

Applying Goal Driven Autonomy to a Team Shooter Game

Héctor Muñoz-Avila¹, David W. Aha², Ulit Jaidee¹, Matthew Klenk², & Matthew Molineaux³

¹Department of Computer Science & Engineering; Lehigh University; Bethlehem, PA 18015

²Navy Center for Applied Research in Artificial Intelligence; Naval Research Laboratory (Code 5514); Washington, DC 20375

³Knexus Research Corporation; Springfield, VA 22153

munoz@cse.lehigh.edu | david.aha@nrl.navy.mil | ulj208@lehigh.edu | klenk@aic.nrl.navy.mil | matthew.molineaux@knexusresearch.com

Abstract

Dynamic changes in complex, real-time environments, such as modern video games, can violate an agent's expectations. We describe a system that responds competently to such violations by changing its own goals, using an algorithm based on a conceptual model for *goal driven autonomy*. We describe this model, clarify when such behavior is beneficial, and describe our system (which employs an HTN planner) in terms of how it partially instantiates and diverges from this model. Finally, we describe a pilot evaluation of its performance for controlling agent behavior in a team shooter game. We claim that the ability to self-select goals can, under some conditions, improve plan execution performance in a dynamic environment.

Introduction

AI researchers have repeatedly acknowledged the limiting assumptions of classical algorithms for automated planning (Ghallab *et al.*, 2004; Nau, 2007). For example, they assume *static environments* (where changes in the environment are due only to the planner's actions), *off-line planning* (where the planner does not monitor execution), and that all goals are fixed/unchanging. Naturally, these assumptions do not always apply. For example, team shooter games are *dynamic environments* that are populated by multiple agents resulting in exogenous events. Also, the agents must perform *online planning* by executing their plans during the game. Finally, the goals of the game change as the game state changes (e.g., if a win is infeasible, then the agent should attempt to gain a draw).

We present a system, GDA-HTNbots, which reasons about the events occurring in its environment, changes its own goals in response to them, and replans to satisfy these changed goals. To do this, GDA-HTNbots constantly monitors its environment for unexpected changes and dynamically formulates a new goal when appropriate.

In this paper, we briefly summarize related research, review a conceptual model for *goal driven autonomy*

(GDA) (Klenk *et al.*, 2010), describe GDA-HTNbots as a partial instantiation of this model, and present a limited empirical study of its performance. The results support our primary claim: agent performance in a team shooter domain with exogenous events can, for some conditions, be improved through appropriate self-selection of goals.

Related Work

Plan generation is the problem of generating a sequence of actions that transform an initial state into some desired state (Ghallab *et al.*, 2004). GDA-HTNbots controls plan generation in two ways: first, it determines when the planner must start working on a new goal. Second, it determines what goal the planner should attempt to satisfy.

A considerable amount of research exists on relaxing the assumptions of classical planning. For example, *contingency planning* permits dynamic environments (Dearden *et al.*, 2003). Agents that use this approach create a plan that assumes the most likely results for each action, and generate contingency plans that, with the help of monitoring, are executed only if a plan execution failure occurs at some anticipatable point(s).

Another assumption of classical planning concerns the set of goals that the agent is trying to achieve. If no plan exists from the initial state that satisfies the given goals, then classical planning fails. *Partial satisfaction planning* relaxes this all-or-nothing constraint, and instead focuses on generating plans that achieve some "best" subset of goals (i.e., the plan that gives the maximum trade-off between total achieved goal utilities and total incurred action cost) (van den Briel *et al.*, 2004).

While these approaches each relax an important assumption of classical planning, neither addresses how to respond to unexpected events that occur during execution. One straightforward solution is *incremental planning*, which plans for a fixed time horizon. After plan execution, these planners then generate plans for the next horizon. This process iterates until the goal state is reached. Another approach is *dynamic replanning*, which monitors the plan's execution. If it is apparent that the plan will fail, the planner will replan from the current state. For example,

HOTRiDE (Ayan *et al.*, 2007) employs this strategy for non-combatant evacuation planning. These approaches can also be combined. For example, CPEF (Myers, 1999) incrementally generates plans to achieve air superiority in military combat and replans when unexpected events occur during execution (e.g., a plane is shot down).

However, these approaches do not perform *goal formulation*; they continue trying to satisfy the current goal, regardless of whether their focus should dynamically shift towards another goal (due to unexpected events).

Fortunately, some other recent research has addressed this topic. For example, Coddington and Luck (2003) bestowed agents with *motivations*, which formulate goals in response to thresholds on specific state variables (e.g., if a rover's battery charge falls below 50%, then a goal of full battery charge will be formulated (Meneguzzi & Luck, 2007)). Here we adopt an alternative rule-based approach whose antecedents can match to complex games states.

Research on game AI takes a different approach to goal formulation in which specific states lead directly to behaviors (i.e., sequences of actions). This approach is implemented using *behavior trees*, which are prioritized topological goal structures that have been used in HALO 2 and other high profile games (Champanand, 2007). Behavior trees, which are restricted to fully observable environments, require substantial domain engineering to anticipate all events. GDA can be applied to partially observable environments by using explanations that provide additional context for goal formulation.

Cox's (2007) work inspired the conception of GDA, with its focus on integrated planning, execution, and goal reasoning. We extend this concept here in multiple ways, including by embedding it in the context of adversarial gaming environments.

Goal Driven Autonomy

Goal-driven autonomy is a process for online planning in autonomous agents (Klenk *et al.*, 2010). Figure 1 illustrates how GDA extends Nau's (2007) model of online planning. The GDA model primarily expands and details the scope of the *Controller*, which interacts with a *Planner* and a *State Transition System* Σ (an execution environment). We present only a simplified version of this model, and our system is only a partial implementation of this model.

System Σ is a tuple (S, A, V, γ) with states S , actions A , exogenous events V , and state transition function $\gamma: S \times (A \cup V) \rightarrow 2^S$, which describes how the execution of an action or the occurrence of an event transforms the environment from one state to another. For example, given an action a in state s_i , γ returns the updated state s_{i+1} .

The Planner receives as input a planning problem (M_Σ, s_0, g_c) , where M_Σ is a model of Σ (the environment), s_0 is the current state, and $g_c \in G$ is a goal that can be satisfied by some set of states $S_g \subseteq S$. The Planner outputs a plan $p_c = \{A_c, X_c\}$, which is a sequence of actions $A_c = [a_c, \dots, a_{c+n}]$ paired with a sequence of expectations $X_c = [x_c, \dots, x_{c+n}]$. Each $x_i \in X_c$ is a set of state constraints corresponding to

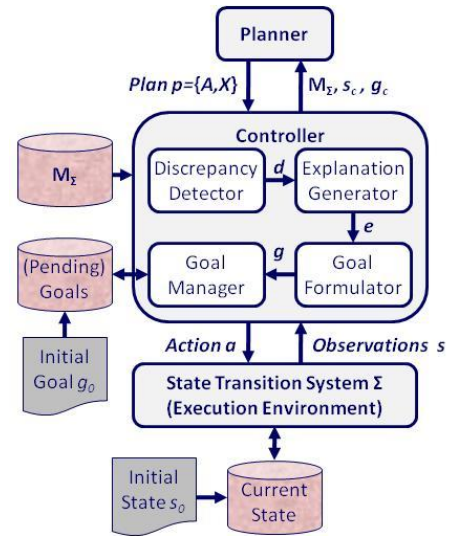


Figure 1: A Conceptual Model for Goal Driven Autonomy

the sequence of states $[s_c, \dots, s_{c+n}]$ expected to occur when executing A_c in s_c using M_Σ .

The Controller sends the plan's actions to Σ and processes the resulting observations. The GDA model takes as input initial state s_0 , initial goal g_0 , and M_Σ , and sends them to the Planner to generate a plan p_0 consisting of action sequence A_0 and expectations X_0 . When executing A_0 , the Controller performs the following four knowledge-intensive tasks, which distinguish the GDA model:

1. *Discrepancy detection*: This compares the observations s_c obtained from executing action a_{c-1} in state s_{c-1} with the expectation $x_c \in X$ (i.e., it tests whether any constraints are violated, corresponding to unexpected observations). If one or more discrepancies $D_c \subseteq D$ is found, then they are given to the following function.
2. *Explanation generation*: Given a state s_c and a set of discrepancies $D_c \subseteq D$, this hypothesizes one or more explanations $e_c \in E$ of D_c 's cause(s), where e is a belief about (possibly unknown) aspects of s_c or M_Σ .
3. *Goal formulation*: This creates a goal $g_c \in G$ in response to a set of discrepancies D_c , given their explanation $e_c \in E$ and the current state $s_c \in S$.
4. *Goal management*: Given a set of existing/pending goals $G' \subseteq G$ (one of which may be the focus of the current plan execution) and a new goal $g_c \in G$, this may update G' to create G'' (e.g., by adding g_c and/or deleting/modifying other pending goals) and will select the next goal $g' \in G''$ to be given to the Planner. (It is possible that $g = g'$.)

GDA makes no commitments to specific types of algorithms for the highlighted tasks, and treats the Planner as a black box. This description of GDA's conceptual model is necessarily incomplete due to space constraints. For example, it does not describe the reasoning models used by Tasks 1-4 (each of which may perform substantial inferencing) nor how they are obtained, it assumes multiple plans are not simultaneously executed, and it does not

address goal management issues such as goal prioritization or *goal transformation* (Cox & Veloso, 1998). However, this description should suffice to frame the general model, which we use to implement our system, GDA-HTNBots.

The Domination Game (DOM)

In this paper, we describe a simple GDA system and its application to controlling a set of agents' actions in a team shooter game, called DOM. In DOM games, two teams (of multiple agents) compete over specific locations in the game world called *domination locations*. A team receives a point for every five seconds that each domination location remains under its control (i.e., when the only agents in that location are members of their team). The game is won by the first team to score a pre-specified number of points.

Domination games are interesting because individual deaths have no direct impact on the final score; any agent that is killed will continue playing after a short pause, starting in a new location (this is called *respawning*). This allows for an overall strategy and organization to have a large impact on game play.

DOM is a good test for GDA systems because its environment is dynamic: the world changes due to the opponent's actions, which our system cannot predict. Also, some actions in the game are non-deterministic (e.g., adjudicating when members of two teams shoot each other). Hence, our system must react to unexpected events and dynamically generate new plans that satisfy different goals. Manually engineering these plans a priori is difficult, and infeasible for suitably complex task environments. GDA removes the need to create these beforehand by providing an agent with the ability to reason about its goals and dynamically determine which to pursue.

A Goal Driven Autonomy System

GDA-HTNBots is an extension of HTNBots (Hoang *et al.*, 2005) in which the Controller performs the four tasks of the GDA model. HTNBots uses SHOP (Nau *et al.*, 1999) to generate game-playing strategies for DOM based on an external hierarchical task network (HTN). These strategies are designed to control a majority of the domination locations in the game world. Whenever the situation changes (i.e., when the owner of a domination location changes), HTNBots generates a new plan. Therefore, HTNBots is a dynamic replanning system. It calls SHOP to find the first method that is applicable to a given task, and uses it to generate subtasks that are recursively decomposed by other methods into a sequence of actions to be executed in the environment.

Unlike HTNBots, GDA-HTNBots reasons about its goals, and can dynamically formulate which goal it should plan to satisfy. GDA-HTNBots extends HTNBots to instantiate the GDA conceptual model as follows:

State Transition System (Σ) (task environment): We apply GDA-HTNBots to the task of controlling an agent playing

DOM. We described this task and game environment in the preceding section, and describe an example of this application in the next section.

Model of the State Transition System (M_Σ): We describe the state transition function for DOM using SHOP axioms and operators. Exogenous events are not directly modeled in SHOP. HTNBots play DOM by monitoring the game state and replan as needed.

Planner: GDA-HTNBots uses SHOP, although other planners can be used. Given the current state s_c (initially s_0), current goal g_c (initially g_0), and M_Σ , it will generate an HTN plan p_c designed to achieve g_c when executed in Σ starting in s_c . This plan includes the sequence of expectations X_c determined by the HTN's methods that are anticipated from its execution.

Discrepancy Detector: This continuously monitors p_c 's execution in Σ such that, at any time t , it compares the observations of state s_t provided by Σ with the expected state x_t . If it detects any discrepancy d_t (i.e., a *mismatch*) between them, then it outputs d_t to the Explanation Generator.

Explanation Generator: Given a discrepancy d_t for state s_t , this generates an explanation e_t of d_t . GDA-HTNBots tracks the history of the game by counting the number of times agents from the opposing team have visited each location. Using this information and the discrepancy d_t , GDA-HTNBots identifies an explanation e_t , which is the strategy that the opponent is pursuing.

Goal Formulator: Given an explanation e_t representing the opponent's current strategy, GDA-HTNBots formulates a goal g_t using a set of rules of the form:

if e then g

The new goal g_t directs GDA-HTNBots to counter the opponent's strategy.

Goal Manager: GDA-HTNBots employs a trivial goal management strategy. Given a new goal g_t , it immediately selects this as the current goal, which the Controller submits to the Planner for plan generation.

Example in the DOM Game

In this paper, we report on a case study in which the system's task is to control a team of agents in DOM. Figure 2 shows an example of a map in a domination game with five locations. Our scenario began with the following initial state and goal:

Initial State (s_0): This includes the locations of all the agents in the game and which team (if any) controls each domination location.

Initial Goal (g_0): The initial goal is to win the game (i.e., be the first to accumulate 20,000 points). GDA-HTNBots sends g_0 to SHOP, which generates a plan to dispatch GDA-HTNBots' agents to each domination location and control them. Given the uncertainties about the opponent's actions and the stochastic outcome of engagements, this

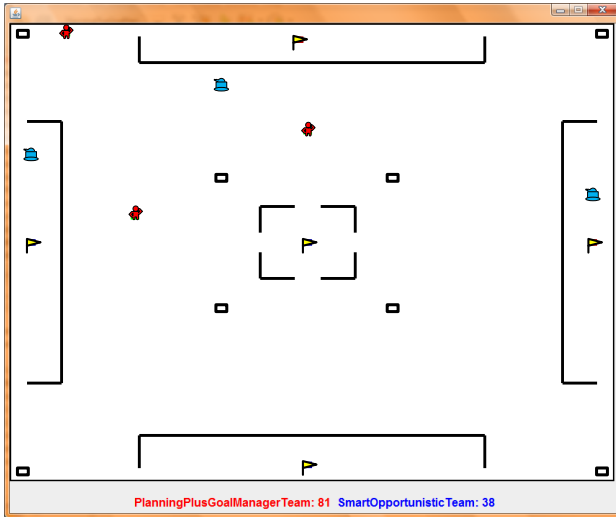


Figure 2: An example DOM game map with five domination locations (yellow flags), where small rectangles identify the respawning locations for the agents and the remaining two types of icons denote each player’s agents.

Table 1: Example explanations of discrepancies (with some expectations and observations shown), and the corresponding recommended goals.

Discrepancy	Explanation	Next Goal
$x_t: \text{Loc}(\text{bot3}, \text{loc2})$ $s_t: \neg \text{Loc}(\text{bot3}, \text{loc2})$	Defended(loc2)	$\text{Loc}(\text{bot3}, \text{loc1})$
$x_t: \text{OwnPts}(t) > \text{EnemyPts}(t)$ $s_t: \text{OwnPts}(t) < \text{EnemyPts}(t)$	EnemyCtrl(loc1) EnemyCtrl(loc2)	OwnCtrl(loc2)

plan may not yield the expected results. For example, Table 1 presents some sample explanations for the DOM game (we do not display the full state due to space constraints). The first row highlights a situation where the bot3 agent was expected to be at location 2, but this did not happen. By examining the history of enemy agents at that location, GDA-HTNbots assumes the opponent is executing a strategy to heavily defend location 2. Using the explanation goal rule set, GDA-HTNbots counters this strategy by setting a goal to have bot3 at an alternative location, namely location 1.

The second row shows a discrepancy where GDA-HTNbots expected to, over the last time period t , earn more points than the enemy. However, this did not happen because the enemy controlled two of three (total) locations. The rule set determines that the next goal should be to control one of the locations controlled by the opponent (e.g., loc2). Given this, our system generates a plan to send two agents to location 2.

This example illustrates how GDA-HTNbots explains discrepancies by reasoning about the opponent’s strategies. This enables GDA-HTNbots to formulate goals that counter the opponent’s actions.

Table 2: Domination Teams and Descriptions

Opponent Team	Description	Diff.
Dom1 Hugger	Sends all agents to domination location 0	trivial
First Half Of Dom Locations	Sends an agent to the first half +1 domination locations. Extra agents patrol between the 2 locations	easy
2 nd Half Of Dom Locations	Sends an agent to the second half +1 domination locations; extra agents patrol between the two locations	easy
Each Agent to One Dom	Each agent is assigned to a different dom. loc. and remains there for the entire game	med.-easy
Smart Opportunistic	Sends agents to each dom. loc. the team doesn’t own; if possible, it will send multiple agents to each unowned location	hard
Greedy Distance	Each turn the agents are assigned to the closest domination loc. they do not own	hard

Pilot Evaluation

We conducted a pilot study to assess the performance utility of HTNbots’ GDA enhancements. We claim that GDA increases our system’s ability to win DOM games versus a set of opponents. To test this claim, we performed an ablation study that isolates the GDA functionality.

In particular, we compared the performances of GDA-HTNbots and HTNbots. Hoang et al. (2005) and Muñoz-Avila and Hoang (2006) reported that HTNbots performs well versus several hard-coded opponents. Thus, HTNbots should provide a good baseline for our evaluation. However, we expected GDA-HTNbots would outperform HTNbots for opponents whose behaviors motivate the dynamic formulation of new goals.

We recorded and compared the performance of these systems versus the same set of hard-coded opponents. Our performance metric is the difference in the score between the system and opponent while playing DOM, divided by the system’s score.

We ran both systems against each of the six opponents summarized in Table 2. The first three were the same used by Hoang et al. (2005) to test HTNbots, which was found to perform well on them. Hence, these are challenging DOM opponents for testing whether GDA enhancements can improve HTNbots’ performance. The final three opponents were created in subsequent studies of HTNbots to test reinforcement learning (Smith *et al.*, 2007) and case-based reasoning (Auslander *et al.*, 2008) algorithms. Among these, the final two opponents were found to be particularly difficult to beat. In summary, these opponents form a challenging and varied testbed to measure the utility of GDA-HTNbots.

The experimental setup was as follows: Both systems were tested versus each of these opponents on the map shown in Figure 2. This is the same map that was used in the previously mentioned experiments. Each game was run three times to account for the randomness introduced by non-deterministic game behaviors.

Table 3: Avg. Percent Normalized Difference in Game AI System vs. Opponent Scores (with avg. scores in parentheses)

Opponent Team (controls enemies)	Game AI System (controls friendly forces)	
	HTNbots	GDA-HTNbots
Dom1 Hugger	81.2% † (20,002 vs. 3,759)	80.9% (20,001 vs. 3,822)
First Half Of Dom Locations	47.6% (20,001 vs. 10,485)	42.0% (20,001 vs. 11,605)
2 nd Half Of Dom Locations	58.4% (20,003 vs. 8,318)	12.5% (20,001 vs. 17,503)
Each Agent to One Dom	49.0% (20,001 vs. 10,206)	40.6% (20,002 vs. 11,882)
Smart Opportunistic	-19.4% (16,113 vs. 20,001)	-4.8% (19,048 vs. 20,001)
Greedy Distance	-17.0% (16,605 vs. 20,001)	0.4% (19,614 vs. 19,534)

†**Bold face** denotes the better average measure in each row

The results are shown in Table 3, where each row displays the normalized average difference in scores (computed over three games) versus each opponent. It also shows the average scores for each player. We repeated the same experiment with a second map and obtained results consistent with the ones discussed here.² The limited number of trials in this pilot study prevents us from computing statistical significance. Therefore, we focus our discussion on general trends and game analysis.

Discussion

The results can be summarized as follows: Against difficult opponents (the final two opponents in Table 2), GDA-HTNbots outperforms HTNbots. Against easy opponents (the first four listed in Table 2) HTNbots outperforms GDA-HTNbots. We examined game-play records to investigate why this occurred, and concluded that the initial strategy chosen by HTNbots is frequently sufficient to win the game. For example, the Dom1 Hugger (opponent) team sends all agents to one location. It is easy for HTNbots to immediately generate a winning plan against this strategy and start winning from the outset. Indeed, in situations where the goals should not be changed, this implementation of GDA should not be used.

The more difficult opponents reason about the distance between the agent locations and the domination locations as part of their strategy. These strategies are particularly effective versus HTNbots and GDA-HTNbots, which encode their knowledge symbolically without metric information. Indeed, the two hard opponents soundly defeat HTNbots. The advantage of using a specialized component to reason about goals becomes apparent in this study. By tracking which domination locations the opponent is trying to control and which goal was used to generate the current plan, GDA-HTNbots can react quickly to the opponent’s strategy. This allowed GDA-HTNbots to

² For the second map we didn’t obtain results for greedy distance because of some path finding issues.

outperform the Greedy Distance opponent (which outperformed HTNbots) and almost perform as well as the Smart Opportunistic opponent.

GDA-HTNbots is simple implementation of the four GDA tasks. In future research, we plan to study other methods for each of these tasks. First, during discrepancy detection, GDA-HTNbots observes discrepancies only between discrete variables. A more complete system would consider the continuous attributes of the environment (e.g., precise agent locations, health, and score). It would also represent and reason about relations among these attributes. Second, the explanation generation process allows the system to consider reasons for a given discrepancy. In this paper, our system does not dynamically derive the explanation using a comprehensive reasoning mechanism, and it only considers the opponent’s strategy. In open domains, it is important to hypothesize new entities and events that were not represented in M_{Σ} . This capability would substantially diverge from current approaches to online planning. Third, during goal formulation, GDA-HTNbots uses simple rules to map explained discrepancies to specific goals. However, not all discrepancies require goal formulation (e.g., some should be ignored). A more complete system should reason about alternative responses to detected discrepancies, such as by reasoning about game state information. This is the reason why GDA-HTNbots underperformed versus the easier opponents compared to HTNbots; if GDA-HTNbots had a more sophisticated Discrepancy Resolver, it 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. Finally, GDA-HTNbots performs only goal replacement during goal management. This was effective for DOM because it does not require balancing various goals and focuses solely on defeating an individual opponent. In more complex environments, a list of pending goals must be maintained, and the system will need to consider which goal(s) to pursue at any given time by reasoning about resources, tradeoffs, and priorities.

In future work, we plan to run more trials, which will allow us to statistically analyze our claims. In addition, we will investigate other methods for the four GDA tasks. We also envision ways in which the background knowledge used in the GDA process can be learned automatically. Finally, we will examine the behavior of GDA systems for tasks in other complex domains that have a greater need to reason dynamically about which goals to satisfy (e.g., the TAO Sandbox (Auslander *et al.*, 2009), which is a simulator used to train Naval officers for decision making in anti-submarine warfare missions).

Final Remarks

Plan generation is the problem of generating a sequence of actions that transform the initial state into some goal state (Ghallab *et al.*, 2004). Goal driven autonomy (GDA) enhances plan generation by addressing two important

questions. First, it answers *when*: it determines when the planner should start working on a new goal. Second, it answers *what*: it determines which goal should be satisfied in response to a detected discrepancy. Our initial implementation of GDA illustrates this concept. GDA-HTNbots can interrupt the execution of a strategy based on an earlier goal, and instead generate and execute a plan that achieves a different goal that now has higher priority.

Our implementation barely scratches the surface, as it is a simple instantiation of the GDA conceptual model. However, our investigation with GDA-HTNbots illustrates its potential. The DOM game, when played against challenging opponents, has all of the elements targeted by GDA: the environment is dynamic, it requires game AI agents to conduct online planning, and it must constantly change its goals to perform well (versus the more challenging opponents). Furthermore, the baseline system, HTNbots, performed well on this DOM task. Hence, GDA-HTNbots' performance improvement versus the more challenging opponents is encouraging, and we look forward to assessing a more complete GDA model's performance in future work.

Acknowledgements

Thanks to our reviewers. 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.
- Auslander, B., Molineaux, M., Aha, D.W., Munro, A., & Pizzini, Q. (2009). *Towards research on goal reasoning with the TAO Sandbox* (Technical Report AIC-09-155). Washington, DC: Naval Research Laboratory, Navy Center for Applied Research on Artificial Intelligence.
- Ayan, N.F., Kuter, U., Yaman F., & Goldman R. (2007). HOTRiDE: Hierarchical ordered task replanning in dynamic environments. In F. Ingrand, & K. Rajan (Eds.) *Planning and Plan Execution for Real-World Systems – Principles and Practices for Planning in Execution: Papers from the ICAPS Workshop*. Providence, RI.
- van den Briel, M., Sanchez Nigenda, R., Do, M.B., & Kambhampati, S. (2004). Effective approaches for partial satisfaction (over-subscription) planning. *Proceedings of the Nineteenth National Conference on Artificial Intelligence* (pp. 562-569). San Jose, CA: AAAI Press.
- Chamandard, A. (2007). Behavior trees for next-gen game AI. In *Proceedings of the Game Developers Conference*. Lyon, France.
- Coddington, A.M., & Luck, M. (2003). Towards motivation-based plan evaluation. *Proceedings of the Sixteenth International FLAIRS Conference* (pp. 298-302). Miami Beach, FL: AAAI Press.
- Cox, M.T. (2007). Perpetual self-aware cognitive agents. *AI Magazine*, **28**(1), 32-45.
- Cox, M.T., & Veloso, M.M. (1998). Goal transformations in continuous planning. In M. desJardins (Ed.), *Proceedings of the Fall Symposium on Distributed Continual Planning* (pp. 23-30). Menlo Park, CA: AAAI Press.
- 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.
- Ghallab, M., Nau, D.S., & Traverso, P. (2004). *Automated planning: Theory and practice*. San Mateo, CA: Morgan Kaufmann.
- 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.
- Klenk, M., Molineaux, M., & Aha, D.W. (2010). Goal directed autonomy for flexible planning and acting. Manuscript submitted for publication.
- Meneguzzi, F.R., & Luck, M. (2007). Motivations as an abstraction of meta-level reasoning. *Proceedings of the Fifth International Central and Eastern European Conference on Multi-Agent Systems* (pp. 204-214). Leipzig, Germany: Springer.
- 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.
- Myers, K.L. (1999). CPEF: A continuous planning and execution framework. *AI Magazine*, **20**(4), 63-69.
- Nau, D.S. (2007). Current trends in automated planning. *AI Magazine*, **28**(4), 43-58.
- 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.
- Smith, M., Lee-Urban S., & Muñoz-Avila, H. (2007). RETALIATE: Learning winning policies in first-person shooter games. *Proceedings of the Seventeenth Innovative Applications of Artificial Intelligence Conference* (pp. 1801-1806). Vancouver, (BC) Canada: AAAI Press.