Goal-driven Autonomy Without GDA

Bryan McKenney and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire, USA
Bryan.McKenney@unh.edu, ruml@cs.unh.edu

Abstract

Goal-driven autonomy (GDA) is an agent control architecture proposed to enable long-term autonomous operation. At its heart is the idea of goal reasoning, in which a dedicated subsystem decides which goals the system should pursue given the current context. In this position paper, we summarize and extend a line of work that argues that a specialized goal reasoning subsystem is unnecessary and that long-term autonomous operation of complex intelligent systems should simply be directed by a planner (although perhaps not a classical one). We present several examples of how functionality that has been described as goal reasoning can be performed by modern planning techniques.

Introduction

There are many situations in which it is useful to have robots that can operate autonomously for extended periods of time in unknown environments. Autonomous cars, unmanned military craft, Mars rovers, and rescue robots are but a few examples of agents that must plan intelligently on their own, revising goals when necessary as new information is learned about the environment or if the environment changes in unexpected ways. This kind of planning is known as *online open-world planning*.

Molineaux, Klenk, and Aha (2010); Klenk, Molineaux, and Aha (2013) introduce a framework for solving online open-world problems called *goal-driven autonomy* (GDA), in which the agent manages a set of goals while trying to achieve them. In GDA, goal reasoning is made explicit and separated from planning. They test their GDA-driven agent, ARTUE, on three Navy-themed domains, two of which involve a Navy ship agent and a hidden submarine adversary, and they claim on the basis of these results that GDA is necessary to achieve high performance in such scenarios.

In this paper, we strengthen and extend the argument of Paredes and Ruml (2017) that GDA is not necessary. We use a domain more similar to that of Molineaux, Klenk, and Aha, with a Navy ship agent and an unknown number of hidden submarine adversaries that are trying to destroy cargo ships, to show that a much simpler pure-planning approach called *hindsight optimization*, in which the agent samples possible worlds and plans in all of them, works just as well as GDA. Interestingly, we find that our planner gives rise to patrolling behavior without the need to explicitly program that behavior. A Navy ship agent using hindsight optimization can imagine possible actions of possible adversaries and how its actions might thwart them, thereby implicitly employing an intelligent patrolling strategy to protect the cargo ships without needing to explicitly reason about its multiple goals. We also find that the agent can implicitly switch goals in the face of unexpected events.

Background

In this section, we explain goal-driven autonomy and hindsight optimization in the context of previous work.

Goal-driven Autonomy

Molineaux, Klenk, and Aha (2010) observe that agents in partially-observable, open-world, adversarial, stochastic domains with continuous time and space (such as video games and simulations) need a way to deal with unexpected events that ruin plans. They introduce the goal-driven autonomy (GDA) algorithm framework as the solution to this problem. GDA has three sub-systems: a Planner, a State Transition System, and a Controller, the last of which deals with goal reasoning. The Controller has four components: 1) The Discrepancy Detector, which compares the observed state to the expected state to find discrepancies; 2) The Explanation Generator, which hypothesizes causes for the discrepancies based on what it knows about the environment; 3) The Goal Formulator, which creates new goals based on the explanations; and 4) The Goal Manager, which prioritizes goals. Molineaux, Klenk, and Aha claim that only GDA simultaneously relaxes four classical planning assumptions — deterministic environments, static environments, discrete effects, and static goals.

They introduce a specific instantiation of GDA called ARTUE (Autonomous Response to Unexpected Events). For its Planner, ARTUE uses a Hierarchical Task Network (HTN) planner that predicts the future to anticipate exogenous events. ARTUE's Goal Formulator relies on domain-dependent *principles* to map explanations of discrepancies to new goals, and the Goal Manager prioritizes goals based on their hard-coded intensity levels. Molineaux, Klenk, and Aha test ARTUE with success on three scenarios in the Tactical Action Officer (TAO) Sandbox, a naval simulation in which the agent is a Navy ship. These three scenarios are: 1) *Scouting*, in which the initial goal is to identify nearby ships until an unexpected submarine attack adds the goal of identifying and destroying the sub; 2) *Iceberg*, in which the initial goal is to transport cargo between points until lightning strikes an iceberg and a severe storm arises, adding the additional goals of seeking shelter and rescuing members of a sinking ship; and 3) *SubHunt*, in which the goal is to seek and destroy a submarine while also sweeping mines that it lays.

Hindsight Optimization

Hindsight optimization is a planning algorithm that finds suboptimal solutions to problems involving uncertainty by sampling possible worlds, each representing a possible resolution of the uncertainty, and planning in those. It was introduced to the planning community by Yoon et al. (2008) and has since been successfully applied to a wide range of problems, from manufacturing and unmanned aerial vehicle flight patterns (Burns et al. 2012) to robot search-and-rescue and making omelettes (Kiesel et al. 2013).

Paredes and Ruml (2017) use hindsight optimization to argue against Molineaux, Klenk, and Aha (2010)'s claim that goal reasoning needs to be separate from planning for complex domains. They create a partially-observable, open-world, online, stochastic, multi-unit, adversarial domain called Harvester World to aid their argument. In this grid-based domain, the agent controls a Harvester and a Defender, there is an Enemy that can be seen from one unit away from them but is otherwise invisible, and there are also hidden obstacles and food around the map. The agent knows the Enemy's policy. Paredes and Ruml successfully use hindsight optimization in three different Harvester World scenarios and argue that it implicitly has all of the components of goal-driven autonomy. Hindsight optimization is a type of *multilevel planner*, as the high-level deterministic planner acts as a heuristic function for the low-level hindsight algorithm, and this can be viewed as reasoning about goals and choosing the best one to pursue.

Like Paredes and Ruml, we use a domain (described more fully below) that is grid-based, partially-observable, open-world, online, stochastic, and adversarial. The adversaries can be detected if they are one space away from the agent. However, our domain has more similarities with TAO Sandbox than Harvester World does: it has an unknown number of adversaries, instead of just one, and it has only one unit controlled by the agent, instead of two. Additionally, the agent has an inaccurate (but reasonable) model of the adversaries, which makes things more challenging. We are the first to show that hindsight optimization can lead to emergent patrolling behavior, which can be seen as a strategic type of goal of maintenance.

Algorithm 1 outlines hindsight optimization. The algorithm first samples a certain number of states from the agent's *belief state* (line 1), which is the set (or, more realistically, a subset) of all possible current states based on the agent's complete history of observations. A seed is associated with each state that will be used to resolve stochastic effects. Then, for each possible action, a deterministic planner is run in each of these sampled possible states up to a certain horizon (line 6). The action that led to the lowest average cost across the possible states is chosen (line 11).

Algorithm 1: Hindsight Optimization(o, N, H)

```
1:  $sampleStates \leftarrow hallucinate(N)$
2:  **for all** actions $a$ applicable in $o$ **do**
3:      $sampleCosts \leftarrow [\,]$
4:      **for all** states $s$ in $sampleStates$ **do**
5:          $s', aCost \leftarrow f(s, a)$
6:          $c \leftarrow aCost + planCost(s', H)$
7:          append $c$ to $sampleCosts$
8:      **end for**
9:      $Q(o, a) \leftarrow mean(sampleCosts)$
10: **end for**
11: **return** $argmin_a Q(o, a)$
```

Algorithm 2: planCost(s, H, $t \leftarrow 0$)

```
12: **if** $t = H$ **then**
13:     **return** $0$
14: **end if**
15: **for all** actions $a$ applicable in $s$ **do**
16:     $s', aCost \leftarrow f(s, a)$
17:     $costToGo \leftarrow planCost(s', H, t + 1)$
18:     $Q(s, a) \leftarrow aCost + costToGo$
19: **end for**
20: **return** $min_a Q(s, a)$
```

Algorithm 2 outlines a basic deterministic depth-first horizon-limited planner. We use branch-and-bound and memoization to improve efficiency. (Our implementation is sufficiently fast and we have not optimized runtime — child ordering using a heuristic function would likely make the planner even faster.)

Implicit Goal-driven Autonomy

GDA has a dedicated sub-system for goal reasoning and requires a host of hard-coded goals and principles, but this is not the only way to achieve autonomous goal-driven behavior. Another way is to abstract the agent's goals down to cost values that are incurred by certain events and then make its single goal to minimize cost. When paired with an appropriate planner and a belief state that tracks knowledge about the environment (fulfilling the tasks of the Discrepancy Detector and the Explanation Generator), these costs allow trading off between multiple goals and there is no need for a Goal Formulator or Goal Manager — goal formulation and management will be done implicitly while following the prime directive to minimize cost. This eliminates the need for a separate Controller. We present hindsight optimization as just one possible appropriate planner — other possibilities include approximate POMDP planners like POMCP (Silver and Veness 2010). To provide empirical evidence for our position, we ran an experiment with an agent using hindsight optimization in a simple domain, described below, which shares similarities with the TAO Sandbox domains used by Molineaux, Klenk, and Aha.

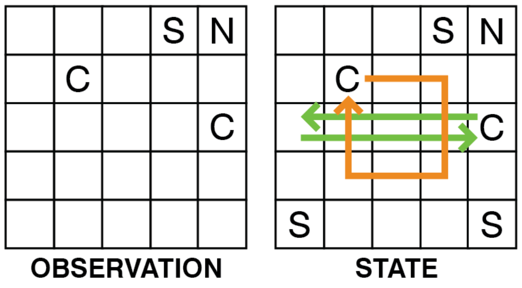

Figure 1: An example of the Navy Defense domain showing the agent's observation on the left and the true state on the right.

Experimental Study

In this section, we will first describe the Navy Defense domain and its relation to the TAO Sandbox, then the test instances and benchmark algorithms that we used in our experiment, and finally the experiment itself and its results.

The Navy Defense Domain

In this domain, a Navy ship must defend cargo ships from being destroyed by hidden submarines for as long as possible in a grid-world of variable size. There is one Navy ship (the agent) and a variable number of cargo ships (allies) and submarines (adversaries) in different starting positions. A 5x5 example is shown in Figure 1 with one agent/Navy ship (marked N), two cargo ships (C) to protect, and three invisible submarines that are trying to destroy them (S). The arrows show the fixed paths that the cargo ships move along. The Navy ship, cargo ships, and submarines each occupy a single space on the grid and have 2 health. When a ship or sub is reduced to 0 health, it is destroyed (removed from the world). Multiple ships and subs can occupy the same space. The agent can only observe submarines that are within a 1-space sonar radius (including diagonals) of it, and it does not know how many submarines are in the world (but it does know the maximum number that there could be). The agent has unlimited time to contemplate its next move while the world stands still. After the agent takes an action, cargo ships and then submarines (in the order that they were created in) make their moves. All submarines decide what they will do before any of them acts. We define On Hit to mean that a vessel has sustained damage but still has more than 0 health afterwards, and On Destroy to mean that it is destroyed. The ships and subs work in the following ways:

Navy Ship (Agent) Can stay still or move one space in any of the four cardinal directions (within the grid boundary), then deals 1 damage to all submarines within sonar radius. Moving incurs a cost of 1; On Hit, incurs a cost of 10; On Destroy, incurs a cost of 40.

Cargo Ship (Ally) Travels automatically in a rectangle, staying a constant distance from the edge of the grid, in a predetermined direction (clockwise or counterclockwise). On Hit, incurs a cost of 20; On Destroy, incurs a cost of 80. This reflects the idea that the cargo ships are more valuable than the Navy ship; for example, if there is one time step left and either the cargo ship or the Navy ship will be destroyed, the agent would be expected to sacrifice itself for its mission. The future value of the Navy ship, insofar as it can protect cargo ships over time, is up to the planner to reason about.

Submarine (Adversary) Moves orthogonally, avoiding the Navy ship's sonar radius, to the nearest point of intersection along a cargo ship's route. Can also stay still to lie in wait. After moving or staying still, deals 1 damage to all ships on its space. If inside the Navy ship's sonar radius, will move out of it, if possible, or move onto the Navy ship's space otherwise. Breaks ties between equally good moves randomly. The agent believes that subs choose to target a cargo ship at random and then hunt it down until it is destroyed before switching target, but this is not the true behavior (a sub will switch targets to whichever cargo ship it can intercept more quickly).

For small worlds, the agent can keep an exact belief state (the probability of each possible current state), but for larger worlds an unweighted particle filter belief state (which is used by POMCP (Silver and Veness 2010)) is ideal.

Navy Defense vs. TAO Sandbox

Like the TAO Sandbox, the Navy Defense domain is online, partially observable, open world, stochastic, adversarial, infinite horizon, and features a Navy ship agent. Navy Defense is grid-based, however, while TAO Sandbox is not.

Navy Defense is similar to the *Scouting* scenario because there is a hidden submarine that attacks ships and the agent can see it only by using sensors and is able to destroy it. In *Scouting*, however, the agent gets rewarded for destroying the submarine, which is not the case in Navy Defense, and in Navy Defense there can be more than one submarine.

Navy Defense is similar to the *SubHunt* scenario as well, but in *SubHunt* the goal is to find and destroy the submarine, while in Navy Defense destroying subs is not necessary as long as the cargo ships are protected. The mines in *SubHunt* can be considered an unknown number of static adversaries, and in Navy Defense there are an unknown number of dynamic adversaries.

Test Set-up

Navy Defense worlds were randomly generated for the following experiment. They each have 7 rows and columns, 4 cargo ships, and 1-3 subs. We compared the performance of our hindsight optimization planner to the following agent strategies:

Static Does not move.

Random Chooses an action to take at random.

Patrol Moves in a rectangle with size based on initial location (just like the cargo ships). There are two versions of this algorithm, CW and CCW, which determine which direction the agent will move in (clockwise or counterclockwise). Patrol is similar to the PLAN1 benchmark algorithm for ARTUE.

Reactive Like Patrol, except after a nearby cargo ship is attacked, moves to protect it and then updates its rectangular course based on its new location and its direction

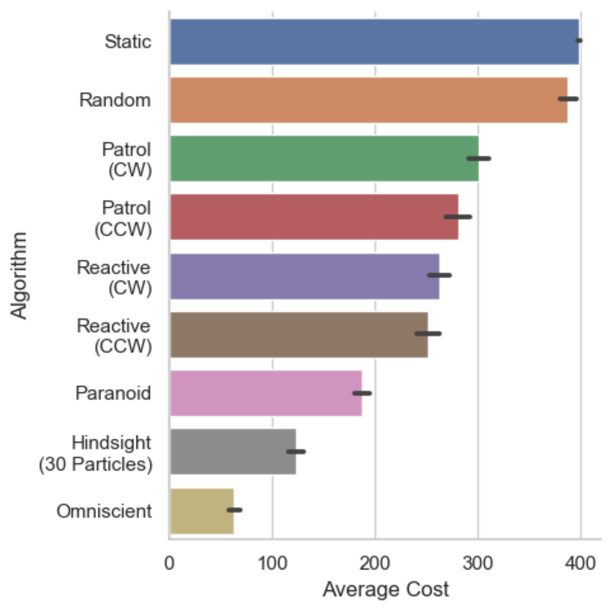

Figure 2: The average cost achieved by each algorithm.

based on that cargo ship's direction. Reactive is similar to the REPLAN benchmark algorithm for ARTUE.

Paranoid Like Hindsight but never updates its belief state (so it never learns where subs actually are).

Omniscient Uses the same planner as Hindsight (and so has a lookahead horizon) but knows the true state of the world (can see all subs) and uses that instead of hallucinating possible worlds.

Results

Each algorithm was run for 30 time steps on 200 random worlds. Hindsight used an unweighted particle belief state with 30 particles. Paranoid and Hindsight used sample sizes of 30 and horizons of 5 and Omniscient used a horizon of 5. 3 trials were done per world with each algorithm. The same seeds were used for each set of trials.

Figure 2 presents the mean cost experienced by each method, along with 95% confidence intervals. The results show that Hindsight beats all of the benchmark algorithms except Omniscient, which we would expect to represent an upper bound on achievable performance. The two versions of Patrol performed about the same, and so did the two versions of Reactive. This is not surprising, as the only difference between each version is starting direction, and the worlds are random, so neither starting direction should be better than the other on average. Sampling possible worlds and planning in them, even without updating belief state, is much more effective than the Static, Random, Patrol, and Reactive strategies, as evidenced by the large drop in cost from Reactive to Paranoid. Hindsight exhibits emergent patrolling behavior, an example of which can be viewed here: https://youtu.be/Gh6Ku05Y880. It also displays implicit goal reasoning in an example where it believes that there are no submarines to begin with (and thus stays still to avoid the movement cost) but has to reevaluate its beliefs and "goals" when both cargo ships are simultaneously attacked: https://youtu.be/FbqZVc9gCJg.

Discussion

The results show that hindsight optimization works well on a small grid-based domain that shares some similarities with the TAO Sandbox domains used by Molineaux, Klenk, and Aha. The agent is never given the goal to patrol or a rule stipulating that it destroy a submarine when doing so would not put cargo ships in danger from other subs, and yet it acts in these ways. In other words, it displays goal-driven autonomy without GDA.

Although Molineaux, Klenk, and Aha (2010) showed that their system did not perform as well when its goal reasoning component was removed, this provides only weak evidence that goal reasoning is useful. After all, removing the carburetor from an internal combustion engine does not prove that it is a necessary component — fuel injection or an electric motor might in fact be superior. Similarly, we expect that using a full-fledged planner would provide superior performance to a hand-tuned goal prioritization scheme, as only a planner can properly estimate the cost of achieving each possible outcome in the current context. One could call this goal reasoning, but the fact remains that planning-type reasoning is essential in action selection.

In future work, we could modify the Navy Defense domain by adding features from the TAO Sandbox *Iceberg* scenario, such as allowing the agent to rescue people from sinking cargo ships, to show that the agent is able to display goal reasoning in more complex situations.

Conclusion

In this paper, we add to prior arguments that planning can enable goal-driven autonomy without explicit goal reasoning. To provide an example, we introduced the open-world adversarial Navy Defense domain, which mimics some features of the TAO Sandbox, and then showed that a planner based on hindsight optimization can be used to find good online suboptimal solutions in Navy Defense scenarios. With hindsight optimization, patrolling and goal reasoning behavior naturally emerge from the single goal of minimizing cost. This is a simpler and more robust approach than GDA, as most of the reactive behavior in GDA is hard-coded. This work provides support for the notion that a single capable planner is a better solution than the complex GDA architecture for long-term autonomous operation.

Acknowledgments

Funding for this research was provided by the UNH Hamel Center for Undergraduate Research via the SURF USA program. Funding for AAAI 2023 registration and travel were provided by AAAI, the UNH Department of Computer Science, and the UNH Hamel Center for Undergraduate Research.

References

Burns, E.; Benton, J.; Ruml, W.; Yoon, S.; and Do, M. 2012. Anticipatory On-line Planning. In *Proceedings of ICAPS-12*.

Kiesel, S.; Burns, E.; Ruml, W.; Benton, J.; and Kreimendahl, F. 2013. Open World Planning for Robots via Hindsight Optimization. In *Proceedings of the ICAPS-13 PlanRob Workshop*.

Klenk, M.; Molineaux, M.; and Aha, D. W. 2013. Goal-Driven Autonomy for Responding to Unexpected Events in Strategy Simulations. *Computational Intelligence*.

Molineaux, M.; Klenk, M.; and Aha, D. 2010. Goal-Driven Autonomy in a Navy Strategy Simulation. In *Proceedings of AAAI-10*.

Paredes, A.; and Ruml, W. 2017. Goal Reasoning as Multilevel Planning. In *Proceedings of the ICAPS-17 IntEx Workshop*.

Silver, D.; and Veness, J. 2010. Monte-Carlo Planning in Large POMDPs. In *Proceedings of NIPS-10*.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *Proceedings of AAAI*.