institute
CSIC, Spanish Council for Scientific Research
`santi@iiia.csic.es`

Abstract. A major challenge in the field of case-based reasoning is building case libraries representative of the state space the system will encounter. We approach this problem by automating the process of converting expert demonstrations, in the form of game replays, into cases. To achieve this, we present a technique for annotating traces with goals that can be used by a case-based planner. We have implemented this technique in the case-based planning system Darmok 2 and applied it to the task of playing complete games of StarCraft. By automating the process of case generation, we enable our system to harness the large number of expert replays available on the web.

1 Introduction

Video games provide an excellent testbed for research in case-based reasoning. Games are increasingly providing data that can be used to automate the process of building game AI, such as replays. Harnessing this data is challenging, because game replays contain only game state and actions performed by players and do not contain a player's goals or intentions. Utilizing this data in agents that reason about goals is problematic, because there is often a gap between the game actions contained in a replay and the agent's goals. This problem becomes apparent in complex games in which players reason about the game at multiple levels of granularity while performing tasks.

Real-time Strategy (RTS) games in particular are an interesting domain for evaluating AI techniques [1], because they provide several challenges for building game AI. The decision complexity of RTS games is huge [2], and requires simultaneously reasoning about both strategic and tactical goals. RTS games are also real-time environments in which agents must react to events including second-by-second world changes at the tactical level, as well as exogenous events at the strategic level, such as an opponent switching strategies midgame.

There has been recent interest in addressing these challenges by building agents that learn from demonstration, such as replays [3, 4]. The goal is to develop techniques that automatically acquire domain knowledge by analyzing examples of gameplay from skilled human players. Building agents that learn from

replays poses several challenges: defining a suitable representation for encoding game state, extracting cases automatically from replays, specifying similarity metrics for retrieval of cases, modeling the domain in order for the agent to reason about goals, and supporting real-time execution in the game environment. Previous work has addressed the issue of case extraction by either requiring manual annotation of replays [3], or building hierarchical plans based on dependencies between actions [5].

In this paper, we present a technique for annotating replays for use in case-based planning. Our approach automates the process of extracting cases from replays by labeling the actions performed in replays with the goals being pursued. This enables the use of real-world data for generating huge case libraries. We have implemented this technique within the Darmok 2 framework [5] using the real-time strategy game StarCraft as our application domain.

2 Related Work

Applying case-based planning for building game AI requires formally modeling the domain. Goal oriented action planning (GOAP) is a planning-based approach to building AI for non-player characters in games [6]. In GOAP architectures, a character has a set of goals that become activated based on a set of criteria. Upon activation, a plan is generated to achieve the goal and then executed in the game world. The main challenge in implementing this approach is defining a suitable world representation and operators for building plans in real-time. Our system differs from this approach, because Darmok 2 performs case-based planning, while GOAP architectures utilize generative planning. Another aspect of applying planning to game AI is specifying goals for the agent to pursue. An approach implemented in the RTS game Axis & Allies is triggering goal formulation when specific game events occur, such as capturing an enemy city [7]. Darmok 2 does not explicitly perform goal formulation, but incorporates it in the subgoaling process of plan retrieval.

In the domain of RTS games, two approaches have been utilized to build game AI with case-based reasoning: systems that learn online, and systems that bootstrap the learning process by generating an initial case library from a set of replays. The first approach has been applied to strategy selection in Wargus [2] and micro-management in WarCraft III [8]. The second approach has been applied to building a complete game playing agent for Wargus [3] and build order specifically in Wargus [4]. Our system differs from previous work in that we are using a much larger case library in order to manage the complexity of StarCraft.

There has also been related work on annotating replays for RTS games. Weber and Mateas applied data mining to predicting an opponent's strategy in StarCraft [9]. Their approach labeled replays with specific strategies and represented strategy prediction as a classification problem. Metoyer et al. analyzed how players describe strategies while playing Wargus and identified several patterns [10]. There is also a large online community supporting StarCraft that

manually annotates replays by identifying specific strategies [11] and providing high-level commentary[3].

3   StarCraft

StarCraft[4] is a science fiction RTS game in which players manage an economy, produce units and buildings, and vie for control of the map with the goal of destroying all opponents. Real-time strategy games (and StarCraft in particular) provide an excellent environment for AI research, because they involve both low-level tactical decisions and high-level strategic reasoning. At the strategic level, StarCraft requires decision-making about long-term resource and technology management, while at the tactical level, effective gameplay requires both micro-management of individual units in small-scale combat scenarios and squad-based tactics such as formations.

StarCraft is an excellent domain for evaluating decision making agents, because there are many complex tradeoffs. One of the main tradeoffs in StarCraft is selecting a "build order", which defines the initial actions to perform in the game and is analogous to chess openings. There is no dominant strategy in StarCraft and a wide variety of build orders are commonly executed by top players: economic-heavy build orders focus on setting up a strong resource infrastructure early in the game, while rush-based strategies focus on executing a tactical assault as fast as possible. The most effective build order for a given match is based on several factors, including the map, opponent's race, and opponent's predicted strategy. An agent that performs well in this domain needs to include these factors into its strategy selection process.

Another tradeoff in StarCraft is deciding where to focus attention. StarCraft gameplay requires simultaneously managing high-level strategic decisions (macro-management) with highly-reactive actions in tactical scenarios (micro-management). Players have a finite amount of time and must decide where to focus their attention during gameplay. This is also an issue for game-playing agents, because decisions must be made in real time for both macro-management and micro-management actions.

StarCraft has a vast decision complexity, not just because of the game state, but also due to the number of viable strategies in the strategy space. One of the interesting aspects of StarCraft is the level of specialization that is applied in order to manage this complexity. Professional StarCraft players select a specific race and often train for that race exclusively. However, there is large amount of general knowledge about StarCraft gameplay, and it provides an excellent domain for transfer learning research (e.g. adapting strategies learnt for one race to another race).

Our choice of StarCraft as a domain has additional motivating factors: despite being more than 10 years old, the game still has an ardent fanbase, and there is

---

[3] http://www.gomtv.net

[4] StarCraft and StarCraft: Brood War were developed by Blizzard Entertainment[TM]

even a professional league of StarCraft players in South Korea[5]. This indicates that the game has depth of skill, and makes evaluation against human players not only possible, but interesting.

4 Darmok 2

Darmok 2 (D2) [5] is a real-time case-based planning system designed to play RTS games. D2 implements the *on-line case-based planning* cycle (OLCBP) as introduced in [3]. The OLCBP cycle attempts to provide a high-level framework to develop case-based planning systems that operate on-line, i.e. that interleave planning and execution in real-time domains. The OLCBP cycle extends the traditional CBR cycle by adding two additional processes, namely *plan expansion* and *plan execution*. The main focus of D2 is to explore learning from unannotated human demonstrations, and the use of adversarial planning techniques. The most important characteristics of D2 are:

- It acquires cases by analyzing human demonstrations.
- It interleaves planning and execution.
- It uses an efficient transformational plan adaptation algorithm for allowing real-time plan adaptation.
- It can use a *simulator* (if available) to perform adversarial planning.

D2 learns a collection of cases by analyzing human demonstrations (traces). Demonstrations in D2 are represented as a list of triples $[\langle t_1, S_1, A_1 \rangle, ..., \langle t_n, S_n, A_n \rangle]$, where each triple contains a time stamp t_i game state S_i and a set of actions A_i (that can be empty). The set of triples represent the evolution of the game and the actions executed by each of the players at different time intervals. The set of actions A_i represent actions that were issued at t_i by any of the players in the game. The game state is stored using an object-oriented representation that captures all the information in the state: map, players and other entities (entities include all the units a player controls in an RTS game: e.g. tanks).

Each case $C = \langle P, G, S \rangle$ consists of a plan P (represented as a *petri-net*), a goal G, and a game state S. A case states that, in a game state S, the plan P managed to achieve the goal G. An example case can be seen in Figure 1. Plans in cases are represented in D2 as *petri-nets* [12]. Petri nets offer an expressive formalism for representing plans that can have conditionals, loops or parallel sequences of actions (in this paper we will not use loops). In short, a petri net is a graph consisting of two types of nodes: *transitions* and *states*. Transitions contain conditions, and link states to each other. Each state might contain *tokens*, which are required to fire transitions. The distribution of tokens in the petri net represent its status. For example, in Figure 1 there is only 1 token in the top-most state, stating that none of the actions has executed yet.

When D2 executes a plan contained in a case, the actions in it are adapted to fit the current situation. Each action contains a series of parameters referring to
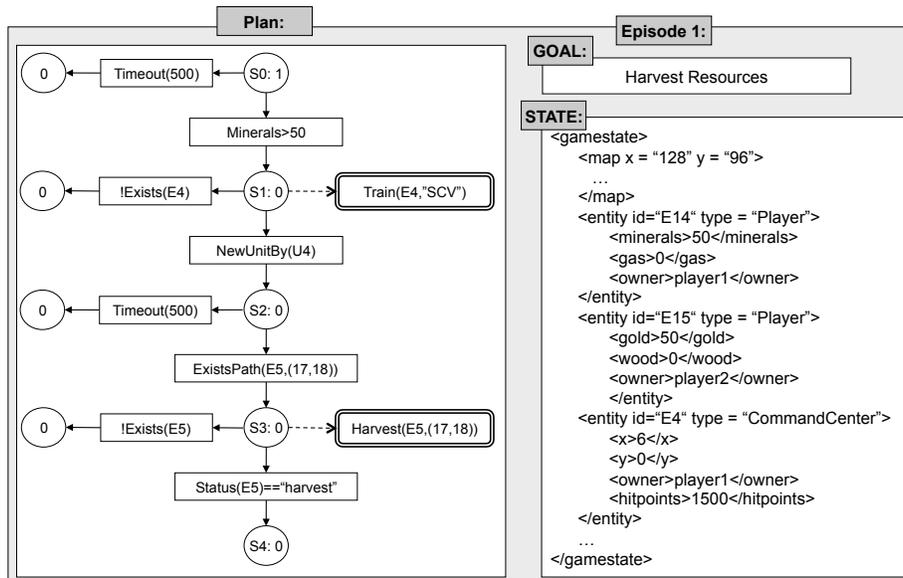

Fig. 1. A case in D2 consisting of a plan, goal and game state. The snippet contains two actions and the game state representation is not fully included due to space limitations.

locations or units in the map, and other constants. For example, if a parameter refers to a location, then a location in the current map which is the most similar to the location specified in the action is selected. For assessing location similarity, D2 creates a series of potential fields (one for each entity type in the game, in the case of StarCraft: friendly-marines, enemy-marines, friendly-tanks, enemy-tanks, minerals, etc.). Each location in the map, is thus assigned a vector, which has one value for each potential field. This allows D2 to compare map locations and assess which ones are more similar to others. For example, it can detect that a location is similar to another because both are very close to enemy units.

In previous work [5] we explored a technique based on ideas similar to those of the HTN-maker [13] in order to automatically learn cases from expert demonstrations (traces). This strategy lets D2 effectively learn from traces automatically without requiring an expert to annotate the traces as previous work required [3, 14]. However, it relies on the assumption that each action the expert executed was carefully selected, and that each condition that becomes true during the game as an effect of the executed actions was intended by the expert. Since those conditions are too restrictive, sometimes the system learns spurious plans for goals that were achieved only accidentally.

In this paper, we explore an alternative approach which exploits goal recognition techniques [15]. Our approach incorporates an expert provided goal ontology to automate the process of case extraction. While it requires the application of additional domain knowledge, it enables the extraction of cases that achieve

high-level goals, rather than grouping action sequences based only on unit dependencies. High quality annotations are important, since they provide a means for D2 to break up a trace into smaller pieces, which will constitute cases.

5   Replay Annotation

Our goal is to make use of real-world StarCraft replays in order to build a case library for D2. Our approach to achieve this goal requires automating the trace annotation process, because manual annotation is impractical for large datasets and presents language barriers for StarCraft, because the majority of professional StarCraft players are non-English speakers. By automating the process of annotating traces, we enable D2 to make use of the large amount of StarCraft traces available on the web.

There are several challenges faced when utilizing real-world examples. First, unlike previous work in Wargus [4, 5], we were unable to specify a trace format for storing examples. The replay format we use is a proprietary binary format specified by Blizzard and we have no control over the data that is persisted. Second, real-world replays are noisy and contain non-relevant actions, such as spamming of actions. For example, in previous work [3] the traces used for learning were carefully created with the purpose of learning from demonstration, and did not have any noise. Third, our StarCraft traces contain large numbers of actions, because players execute hundreds of actions per minute, which can result in traces containing over a thousand actions. To overcome these challenges, we developed a technique for automatically annotating replays, which is used to break up the replays into cases which are usable by D2.

To automatically annotate actions in game traces with goals, we defined a goal ontology for StarCraft and developed a rule set for recognizing when goals are being pursued. To build our case library, we extracted action logs from replays and labeled actions with a set of goals. The result of this approach is annotated game traces that can be used to build a case library for D2.

5.1   Goal Ontology

Our approach utilizes a goal ontology which is used to specify the goals that are being pursued by actions in a game trace. The goal annotations are equivalent to tasks in a HTN planner [5] and are used by the case-based planner to decompose the goal of winning the game into subgoals, such as the goal of setting up a resource infrastructure. The case retrieval process incorporates goal similarity as well as game state similarity when performing retrieval. In order to effectively make use of OLCBP, the goal ontology should model the domain at the level at which players reason about goals.

The goal ontology was formulated based on analysis of professional StarCraft gameplay [11] as well as previous work [16], and is shown in Figure 2. The economy subgoal contains tasks that achieve the goal of managing a resource infrastructure, while the strategy subgoal contains tasks that achieve the goal

- Win StarCraft
  - Economy
    * Produce worker units
    * Harvest resources
    * Build resource facilities
    * Manage supply
  - Strategy
    * Build production facilities
    * Build tech building
    * Produce combat units
    * Research upgrade
  - Tactics
    * Attack opponent
    * Use tech ability or spell

Fig. 2. Our goal ontology for actions in StarCraft

of expanding the tech tree and producing combat units. The ontology is not specific to a single StarCraft race, and could be applied to other games such as Wargus. It decomposes the task of playing StarCraft into the subgoals of managing the resource infrastructure, expanding the tech tree and producing combat units, and performing tactical missions. The main difference between our ontology and previous work is more of a focus on the production of units, rather than the collection of resources, because players tend to follow rules of thumb for resource gathering and spend resources as soon as they are available. Additionally, we grouped the expansion of the tech tree and production of combat units into a shared subgoal, to manage contention of in-game resources.

Each goal in the ontology has a set of parameters that are used to evaluate whether the given goal is applicable to the current game situation. Upon retrieval of a subgoal, D2 first queries for goals that can be activated, and then searches for cases that achieve the selected goal. Each subgoal contains parameters that are relevant to achieving that specific goal. For example, the *economy* goal takes as input the following parameters: the current game time, the number of units controlled by the agent, the maximum number of units the agent can support, the number of worker units, expansions and refineries, and the number of worker units harvesting gas.

5.2 Case Extraction

The system uses a two part process in which cases are extracted from traces and then annotated with goals. In the first part of the process, actions in a trace are placed into different categories based on the subgoal they achieve and then grouped into cases based on temporal locality. For example, an attack command issued in StarCraft would be placed into the *tactics* category and grouped into

a case with other attack actions that occur within a specific time period. The second part of the process is the actual annotation phase in which cases are labeled based on the game state of the first action occurring in the case. Our approach currently groups actions into three categories, which correspond to the direct subgoals of the *Win StarCraft* goal in the ontology. Different techniques are used for grouping actions into cases for the different categories.

The *economy* and *strategy* categories group actions into cases using a model based on goal recognition, which exploits the linear structure of gameplay traces [15]. Given an action at time t in a trace, we can compute the game state at time $t + n$ by retrieving it directly from the trace, where n is the number of actions that have occurred since time t. The game state at $t + n$, S_{t+n}, can be inferred as the player's goal at time t, given the player has a plan length of size n. This model of goal recognition relies on the assumption that the player has a linear plan to achieve a specific economic or strategic goal. This assumption is supported by what is referred to as a "build order" in StarCraft, which is a predefined sequence of actions for performing a specific opening.

Cases are built by iterating over the actions in a trace and building a new case every n actions using the following algorithm:

$$C.S = S_t$$
$$C.P = petri\_net(A_t, A_{t+1}, \ldots, A_{t+n})$$
$$C.G = category.G$$
$$C.G.params = compute\_params(S_{t+n})$$

where C is the generated case, S is the game state, P is a petri-net plan, $petri\_net$ is a method for building a petri net from a sequence of actions, A is an action performed by the player in the trace, $category.G$ is the *economy* or *strategy* subgoal, $C.G.params$ is the goal parameters, and $compute\_params$ is a goal specific function for computing the goal parameters given a game state. For *economy* cases, n was set to 5 to allow the agent to react to the opponent, while for *strategy* cases, n was set to 10 to prevent the agent from dithering between strategies [15]. This approach can result in cases with overlapping actions.

The *tactics* category groups actions into cases using a policy similar to the previous approach, but groups actions based on the game time at which they occur, rather than the index in the trace. The motivation for this approach to grouping actions is based on the tendency of players to rapidly perform several attack actions when launching a tactical assault. A new case is created each time an attack issue is ordered to a unit where the distance to the attack location is above a threshold. Given an attack action, A_i, occurring at cycle t, subsequent attack actions are grouped into a case using the following policy:

$$C.P = petri\_net(A_i, \ldots, A_j)$$
$$A_j = \max \{A_n | A_n.gameframe \leq (t + delay)\}$$

where $game\_frame$ is the time at which attack A_n occurred, in game cycles, and $delay$ is a threshold for specifying how much of a delay to allow between the initial attack command and subsequent commands.

6 Experimental Evaluation

We implemented our approach for automated replay annotation in the StarCraft domain. To build a case library, we first collected several replays from professional players. Next, we ran the replays in StarCraft and exported traces with complete game information, i.e. we generated D2 traces from the replay files. Finally, we ran the case extraction code to build a library of cases for D2 to utilize.

D2 was interfaced with StarCraft using BWAPI [6], which enables the querying of game state and the ability to issue orders to units. D2 communicates with StarCraft through BWAPI using sockets. This interface enables D2 to keep a synchronized state of the game world and provides a way for D2 to perform game actions.

We performed an initial evaluation with a case library generated from four traces to explore the viability of the approach. While D2 was able to set up a resource infrastructure and begin expanding the tech tree, the system is currently unable to defeat the built in AI of StarCraft, which performs a strong rush strategy. A full evaluation of this technique is part of our future work.

7 Conclusion and Future Work

We have presented a technique for automating the process of annotating replays for use by a case-based planner. This technique was applied to replays mined from the web and enables the generation of large case libraries. In order to annotate traces with goals, we first defined a goal ontology to capture the intentions of a player during gameplay. We then discussed our approach to breaking up actions in a trace into cases and how we label them. Our system was evaluated by collecting several StarCraft replays and converting them into cases usable by D2. D2 was then applied to the task of playing complete games of StarCraft.

While our system hints at the potential of our approach, there are several research issues to address. For example, the potential field technique used by D2 to adapt examples to the current game situation provides a domain independent mechanism for adaptation. However, there are several specific instances in StarCraft where it may be necessary to hand author the adaptation functionality. Some units in StarCraft have several different uses, and determining the intended result of an action may require the application of additional domain knowledge.

Another aspect of related work is expanding the goal ontology to include a larger number of player goals. For example, a player attacking an opponent may be pursuing one of several goals: destroying an expansion, harassment of worker units, pressuring the opponent, gaining map control, or reducing the opponent's army size. Increasing the granularity of the goal ontology will require more sophisticated techniques for labeling traces.

---

[6] http://code.google.com/p/bwapi/

Finally, as part of our future work, we plan to study scaling-up issues related to using case-based planning systems in complex domains such as StarCraft with a large collection of complex cases.

References

1. Buro, M.: Real-Time Strategy Games: A New AI Research Challenge. In: Proceedings of the International Joint Conference on Artificial Intelligence. (2003) 1534–1535
2. Aha, D.W., Molineaux, M., Ponsen, M.: Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game. Lecture notes in computer science **3620** (2005) 5–20
3. Ontanón, S., Mishra, K., Sugandh, N., Ram, A.: On-Line Case-Based Planning. Computational Intelligence **26**(1) (2010) 84–119
4. Weber, B., Mateas, M.: Case-Based Reasoning for Build Order in Real-Time Strategy Games. In: Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference, AAAI Press (2009) 106–111
5. Ontañón, S., Bonnette, K., Mahindrakar, P., Gómez-Martín, M., Long, K., Radhakrishnan, J., Shah, R., Ram, A.: Learning from Human Demonstrations for Real-Time Case-Based Planning. IJCAI Workshop on Learning Structural Knowledge from Observations (2009)
6. Orkin, J.: Applying Goal-Oriented Action Planning to Games. In: AI Game Programming Wisdom 2. Charles River Media, S. Rabin editor (2003) 217–228
7. Dill, K., Papp, D.: A Goal-Based Architecture for Opposing Player AI. In: Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference, AAAI Press (2005)
8. Szczepański, T., Aamodt, A.: Case-based reasoning for improved micromanagement in real-time strategy games. In: Proceedings of the ICCBR 2009 Workshop on CBR for Computer Games. (2009)
9. Weber, B., Mateas, M.: A Data Mining Approach to Strategy Prediction. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games, IEEE Press (2009) 140–147
10. Metoyer, R., Stumpf, S., Neumann, C., Dodge, J., Cao, J., Schnabel, A.: Explaining How to Play Real-Time Strategy Games. Research and Development in Intelligent Systems XXVI (2010) 249–262
11. Team Liquid: Liquipedia: The StarCraft Encyclopedia (April 2010) http://wiki.teamliquid.net/starcraft.
12. Murata, T.: Petri nets: Properties, Analysis and Applications. Proceedings of the IEEE **77**(4) (1989) 541–580
13. Hogg, C., Munoz-Avila, H., Kuter, U.: HTN-MAKER: Learning HTNs with Minimal Additional Knowledge Engineering Required. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence. (2008)
14. Könik, T., Laird, J.E.: Learning goal hierarchies from structured observations and expert annotations. Mach. Learn. **64**(1-3) (2006) 263–287
15. Weber, B., Mateas, M., Jhala, A.: Case-Based Goal Formulation. In: Proceedings of the AAAI Workshop on Goal-Driven Autonomy. (2010)
16. Ontañón, S., Mishra, K., Sugandh, N., Ram, A.: Learning from Demonstration and Case-Based Planning for Real-Time Strategy Games. Soft Computing Applications in Industry (2008) 293–310