Goal-Driven Autonomy for Responding to Unexpected Events in Strategy Simulations

Matthew Klenk[1*], Matt Molineaux[2], and David W. Aha[1]
[1]Navy Center for Applied Research in Artificial Intelligence;
Naval Research Laboratory (Code 5514); Washington, DC 20375
[2] Knexus Research Corporation; Springfield, VA 22153
matthew.klenk@parc.com | david.aha@nrl.navy.mil | matthew.molineaux@knexusresearch.com

Abstract

To operate autonomously in complex environments, an agent must monitor its environment and determine how to respond to new situations. To be considered intelligent, an agent should select actions in pursuit of its goals, and adapt accordingly when its goals need revision. However, most agents assume that their goals are given to them; they cannot recognize when their goals should change. Thus, they have difficulty coping with the complex environments of strategy simulations that are continuous, partially observable, dynamic, and open with respect to new objects. To increase intelligent agent autonomy, we are investigating a conceptual model for goal reasoning called *Goal-Driven Autonomy* (GDA), which allows agents to generate and reason about their goals in response to environment changes. Our hypothesis is that GDA enables an agent to respond more effectively to unexpected events in complex environments. We instantiate the GDA model in ARTUE (Autonomous Response to Unexpected Events), a domain-independent autonomous agent. We evaluate ARTUE on scenarios from two complex strategy simulations, and report on its comparative benefits and limitations. By employing goal reasoning, ARTUE outperforms an off-line planner and a discrepancy-based replanner on scenarios requiring reasoning about unobserved objects and facts and on scenarios presenting opportunities outside the scope of its current mission.

Keywords: Goal-Driven Autonomy, Autonomous Agents, On-line Planning, Goal Reasoning

1. Introduction

Modern video games and training simulations are *complex* environments: they are continuous, partially observable, dynamic, and open with respect to the introduction of new objects. To operate autonomously in these environments, intelligent agents must continuously perform situation assessment, recognize when and how to select appropriate goals, create plans to satisfy these goals, and execute them. Indeed, goal-driven action selection is a hallmark of intelligent behavior (Newell and Simon 1972). During execution, an agent may encounter opportunities and obstacles that lie outside the scope of the agent's currently *active* goals (i.e., the goals it is currently trying to satisfy), but which nevertheless require response, either because they have higher priority or benefit the agent's long-term welfare. Because these new goals are outside the scope of the active goals, subgoaling (Laird *et al.* 1986) is insufficient to generate them. Traditionally, this process has been left to human users, as in on-line planning (Nau 2007). In this paper, we extend on-line planning to allow an agent to introduce, manage, and pre-empt goals when necessary. We define a conceptual model of this goal reasoning process called *Goal-Driven Autonomy* (GDA). Our claim is that GDA-enabled agents will outperform non-GDA approaches when unanticipated situations occur in scenarios from complex environments.

To illustrate the importance of goal reasoning, consider a reconnaissance unit with the goal of returning to its base. An applicable plan would include actions for it to move to the base's location. Suppose an enemy force attacks during this plan's execution. An agent that reasons about its goals can preempt its original goal, in favor of a new goal to reinforce the defenses and repel the assault. This is outside the scope of returning the reconnaissance unit to the base, but results in a superior outcome (i.e., the base's defenses remain intact).

To investigate goal reasoning, we implemented the GDA model in ARTUE (Autonomous Response to Unexpected Events; Molineaux *et al.* 2010a), an agent which integrates: a novel Hierarchical Task Network (Erol *et al.* 1994) planner that reasons about exogenous events and continuous environments, an explanation generator that reasons about unobserved information in the environment, and goal formulation and management components that use structured pieces of domain knowledge called *principles* to reason about the agent's goals when new information about the environment becomes available. GDA enables ARTUE to competently respond to potential problems and

[*] Present address: Palo Alto Research Center, 3333 Coyote Hill Rd, Palo Alto, CA, 94304

opportunities in complex environments by changing its goals. Unlike most agents, it explicitly reasons about unobserved facts and unknown objects, the passage of time using process models, continuous and discrete state descriptions, *and* exogenous events.

We evaluate GDA's contributions to ARTUE on scenarios defined using two complex environments: *Battle of Survival* (BoS), an open source real-time strategy game built on the Stratagus engine (Ponsen *et al.* 2005), and the Navy training simulation *TAO Sandbox* (Auslander *et al.* 2009). Through an ablation study, we demonstrate GDA's significant performance gains versus off-line planning and replanning agents. These gains are largest for scenarios requiring reasoning about unknown objects and unobservable aspects of the environment. This work extends our previous research (Molineaux *et al.* 2010a) by reporting an investigation with an additional domain, providing additional detail on the GDA implementation and its evaluation in ARTUE, and placing ARTUE and goal reasoning in a broader research context.

We define GDA, introduce the simulation environments, and describe the planning and reasoning representations employed by ARTUE. We then provide an illustrative example and report on our evaluation. We close with a discussion of related work and future directions.

2. Goal-Driven Autonomy

Cox's (2007) INTRO system provides inspiration for several concepts in Goal-Driven Autonomy, with its focus on integrated planning, execution, and goal reasoning. We extend these ideas and integrate them with Nau's (2007) on-line planning framework (Figure 1). Also shown in Figure 1 is our GDA conceptual model of goal reasoning. 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 briefly describe this model and its component tasks.

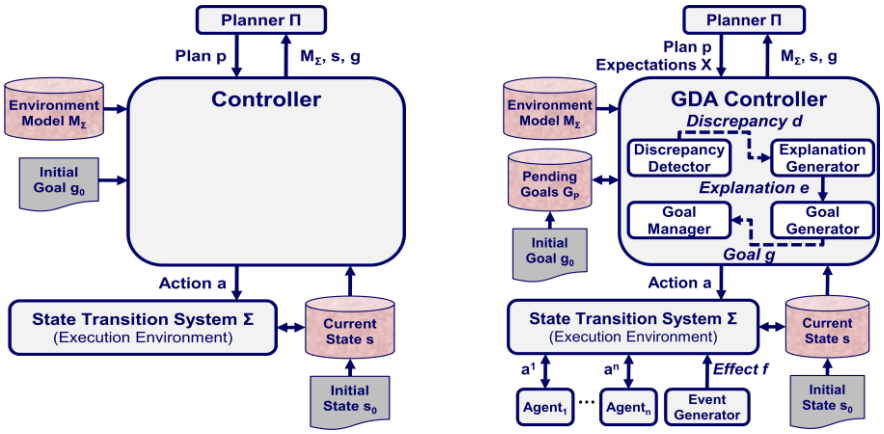

Figure 1: Conceptual models of on-line planning (left) (Nau, 2007) and Goal-Driven Autonomy (right)

Following Nau's (2007) notation, the environment is a tuple (S,A,E,γ) with sets of states S, actions A, and exogenous events E, as well as a state transition function γ: $S\times(A\cup E)\rightarrow 2^S$, which describes how an action's execution and/or an event's occurrence transforms the environment from one state to another (Ghallab *et al.* 2004). In this context, partial observability means that, in addition to having only partial knowledge of the current state, the agent is told neither the set of all possible states nor given the definition of γ. Therefore, the agent must reason with M_Σ, a (possibly incorrect and incomplete) model of the environment.

In both models, the Planner receives as input a planning problem (M_Σ,s_c,g_c), where s_c is the current observed state, and g_c is a goal that can be satisfied by some set of states $S_g\subseteq S$. The Planner outputs a plan p_c, which is a sequence of actions $A_c=[a_c,\ldots,a_{c+n}]$ that result in satisfying g_c. Unlike the on-line planning model, the GDA model's Planner generates a corresponding sequence of expectations $X_c=[x_c,\ldots x_{c+n}]$, where each $x_i\in X_c$ is a set of state constraints corresponding to the sequence of states $[s_{c+1},\ldots,s_{c+n+1}]$ expected to occur when executing A_c in s_c using M_Σ.

In both models, the Controller sends the actions in the plan to the environment and processes the resulting observations. The on-line planning model's Controller monitors the observations to determine whether it should submit the next action of p_c or replan for the initial goal g_0. The GDA Controller likewise has these options, but it also compares its observations with X_c to determine whether a change in goals may be warranted. In particular, it performs the following four knowledge-intensive tasks, which together comprise goal reasoning in the GDA model:

1. *Discrepancy detection*: This compares the observations obtained from executing action a_c in state s_c with the expectation x_c (i.e., it tests whether any constraints are violated, corresponding to unexpected observations). If the set of discrepancies D found in this comparison is non-empty, then they are explained in the following task.
2. *Explanation generation*: Given the current observed state s_c, prior observed state s_p, and a set of discrepancies D, this process hypothesizes one or more explanations E of their cause.
3. *Goal formulation*: This generates zero or more goals G in response to the set of discrepancies D, explanations E, and the current observed state s_c.[1]
4. *Goal management*: Given a set of pending goals G_{pend} (some of which may be active) and new goals G, this may update G_{pend} (e.g., by adding G and/or deleting/modifying other pending goals) and will select the next goals G' to be given to the Planner. (It is possible that $G=G'$.)

GDA makes no commitment to specific types of algorithms for the highlighted tasks and treats the Planner as a black box. For example, goal formulation and management may involve comprehensive goal transformations (Cox and Veloso 1998) instead of ARTUE's use of principles. In addition, GDA agents may employ task goals, as in ARTUE, or state goals depending on the planning algorithm being used.

3. Strategy Simulations and the PDDL+ Planning Formalism

Before describing our implementation of GDA in the ARTUE agent, we describe the task environments and corresponding planning domain representations.

3.1. Strategy Simulations as Complex Environments

We employ two strategy simulations as environments to demonstrate ARTUE: Battle of Survival 1.0 (BoS) and the TAO Sandbox. BoS[2] is an open source real-time strategy game implemented using the Stratagus game engine (Ponsen *et al.* 2005). The TAO Sandbox is a strategy simulator used by the Navy to train Tactical Action Officers in anti-submarine warfare and other types of missions (Auslander *et al.* 2009). In each of these strategy simulations, players accomplish their goals by giving orders to *units* – entities in the environment capable of carrying out tasks. The effects of orders may be instantaneous (e.g., launch a helicopter), of fixed duration (e.g., move to a specific location), or of indefinite duration (e.g., gather resources from the environment). Therefore, players interacting with the simulation must reason about instantaneous occurrences (i.e., the orders themselves) and continuous effects (i.e., the units carrying out the orders). During the simulation, potential opportunities and failures can arise requiring the player's response and possibly changes to the player's goals. Figure 2 displays brief descriptions of the units from these two simulations.

Battle of Survival	TAO Sandbox
• Harvester – Non-combat unit used to gather resources from the environment • Tank – Mobile unit used for combat • Scout – Fast ground unit used for reconnaissance • Helicopter – Aerial unit used for transportation and combat • Gun Turret – Immobile unit used for defense	• Navy Ship – Anti-submarine maritime surface unit that can launch torpedoes and helicopters; it can use a variety of sensors • Submarine (sub) – Underwater combat unit that can lay mines and launch torpedoes to destroy surface ships • Helicopter – Aerial unit that can detect submarines (by launching buoys) and launch torpedoes • Transport – Surface units used to move people and cargo

Figure 2: TAO Sandbox and BoS unit types and descriptions

[1] More generally, this is a discrepancy resolution step; a GDA agent could decide to ignore the discrepancy, dismiss the explanation, query for additional information before making a decision on whether to generate a goal, etc. However, in this paper we simplify this model to better align it with our instantiation of GDA in ARTUE.

[2] http://www.boswars.org/

In Section 1, we explained that a *complex* environment is one that is *continuous*, *partially observable*, *dynamic*, and *open* with respect to new objects. BoS and the TAO Sandbox are complex environments. Unit positions, health, and resources are important *fluents* (i.e., continuously varying numeric quantities). These simulations are partially observable because the players do not have access to the complete state when selecting actions. Therefore, from the agent's perspective, these environments are *nondeterministic*. Furthermore, the environment changes not only as a result of the player's actions, but also as a result of dynamic processes and events, such as enemy attacks and changing weather conditions. Finally, these environments are open with respect to the introduction of new objects. In BoS, new units (neutral and enemy) may be discovered as terrain is explored, and in the TAO Sandbox, new submarines, torpedoes, and mines may all be discovered during execution.

3.2. PDDL+ Planning Formalism

To model these complex environments, we use the PDDL+ domain language, which was designed to support reasoning about mixed discrete-continuous domains (Fox and Long 2006). In PDDL+, states are divided into two components: a discrete state described logically, and a continuous state consisting of fluents and their numeric values. In addition to *actions,* which (as in classical planning) describe instantaneous changes to the discrete state and are initiated by the agent, PDDL+ describes continuous and dynamic changes to the state from the environment using *processes* and *events*. These allow us to model the interaction between the agent's behavior and the environment. Processes occur over time and have continuous effects on fluent values. They are initiated or terminated by discrete state changes, which are modeled as effects of actions and events. Figure 3 illustrates how moving a ship in the TAO Sandbox environment can be modeled using PDDL+.

Action Name	MoveShip
Participants	?unit type=NavalShip, ?x type=Number, ?y type=Number
Conditions	(not (movingTo ?unit ?x ?y))
Effects (Discrete Only)	(movingTo ?unit ?x ?y)

Process Name	ShipMovement
Participants	?ship type = NavalShip
Conditions - (Discrete Only)	(movingTo ?ship ?x ?y) (speedOf ?ship ?speed)
Effects - (Continuous Only)	(increase (atX ?ship) (* (cos (headingOf ?ship)) ?speed #t)) (increase (atY ?ship) (* (sin (headingOf ?ship))?speed #t))

Event Name	EndOfShipMovement
Participants	?ship type = NavalShip
Conditions - (Discrete and Continuous)	(movingTo ?ship ?x ?y) (speedOf ?ship ?speed) (<=(dist(atX ?ship)(atY ?ship)?x ?y)0.5)
Effects - (Discrete)	¬(movingTo ?ship ?x ?y) ¬(speedOf ?ship ?speed) (speedOf ?ship 0)

Figure 3: PDDL+ action, process, and event describing the movement of a naval ship.

While actions are controlled by the agent and may be taken when a set of objects satisfies the participant types and conditions, processes and events exist in the environment as follows. A process is *active* for each set of objects that satisfy its participant types and conditions, modeling continuous changes in the environment. An event *occurs* when a set of objects satisfy its participant types and conditions, resulting in changes to the discrete state. For example, when an agent initiates the `MoveShip` action, with assignments for its participants, a `movingTo` literal is added to the state, creating a `ShipMovement` process that models the participant ship's change in position over time. The `ShipMovement` process definition's conditions include the ship's destination and speed. The `ShipMovement` process's continuous effects define functions describing the ship's location with respect to time. In this case, when the ship reaches its destination, its new position will satisfy the conditions of an `EndOfShipMovement` event, causing that event to occur. The result of this event is that the ship's `ShipMovement` process is no longer active (i.e., the ship has stopped).

3.3. Domain Extensions to Support GDA

To support discrepancy detection and explanation generation from the environment model, we extended PDDL+ with three properties for domain predicates: observable, hidden, and ignorable. To account for partial observability, we consider each predicate to be either observable or hidden. A literal with an *observable* predicate is always observed when true, and a literal with a *hidden* predicate is never observed regardless of its truth-value. Thus, the truth of a hidden fact can never be established by direct observation. For example, the hidden predicate `enemyProducingRegion` describes an unexplored region that may introduce enemy units in the BoS environment model. *Ignorable* predicates describe aspects of the observable state that the agent may safely ignore during execution. For example, the BoS environment model uses the ignorable predicate `regionBoundary` to relate a region to a sequence of coordinates describing the region's boundary. While necessary for planning, the exact coordinates and their order may be ignored during execution. In the next section, we describe how ARTUE uses this domain representation to perform GDA tasks in the BoS and TAO Sandbox environments.

4. Goal-Driven Autonomy in ARTUE

To highlight ARTUE's novel features and its implementation of GDA components, we use the Escort scenario from BoS as a running example throughout this section. The agent's initial goal in this scenario is to return a harvester unit to its base, which is distant from the harvester's initial location. The map is divided into five regions. A corridor lies between the harvester and base; it is surrounded by impassible terrain except for two bottleneck regions on either side. Each bottleneck is defended by two friendly units: an immobile gun turret and a tank. The bottlenecks and corridor regions are visible to the agent (i.e., any enemy units in these regions will be observed by the agent). Adjacent to each bottleneck is an unseen region, from which enemy units may attack. The agent must navigate the harvester safely to its base by generating, executing, and monitoring plans for the user-defined navigation goal and any generated goals. Figure 4 shows a screenshot of a tank and a gun turret defending a bottleneck and a small subset of the initial state for this scenario. `Region4` is one bottleneck region, `Region5` is an unseen by the agent, and

Discrete State:

```
(isa Base27 Base)
(isa Region4 BottleneckRegion)
(isa Region5 UnseenRegion)
(regionBorders Region4 Region5)
(isa Harvester17 Harvester)
(unitAt Harvester17 114 122)
(isa GunTurret19 GunTurret)
(inRegion GunTurret19 Region4)
…
```

Continuous State:

```
((unitHealth Tank20) 200)
((playerResources Player0 Titanium) 0)
…
```

Goal:

```
(moveUnitToUnit Harvester17 Base27)
```

Figure 4: Escort scenario with screenshot of units defending a bottleneck adjacent to unknown region.

the corridor that the harvester must traverse is to the left of `Region4`. The discrete state defines the regions, their adjacencies, units, and their locations. The continuous state describes the health of the units and the agent's resources.

4.1. Hierarchical Task Network Planning with Processes and Events

In GDA, given a goal, the Planner generates plans, which are sequences of actions, and expectations about how the environment will change during plan execution. For ARTUE, we extended SHOP2, a popular Hierarchical Task Network (HTN) planner (Nau *et al.* 2003), to reason about PDDL+ domains. Our extensions include the addition of a *wait* action, which allows the Planner to reason about time within the SHOP2 framework, and a *state projection algorithm*, which determines the continuous effects of active processes and the timing of exogenous events. These extensions are described in detail in (Molineaux *et al.* 2010b) and summarized here.

HTN planning (Erol *et al.* 1994) formulates plans by decomposing *tasks* into *subtasks*, bottoming out into a sequence of *actions*. Therefore, for an HTN Planner, the goals generated and selected by ARTUE are tasks. SHOP2 is a state space planner; at each step in its search, applicable task decompositions are considered to extend an

```
(:method (moveUnitToUnit ?unit ?destination-unit)
  :name MoveUnitBasic
  :preconditions ((unitAt ?unit ?x1 ?y1)
                  (unitAt ?destination-unit ?to-x ?to-y)
                  (speedOfUnit ?unit ?speed)
                  (assign ?dur
                          (durationOfMovement ?speed ?x ?y ?to-x ?to-y)))
  :subtasks ((MoveTo ?unit ?to-x ?to-y)
             (Wait ?dur)))
```

Figure 5: An HTN method for decomposing the task `moveUnitToUnit` into a sequence of two actions: a move order and a wait action for the estimated duration of the movement.

existing plan, and, for each selected action, a new state is extrapolated. To process the passage of time within tasks, we introduce a special *wait* action, which takes one argument, *wait-time*, representing the action's duration.[3] It is always applicable, and because time passes during this action, it is necessary to compute the effects of the active processes and events that occur using the state projection algorithm. All other actions are instantaneous and added sequentially with any number of actions occurring between wait actions. Because the actions in these domains are orders to units, during the wait action, the units carry out their orders concurrently. In this manner, our planner can predict the state at any future time point. To use the wait action effectively, we use HTN methods to determine the appropriate durations as illustrated in the example in Figure 5.

Given the initial goal `(moveUnitToUnit Harvester17 Base27)`, the HTN method in Figure 5 reduces this task into two actions: `(MoveTo Harvester17 2 2)` and `(Wait 3500)`. The method's preconditions specify the two units' locations and the speed of the moving unit, and also compute how long the movement will take. In this case, one effect of the `MoveTo` action is that the fact (`movingTo Harvester17 2 2)` is added to the state activating a movement process. After 3500 time units, the harvester is predicted to arrive at its destination, satisfying the initial goal.

4.2. Discrepancy detection

Discrepancy detection is the GDA task responsible for identifying prediction failures that may indicate problems or opportunities. ARTUE monitors the observed state at fixed intervals throughout execution and after each wait action completes. Discrepancies are found by comparing the observed state to the expected state, which was generated by the extended SHOP2 system given the environment model and the observed state when the plan was requested.

Before comparing the observed and expected states, the facts and fluents containing ignorable predicates are removed from consideration. For the discrete state, any fact that appears in one state but not the other is considered a discrepancy. For the continuous state, if a fluent is defined in one of the states and not the other, or if there is more than 1% difference between its expected and observed value, then it is considered a discrepancy. While this error tolerance will vary by the fidelity of environment model, a 1% difference was sufficient in each of these domains.

Returning to our example, an enemy Scout unit is observed in `Region5` at time step 300. None of the discrete or continuous facts describing the enemy Scout unit were in the expected state that was projecting the effects of the moving process on the harvester's position. Therefore, these facts are considered a discrepancy and provided to the explanation generation component.

4.3. Explanation Generation

Discrepancies between a projected state and an observed state arise as a result of one of three circumstances:

1. A hidden factor influences the state.
2. The agent's domain knowledge is flawed, resulting in false expectations.
3. The perception of the actual state is incorrect.

While a complete model would address each type of discrepancy, we focus here on hidden factors influencing the state and discuss possible approaches for explaining and responding to flawed domain knowledge and misperceptions later in this section. Hidden factors influencing the state may threaten an agent's performance. Therefore, ARTUE generates *explanations* (i.e., sets of assumptions about the hidden state) of discrepancies to

[3]This could be extended to include an arbitrary condition. In that case, the duration would last until the condition was satisfied in the environment.

identify hidden factors. ARTUE employs an Assumption-based Truth Maintenance System (ATMS; de Kleer 1986) to generate these explanations using causal inference over the environment model and the previous, expected, and observed states.

Because complex environments are partially observable and open with respect to new objects, ARTUE must reason about two types of hidden factors: hidden facts and unknown objects. For each hidden predicate in the environment model, ARTUE creates a set of assumptions that include all possible groundings of that predicate with known objects. To account for the introduction of new objects, ARTUE will also generate additional assumptions for hidden predicates with reified unknown objects using the argument types of the predicate. For example, the predicate `enemyProducingRegion` is a hidden predicate whose one and only argument is of type `Region`. Therefore, if there are five known region objects in the environment, then six `enemyProducingRegion` assumptions will be created, one for each known region and one for an `UnknownRegion` object.

To determine which of these assumptions explains the discrepancies, ARTUE automatically generates a set of ATMS rules from the PDDL+ domain, which incorporates hidden and observable state facts. These rules relate the state before an action is executed (or an event occurs) to the resulting state.

Event Name	`EnemyAppears`
Participants	`?unit` type = `Unit`

Conditions	`(enemyProducingRegion ?region EnemyPlayer)  // hidden`
	`(enemyBuiltIn ?unit ?region)  // hidden`
	`(unseenRegion ?region)`

Effects	`(inRegion ?unit ?region)` `(unitOwner ?unit EnemyPlayer)`

Figure 6: Event definition describing the appearance of an enemy unit using two hidden facts

Figure 6 includes a PDDL+ event definition for an appearing enemy unit. ARTUE translates this into the following rule:

```
(occurs (EnemyAppears ?unit)) ∧ (unseenRegion ?region BeforeEvent)
                              ∧ (enemyProducingRegion ?region BeforeEvent)
                              ∧ (enemyBuiltIn ?unit ?region BeforeEvent)
⇒ (inRegion ?unit ?region AfterEvent) ∧ (unitOwner ?unit Enemy AfterEvent)
```

Using the rules and observed literals, the ATMS can deduce the possible worlds that result from different sets of assumptions. Ideally, there is a possible world in which each discrete fact in the actual successor state is true and each fluent value in the successor state is within the error tolerance of discrepancy detection. However, no set of assumptions may explain certain values or facts in the successor state; this can occur due to flaws in ARTUE's domain knowledge or flawed perceptions. Currently, ARTUE does nothing with these discrepancies and ignores them in its search for possible worlds. In the future, explanation failure could lead to the construction of a learning goal (Ram and Leake 1995) to refine ARTUE's knowledge. This could involve altering the error tolerance for misperceptions or performing model-based diagnosis to identify errors in action models as in (Roos and Witteveen 2009).

After all unexplainable facts are discarded, ARTUE searches for possible worlds in which all of the remaining facts and fluents are explained. Each possible world is one *explanation* (i.e., a set of assumptions that predicts as many of the discrepancies as possible). Assumptions shared across all explanations are added to the ARTUE's beliefs about the current state.

In the case of the enemy Scout's appearance, the `EnemyAppears` event includes effects that match the discrepancies in the observed state. Explanation generation produces an explanation that includes the hidden facts that would cause the event to occur (i.e., assumptions that `Region5` is an enemy producing region and that the Scout unit was built in it). ARTUE adopts these facts as beliefs, and goal formulation is triggered.

4.4. Goal Formulation

After explaining the discrepancy, the agent determines which goals should be generated (if any) and added to the set of pending goals. Not all discrepancies require the creation of new goals; in fact many discrepancies may be ignored, and some explanations involve irrelevant assumptions that do not require action.

ARTUE performs goal formulation using *principles,* which are schemas whose components are *participants*, *conditions*, an *intensity level*, and a *goal form*. Each participant is assigned a type (e.g., `Region`). Conditions are statements concerning the participants that must hold in the agent's beliefs to generate the goal specified by the goal form. The intensity level specifies the importance of the generated goal to the agent and is currently a fixed qualitative value. In the future, ARTUE will support functions over future states to allow for finer fidelity in prioritizing the agent's goals. For example, the larger the opposing force outside the bottleneck, the higher the intensity should be to reinforce it. ARTUE checks its principles to determine which goals are generated (if any). A principle will generate a goal for each set of objects in ARTUE's beliefs that correspond to the principle's participant types and satisfy its conditions. A principle may generate multiple goals in either a single goal formulation step or over multiple goal reasoning cycles. We consider the implications of providing the principles to this version of ARTUE and extensions that learn this knowledge in the discussion of the evaluation.

Name	`ReinforceBottleneck`
Participants	`?bottleneck` type = `Bottleneck` , `?region` type = `Region`
Conditions	`(regionBorders ?bottleneck ?region)` ∧ `(enemyProducingRegion ?region ?enemy)`
Intensity Level	`HighIntensity`
Goal form	`(reinforceBottleneck ?bottleneck)`

Figure 7: Principle used by ARTUE to reinforce a bottleneck between its forces and the enemies

In our example, given the observed state with the enemy Scout unit and the explanation that `Region5` is an enemy producing region, ARTUE uses the principle in Figure 7 to generate the high priority goal of reinforcing the bottleneck in `Region4`.

4.5. Goal Management

Given multiple goals, the agent must decide which pending goal to pursue next (i.e., make active). Principles play an important role in *goal management*: pending goals are ordered by their intensity. Currently, ARTUE considers goals individually. Thus, it selects a goal with a highest intensity. However, if the Planner cannot generate a plan to achieve a selected goal, then ARTUE selects the goal with the next highest intensity, until an achievable goal is found. This is a simple method for goal retraction. A more complete account of goal management would include goal transformations and other methods for goal retraction (Cox and Veloso 1998).

In our example, two goals are now pending: `(reinforceBottleneck Region4)` and `(moveUnitToUnit Harvester17 Base27)`. The first goal has a higher intensity and given that a plan for can be created by the Planner, `(reinforceBottleneck Region4)` becomes the active goal.

5. Evaluation

In this section, we describe an evaluation assessing ARTUE's performance in six scenarios drawn from two complex environments: Battle of Survival and the TAO Sandbox. While the TAO Sandbox results were originally reported in (Molineaux *et al.* 2010a), we include additional analysis and discussion here. To evaluate the impact of Goal-Driven Autonomy, we compare ARTUE against two baselines: a discrepancy-based replanning agent (REPLAN) and an off-line planning agent (PLAN1). In these scenarios, REPLAN should outperform PLAN1 due to the environments' partial observability and openness with respect to new objects. REPLAN employs the same discrepancy detection algorithm as ARTUE, but does not perform explanation and goal management. We hypothesize that ARTUE will outperform the baseline agents in scenarios where unobserved facts affect the state, due to explanation, and where important events occur outside the scope of the mission, due to goal reasoning. We measure performance based on achieving the initial goal and a combination of scenario-specific quantitative metrics (e.g., execution time). Table 1 organizes the six scenarios with respect to the types of unexpected events, whether

unobservable facts affect the environment, whether adversaries are present, and the scenario-specific scoring metrics. After describing these six scenarios, we describe results from 25 trials on each.

Table 1: Evaluation Scenarios

Scenario Name	Unexpected Event Types	Unobservable Facts (Hidden)?	Adversaries?	Additional Quantitative Metrics
BoS: Resource Gathering	Opportunity			Execution time
BoS: Escort	Problem	✓	✓	Distance from goal and health of other friendlies
BoS: Exploration	Problem	✓		Area explored
TAO: Identification	Opportunity	✓	✓	Identifying and destroying enemy sub
TAO: Iceberg	Problem + Opportunity	✓		Execution time and rescue results
TAO: Sub Hunt	Problem	✓	✓	Clearing discovered mines

5.1. Scenarios: Battle of Survival

Resource Gathering: The agent is given the goal to retrieve 100 units of titanium. While a harvester is en route to known titanium fields, it discovers a closer titanium deposit. Score is calculated using the time required to achieve the goal. This unexpected event does not require reasoning about unobserved facts.

Escort: A harvester unit must safely return to base. Its path is a corridor with impassible terrain on either side except for two bottlenecks. Each bottleneck is defended by a friendly tank and gun turret. All regions outside the bottleneck are unexplored, and therefore any enemy units in these regions are initially unknown to the agent. After a short time, an enemy Scout unit appears outside one of the bottlenecks. Later, three enemy tanks attack the same bottleneck. Score is calculated based on the survival time of the harvester and the remaining health of friendly units. The unexpected event can be explained by unobserved facts.

Exploration: Friendly buildings are clustered in one corner of the map along with a scout unit, and the agent is given the goal to explore all of the terrain tiles. During exploration, land-based movement actions fail to achieve their intended effects due to impassable terrain in part of the map. Score is calculated based on the resources remaining, the time spent exploring the map, and the percentage of the map explored. The Planner is allowed to order agents to move to any tile on map. Therefore, the unobservable facts that some areas are inaccessible are needed to explain why the land-based explorer cannot reach its destination.

The PDDL+ planning environment model for these scenarios consists of 13 actions, 3 processes, and 6 events. ARTUE uses 13 HTN methods to formulate plans in this domain, and 3 principles to generate goals and assign goal intensities in these scenarios.

5.2. Scenarios: TAO Sandbox

Identification: The agent is given a goal to identify nearby ships. To do so, it must send a task group unit within visual range of each ship. During this exercise, an unseen sub torpedoes a nearby ship, causing it to sink. This sub is an unknown object that cannot be directly observed. Score is based on identifying each of the nearby ships as well as the sub (which requires special sensors), and also destroying the sub, which is outside the scope of the user-supplied goal.

Iceberg: The agent is given a goal to safely guide a transport ship to a destination in Norway. During its travel, a storm arises, which is presaged by a lightning strike. The strike causes a large iceberg to calve, blocking the entrance to a nearby port. Due to the storm's severity, all ships must seek shelter, and a nearby ship, which does not detect the

iceberg, founders on it. Score for this scenario is based on how close the ship comes to its destination, how long it survives, and how early it arrives to rescue the passengers of the foundering ship, which is outside the scope of the user-supplied goal. The presence of an approaching storm is an unobservable fact that is necessary to explain the observations of thunder and lightning.

SubHunt: An enemy sub has been spotted nearby. The agent's goal is to destroy the sub. However, this sub has been laying mines that can incapacitate the searching ship. Points are awarded for finding and destroying the sub, as well as sweeping the mines. The existence of the mines cannot be detected safely, but must be inferred from explosions on board a nearby ship.

The PDDL+ planning environment model for these scenarios consists for 33 actions, 7 processes, and 37 events. ARTUE uses 46 HTN methods to formulate plans in this domain, and 11 principles to generate goals and assign goal intensities in these scenarios.

5.3. Results

In each scenario, we performed 25 trials by varying the objects' starting locations and the timing of unexpected events, which were held constant across the three agents. Table 2 lists the average scores for each agent on each scenario and the percentage of the trials in which they achieved the initial goal.

Table 2: Agent performance on each scenario (average scenario-specific metric and percent of the trials in which the agent achieves the initial goal)

Scenario	ARTUE	REPLAN	PLAN1
BoS: Resource Gathering	0.61 (100%)	0.61 (100%)	0.04 (100%)
BoS: Escort	0.93 (92%)	0.63 (4%)	0.58 (0%)
BoS: Exploration	0.98 (96%)	0.26 (0%)	0.83 (0%)
TAO: Identification	0.73 (100%)	0.40 (100%)	0.32 (40%)
TAO: Iceberg	0.72 (76%)	0.48 (48%)	0.35 (4%)
TAO: Sub Hunt	0.97 (96%)	0.48 (36%)	0.34 (0%)

Overall, ARTUE achieved the user-provided goal on 93% of the trials compared to 48% and 24% for REPLAN and PLAN1, respectively. ARTUE achieved the user-provided goal at a greater rate than either of the ablations on five out of six scenarios, with the three agents performing equally well on the *Resource Gathering* scenario. All scenario-specific metric scores are scaled between 0 and 1, with 1 being the maximum performance. While they are not comparable between scenarios, the scenario-specific metrics are used here to establish ordinal relationships. Within each scenario, the differences between each agent's performance are statistically significant ($p<.05$) on all trials except between ARTUE and REPLAN on the *Resource Gathering* scenario. On five of the six scenarios, ARTUE outperformed REPLAN, and REPLAN outperformed PLAN1. On the Exploration scenario, PLAN1 achieved a higher average scenario-specific score than REPLAN. We discuss these results in more detail in the following section.

5.4. Discussion

These results demonstrate that ARTUE can competently respond to unexpected events in scenarios defined for these complex environments. Furthermore, the comparisons with each ablation support our hypothesis that Goal-Driven Autonomy is responsible for ARTUE's performance gains. Further analysis indicates the differences in performance are due to ARTUE's reasoning about hidden factors affecting the environment and goal generation. ARTUE outperformed REPLAN on all of the scenarios involving reasoning about unobservable facts. In the *Resource Gathering* scenario, ARTUE and REPLAN respond in the same manner to the discovered titanium because the best response is to retain the goal of gathering titanium and gather it from the newly discovered field. In the *Escort* scenario, while REPLAN does not respond to the Scout unit, ARTUE changes goals to add another tank to defend the bottleneck. When the enemy tanks arrive, REPLAN still has a tank unnecessarily defending the other bottleneck. At this point, REPLAN directs this tank to attack the enemy force, but it arrives too late to assist and is also destroyed. A surprising result is that PLAN1 outperformed REPLAN on the *Exploration* scenario. While neither agent accomplished the goal of exploring all the terrain, PLAN1 uncovered more terrain. The unexpected event in this scenario is that impassable terrain prevented the exploring Scout unit from reaching the first waypoint. While

PLAN1 continued its plan by ordering the Scout unit to the next waypoint, REPLAN persists in ordering the Scout unit to the first waypoint and fails to explore the rest of the map.

The performance differences in the TAO Sandbox scenarios demonstrate the types of scenarios that each agent can address. PLAN1 cannot tolerate even minor changes to its expectations. In the *Iceberg* scenario, PLAN1 accomplished the initial goal in the rare occasion that the port was near enough to be reached at top speed before the storm arrives. REPLAN responds correctly to minor exogenous events (e.g., course changes) that occur during the *Identification* scenario. This requires that the event and its repercussions be immediately apparent. ARTUE outperforms REPLAN when exogenous events are caused by unobserved aspects of the environment and in situations outside the bounds of the active goals. For example, saving a foundering ship would not be a subgoal for every possible goal in the domain, although it is required by the *Iceberg* scenario.

GDA as instantiated in ARTUE improves agent performance in scenarios that require reasoning about unobservable facts and unseen objects. In HTN planning, the extent to which these scenarios require explicit goal formulation in addition to reasoning about unknown facts is difficult to ascertain. An alternative approach would involve adding levels of abstraction to the task hierarchy, thereby stretching the semantics of HTNs. For example, the first decomposition in *every* plan could involve reasoning about what task to pursue in the current state. This not only raises the knowledge engineering burden of intelligent system designers, but also increases the difficulty of learning goal reasoning knowledge. Supporting this claim, a later version of this system, T-ARTUE, demonstrates that principles can be replaced with an active learning process (Powell *et al*. 2011). In the future, we plan to explore additional ways for agents to select goals, and, by confining planning to action selection, this framework enables that functionality.

6. Related Work

Intelligent action selection is a central aim of research on intelligent agent architectures. These architectures range from purely deliberative (Fikes and Nilsson 1971) to purely reactive architectures (Brooks 1991) with many agent architectures containing both reactive and deliberative components. Reactive components directly link observed states to particular actions. They can be hardcoded and fixed, as in the finite state machines (Gill 1962) and behavior trees algorithms used in current video games (Champandard 2007), or learned using methods such as reinforcement learning (Sutton and Barto 1998). GDA is a deliberative model. Consequently, our discussion of related work focuses on the deliberative components of other approaches. To highlight the differences between GDA, its implementation in ARTUE, and other approaches, we begin by discussing other approaches to planning in complex environments, and then discuss other approaches for goal reasoning.

6.1. Related Approaches for Planning in Complex Environments

As defined in Section 1, complex environments are *partially observable*, *dynamic*, *continuous*, and *open* with respect to new objects. In partially observable environments, the results of actions are often *nondeterministic* from the perspective of the agent. One approach for coping with nondeterministic environments is *contingency planning* (Dearden *et al.* 2003). Contingency planning agents generate contingency branches that are executed only when an action does not achieve its intended effects. Determining the actions that require conditional plans is a difficult problem. Furthermore, contingency planning typically requires that the possible outcomes of actions are known beforehand, which is not the case in our domains due to unknown objects. An alternative approach is to represent uncertainty about the current state directly using Partially Observable Markov Decision Processes (Kaelbling *et al.* 1998). Likhachev and Stentz (2007) observe that these approaches require an enumeration of the state space, which is impossible in open domains, and cannot incorporate domain-specific heuristic knowledge. To address these concerns, their PCPP planner reasons about preferences between unknown values of the state when generating the plan. While ARTUE (currently) generates a single plan without any probabilities, it uses an HTN planner that can leverage domain knowledge. Incorporating probability distributions over future expected states can reduce the number of false positives during discrepancy detection and is an important direction for future work.

Plan monitoring can be used to detect changes in dynamic environments that can cause plan failure. For example, HoTRiDE replans for portions of its plan when an action fails (Ayan *et al*. 2007). Another replanner, MRP-Agent, continually searches for the plan that can most easily be replanned in the event of plan failure (Corchado *et al.* 2008). Instead of generating complete plans that are likely to be invalidated by exogenous events, *incremental*

planners (1) plan for a fixed time horizon, (2) execute the plan, and then cycle on these steps. This process continues until a goal state is reached. Incremental planning and plan monitoring may be combined. For example, CPEF (Myers 1999) incrementally generates plans to achieve air superiority in military combat and replans when unexpected events occur (e.g., a plan is shot down). As illustrated in our evaluation, replanning without reasoning about the cause of the failure or the goals of the agent is insufficient for the scenarios in our evaluation.

Complex environments are subject to continuous change. Only a few planning systems can process continuous effects. One such system is COLIN, a domain-independent planner that can reason with the linear continuous effects of durative actions (Coles *et al.* 2009). However, COLIN does not consider exogenous changes in the environment. Our extensions to the popular SHOP2 planner allow ARTUE to reason about continuous effects present in the domain description.

In open environments, objects unknown at initial planning time may be relevant to the active goals. To account for this, several researchers have explored extensions to goal specifications. Goldman (2009) describes a system with universally quantified goals that allows planning for sets of entities whose cardinality is unknown at planning time. Beyond just specifying goals for unknown entities, *open world quantified goals* define sensing actions to detect unknown objects as well as a utility for achieving each goal (Talamadupula *et al.* 2009). These approaches place goal reasoning within the central planning mechanism. GDA differs by considering goal reasoning as a knowledge intensive process worthy of independent investigation. We believe that this reduces the knowledge engineering burden for a system's designer and will enable learning of goal reasoning structures in new domains.

6.2. Related Approaches for Goal Reasoning

Goal-Driven Autonomy is a conceptual model of goal reasoning, which was the topic of a recent AAAI 2010 workshop that brought together researchers from a variety of research perspectives.[4] For a recent survey of goal reasoning approaches, see Hawes (2011). In this section, we distinguish our work from other goal reasoning approaches including goal generation structures, extended agent frameworks, mobile robotic applications, and case-based reasoning.

The most straightforward approaches for goal reasoning involve postulating a structure to formulate goals given the current state. Coddington and Luck (2004) bestowed agents with *motivations*, which generate goals in response to changes in specific state variables. For example, if a rover's battery charge falls below 50%, then a goal to recharge the battery will be generated (Meneguzzi and Luck 2007). ARTUE's principles differ in two important ways: (1) they use explanations to infer unobservable facts of the current state, and (2) they are not constrained to operate on individual objects or values.

Intelligent architectures typically address issues related to goal reasoning. Here, we discuss extensions specifically designed for goal reasoning in three agent architectures: Belief-Desire Intention (BDI; Rao and Georgeff 1995), SOAR (Laird 2008) and ICARUS (Langley and Choi 2006). While BDI agents typically change their procedural goals as a result of observed events, CANPlan illustrates how events can trigger declarative goals and invoke deliberative planning (Sardina and Padgham 2010). Extending these semantics, the agent language CAN specifies goal states (pending, waiting, active, and suspended) for three different types of goals (achievement, task, and maintenance) and transitions between them (Harland *et al.* 2010). While providing formal semantics for goal reasoning, these approaches require all the goals to be enumerated prior to execution, which is not possible in open environments, and have not been empirically evaluated. SOAR agents employ subgoaling and chunking to acquire new skills for goals that are not directly attainable. Recently, the selection of operators that change the goals has been modified with an intrinsically motivated reinforcement learning mechanism guided by the agent's appraisals (Marinier *et al.* 2010). Within the ICARUS architecture, Choi (2010) developed top-level constraint goal descriptions that create goals within the recognize-act cycle. In contrast to these approaches, GDA separates environmental and goal reasoning from action selection, which permits additional reflection when required. Furthermore, the goals we consider may differ substantially from the agent's current goals and, consequently, should not be considered subgoals (e.g., rescuing shipwreck survivors during a transportation mission).

Mobile robotics researchers frequently address issues pertaining to complex environments and goal reasoning. To enable autonomous goal formulation and management, *goal generators* produce goals when new objects are detected, which satisfy a set of conditions (Hanheide *et al.* 2010). For each new region detected by their mobile

[4] http://home.earthlink.net/~dwaha/research/meetings/aaai10-gda/

robot, a goal will be generated to identify the region and incorporated into its continuous planner (Hawes *et al.* 2009). Reasoning with open world quantified goals, Scheutz and Schermerhorn (2009) define *affective goal management* as a heuristic to select goals using expected utility, instead of probabilities, based on previous experiences. Open world quantified goals and goal generators formulate goals for newly detected objects in the environment, they do not generate goals for existing objects whose values have changed. For example, a detected submarine may not require a change in goal, but if it changed its course to an intercept a friendly vessel, this may require a reconsideration of the active goals that the above goal formulation methods do not consider. While ARTUE does not yet include a learning mechanism, it is an important direction for future work. In particular, we began investigating the use of *active learning* (Settles 2009) techniques to acquire ARTUE's principles (Powell *et al.* 2011).

In addition to agent architectures and robotics applications, some researchers in computer game AI have employed case-based reasoning (CBR; Aamodt and Plaza 1994) to perform goal reasoning. This reduces the knowledge engineering tasks for system designers to either annotating expert gameplay traces or simply collecting them. For example, CB-gda uses observed discrepancies as the retrieval cue to select task goals in a team shooter game (Muñoz-Avila *et al.* 2010).EISBots performed competently against the built-in AI of Starcraft by selecting goal states using the current state as the retrieval cue from a library of game play traces (Weber *et al.* 2010). Finally, given annotated gameplay traces, Darmok (Ontañón *et al.* 2010) adapts retrieved plans using the current game state and active goal to control agents in a real time strategy game. While useful for strategic decisions, such as, what to build next, these approaches would have difficulty on the scenarios described in our evaluation for the following reasons: First, they do not reason about hidden state explicitly. Instead they rely on the assumption that similar observed states have similar hidden states. While this is likely true in their domains, it is unlikely to hold for training simulations, because they seek to broaden the trainee's reasoning about the domain. Secondly, their state representations are simplified in that they do not, for example, include any spatial knowledge in their goal reasoning or plans. Therefore, spatially distinct situations would not result in different goals, and plans involving spatial elements cannot be constructed, e.g., the plan for defending a specific bottleneck in the BoS *Escort* scenario.

7. Conclusions and Future Work

Strategy simulations provide complex environments for agents that are continuous in time and space, partially observable, open with respect to the introduction of new objects, and dynamic due to hostile opponents and exogenous events. To operate autonomously in these complex environments, intelligent agents must perform continuous situation assessment, select appropriate goals, create plans to satisfy these goals, and execute them. We defined Goal-Driven Autonomy, a conceptual model of goal reasoning, and then instantiated it in ARTUE, an agent that integrates HTN planning, discrepancy detection, explanation generation, goal formulation, and goal management. Our system, ARTUE, monitors the environment for discrepancies between its expectations and its observations during execution. ARTUE then explains these discrepancies using an environment model, which extends its understanding of the state by postulating unobserved objects and facts. For goal formulation and management, ARTUE uses *principles*, which are goal formulation schemas, to introduce and prioritize new goals. This allows ARTUE to generate new plans, designed to avoid potential problems and exploit opportunities, whose goal may be unrelated to its active goal. The results of our evaluation on scenarios from two strategy simulations, Battle of Survival and the TAO Sandbox, provide support for the hypothesis that GDA enables agents to respond competently to unexpected events in scenarios from complex environments.

While our results are promising, future work should investigate implications of the GDA conceptual model and extend ARTUE's capabilities. First, by providing a clearer semantics for tasks and goals, we believe that GDA will reduce the knowledge engineering required to design an intelligent agent and enable learning of domain knowledge. To investigate this claim, we will explore approaches that learn ARTUE's domain knowledge for discrepancy detection, explanation generation, goal formulation, and goal management. Second, there are a variety of amenable AI techniques for each GDA task (Klenk 2010), and ARTUE represents only one possible instantiation. Consequently, additional research is required to understand the strengths and weaknesses of different techniques and their combinations for goal reasoning. Given the breadth of recent work on goal reasoning (see Section 6.2), we expect it will attract more research attention in the future. The fact that ARTUE's principles, which generate tasks for HTN planning, are similar to other approaches, such as, the goal generators (Hanheide *et al.* 2010), which generate state-based goals, suggests that goal reasoning is a distinct process worthy of independent investigation.

Our experience with ARTUE in these domains drives our future work in three directions. First, leveraging the GDA framework, we will continue to investigate methods for learning the knowledge necessary for discrepancy detection, explanation, and goal reasoning. Our first extension has shown that ARTUE can be trained to learn goal formulation and management knowledge in an active learning framework (Powell *et al.* 2011). . Second, we will integrate a spatial reasoning system to replace the current representation, which is tied to specific scenarios. In addition to increasing ARTUE's range of application, abstract spatial relations may also be used to perform cross-domain analogical learning (Klenk 2009) reducing the knowledge engineering burden in new environments. Third, ARTUE's explanation generation component only explains one type of discrepancy – those arising from hidden facts and unknown objects in the environment. An important next step is to generate explanations that trigger learning goals to improve the agent's domain knowledge, and sensing goals to investigate unexplainable discrepancies resulting from inaccurate perception. We will investigate these issues as we extend ARTUE toward our vision of a more robust autonomous agent. While we currently use strategy simulations as a testbed for autonomous behavior, we believe that goal-reasoning autonomous agents will function as realistic teammates and opponents in future strategy simulations.

Acknowledgements

Thanks to DARPA for funding this research, and PM Michael Cox for inspiring work on this topic. The views and opinions 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 NRL, DARPA, or the DoD. Matthew Klenk performed this research while supported by an NRC postdoctoral fellowship.

References

Aamodt, A., and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, **7**(1), 39-59.

Auslander, B., Molineaux, M., Aha, D.W., Munro, A., and 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 AI.

Ayan, N.F., Kuter, U., Yaman F., and Goldman R. (2007). Hotride: Hierarchical ordered task replanning in dynamic environments. In F. Ingrand, and 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.

Brooks, R. (1991). Intelligence without representation. *Artificial Intelligence*, **47**, 139-160.

Champandard, A. (2007). Behavior trees for next-gen game AI. In *Proceedings of the Game Developers Conference*. Lyon, France.

Choi, D. (2010). *Coordinated execution and goal management in a reactive cognitive architecture*. Doctoral dissertation: Department of Aeronautics and Astronautics, Stanford University, Stanford, CA.

Coddington, A., and Luck, M. (2004). A motivation-based planning and execution framework. *International Journal on Artificial Intelligence Tools*, **13**(1), 5-25.

Coles, A., Coles, A., Fox, M., and Long, D. (2009). Temporal planning in domains with linear processes. In *Proceedings of the International Joint Conference on Artificial Intelligence*. Pasadena, CA: AAAI Press.

Corchado, J.M., Gonzalez-Bedia, M., De Paz, Y., Bajo, J., and De Paz, J.F. (2008). Replanning mechanism for deliberative agents in dynamic changing environments. *Computational Intelligence*, **24**(2), 77–107.

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

Cox, M.T., and 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.

de Kleer, J. (1986). An assumption-based TMS. *Artificial Intelligence*, **28**(2), 127-162.

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

Erol, K., Nau, D., and Hendler, J. (1994). HTN planning: Complexity and expressivity. *Proceedings of the Twelfth National Conference on Artificial Intelligence* (pp. 123-1128). Seattle, WA: AAAI Press.

Fikes, R. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence,* **2**, 189–208.

Fox, M. and Long, D. (2006). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, **27**, 235-297.

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated planning: Theory and practice.* San Mateo, CA: Morgan Kaufmann.

Gill, A. (1962). *Introduction to the theory of finite-state machines.* , New York: McGraw-Hill.

Goldman, R.P. (2009). Partial observability, quantification, and iteration for planning: Work in progress. In *Generalized Planning*: *Macros, Loops, Domain Control: Papers from the ICAPS Workshop*. Thessaloniki, Greece: [http://www.cs.umass.edu/~siddhart/genplan09].

Hanheide, M., Hawes, N., Wyatt, J., Göbelbecker, M., Brenner, M., Sjöö, K., Aydemir, A., Jensfelt, P., Zender, H., and Kruijff, G-J. (2010). A framework for goal generation and management. In D.W. Aha, M. Klenk, H. Muñoz-Avila, A. Ram, and D. Shapiro (Eds.) *Goal-Directed Autonomy: Notes from the AAAI Workshop (W4)*. Atlanta, GA: AAAI Press.

Harland, J., Thangarajah, J., Morley, D., and Yorke-Smith, N. (2010). Operational behaviour for executing, suspending, and aborting goals in BDI agent systems. In A. Omicini, S. Sardina, and W. Vasconcelos (Eds.) *Declarative Agent Languages and Technologies: Papers from the AAMAS Workshop*. Toronto, CA.

Hawes, N. (2011). A survey of motivation frameworks for intelligent systems. *Artificial Intelligence*, **175**(5-6), 1020-1036.

Hawes, N., Zender, H., Sjöö, K., Brenner, M., Kruijff, G.J.M., and Jensfelt, P. (2009). Planning and acting with an integrated sense of space. *Proceedings of the First International Workshop on Hybrid Control of Autonomous Systems -- Integrating Learning, Deliberation and Reactive Control* (pp. 25-32). [http://www.hycas.org/2009/HYCAS2009-Proceedings.pdf]

Kaelbling, L., Littman, M., and Cassandra, A. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, **101**(1-2), 99-134.

Klenk, M. (2009). Transfer as a benchmark for multi-representational architectures. In U. Kurup and B. Chandrasekaran (Eds.) *Multi-Representational Architectures for Human-Level Intelligence: Papers from the AAI Fall Symposium* (Technical Report FS-09-05). Washington, DC: AAAI Press.

Klenk, M. (2010). Goal-driven autonomy in planning and acting. In D.W. Aha, M. Klenk, H. Muñoz-Avila, A. Ram, & D. Shapiro (Eds.) *Goal-Directed Autonomy: Notes from the AAAI Workshop (W4).* Atlanta, GA: AAAI Press.

Laird, J.E. (2008). Extending the Soar cognitive architecture. *Proceedings of the First Artificial General Intelligence Conference* (pp. 224-235). Memphis, TN: IOS Press.

Laird, J.E., Rosenbloom, P., and Newell, A. (1986). *Universal subgoaling and chunking*. Kluwer Academic Publishers, Boston, MA.

Langley, P. and Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence*. Boston, MA: AAAI Press.

Likhachev, M. and Stentz, T. (2009). Probabilistic planning with clear preferences on missing information. *Artificial Intelligence*, **173**, 696-721.

Marinier, B., van Lent, M., and Jones, R. (2010). Applying appraisal theories to goal directed autonomy. In D.W. Aha, M. Klenk, H. Muñoz-Avila, A. Ram, and D. Shapiro (Eds.) *Goal-Directed Autonomy: Notes from the AAAI Workshop (W4)*. Atlanta, GA: AAAI Press.

Meneguzzi, F.R., and 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.

Molineaux, M., Klenk, M., and Aha, D. (2010a). Goal-driven autonomy in a Navy training simulation. *In Proceedings of Twenty-Fourth AAAI Conference on Artificial Intelligence*. Atlanta, GA: AAAI Press.

Molineaux, M., Klenk, M., and Aha, D. (2010b). Planning in dynamic environments: Extending HTNs with nonlinear continuous effects. In *Proceedings of Twenty-Fourth AAAI Conference on Artificial Intelligence*. Atlanta, GA: AAAI Press.

Muñoz-Avila, H., Aha, D.W., Jaidee, U., and Carter, E. (2010). Goal-driven autonomy with case-based reasoning. *Proceedings of the Eighteenth International Conference on Case Based Reasoning* (pp. 228-241). Alessandria, Italy: Springer.

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., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., and Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, **20**, 379–404.

Newell, A., and Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

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

Ponsen, M.J.V., Lee-Urban, S., Muñoz-Avila, H., Aha, D.W., and Molineaux, M. (2005). Stratagus: An open-source game engine for research in real-time strategy games. In D.W. Aha, H. Muñoz-Avila, & M. van Lent (Eds.) *Reasoning Representation, and Learning in Computer Games: Papers from the IJCAI Workshop* (Technical Report AIC-05-127). Washington, DC: Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence.

Powell, J., Molineaux, M., & Aha, D.W. (2011). Active and interactive learning of goal selection knowledge. In *Proceedings of the Twenty-Fourth Florida Artificial Intelligence Research Society Conference*. West Palm Beach, FL: AAAI Press

Ram, A., and Leake, D. (1995). *Goal-driven learning*. Cambridge, MA: MIT Press.

Rao, A., and Georgeff, M. (1995). BDI agents: From theory to practice. *Proceedings of the First International Conference on Multi-agent Systems* (pp. 312-319). San Francisco, CA: AAAI Press.

Roos, N., and Witteveen, C. (2009). Models and methods for plan diagnosis. *Autonomous Agents and Multi-Agent Systems*, **19**, 30-52.

Sardina, S., and Padgham, L. (2010) A BDI agent programming language with failure handling, declarative goals, and planning. *Autonomous Agents and Multi-Agent Systems*. Online First: Springer. [http://www.springerlink.com/content/x87k477806p2k15m/fulltext.pdf]

Scheutz, M., and Schermerhorn, P. (2009). Affective goal and task selection for social robots. In J. Vallverdú and D. Casacuberta (Eds.) *The Handbook of Research on Synthetic Emotions and Sociable Robotics: New Applications in Affective Computing and Artificial Intelligence*. IGI Global: Hershey, PA.

Settles, B. (2009). *Active learning literature survey* (Technical Report 1648). Madison, WI: University of Wisconsin-Madison, Department of Computer Sciences.

Sutton, R.S., and Barto, A.G. (1998). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.

Talamadupula, K., Benton, J., Schermerhorn, P., Kambhampati, S., and Scheutz, M. (2009). Integrating a closed world planner with an open world robot: A case study. In M. Likachev, B. Marthi, C. McGann, and D.E. Smith (Eds.) *Bridging the Gap between Task and Motion Planning: Papers from the ICAPS Workshop*. Thessaloniki, Greece.

Weber, B., Mateas, M., and Jhala, A. (2010). Applying goal-driven autonomy to StarCraft. In *Proceedings of Artificial Intelligence and Interactive Digital Entertainment*. Palo Alto, CA: AAAI Press.