Goal Reasoning as Multilevel Planning

Alison Paredes and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
alison, ruml at cs.unh.edu

Abstract

There has been much recent interest in the topic of goal reasoning: where do an agent's goals come from and how is it decided which to pursue? Previous work has described goal reasoning as a unique and separate process apart from previously studied AI functionalities. In this paper, we argue an alternative view: that goal reasoning can be thought of as multilevel planning. We demonstrate that scenarios previously argued to support the need for goal reasoning can be handled easily by an on-line planner, and we sketch a view of how more complex situations might be handled by multiple planners working at different levels of abstraction. By considering goal reasoning as a form of planning, we simplify the AI research agenda and highlight promising avenues for future planning research.

Introduction

It is widely understood that plan synthesis is only part of the functionality that an agent needs with respect to taking intelligent action. For example, Molineaux, Klenk, and Aha (2010) posit a capacity for goal reasoning, which creates the goals that the agent's planner might attempt to achieve, determines which goals the agent will pursue at any particular moment, and monitors goal achievement. (They also include a sophisticated component for estimating the state of the world and updating the agent's model, given its actions and their observed consequences, but this is not our concern here.) In this paper, we consider whether goal reasoning is best thought of as a capability distinct from plan synthesis and action selection, or whether it might be possible to unify these two functionalities, thereby simplifying the AI research agenda. After reviewing the proposals of Molineaux, Klenk, and Aha (2010) regarding goal reasoning and their experimental benchmark scenarios, we introduce a new benchmark domain, called Harvester World, that captures many of the features of prior benchmarks. We then demonstrate a relatively simple planner, called GROH-wOW, that achieves high performance in Harvester World. We argue that this result undercuts the empirical support for goal reasoning claimed by Molineaux, Klenk, and Aha (2010) on the basis of their experiments. We then speculate about how goal reasoning functionality might be exhibited by multiple planners working together in various relationships.

Goal Reasoning as a Separate Module

A planner takes a model of the environment, a state and a goal and returns either an action or an entire plan. A goal reasoner takes a model of the environment, a current state and a goal, and returns either the same goal or a new one (Klenk, Molineaux, and Aha 2013). It also uses a sequence of expected states which should result from each action taken in the plan and the observed result of the previous action in the plan to: 1) detect discrepancies between expected states and the observed current state, 2) explain these discrepancies, which may modify the current model of the environment, 3) generate new goals which might be applicable in this new understanding of the environment, and 4) finally, decide which among the current goals and any newly generated goals should be pursued next.

For example, in a real-time strategy game such as Bos Wars (formally known as Battle of Survival), a game very similar to Starcraft, a planner might be given the goal to move a friendly harvester unit to its home base. The planner returns a sequence of actions such as, move the harvester from point A to B to C to D around an known obstacle until it reaches the base. For the use of the goal reasoner, the planner might also return a sequence of expected states such as after moving the harvester from point A to point B, the harvester is at point B. After each action is executed in a plan, the goal reasoner should get the game's current state, which in this case would be the observation that the harvester is at point B. Given this information, the goal reasoner might conclude that it should continue to pursue its current goal to move the harvester to its home base. In a partially observable environment, however, it is possible that the goal to move the harvester to its home base is not the best one to pursue in the long run, and it is part of the goal reasoner's responsibility to figure that out. For example, it might make more sense to first deploy some kind of defenses before sending the harvester home because we have reason to believe that an enemy may be lurking nearby.

Klenk, Molineaux, and Aha (2013) implement a planner with a goal reasoner and test it in comparison to planning without one. In these experiments, the goal reasoner performed better than either planning once off-line or replanning to the initial goal. In their scenarios, the most successful behavior involved responding to unobserved factors in a partially observable environment. The tests used two different simulations, an instance of the open source Battle of Survival (an older version of Bos Wars) and a proprietary Navy training simulation; in all scenarios there was some element of partial observability.

In this paper we focus on the three scenarios run in Battle for Survival (BoS): 1) Resource Gathering, 2) Escort, and 3) Exploration. In all of these scenarios the current state of the game could be described in terms of the units, such as a friendly harvester or not so friendly enemy, and whether harvestable things like titanium deposits have been gathered.

Actions in the BoS scenarios describe low level movements of units such as moving a harvester one step in some direction. Goals are represented as high level tasks such as moving the harvester to its home base rather than moving the harvester one step north. Consequently, planning involves figuring out which low level actions to string together to accomplish the high level action.

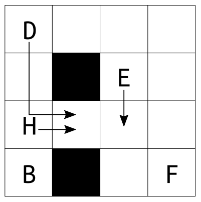
In the resource gathering scenario the location of harvestables may be only partially known. For example, it is possible that there could be a more conveniently located titanium deposit than the one the agent knows about. In the escort scenario, the location of enemy units is initially unknown until an enemy attacks, by which time it is too late to deploy defenses. In the exploration scenario, some paths may be impossible, making some goals unachievable. In all of these scenarios, planning with a goal reasoner performed better than without one. Klenk, Molineaux, and Aha interpreted this success as providing support for goal reasoning as a separate functionality alongside planning.

Goal Reasoning as Planning

But we hypothesize that the right type of planner might do just as well without an explicit goal reasoner. To test this hypothesis, we synthesized a new domain, called Harvester World, that captures the essential features of the benchmarks in Klenk, Molineaux, and Aha (2013) and implemented a planner for it. The previous benchmarks share the following important features: 1) partial observability: the agent does not necessarily see every aspect of the state, such as the locations of harvestables; 2) open world: the agent does not necessarily know about all the objects that exist; 3) on-line sensing: the agent learns more about the world as it takes actions; 4) multi-unit: the agent controls multiple moving objects in the environment which might move simultaneously; and 5) adversarial: there may be other agents whose goals conflict with that of the agent. Harvester World is a single domain that captures all of these features.

Harvester World

Harvester World takes place on a two-dimensional grid similar to BoS (see Figure 1) and supports test cases similar to the three scenarios described in the previous section. While both time, states and actions are discrete, the world state is only partially observable, there may be adversaries, and the agent may need to control multiple movable units. A simulator controls the ground truth of an instance of Harvester World, and at each time step, the simulator requests an action from the agent, attempts the action in the ground truth



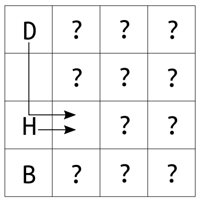


Figure 1: An example Harvester World instance: B = base, D = defender, E = enemy, H = harvester, F = food. Left map represents ground truth; right map represents the agent's current belief state.

representation and moves other factors of the environment forward, and returns an observation to the agent which the agent then uses to update its belief state.

Individual grid cells can allow movement or be fixed impassable obstacles. The harvester's home base (B in Figure 1) is also fixed. The grid also contains harvestable food (F). Food can be harvested by the agent when it is in the same cell. As soon as one food is harvested, another will sprout somewhere else in the map (the law of conservation of food). The agent controls the harvester (H) and a defender unit (D), and the grid may also contain an enemy unit (E). The agent can sense the state of the cell it is in, but cannot see food elsewhere and cannot see an enemy further than one cell away. For example, if an enemy is adjacent to the harvester then the harvester immediately discovers it without any uncertainty about whether or not it is an enemy.[1]

The harvester can move in the four cardinal directions or stay still. It may also choose whether or not a defender should move to the location of the harvester. Hence there are a total of eight actions which may be considered at each step. In BoS it is possible to move a defender independently of the harvester with the intention of strategically placing it in a way that would prevent the harvester from taking damage from an enemy unit. Our simplification does not preclude this effect, it only simplifies the number of actions needed to achieve it. Other actions were hard-wired into the agent and do not require deliberation: if the harvester is not carrying food and enters a cell with food, the food is harvested; if the harvester reaches the base while carrying food, the food is delivered.

The enemy's policy is also hard-wired and known to the agent. An enemy unit will immediately move out of a cell containing the defender. Otherwise, at every timestep the enemy will attempt to move to the next location, either North, South, East or West, along the shortest path toward the harvester's current location, taking obstacles and the defender into account. We compute this via Dijkstra's algorithm outward from the agent.

The objective of the game is to maximize accumulated reward. As in BoS, the successful player should try to maximize the amount of resources it collects while minimizing

[1]For those familiar with BoS, we omit the scout from our domain because we have tried to keep our problem succinct; a scout only increases the range at which an enemy is discovered and thus the size of the problem needed to demonstrate interesting behavior.

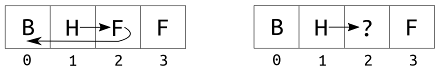


Figure 2: An example Resource Gathering scenario

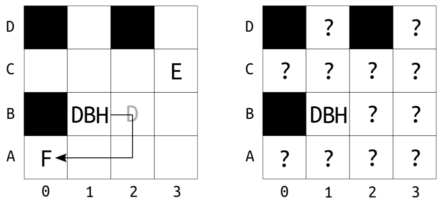


Figure 3: An example Escort scenario

the damage it takes from enemies and the amount of resources it spends to play. Every move of the harvester costs -1, even if the action does not complete because an obstacle is in the target location. Moving a defender costs -1 for each cell it traverses on its way to the harvester. If the harvester delivers food to the base, this earns +50. If the harvester is in the same location as an enemy, this penalizes the agent -10.

Using Harvester World, we were able to reproduce the three BoS scenarios described in the previous section by varying which features should be hidden from our agent. To model Resource Gathering, we configured a world that contained a base, a harvester initially located at the base, and two food sources located in two different locations some distance from the base (see Figure 2 for an illustration). Initially only the base, the harvester and the location of one of the food are known to the agent, making the current state of the world only partially known. An efficient plan should be able to maximize the amount of food it can collect in limited amount of time by harvesting the food closest to the base, including recognizing when newly discovered food is closer than the previously known food.

To model Escort we configured a world that contained a base, a harvester and a defender both initially located at the base, an set of obstacles which partition the world into defensible regions, and one food located some distance from the base (see Figure 3 for an illustration). Initially only the base, the harvester, the defender and the obstacles are known to our agent. The belief state does not include the location of the enemy as well as the location of food. An efficient plan should be able to maximize the amount of food it can collect in a limited amount of time while minimizing the damage it takes from enemies and the resources it spends to defend against them.

To model Explore we configured a world that contained a base, a harvester, a set of obstacles and two food (see Figure 4 for an illustration). Initially only the obstacles are hidden from the planner. An efficient plan should be able to get both food, detouring around any discovered obstacles, if there is an open path to both, or give up on one if it turns out all paths to it are impassable.

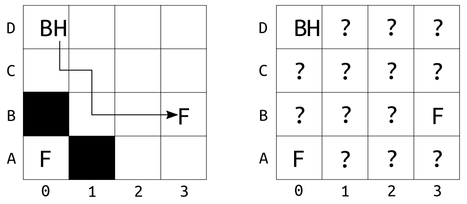


Figure 4: An example Explore scenario

```
HOP($s$)
1. for $i$ from 1 to $N$ do
2.     $w_i \leftarrow$ sample world consistent with current belief
3. foreach action $a$ applicable in $s$
4.     $s' \leftarrow a(s)$
5.     $c \leftarrow (\sum_i^N plancost(s', w_i))/N$
6.     $Q(s,a) \leftarrow C(s,a) + c$
7. return $\mathrm{argmin}_a\, Q(s, aA)$
```

Figure 5: Sketch of hindsight optimization

A Planner for Harvester World

Harvester World exhibits partially observable state, an adversary, and multiple units that need to be controlled by the planner. This is well beyond the scope of classical planning tasks. To handle this problem, we turn to an approximate method for planning in large partially-observable Markov decision processes: hindsight optimization (Yoon et al. 2008; 2010). Hindsight optimization planning has been shown capable of performing better than reactive planning in a partially observable environment when we are unsure if and when new goals may arrive (Burns et al. 2012). In their planner OH-wOW, Kiesel et al. (2013) showed that hindsight optimization can be used to plan on-line in open worlds where the existence of important objects is unknown. We extend the work of Kiesel et al. (2013), hence our planner for Goal Reasoning with Optimization in Highsight in Open Words is called GROH-wOW.

Fundamentally, hindsight optimization works by sampling from a model of all possible realizations of the uncertain features in the current state, and then finding the best deterministic plan in each of these sampled world, and using the values of these plans to estimate the value of the agent's possible successor states and hence determine what the best next action is. See Figure 5 for a pseudocode sketch ($C(s, a)$ is the cost of taking action a in state s). Sampling avoids explicitly representing a belief distribution over a large number of possible states, and the fully specified worlds allow conventional fast deterministic planning techniques to be used. Despite its practicality, unlike methods such as UCT (Kocsis and Szepesvári 2006), hindsight optimization is well-known to not be guaranteed to converge to the optimal action in the limit of infinite sampling. We use the hindsight optimization strategy within a receding horizon paradigm: the deterministic planner looks ahead only to a limited horizon while finding the maximum reward plan for each sampled world, then the agent takes a single action and the cycle repeats.

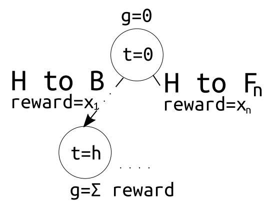


Figure 6: Search tree

Following the algorithm sketched above, GROH-wOW has two main parts: the agent's action planner, which takes a model of Harvester World, samples from it and passes these samples on to the deterministic planner, and the deterministic planner subroutine, which will then find the best plan in each of these samples and return to the action planner the value of each. The hindsight optimization planner will then use these values to find the expected value of each of the immediate successors of the current state and return the action that leads to the next state with the highest expected reward. The action is executed and a set of observations is returned which may be used to update the agent's belief model, which will be given to the hindsight optimization planner at the next timestep to use when determining the next action.

Sampling over the remaining uncertainty in the belief state given to the planner is intended to resolve all ambiguity in the current state in order to leverage deterministic planning. In Harvester World this means resolving two different types of uncertainty and partial observability such as where the enemy is located and stochastic effects such as where food may grow next. We model this uncertainty as a uniform distribution over unknown factors in a belief state. For example, to model Resource Gathering the belief state given to the sampling function might omit the location of food. It might assume that there are at most two food in the world, but the belief state does not provide the location of both of them. Sampling a world from this belief would result in one that contains both foods at a specific locations, e.g. food at positions 2 and 3 in Figure 2. We can then plan to move the harvester to the food at 2 and back to the base for a reward of $+50 - 3$. The belief state might also contain ambiguity about future states. Since in Harvester World the amount of food in the world never diminishes, gathering food implies that new food must have grown somewhere. Hence when we sample a world, we also determine exactly where food might grow in the future, resolving both the uncertainty regarding where food is located now but also how its location might possibly change. Sampling resolves all ambiguity into a set of possible worlds to pass along to the deterministic planner.

Deterministic Planner

The deterministic planner uses a basic breadth-first search from each of the now complete models of the current state out to a given time horizon, enumerating all states which can be reached within $t = horizon - 1$ time steps. It then considers the plan with the highest reward among all of the states reachable at $t = horizon$, and returns this value to the hindsight optimization planner to aggregate.

To make deterministic planning faster, we note that there are only a few 'macro actions' worth considering. In the Resource Gathering scenario there are three macro actions that it makes sense for the deterministic planner to consider: move the harvester to the base, move the harvester to the known food at a specific location, or move it to the unknown food at a different location. This would have been impractical without having first sampled a world where the location of the unknown food can be known, but in hindsight optimization it becomes a simple matter of planning the shortest path from where ever the harvester is now to the closest food and back to the base for the optimal reward, assuming this can be completed within the given horizon. If a goal would take longer to complete than the remaining time, then we consider the macro action incomplete and its result is the state of the world up to the horizon. These macros reduce the branching factor and search depth that the deterministic planner must search. Additional strategies, such as designing a heuristic reward-to-go function, are also possible.

In the Escort scenario the deterministic planner has four macro actions to reason about after sampling determinizes the unknown attributes of the model: move the harvester to a food, move both the harvester and the defender to a food, move the harvester to the base, or move both the harvester and the defender to the base. Recall at each time step we simulate the movement of the enemy. We can do this during planning as well since we assume that, while we do not know exactly where our enemy is, we have a reasonable model of how it moves. (In the absence of a good model, we would just sample possible enemy actions.) If the enemy, wherever sampling has imagined it might be, would interrupt the completion of any of the macro actions above, then we consider the macro action incomplete, like we do when we have run out of time. Its result is the state of the world when encountering the enemy.

In the Exploring scenario, the deterministic planner must reason about the same three macro actions used in Resource Gathering but with some nuance. While the existence of obstacles are initially unknown in the model given to the hindsight optimization planner, for simplicity we do not speculate about where they could be. Instead as obstacles are discovered, macro actions like moving harvester to food at position (A, 0) in Figure 4 potentially become more expensive to complete. The deterministic planner is able to take these changes into consideration when deciding if it should first get the food at (A, 0) and drop it off at the base before getting the food at (B, 3). If the shortest path to (A, 0) becomes too long or even impossible because of obstacles then it should choose to get the food at (B, 3) first.

Experimental Results

The central empirical question at issue in this paper is whether a planner that lacks an explicit goal reasoning component can perform well in Harvester World instances similar to the problems considered by Klenk, Molineaux, and Aha (2013). We addressed this question by running GROH-wOW on each of the BoS scenarios.

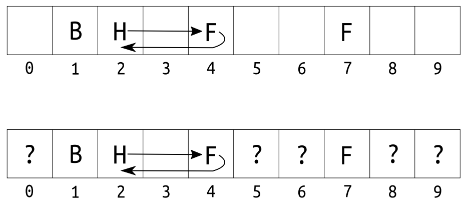


Figure 7: Example sequence in a Resource Gathering instance

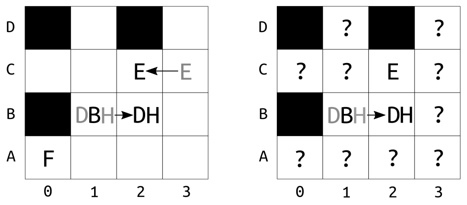


Figure 8: Example step in an Escort instance

Experiment 1: To demonstrate Resource Gathering we ran hindsight optimization on a single instance of Harvester World configured for Resource Gathering as described above for 10 time steps on a 10 x 1 grid, with a sample size of 10 and a horizon of 20, which would allow enough time to foresee the value of getting food located as far away as position 9 and bringing it back to the base. Figure 7 illustrates our results. The top map describes an intermediate state from the perspective of the simulator; it is perfectly known. The bottom map describes the agent's belief state at that time. In the simulation's initial state the harvester (H) is at the base (B) and the planner knows about the food (F) at position 7. At each time step the simulator gives our planner an incomplete model of the world, and the agent returns a single action to the simulator. In this test, the harvester discovered food at position 4 and thereafter the planner determined it should return to base instead of continuing to the original food at position 7. Thus on-line planning can recognize an opportunity as well as a system using a goal reasoner.

Experiment 2: To demonstrate our planner in the Escort scenario, we configured 10 instances of the Escort scenario, varying the initial location of the base, where the harvester and defender start, the hidden enemy and a hidden food. The number of obstacles in each scenario was constant and their location was known to the planner. We ran each of these 10 scenarios for 100 time steps, allowing the hindsight optimization planner to sample 10 worlds at each time step, and gave the deterministic planner a horizon of 10 time steps. Figure 8 illustrates one time step in one problem instance. In this example the agent decided to move the harvester (H) along with the defender (D) East away from the base (B) before the location of the enemy was ever revealed, illustrating the ability of hindsight optimization to consider unobserved factors such as the possible location of the enemy. All problem instances exhibited similar behavior as the agent explored the map and collected food.

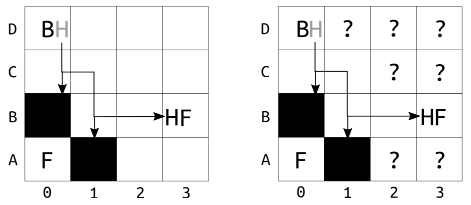


Figure 9: Example sequence in an Explore instance

Experiment 3: To demonstrate hindsight optimization in the Explore scenario, we configured an instance of Harvester World with two food, one of which was unreachable due to obstacles. We made the location of all of the food known but not the location of obstacles, and allowed the deterministic planner a horizon of 10. We did not need to sample more than one possible world since the location of the food was known. As figure 9 illustrates, the agent initially attempted to get the food in the lower left corner by exploring the cells along the left side of the grid where it discovered two obstacles which precluded its ability to get the food it was initially after; it then went for the food on the right side of the grid. This behavior highlights the influence of the objective function in goal reasoning. Once the obstacles at (A, 1) and (B, 0) have been discovered there is no plan which sends the harvester to the food at (A, 0) that could improve the total expected reward measured by the objective function so the food at (A, 0) is abandoned.

Discussion

So far we have shown that a hindsight optimization planner can perform well in the same class of problems as was previously thought to require goal reasoning. That is not to suggest that these problems do not require goal reasoning, just that a discrete goal reasoner is not necessarily required to achieve the desired functionality. We propose that our planner demonstrates many of the aspects of goal reasoning. We explore this with reference to Johnson et al. (2016)'s goal lifecycle.

In previous work goal reasoning has been organized into discrete steps defining a goal lifecycle. A goal must be formulated, selected, expanded, committed, dispatched, and then execution and resolved (Johnson et al. 2016; Roberts et al. 2016). We consider these in turn. If one were to interpret existing food as goals for planning with low-level actions, then we could view grounding a belief state into a set of completely known worlds as formulating a set of goals. For example in Figure 7, in the belief state the location of only one food is known; there is one known goal, retrieving the food at position 7. Sampling however could result in a world in which there is a food at position 0, in which case sampling generates the additional goal, retrieving the food at position 0. Selection is then the process by which our planner decides which food to pursue first; the low-level planner will recommend the order that gets us the most expected reward within the planning horizon. Expansion is finding the action that best follows this longterm plan. Goal evaluation metrics used in previous work like principles and intensity levels (Johnson et al. 2016; Cox, Dannenhauer, and Kondrakunta 2017) are encapsulated in the idea of a reward. For example, the ordering of macro actions in optimal plans represent the most efficient ordering of goals as high-level tasks to achieve the best possible total reward without having to be explicitly prioritized.

In these ways, one can view our planner as engaging in the elements of goal reasoning. However, there is also a more subtle interpretation. One can also find goal reasoning in how hindsight optimization uses the objective function to estimate the expected reward and choose the next low-level action toward this 'estimated goal.' Goal generation in this sense could be seen as the process of finding the number of times we should be able to put the harvester at the base with food within the planning horizon. Selecting one of these goals (or abstract plans) is easy when the world is completely known; there is only one optimal number of times harvester can be at the base with food. Goal expansion then is deciding which low-level action is the best next step in making the harvester arrive at the base with food the expected number of times. This is simply the next action along the plan that puts the world in a state with the best possible reward within the planning horizon according to the objective function.

The selection step in the goal lifecycle becomes more interesting in a partially known world because the true goal, the number of times we can make the harvester at the base with food true, is unknown as well, but with hindsight optimization we can estimate this quantity and estimate the true goal; the expected value of the true goal is the expected value of the optimal plans across all possible worlds. Selection can be interpreted as the process by which hindsight optimization finds this estimate. Given the current belief state, the selected goal is the estimated number of times we could expect to put the harvester at the base with food. Because each grounded version of the belief state has the potential to change the costs involved in achieving the goal state and so change the number of times it can be achieved within the planning horizon, the goal in each possible world may vary, but sampling from the belief space allows us to quickly find an estimate of the true goal. Then in the expansion step of the goal lifecycle we can use this estimate of the true goal to compare how much each low level action is expected to move the agent closer to the estimated goal.

After selection and expansion our agent immediately dispatches the next action with the promise of achieving the selected goal. The effect of the next action, along with any new information about the environment, is then incorporated in the next round of goal reasoning. Because every new observation has the potential to dramatically change the expected value of the goal, such as we saw in the Explore scenario when the discovery of obstacles surrounding the food at (A, 0) made one of the food unreachable, an on-line planer can be said to repeat goal reasoning at every time step, beginning with the evaluation of the state resulting from the previous action and possibly selecting a new goal.

Goal Reasoning as Multilevel Planning

Although we have demonstrated how an appropriate planner like GROH-wOW can handle the types of goal reasoning tasks present in Harvester World and simple scenarios of its type, we do not mean to claim that a single planner can give high performance in truly huge domains such as dealt with by human emergency responders and battalion commanders. It seems clear that for planning over very long horizons or with huge action spaces, hierarchical techniques will be necessary. But again, it does not necessarily follow that a separate goal reasoning component is necessary. Instead, we speculate that multiple levels of planning may be appropriate. These planners might coordinate their computation in various ways — we will sketch three. In the most classic approach, a high-level planner might generate a plan of very high-level actions, such as establish communications network, rescue survivors, and treat wounded, while a low-level planner might treat just one of the high-level actions as a goal to achieve through the generation and coordination of many lower-level actions. In this way, subgoals are created that allow low-level planning to be limited to subproblems with shorter solutions, speeding search. In this sense, the high-level plan is a form of loose guidance using landmarks for the low-level planning.

This arrangement raises the question of domain and goal representation for the high-level planner. Using the concept of reward, we believe that a high-level planner can reason about even abstract concepts such as keeping friends safe by postulating the existence of unseen adversaries, reasoning about their behavior, and selecting actions such as defensive patrolling. As we have shown, there is nothing inherent in a partially observable stochastic adversarial domain that demands a goal reasoner.

This traditional concept of multi-level planning, in which the actions of one level are goals for the next, is just one way in which multiple planners can be coordinated to solve large problems. A second way is by viewing a plan at the higher more abstract level as establishing constraints on the possible state space to be considered by the lower-level planner. The lower-level planner considers only concrete states that project into the abstract states visited by the abstract plan. In this way, the high-level planner specifies a kind of tunnel in the state space that constrains the search of the lower-level planner (Gochev et al. 2013). In this view, the states spaces of the two planners must be similar enough to be aligned with projection, with one just being more abstract than the other.

Finally, a third method of coordination is to use the high-level planner to guide the lower-level planner in a more flexible way. The previous tunnel method can be seen as the high-level planner pruning away concrete states that map to abstract states that are not in the plan. In a sense, such concrete states are given heuristic values of infinity (Holte 1995). A more flexible method is to use the high-level planner directly as a heuristic function, allowing it to return values smaller than infinity and allowing the low-level planner to eventually expand states that project outside of the initial high-level plan if necessary. Planning in an abstract representation of the problem has been shown to be an effective heuristic (Hoffmann and Nebel 2001). One can regard our Harvester World planner as being of this type, as the deterministic planner finds more abstract plans, which are used to evaluate states for the low-level action planner.

Conclusion

Despite agreement on the observable capabilities of agents, such as dealing with multiple goals in a dynamic partially-observable world with adversaries, it is not obvious how those capabilities might be implemented in various computational modules. Locating goal reasoning in a specialized module is certainly one way to go. However, our results show that the empirical evidence presented for that architectural commitment may not uniquely support that choice. While Harvester World is certainly beyond the classical planning problem setting, we have demonstrated that modern planning technology can handle partially-observable open worlds with multiple units and adversaries. We have also speculated about three possible ways in which multiple levels of planners might be arranged to handle more challenging domains.

Acknowledgements

We would like to thank Mak Roberts, Will Doyle, Tianyi Gu, Jordan Ramsdell and our anonymous reviewers for their thoughtful comments. We also gratefully acknowledge support from NSF (grant 1150068).

References

Burns, E.; Benton, J.; Ruml, W.; Do, M. B.; and Yoon, S. 2012. Anticipatory on-line planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-12)*.

Cox, M. T.; Dannenhauer, D.; and Kondrakunta, S. 2017. Goal operations for cognitive systems. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*.

Gochev, K.; Cohen, B.; Butzke, J.; Safonova, A.; and Likhachev, M. 2013. Path planning with adaptive dimensionality. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-13)*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Holte, R. C. 1995. The tradeoff between speed and optimality in hierarchical search. Technical Report 95-19, University of Ottawa Computer Science.

Johnson, B.; Roberts, M.; Apker, T.; and Aha, D. W. 2016. Goal reasoning with information measures. In *Proceedings of the Fourth Annual Conference on Advances in Cognitive Systems (ACS-16)*.

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 Workshop on Planning and Robotics*.

Klenk, M.; Molineaux, M.; and Aha, D. W. 2013. Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence* 29(2):187–206.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML-06)*, 282–293.

Molineaux, M.; Klenk, M.; and Aha, D. W. 2010. Goal-driven autonomy in a navy strategy simulation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*.

Roberts, M.; Shivashankar, V.; Alford, R.; Leece, M.; Gupta, S.; and Aha, D. W. 2016. Goal reasoning, planning, and acting with actorsim, the actor simulator. In *Proceedings of the Fourth Annual Conference on Advances in Cognitive Systems (ACS-16)*.

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

Yoon, S.; Ruml, W.; Benton, J.; and Do, M. B. 2010. Improving determinization in hindsight for on-line probabilistic planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*.