A Case Study of Goal-Driven Autonomy in Domination Games

Hector Munoz-Avila[1] **and David W. Aha**[2]

[1]Department of Computer Science & Engineering;
Lehigh University; Bethlehem, PA 18015
[2]Navy Center for Applied Research in Artificial Intelligence;
Naval Research Laboratory (Code 5514); Washington, DC 20375
munoz@cse.lehigh.edu | david.aha@nrl.navy.mil

Abstract

In this paper we present an overview of a variety of efforts to attain good performance of Game AI for Domination games. These are a very popular kind of game sub-genre in which teams compete for control of locations. Due to the rules of the games, good performance is mostly dependant on overall strategy rather than the skill of individual team members.

Introduction

Domination (DOM) games are played in a turn-based environment in which two teams (of *bots*) compete to control specific locations called *domination points*. Teams compete to be the first to earn a predefined number of points. Domination games have been used in a variety of games genres, including first-person shooters (e.g., Half-life®), role-playing games (e.g., World of Warcraft®), and third-person shooters (e.g., Halo®). Domination games are popular because they reward team effort rather than an individual bot's performance. Domination games are *non-deterministic*; if a bot is told to go to a domination location the outcome is uncertain because the bot may be killed along the way. Domination games are also *adversarial*; two or more teams compete to control the domination points. Finally, domination games are *imperfect information* games; a team only knows the locations of those opponent bots that are within the range of view of one of the team's own bots. These conditions make domination games a good testbed for evaluating algorithms that integrate planning and execution.

Over the years we have devised methods that integrate planning and execution for playing domination games. Table 1 shows a summary of the three algorithms which we will discuss in this paper: HTNBots, GDA-HTNBots, and CB-gda. HTNBots uses hierarchical task network (HTN) representation techniques to generate new plans (Hoang *et al*., 2005). It monitors the current situation in the game; when the circumstances change, it generates new plans on the fly. GDA-HTNBots uses HTNBots' method to generate plans but, instead of generating new plans when the situation in the game changes, it follows a goal-driven autonomy (GDA) process to generate new goals, which are then given as input to HTNBots to generate new plans (Muñoz-Avila *et al*., 2010a). The third system is CB-gda (Muñoz-Avila *et al*., 2010b). CB-gda uses a library of cases that record plans and their expectations, which are conditions expected to be fulfilled when the plans are executed. These plans are executed and the resulting state is compared against the expectations. If there is a mismatch, then a new case is retrieved.

Table 1: Three systems for playing domination games

Game AI	Description
HTNBots	Replanning algorithm. Uses HTN planning techniques to generate plans on-the-fly.
GDA-HTNBots	Instead of replanning techniques, this uses a GDA architecture to generate new goals.
CB-gda	Removes HTNBots altogether. A case base records plans and expectations from executing those plans.

We conducted a study that compares these three approaches. We report on comparisons among the knowledge requirements for each system and a summary of their performance results.

DOM: A Generic Domination Game Environment

We built a game environment, called DOM, which captures the essence of domination games (Auslander *et al*., 2008). The basic rules in DOM are the following: Each time a bot on team *t* passes over a domination point, that point will belong to *t*. Team *t* receives one point for every five seconds that it owns a domination point. Teams

compete to be the first to earn a predefined number of points. No awards are given for killing an opponent team's bot, which *respawns* immediately in a location selected randomly from a set of map locations, and then continues to play. A location is captured by a team whenever one of its bots moves on top of the location and within the next five game ticks no bot from another team moves on top of that location.

Because of the large number of possible game states in DOM, we follow the state abstraction model of Auslander *et al*., (2008) which simply tracks ownership of domination points, and to which domination points bots are sent. This abstraction reduces the number of states to $d^{(t+1)}$, and the number of actions to $(b \times t)^d$ where d is the number of domination points, t the number of teams, and b the number of bots. One is added to the exponent to account for neutral ownership of domination points at the beginning of each game. The total number of possible states in the game is at least $O(2 \times 10^{34})$ assuming a standard map of 70×70 cells, 4 domination locations, and 3 bots per team (Gillespie *et al*., 2010).

The Game AI Systems

We briefly summarize each of the three algorithms we investigated that integrate planning and execution for playing DOM. For further details of these algorithms please see the references.

HTNBots. HTNBots uses HTN planning as the main inferencing algorithm to generate plans (Hoang et al., 2005; Muñoz-Avila & Hoang, 2006). HTNs decompose high-level tasks into simpler tasks. There are two kinds of tasks: compound and primitive. ***Compound tasks*** can be further decomposed into subtasks whereas ***primitive tasks*** cannot. The primitive tasks denote concrete actions. Each level in an HTN adds detail on how to achieve the high-level tasks. The sequencing of the leaves in a fully expanded HTN yields the plan for achieving the high-level tasks. In the context of game AI the decompositions can be used to encode game strategies and the leaves to actual in-game actions such as patrol, attack, etc. To cope with the dynamic nature of DOM, we use standard finite state machines (FSMs) encoded in Java to execute the actions, but we extend them so they can also perform the primitive tasks assigned by the HTN, such as going to a certain waypoint. As a result, a grand strategy is laid out by the HTNs and the FSMs allow the bots to react in this highly dynamic environment while contributing to the overall task.

GDA-HTNBots. GDA-HTNBots extends HTNBots by adding goal-driven autonomy capabilities (Muñoz-Avila *et al*., 2010a). Unlike HTNBots, GDA-HTNBots reasons about its goals, and can dynamically formulate which goal it should plan to satisfy. GDA-HTNBots extends HTNBots as follows: HTNs are annotated so that when a plan p is generated the planner's expectations X for executing p (i.e., the states expected to result from the p's execution) are also computed. The first of the two crucial modules in GDA-HTNBots is the ***Discrepancy Detector***, which continuously monitors the plan's execution by comparing at time t the observations of state s_t with the expected state x_t. If it detects any discrepancy d_t (i.e., a *mismatch*) between them, then it is used to identify an explanation e_t (which is computed based on the history of the game by counting the number of times agents from the opposing team have visited each location). The second crucial module, the ***Goal Formulator***, uses a set of rules of the following form (where e is an explanation and g is a goal):

if e then g

This directs GDA-HTNBots to achieve the new goal g.

CB-gda. CB-gda uses case-based reasoning (CBR) for attaining goal-driven autonomy (Muñoz-Avila *et al*., 2010b). CB-gda uses two case bases as inputs: the planning case base and the mismatch-goal case base. The planning case base (PCB) is a collection of tuples of the form (s_c, g_c, e_c, pl), where s_c is the observed state of the world (formally, this is defined as a list of atoms that are true in the state), g_c is the goal being pursued (formally, a goal is a predicate with a task name and a list of arguments), e_c is the state that the agent expects to reach after accomplishing g_c starting from state s_c, and pl is a plan that achieves g_c. The mismatch-goal case base (MCB) is a collection of pairs of the form (m_c, g_c), where m_c is the mismatch (the difference between the expected state e_c and the actual state s_c) and g_c is the goal to try to accomplish next. In a nutshell, when playing the game, CB-gda retrieves a case c from PCB whose state s_c is most similar to the current state of the game. The case c's plan is then executed. CB-GDA monitors the current state after a period of time. If there is a mismatch m between the current state and case c's expected state, then a case in MCB is retrieved with a similar mismatch m_c and the corresponding goal g_c becomes the system's new goal, which is then used to retrieve a new case in PCB, thus closing the loop.

Knowledge Representation Requirements

Each of the three systems has their own knowledge representation requirements and some comparisons can be made:

GDA-HTNBots requires a larger knowledge engineering effort than HTNBots. HTNBots uses the SHOP HTN planning algorithm (Nau *et al*., 1999). The knowledge artifacts needed as input in SHOP are called methods and operators. A *method* encodes how to achieve a compound task. Methods consists of three elements: (1) The task being achieved, called the *head* of the method, (2)

the set of preconditions indicating the conditions that must be fulfilled for the method to be applicable, and (3) the subtasks needed to achieve the head. The second knowledge artifacts are the operators. An *operator* has preconditions and effects. The preconditions indicate the conditions that must be valid for the operator to be applicable. The effects indicate how the current situation changes as a result of applying the operator. For playing DOM, HTNBots methods represent strategies of how to win domination games (e.g., control half plus one of the domination locations) and operators denote concrete actions to be taken in the game (e.g., send a bot to a specific domination location).

GDA-HTNBots also requires the methods and operators for generating workable plans to play DOM games. In our current implementation GDA-HTNBots use the same methods as HTNBots. In addition, GDA-HTNBots requires methods to be annotated with the expected states from executing the methods, the knowledge about how to generate explanations (which in our implementation includes knowledge about the potential opponent's tactics), and the rules mapping explanations to the next goal to achieve.

CB-gda knowledge acquisition requirements. CB-gda's knowledge acquisition requirements are to fill the two case bases PCB and MCB. The number of cases depends on the similarity metric used and the adaptation procedure for the retrieved plans. In our current implementation we use a fine similarity metric (only very similar cases are retrieved) and the retrieved case's plan is executed without any modification. As a result, we have more than 1000 cases. A natural advantage of the case-based reasoning approach is the modularity of the knowledge required. Richter (1995) identifies containers for the four kinds of knowledge needed in a CBR system: vocabulary, the similarity measure, the case base and the adaptation rules. This modularity is believed to facilitate the knowledge engineering effort; rather than, for example, encode a set of HTN methods that encompass every situation, cases can be captured as they occur.

In-Game Performance

In Muñoz-Avila *et al*. (2010a; 2010b) we reported on experiments where the performance metric is the number of points obtained versus an opponent. Games were played until the first team reached a predefined number of points. We used 6 static opponents, two of which were considered easy opponents, two intermediate, and the final two hard. This classification is the result of our extensive testing on this domain with several AI opponents.

CB-gda has the best performance versus the difficult static opponents. CB-gda was able to beat all six static opponents in the two maps we tested. This is particularly significant because it beats ***smart opportunistic***. The smart opportunistic opponent sends agents to each domination location that its team doesn't own. If possible, it will send multiple agents to each such location. It always selects the bot that is the closest to the location it wants to control. This makes it difficult for AI agents to beat and in fact smart opportunistic beat both HTNBots (by around a 19% score difference) and GDA-HTNBots (5%). All of these results were statistically significant. We conjecture that CB-gda's better performance is due to the flexibility of the case-based reasoning process that allows it to find suitable cases that address every situation encountered. Obviously, part of this flexibility was dependent on the more than 1000 cases that were manually input to the system.

GDA-HTNBots outperformed HTNBots on the most difficult opponents. GDA-HTNBots has a better performance than HTNBots on the most difficult opponents. We mentioned that both were beaten by smart opportunistic but that GDA-HTNBots' scores were more similar to HTNBots' scores. Furthermore, GDA-HTNBots was able to beat –albeit by a margin of less than 1%- the other hard opponent, whereas HTNBots was soundly beaten. We attribute this difference to the additional knowledge about goal-driven autonomy that by GDA-HTNBots leverages in contrast to the simple replanning process executed by HTNBots.

HTNBots has the best performance versus the easy and medium static opponents. All three AI systems were able to beat the easy and medium-difficulty opponents but the score difference was generally larger for HTNBots than for the two GDA systems. For example, versus one of the medium-difficulty opponents the score difference was 58% in HTNBots' favor whereas for GDA-HTNBots it was 12% and for CB-gda it was 46%. The reason for this is that during goal formulation both GDA variants use simple rules/cases to map explained discrepancies to specific goals. However, not all discrepancies require goal formulation (e.g., some should be ignored). A more sophisticated Discrepancy Resolver could reason that a given discrepancy does not warrant goal formulation, in which case it would continue pursuing the same goal as selected by HTNBots and, hence, would achieve the same performance as HTNBots.

CB-gda has the best performance among the AI Systems. CB-gda was able to outperform HTNBots and GDA-HTNBots in direct, one-on-one competition. This again points towards the flexibility of having a large case base that allows CB-gda to quickly counter any situation encountered.

Final Remarks

There are several related research efforts. For a comprehensive discussion, please refer to Muñoz-Avila *et al*. (2010a; 2010b). Briefly, Cox's (2007) investigation of self-aware agents inspired the conception of GDA, with its

focus on integrated planning, execution, and goal reasoning. An alternative to GDA is *contingency planning* (Dearden *et al*., 2003), in which the agent plans in advance for plausible contingencies, and *conformant planning* (Goldman & Boddy, 1996), where the generated plan is guaranteed to succeed.

In summary, we presented three alternative techniques for reasoning in dynamic environments. The first one is HTNBots, a replanning system based on HTN planning. HTNBots executes a plan until an action becomes inapplicable. At this point, the replanning agent simply generates a new plan from the current state to achieve its goals. The difference between replanning and GDA is that replanning agents retain their goals while GDA agents can reason about which goals should be satisfied. GDA agents reason about goals. We discussed two variants of GDA: a rule-based variant (GDA-HTNBots) and a case-based variant (CB-gda). Among these variants, CB-gda demonstrated the best performance, provided that it was given proper sets of cases.

Acknowledgements

This work was sponsored by DARPA/IPTO and NSF (#0642882). Thanks to PM Michael Cox for providing motivation and technical direction. The views, opinions, and findings contained in this paper are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of DARPA or the DoD.

References

Auslander, B., Lee-Urban, S., Hogg, C., & Muñoz-Avila, H. (2008). Recognizing the enemy: Combining reinforcement learning with strategy selection using case-based reasoning. *Proceedings of the Ninth European Conference on Case-Based Reasoning (*pp. 59-73). Trier, Germany: Springer.

Cox, M.T. (2007). Perpetual self-aware cognitive agents. *AI Magazine*, **28**(1), 32-45.

Dearden R., Meuleau N., Ramakrishnan S., Smith, D., & Washington R. (2003). Incremental contingency planning. In M. Pistore, H. Geffner, & D. Smith (Eds.) *Planning under Uncertainty and Incomplete Information: Papers from the ICAPS Workshop*. Trento, Italy.

Gillespie, K., Karneeb, J., Lee-Urban, S., & Muñoz-Avila, H. (2010). Imitating inscrutable enemies: Learning from stochastic policy observation, retrieval and reuse. To appear in *Proceedings of the Eighteenth International Conference on Case-Based Reasoning*. Alessandria, Italy: Springer.

Goldman, R., & Boddy, M. (1996). Expressive planning and explicit knowledge. *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems* (pp. 110-117). Edinburgh, UK: AAAI Press.

Hoang, H., Lee-Urban, S., & Muñoz-Avila, H. (2005). Hierarchical plan representations for encoding strategic game AI. *Proceedings of the First Conference on Artificial Intelligence and Interactive Digital Entertainment.* Marina del Ray, CA: AAAI Press.

Muñoz-Avila, H., & Hoang, H. (2006). Coordinating teams of bots with hierarchical task network planning. In S. Rabin (Ed.) *AI Game Programming Wisdom 3*. Boston, MA: Charles River Media.

Muñoz-Avila, H., Aha, D.W., Jaidee, U., Klenk, M., & Molineaux, M. (2010a). Applying goal directed autonomy to a team shooter game. *Proceedings of the Twenty-Third Florida Artificial Intelligence Research Society Conference* (pp. 465-470). Daytona Beach, FL: AAAI Press.

Muñoz-Avila, H., Jaidee, U., Aha, D.W., & Carter, E. (2010b). Goal directed autonomy with case-based reasoning. To appear in *Proceedings of the Eighteenth International Conference on Case-Based Reasoning*. Alessandria, Italy: Springer.

Nau, D.S., Cao, Y., Lotem, A., & Muñoz-Avila, H. (1999). SHOP: Simple hierarchical ordered planner. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (pp. 968-973). Stockholm: AAAI Press.

Richter, M.M. (1995). *The knowledge contained in similarity measures*. Invited talk at the First International Conference on Case-Based Reasoning.