# Online Open-World Adversarial Planning

**Bryan McKenney and Wheeler Ruml**

Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
Bryan.McKenney@unh.edu, ruml@cs.unh.edu

### Abstract

Hindsight optimization is a simple but powerful online planning algorithm that has applications in games (Paredes and Ruml 2017), robot search-and-rescue, making omelettes (Kiesel et al. 2013), manufacturing, unmanned aerial vehicle flight patterns (Burns et al. 2012), and more. We show that it also finds good suboptimal solutions in an open-world domain with an unknown number of invisible adversaries, and thus provides a straightforward alternative to Molineaux, Klenk, and Aha's complex goal-driven autonomy algorithm.

## Introduction

When an agent in some world must decide upon a sequence of actions to perform to achieve some goal, that is known as *planning*. In many areas of planning, it is assumed that everything about the world is known and visible to the agent, but real-world situations seldom have these characteristics. A more realistic problem is one that is *partially observable* and *open world* — in other words, the agent cannot see everything in the world and it does not even know exactly what is out there. In problems like this, it is prohibitively time-consuming to compute an optimal strategy for every possible situation, as *offline* planners do — a suboptimal *online* agent that observes the world after each action and uses its current information to plan its next action is necessary. When *adversaries* that actively work against the interests of the agent are introduced to the world, the problem becomes known as a *strategic game*. The focus of this research is suboptimal online planning for single-agent open-world strategic games with discrete state and action spaces, infinite horizon, and no time pressure.

Molineaux, Klenk, and Aha introduce a complex method for solving online open-world problems called *goal-driven autonomy* (GDA) in which the agent manages a set of goals whilst trying to achieve them; goal reasoning is made explicit and separated from planning. They test their GDA algorithm, ARTUE, on three Navy-themed domains, two of which involve a Navy ship agent and a hidden submarine adversary, and they claim that GDA is necessary to solve such situations.



In this paper, we use a similar domain with a Navy ship agent and an unknown number of hidden submarine adversaries that are trying to destroy cargo ships to show that GDA is *not* necessary — a much simpler approach called *hindsight optimization*, in which the agent samples possible worlds and plans in all of them, works just as well. A Navy ship agent using hindsight optimization will employ an intelligent patrolling strategy to protect the cargo ships without needing to explicitly reason about its goals.

## Previous Work

In this section, we summarize papers that preceded this work and how they are relevant to our research. We first discuss two papers about solving games, then one paper about goal-driven autonomy, and finally three papers that utilize hindsight optimization.

### Games

The following two papers describe methods of solving partially observable games. One of the goals of this research is to show that hindsight optimization works on open-world adversarial games, but at the current stage the domain is simplified and does not actually represent a game. This is because the adversaries do not reason about the agent's actions and try to plan around them, so they are more akin to dynamic environmental hazards than hostile agents.

**Hansen, Bernstein, and Zilberstein, "Dynamic Programming for Partially Observable Stochastic Games," AAAI 2004.** In this paper, Hansen, Bernstein, and Zilberstein describe a method of solving POSGs (Partially Observable Stochastic Games) that involves combining dynamic programming for POMDPs (Partially Observable Markov Decision Processes) with iterated elimination of dominated strategies for normal-form games. In POMDPs, beliefs are about the underlying state, in normal-form games (which do not have states or transition or reward functions), they are about the strategies of other agents, and in POSGs, they are about both. A POSG with a single agent is a POMDP, and a POSG can be converted into a normal-form game with hidden state, which is what the algorithm does. It is an exact algorithm, so it is guaranteed to find the optimal solution for co-op games, but it only works on small problems. A *strategy* is a complete conditional plan that can be represented by

a *policy tree*. All possible strategies are constructed for each agent, pruning *very weakly dominated* strategies, or strategies that have less or equal value to all other strategies, along the way, and then the algorithm alternates between agents, eliminating strategies that are not optimal against the others, until all agents have been narrowed down to a single strategy.

This paper was useful for its introduction to POSGs, even though our work does not currently involve POSGs, due to our deterministic domain and non-agent adversaries. With one agent, states, and a transition function, our domain is a POMDP. Hansen, Bernstein, and Zilberstein's algorithm is also offline and optimal, while our approach is the complete opposite. The one similarity is partial observability.

**Zinkevich et al., "Regret Minimization in Games with Incomplete Information," NIPS 2007.** In this paper, Zinkevich et al. introduce an algorithm for finding approximate solutions to extensive games with imperfect information with up to $10^{12}$ game states. An extensive game with imperfect information can be modelled as a game tree and information sets for each player about the part of the state that they cannot observe. Chance is considered a player. The algorithm involves finding the $\epsilon$-*Nash equilibrium* (where each player chooses roughly the best strategy for themselves) by minimizing *counterfactual regret*, a novel application of regret minimization that is akin to Bellman backups. *Regret* is essentially the utility missed out on by not playing a certain strategy. Zinkevich et al. test their algorithm on two-player limited Texas Hold'em poker and beat champion algorithms.

This paper was useful for its introduction to extensive games with imperfect information, which our research might eventually use, as our domain is partially observable and is planned to become a game. The counterfactual regret algorithm is approximate due to the large number of states, and we are using an approximate algorithm for the same reason.

## Goal-Driven Autonomy

The following paper is the origin of goal-driven autonomy.

**Molineaux, Klenk, and Aha, "Goal-Driven Autonomy in a Navy Strategy Simulation," AAAI 2010.** In this paper, Molineaux, Klenk, and Aha state 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, which separates goal reasoning from planning. GDA has five components: 1) The Hierarchical Task Network Planner, which predicts the future to anticipate exogenous events; 2) The Discrepancy Detector, which compares the observed state to the expected state to find discrepancies; 3) The Explanation Generator, which hypothesizes causes for the discrepancies based on what it knows about the environment; 4) The Goal Formulator, which uses domain-dependent *principles* to map explanations of discrepancies to new goals; and 5) The Goal Manager, which prioritizes goals based on their hard-coded intensity levels. Molineaux, Klenk, and Aha claim that only GDA simultaneously relaxes all four classical planning assumptions — deterministic environments, static environments, discrete effects, and static goals. They introduce a possible implementation of GDA called *ARTUE* (Autonomous Response to Unexpected Events) and test it 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) *Sub-Hunt*, in which the goal is to seek and destroy a submarine while also sweeping mines that it lays.

This paper was useful for introducing GDA, as we are trying to show that GDA is not necessary by using a similar Navy-themed domain. A detailed comparison between the TAO Sandbox scenarios and our domain is given in the Evaluation section. As described by Paredes and Ruml, hindsight optimization fulfills the five components of GDA without having them be explicit parts of the algorithm and without the need for hard-coded goals.

## Hindsight Optimization

The following three papers show that hindsight optimization can be used to solve various problems.

**Burns et al., "Anticipatory On-line Planning," ICAPS 2012.** In this paper, Burns et al. define *on-line continual planning problems* (OCPPs) as problems where new goals can arrive while the agent is working towards other ones, which may require a change in plans. They argue that *reactive planning*, which can change its strategy only after something unexpected happens, does not do well on OCPPs because it ignores information about possible future goals and thus is not prepared for their arrival. Burns et al. introduce *anticipatory on-line planning*, which is hindsight optimization applied in the context of online planning. In contrast to reactive planning, anticipatory on-line planning/hindsight optimization samples possible futures and solves each future using a deterministic planner to determine the best course of action, which is slower than reactive planning but more effective. In order to use hindsight optimization with an OCPP, the OCPP must formulated as a Markov Decision Process (MDP) where each state is comprised of a world state and a set of goals, and possible futures should be solved to minimize cost up to a certain horizon (the larger the horizon, the better the results). Burns et al. test this algorithm with success in a domain where unmanned aerial vehicles (UAVs) keep getting requests to fly over different strips of land and in a domain where orders to manufacture widgets can come in and machine parts can break every time step.

This paper was useful for introducing hindsight optimization and proving its proficiency in online settings. Like Burns et al., we use a domain that is an OCPP formulated as an MDP, although the only "goal" is minimizing cost. Our domain is grid-based, just like the UAV domain, but

ours has 4-way connectivity while the UAV grid has 8-way connectivity. We compare reactive planning to hindsight optimization, but our hindsight optimization maintains a belief state, which is beneficial in our partially-observable domain but would not be in the two domains tested in this paper (because they are fully observable).

**Kiesel et al., "Open World Planning for Robots via Hindsight Optimization," ICAPS 2013.** In this paper, Kiesel et al. argue that hindsight optimization is necessary for robotics, because the assumption that the world is *closed*, or that everything that is in it is known, is unrealistic. They give the example of a rescue robot searching a partially-destroyed building for survivors — the layout of the building is unknown, but the algorithm must work quickly, so it must generate possible world states (floor plans and victim locations) and plan in those. Kiesel et al. introduce *OH-wOW*, or Optimization in Hindsight with Open Worlds, which is hindsight optimization applied in the context of open worlds. It is assumed that the agent possesses probabilistic knowledge about the domain, even if it is not accurate, and the agent maintains a *belief state* that tracks current knowledge of the world so that sampled possible worlds make sense. The agent must plan to take sensing actions before interacting with hypothesized objects to make sure that they are actually there when executing a plan (if not, the plan will have to change). Kiesel et al. test this algorithm with success in a domain where a good omelette has to be made out of eggs that have different probabilities of being bad, a domain where packages that could be bombs have to be diffused in toilets, and the aforementioned robot search-and-rescue domain, which they even tested with a physical robot.

This paper was useful for proving hindsight optimization's proficiency in complex open-world settings. Like Kiesel et al., we use hindsight optimization with a belief state in an open-world domain. Our agent has accurate probabilistic knowledge of the domain in that it knows that the number of adversaries is in a certain range and that each possibility has equal probability. However, it does not perform sensing actions because sensing of nearby submarines is automatic.

**Paredes and Ruml, "Goal Reasoning as Multilevel Planning," ICAPS 2017.** In this paper, Paredes and Ruml argue against Molineaux, Klenk, and Aha's claim that goal reasoning needs to be separate from planning for complex domains. They create a partially-observable, open-world, online, multi-unit, adversarial domain called Harvester World (which is based on a game called Battle for Survival, which is similar to StarCraft) 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 introduce *GROH-wOW*, which is OH-wOW but it includes the horizon of the deterministic planner in the pseudo-code. They test this algorithm with success in three different Harvester World scenarios and show 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. Paredes and Ruml also use *macro actions* that are like easily-achievable goals to speed up the algorithm.

This paper was useful for proving hindsight optimization's proficiency in domains with invisible adversaries. Like Paredes and Ruml, we use a domain that is grid-based, partially-observable, open-world, online, and adversarial. The adversaries can also only be seen if they are one space away from the agent and the agent knows how they operate. In our world, however, there are an unknown number of adversaries instead of just one, and only one unit controlled by the agent instead of two.

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

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

## Approach

In this section, we will first formally define the problem, then describe the solution — hindsight optimization.

### Formal Problem Definition

An online partially-observable deterministic planning problem can be formally defined as a 7-tuple $\langle s_0, o_0, \mathcal{S}, \mathcal{A}, \mathcal{O}, f, g \rangle$. $s_0 \in \mathcal{S}$ is the initial state (unknown to the agent), $o_0 \in \mathcal{O}$ is the initial observation (what the agent knows about the initial state), $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, and $\mathcal{O}$ is the observation space. After the agent takes an action in a state, the transition function $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \mathbb{R}$ returns the next state and the cost of the action, but the concealer function $g : \mathcal{S} \to \mathcal{O}$ takes that state and converts it into an observation that is then given to the agent. The algorithm must choose an action to take on each of its turns with the goal of minimizing long-term cost. The agent has full knowledge of $o_0$, $\mathcal{S}$, $\mathcal{A}$, $\mathcal{O}$, $f$, and $g$ — only $s_0$ is unknown. For a strategic game with invisible deterministic adversaries, the adversaries are represented in states, but not in observations, and $f$ defines their behavior.

### Hindsight Optimization

Algorithm 1 outlines hindsight optimization. This algorithm works by sampling states from the agent's *belief state* (line 1), which is the set of all possible current states based on the history of observations. (In this case, the belief state

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

```
1: if t = H then
2:     return 0
3: end if
4: for all actions a applicable in s do
5:     s', aCost ← f(s, a)
6:     costToGo ← planCost(s', H, t + 1)
7:     Q(s, a) ← aCost + costToGo
8: end for
9: return min_a Q(s, a)
```

tracks the possible number and positions of invisible adversaries.) 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). This pseudo-code is slightly different from the way it is traditionally written because the *hallucinate* function (line 1) samples *up to* $N$ possible states from the agent's belief state, so if the possibilities have been narrowed down to less than $N$, the same state will not be sampled more than once. This improves efficiency.

Algorithm 2 outlines a basic deterministic planner. Our planner followed this blueprint but used the branch-and-bound and memoization techniques to improve efficiency. Child ordering using a heuristic function would also be a good addition to make the planner even faster.

## Evaluation

In this section, we will first describe our Navy Defense domain, then the benchmark algorithms that we compared hindsight optimization to, then how results were normalized, and finally the four experiments that we conducted.

### The Navy Defense Domain

In this domain, a Navy ship must defend cargo ships from being destroyed by hidden submarines in a grid-world of variable size (for a visual, see Figure 1). There is one Navy ship (the agent) and a variable number of cargo ships (allies) and submarines (adversaries) in different starting positions. The Navy ship, cargo ships, and submarines each take up 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 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. The ships and subs work in the following ways (a ship/sub's On Hit ability triggers when it takes damage but still has more than 0 health afterwards, and its On Destroy ability triggers when it is destroyed):

- Navy Ship (Agent)
  - Available Actions:
    - Move (in a valid orthogonal direction): Move 1 space in the chosen direction, then deal 1 damage to all submarines within sonar radius. (The agent does not have the option of moving in a certain direction if there is an edge of the grid there.)
  - On Hit: Incur cost of 10.
  - On Destroy: Incur cost of 40.
- Cargo Ship (Ally)
  - AI: Travel in a rectangle with size based on initial location (e.g. if it was originally in the top left corner of the grid it will sail around the border of the grid) in a predetermined direction (clockwise or counterclockwise).
  - On Hit: Incur cost of 20.
  - On Destroy: Incur cost of 80.
- Submarine (Adversary)
  - AI: Move 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, deal 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.

**Updating the Belief State** An agent using hindsight optimization on this domain can keep track of its belief state simply by keeping a list of known submarine locations and updating them via simulation each turn, then comparing these known locations to the observation, which includes ships and subs that were attacked. When a submarine attacks or is attacked, its precise location is learned, and it can be tracked because it acts deterministically.

**Navy Defense vs. TAO Sandbox** The Navy Defense domain is similar to the TAO Sandbox *Scouting* scenario that Molineaux, Klenk, and Aha tested ARTUE in because there is a hidden submarine that attacks ships and the agent — a Navy ship — 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 can be compared to the *SubHunt* scenario as well, but only in that the Navy ship can discover where the submarine is — 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 (and, again, there can be more than one sub).

Navy Defense is a grid-based domain, while TAO Sandbox is not.

**Test Instances** Three different types of Navy Defense worlds were randomly generated for the following experiments. These worlds are classified by size and have different characteristics:

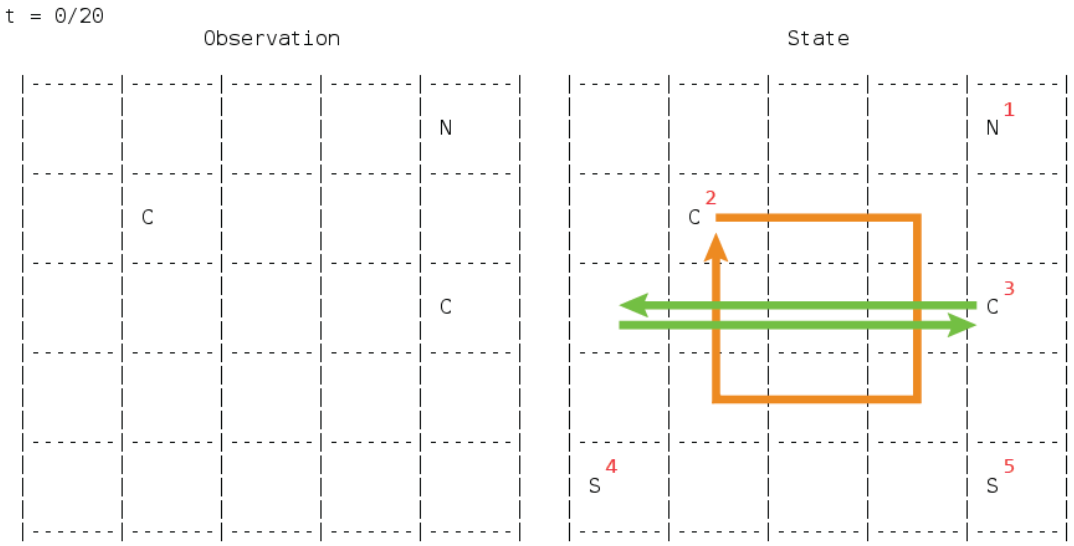


Figure 1: A 5x5 world in the Navy Defense domain with one agent/Navy ship (N), two cargo ships (C) to protect, and two invisible submarines that are trying to destroy them (S). The observation (what the agent sees) is on the left and the true state is on the right. The arrows show the fixed paths that the cargo ships move along, and the numbers indicate turn order.

- Small – Has 5-7 rows and columns, 1-2 cargo ships, and 1-3 subs.
- Medium – Has 8-10 rows and columns, 2-3 cargo ships, and 1-4 subs.
- Large – Has 11-13 rows and columns, 2-4 cargo ships, and 1-5 subs.

## Benchmark Algorithms

The basic algorithms that were compared to hindsight optimization (abbr. Hindsight) in the Navy Defense domain are:

- 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 counter-clockwise).
- 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 based on that cargo ship's direction.
- 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.

To see these algorithms (and Hindsight) in action in the world from Figure 1, go to https://imgur.com/a/rtK5vVV.

## Normalizing Results

The results from the following experiments mostly focus on the average normalized cost achieved by the algorithms. The normalized cost is calculated by dividing the cost accrued by the agent in a world by the maximum cost that could theoretically be attained in that world (for instance, a world with two cargo ships has a maximum cost of 250, which could be achieved if both of the cargo ships and the Navy ship were destroyed).

## Experiment 1: Finding Suitable Benchmark Instances

In this experiment, each algorithm was run for 30 time steps on 20 random worlds of each size. For the algorithms that sample possible worlds (Paranoid and Hindsight), sample sizes of 5, 10, and 15 were tested, and for the algorithms that have lookahead horizons (Paranoid, Hindsight, and Omniscient), horizons of 1, 3, and 5 were tested. 2 trials were done per combination for stochastic algorithms and 1 trial was done per combination for deterministic algorithms. The same seeds were used for each set of trials. The results of these algorithms were averaged across their different combinations, as were the results of the CW and CCW variants for Patrol and Reactive.

Figure 2 shows that, for Small and Medium worlds, Random performed the worst, then Patrol, then Reactive, then Paranoid, then Hindsight, and Omniscient performed the best. These results are as expected — the more the agent reasons or knows about the state of the world, the better it is able to protect the cargo ships. In Large worlds, however, Random and Patrol slightly outperformed Reactive, and Hindsight slightly outperformed Omniscient. This was

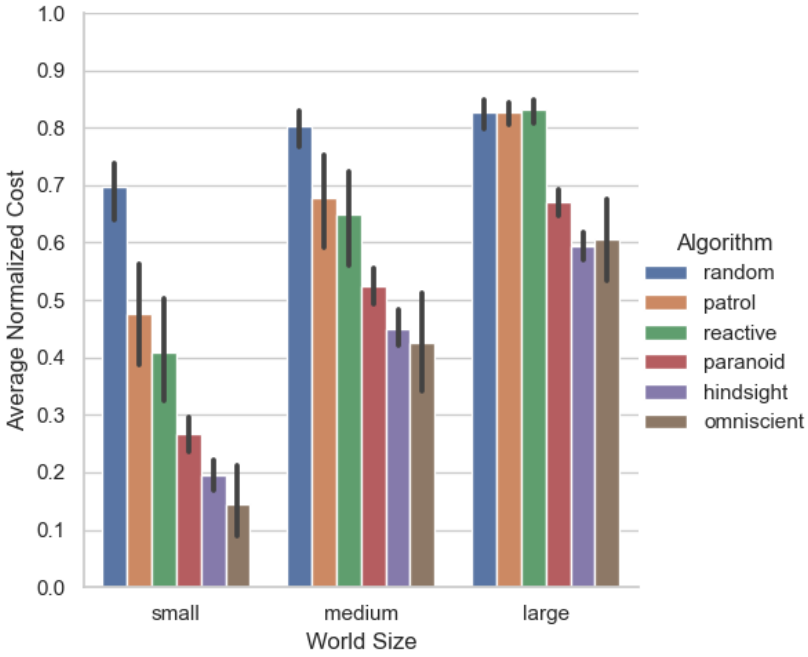


Figure 2: The average normalized cost achieved by each algorithm in the three world sizes. The black lines are 95% confidence intervals.

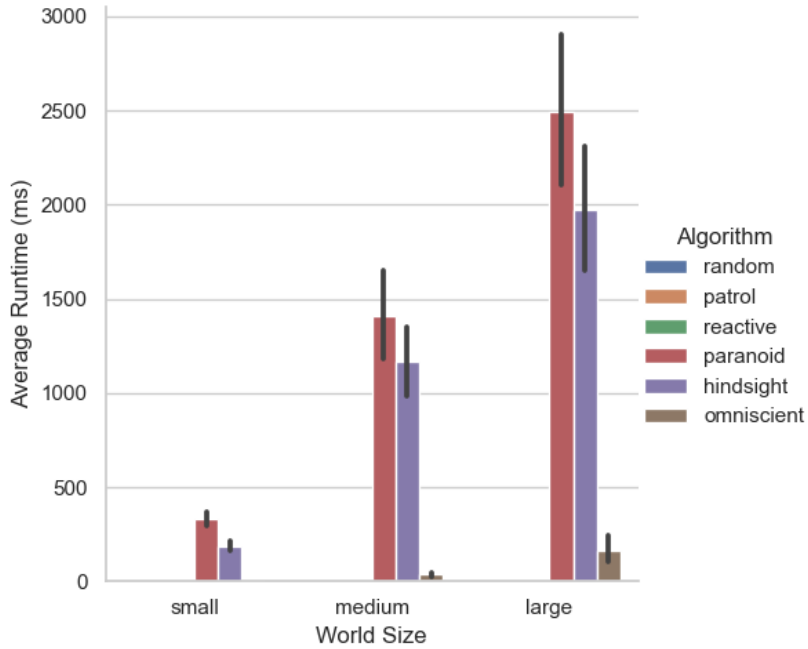


Figure 3: The average runtime of each algorithm in the three world sizes. The black lines are 95% confidence intervals.

probably just due to luck and the small number of worlds and trials. Since the difference between the algorithms' achieved average normalized costs in Large worlds was fairly small and the largest differences were seen in Small worlds, it was judged that Large worlds were too difficult for any algorithm (the agent could start too far away from the cargo ships to be able to reach them before the subs) and that only Small worlds would be used for further experimentation.

Figure 3 shows that, as expected, Random, Patrol, and Reactive take barely any time to run and this time does not increase with world size. Paranoid takes the most time to run (but still under 3 seconds in Large worlds), Hindsight the second-most, and Omniscient the third-most, and these algorithms do take longer to run the larger the world is because there are more valid directions to test travelling in. Hindsight is faster than Paranoid because, in some cases, it can narrow down the world to one possibility and then not have to hallucinate possible worlds and plan in them anymore (and since Omniscient starts with one possibility that is why it is dramatically faster than both).

### Experiment 2: Tuning Hindsight Optimization

In this experiment, Hindsight was run for 30 time steps on 100 random Small worlds. Sample sizes and horizons of 1, 2, 3, 5, 10, 15, 20, and 25 were tested. 2 trials were done for each combination. The same seeds were used for each set of trials.

Figure 4 shows that, no matter what the sample size is, increasing the horizon from 1 to 2 significantly reduces cost, but after that increasing it more does not have a significant effect (except for a sample size of 5). The plot also shows that, no matter what the horizon is, increasing the sample size up to 10 reduces cost, but beyond 10 does not make much difference. The latter result was expected, because the more possible worlds that are considered should increase the probability that the agent makes a good decision in the true world (but considering too many worlds is unnecessary because they are small and likely to have similar good actions). The former result was unexpected but makes sense — the Small worlds are just too small for the horizon to matter very much, because the agent quickly learns the true state once all of the submarines attack or are attacked, and once it is known the best action is usually obvious without looking too far ahead (but looking 1 step ahead is generally not enough). Thus, it was decided that for further experimentation, the algorithms that use a horizon would have it set to 5 and the algorithms that use a sample size would have it set to 20 (horizon 2 and sample size 10 could probably have been used, but using higher numbers just ensures that the algorithms perform as well as they can on these worlds).

### Experiment 3: Determining Necessity of Multiple Trials

In this experiment, the three stochastic algorithms — Random, Paranoid, and Hindsight — were run for 30 time steps on 5 random Small worlds. Paranoid and Hindsight used sample sizes of 20 and horizons of 5. 1,000 trials were done for each combination. The same seeds were used for each set of trials.

Figure 5 shows that Paranoid and Hindsight achieved similar cost distributions in each of the worlds, but Paranoid has greater variance than Hindsight in all worlds but smallWorld101, where they are equal. Random has very high variance in smallWorld0 and smallWorld10 and Paranoid and Hindsight have the highest variance in smallWorld10 and smallWorld100, which shows that these algorithms cannot perform consistently in some worlds. Thus, it was decided that when doing further experimentation with randomly generated worlds, multiple trials of each stochastic algorithm would be run on each world to provide more accurate results.

### Experiment 4: Comparing Algorithms

In this experiment, each algorithm was run for 30 time steps on 1,000 random Small worlds. Paranoid and Hindsight used sample sizes of 20 and horizons of 5 and Omniscient used a horizon of 5. 5 trials were done per combination for stochastic algorithms and 1 trial was done per combination for deterministic algorithms. The same seeds were used for each set of trials.

Figure 6 and Figure 7 reinforce the results found in Experiment 1 — Hindsight beats all of the benchmark algorithms except Omniscient, as is expected. 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 Random, Patrol, and Reactive strategies, as evidenced by the large drop in cost from Reactive to Paranoid. Hindsight exhibited patrolling behavior, which can be viewed here: https://imgur.com/a/KjFuBbU.

## Discussion

We have shown 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. However, the Navy Defense domain is rather simple, as it is deterministic and submarines can only attack ships on their grid space, so once a sub attacks or is attacked the agent knows exactly where it is and can accurately track it for the rest of the simulation. Adding some stochasticity (such as random tie-breaking when subs have two equally good paths to a cargo ship) and allowing subs to attack a nearby space will increase the number of possibilities for where subs can be and force the agent's belief state to be more advanced. The sub will also be made to reason about what the agent will do and plan around it, which will make the domain much harder (and a true game). To make the domain closer to the TAO Sandbox domains, features from the third Navy scenario Molineaux, Klenk, and Aha tested GDA on — *Iceberg*, which involves seeking shelter from a storm and rescuing people from a sinking ship — will be introduced. If hindsight optimization can still perform well in a Navy Defense domain that is made more difficult and complex in these ways, it will be even greater evidence that goal-driven autonomy is not the only solution to such problems.

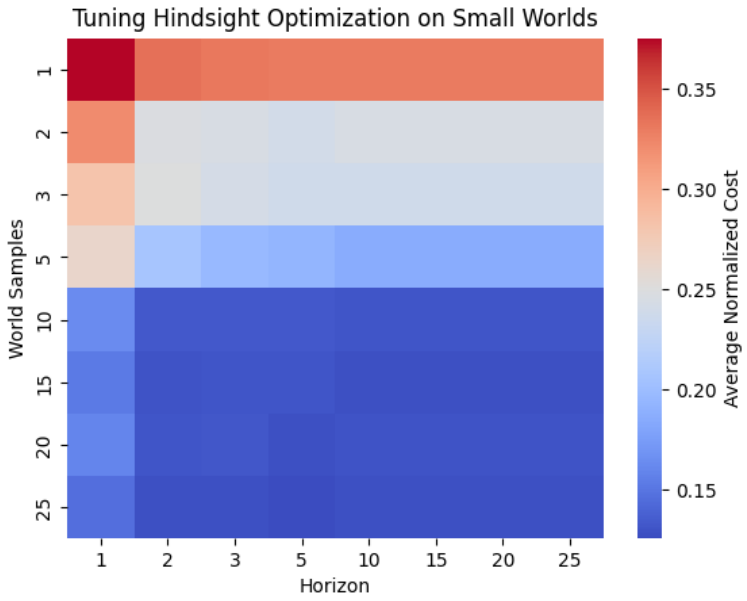


Figure 4: A heatmap showing how the average normalized cost is affected by running hindsight optimization with eight different sample sizes and horizons.

Another avenue for future research is to see if hindsight optimization (or some mutation of it) can succeed against problems that only neural networks are considered suitable for, such as playing very complex games. One such game is *Duelyst*, an online collectible card game that features a 9x5 grid board on which each player's General and minions do battle. *Duelyst* is partially observable and open world because you do not know the cards in your opponent's deck or in their hand. The game's stochasticity is not limited to drawing random cards from your deck, as some of the cards also have random effects when played, and the action space can be very large, depending on the number of cards in your hand and the number of minions you have on the board. In addition, several actions must be taken within a 90-second turn, which means that many observations must be quickly reacted to each turn. Speeding up hindsight optimization with child ordering and multithreading may be necessary to even give it a chance at finding decent solutions to such a complex domain.

## Conclusion

In this paper, we described hindsight optimization and introduced the open-world adversarial Navy Defense domain, which mimics some features of the TAO Sandbox. We then showed that hindsight optimization can be used to find good online suboptimal solutions in Navy Defense scenarios, making it a contender against goal-driven autonomy. Further research will show if it is truly superior, but it already displays emergent patrolling behavior only from attempting to minimize cost, while most of the reactive behavior in GDA is hard-coded.

## Acknowledgments

Thanks to the Hamel Center for Undergraduate Research and the donors who provided me with the funding for this research and to Professor Ruml for being a great mentor!

## References

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

Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic Programming for Partially Observable Stochastic Games. In *Proceedings of AAAI-04*.

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

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 Multi-level Planning. In *Proceedings of ICAPS-17*.

Zinkevich, M.; Johanson, M.; Bowling, M.; and Piccione, C. 2007. Regret Minimization in Games with Incomplete Information. In *Proceedings of NIPS-07*.

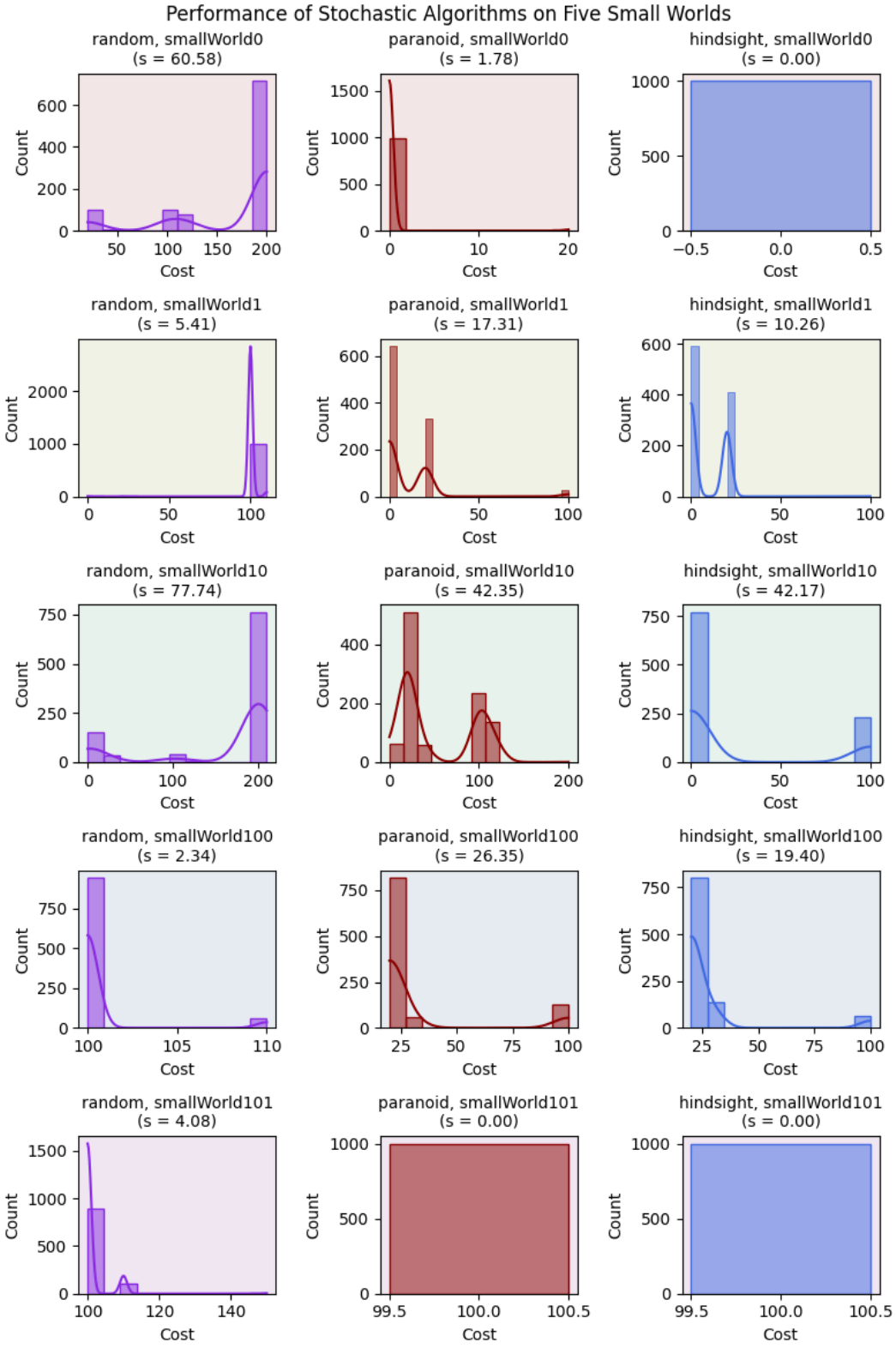


Figure 5: Distributions of the cost achieved by Random, Paranoid, and Hindsight in five worlds. The sample standard deviation for each combination is in parentheses.

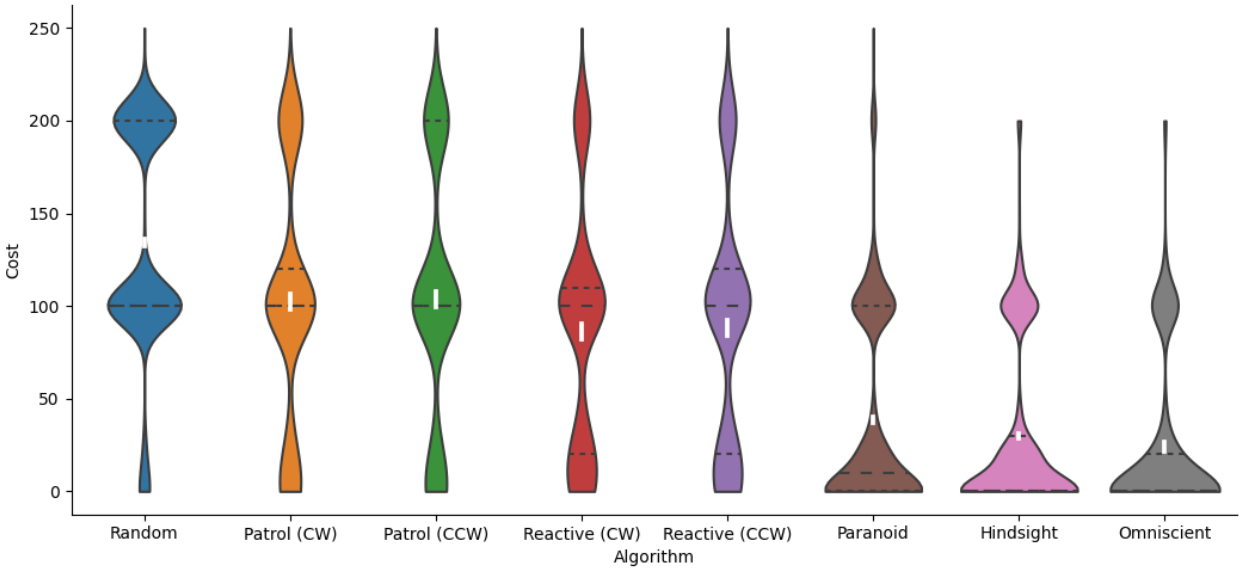


Figure 6: Distributions of the cost achieved by each algorithm across 1,000 worlds. The white lines show the average cost for each algorithm (at the center) with 95% confidence intervals.

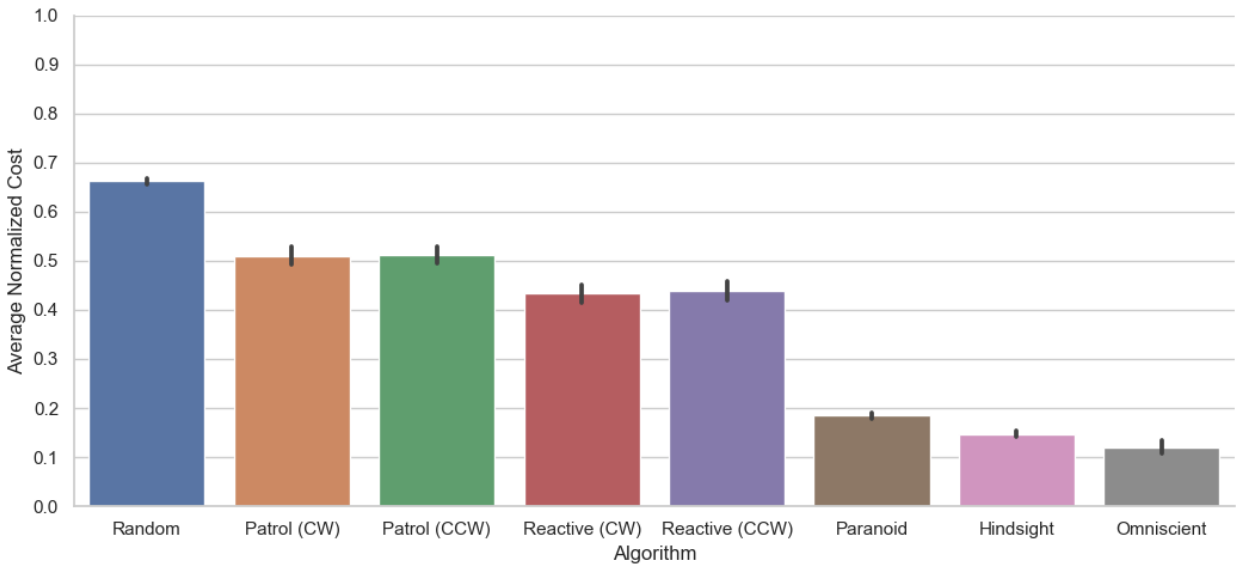


Figure 7: The average normalized cost achieved by each algorithm across 1,000 worlds. The black lines are 95% confidence intervals.