Open World Planning for Robots via Hindsight Optimization

Scott Kiesel[1] and **Ethan Burns**[1] and **Wheeler Ruml**[1] and **J. Benton**[2] and **Frank Kreimendahl**[1]

[1]Department of Computer Science
University of New Hampshire
skiesel, eaburns, ruml, fri2 at cs.unh.edu

[2]Smart Information Flow Technologies (SIFT), LLC
Minneapolis, MN USA
jbenton@sift.net

Abstract

Classical planning makes the closed world assumption in which all relevant aspects of the world are known at planning time. While this assumption holds in some domains, in many practical robotics domains the existence of relevant objects or the states of relevant fluents are initially unknown and must be actively discovered. Previous proposals for open-world planning either employ complex and expensive knowledge representations or depend on ad hoc assumptions. In this paper, we show how hindsight optimization provides a simple and general approach to planning in open and partially observable worlds. Hindsight optimization samples multiple possible worlds that are consistent with the agent's current knowledge, generates a plan in each respective world, and then selects the action that maximizes expected reward over these samples. While this approach is approximate, we demonstrate both in simulation and on a physical robot that this simple technique performs well and is more scalable than previous methods on standard benchmarks.

Introduction

Imagine a rescue robot entering a partially-destroyed building to search for survivors of an earthquake. The agent does not know the initial layout of the building, what new obstructions may exist, the locations of potential victims, or even how many victims there are. In open-world planning problems like this, the agent is not given a complete description of the initial state of the world, but it can perform sensing actions to determine the existence of relevant objects and the values of important fluents. To be useful, the planner must be fast enough to not materially delay the actions of the robot. It must be able to plan to discover and take into account newly sensed information, and ideally it would be expressive enough to handle soft goals, durative actions, temporal constraints, and actions with uncertain outcomes.

In this paper, we propose a simple on-line planning approach that handles the requirements of open-world domains. We call this new approach *Optimization in Hindsight with Open Worlds* (OH-wOW). Rather than using traditional techniques that compute a policy or contingent plan in advance, we estimate on-line at each step which action is best in light of our current knowledge of the world. The OH-wOW approach is domain agnostic and does not commit to a particular representation for open-world knowledge or goals. Instead, it can leverage any closed-world planner appropriate for the underlying domain. Our central assumption is that the agent possesses some knowledge, likely probabilistic, about the domain. In our view, performing well in an open-world depends on having expectations about that world, e.g., building dimensions are typically tens or hundreds of meters rather than centimeters or kilometers, or that people are usually found in certain densities per square meter, or are more often found in certain areas, such as offices. This type of default or prior information can be overridden by direct experience, but ought to play a role in planning until it is discovered to be inaccurate. We use these expectations to generate possible states of the world consistent with the agent's current knowledge, use a closed-world planner to estimate the future reward achievable in those worlds after taking each currently-applicable action, and then select the action with the highest expected reward.

After describing OH-wOW in detail, we contrast it with previous work. We then report on the method's empirical performance, both in simulated domains and when deployed on a physical mobile robot fully integrated with the Robot Operating System (ROS), Simultaneous Localization And Mapping (SLAM) and standard navigation. Our experience indicates that the method is surprisingly general and practical, achieving results as good as those of previous systems but with lower planning times and fewer ad hoc assumptions. This work showcases the power of Monte Carlo techniques and adds open-world planning to the list of non-classical planning settings in which simple planners can be leveraged to provide state-of-the-art performance.

A Hindsight Optimization Approach

Optimization in hindsight was originally developed for scheduling and networking problems (Chong, Givan, and Chang 2000; Mercier and van Hentenryck 2007; Wu, Chong, and Givan 2002) and has recently been applied to probabilistic planning (Yoon et al. 2008; 2010). In these previous settings, sampling is used to resolve uncertainty in the outcome of actions. In our context of open-world planning, each sample forms a concrete hypothesis about the world—which objects might exist and which fluents might hold. While these will likely be revealed to the agent as it performs actions that have, a priori, uncertain outcomes, the sampling process for open-world planning is more involved

than choosing an outcome in a PPDDL (Younes and Littman 2004) action or RDDL (Sanner 2011) description. For example, a rescue robot will generate possible world states with conceivable floor plans for the building, each with sets of victims distributed in various plausible locations. Each of these sampled worlds may potentially determine the outcome of multiple sensing actions during the course of the corresponding planning episode. Demonstrating the practicality of this approach is the central contribution of this paper.

While these samples of possible worlds are intentionally not exhaustive, they are intended to provide useful relative judgements on the expected value of actions. In order to estimate the value of an action, we apply that action in each of the sampled possible worlds, find closed-world plans from the resulting states, and average over the resulting plan costs. The action with the lowest average plan cost over the sampled worlds is chosen to be executed.

More formally, we define the value of being in a state s_1 as the minimum expected cost over plans that extend from s_1. That is, the minimum cost over all possible future action sequences of the total cost over all expected future states:

$$V^*(s_1) = \min_{A=\langle a_1,\ldots,a_{|A|}\rangle} \; \underset{\langle s_2,\ldots,s_{|A|}\rangle}{E} \left[\sum_{i=1}^{|A|} C(s_i, a_i) \right]$$

where $C(s, a)$ represents the cost of performing action a in state s. In open-world planning, these future states incorporate the sensed knowledge of the agent and the expectation is over the distribution of sensing outcomes. The agent will expect different outcomes based on its beliefs about the world. Given our expectations about sensing outcomes, we would like to find the action sequence $A = \langle a_1, ..., a_{|A|}\rangle$ that minimizes the expected sum of action costs. To compute V^* exactly, we would need to compute the expectation for each of exponentially many plans.

In optimization in hindsight, we approximate the value function by exchanging expectation and minimization, so that we are taking the expected value of minimum-cost plans instead of the minimum over expected-cost plans:

$$\hat{V}(s_1) = \underset{\langle s_2,s_3,\ldots\rangle}{E} \left[\min_{A=\langle a_1,\ldots,a_{|A|}\rangle} \sum_{i=1}^{|A|} C(s_i, a_i) \right]$$

This approximation of $V^*(s)$ uses fixed sensing outcomes in each minimization. As in other applications of optimization in hindsight, the stochastic elements have been reduced to known outcomes by sampling. For each possible outcome in the expectation, the problem is to minimize cost given a known world, i.e., standard, closed-world, cost-minimizing, deterministic planning. In OH-wOW, fixed sensing outcomes are generated using concrete hypotheses about the state of the world. For each fully-known, deterministic world hypothesis, the agent can compute the result of different sensing outcomes when solving the minimization in the equation for V^*. For example, the result of querying a vision system to look for an injured person depends on whether or not there is an injured person in the sensed portion of the world—this is fully-known for each hypothesis.

OH-wOW($s = \langle agent, world\rangle, N$)
1. for i from 1 to N do
2. $\quad w_i \leftarrow sample\_world(world)$
3. foreach action a applicable in s
4. $\quad s' \leftarrow a(s)$
5. $\quad c \leftarrow (\sum_{i=1}^{N} solve(s', w_i))/N$
6. $\quad Q(s, a) \leftarrow C(s, a) + c$
7. Return $argmin_a\, Q(s, a)$

Figure 1: The OH-wOW algorithm.

The agent is aware of what features are truly known and which are merely hypothesized, as a result the deterministic problem can require sensing actions before the agent interacts with hypothesized portions of the world. In this way, the system will still be required to plan to sense. A dummy precondition is added to all actions that involve a hypothesized variable. This precondition enforces that the value of that variable is sensed before actions requiring the value are executed. This ensures that the resulting plan executes sensing actions appropriately. More concretely, if the agent hypothesizes that there is an injured person in a room, then the deterministic planner will require a sensing action before that person can be reported. When a sensing action is carried out in the physical world, its result may differ from the hypothesis. This new information will be reflected in the samples taken at the next planning step.

We define the Q-value to be the cumulative expected cost of taking an action a_1 in state s_1:

$$Q(s_1, a_1) = C(s_1, a_1) + \underset{\langle s_2,s_3,\ldots\rangle}{E} \left[\min_{A=\langle a_2,\ldots,a_{|A|+1}\rangle} \sum_{i=2}^{|A|+1} C(s_i, a_i) \right]$$

From this, we estimate the best action choice in s_1 as $\min_a Q(s_1, a)$. Using this technique, we are said to be performing optimization with the benefit of 'hindsight' knowledge about how future uncertainty will be resolved.

The pseudocode in Figure 1 summarizes the algorithm. At each time step, the algorithm is used to find the next action to execute from the current state s, which includes information about both the agent's current configuration and its current knowledge about the world. First, we generate a set of N possible worlds that are consistent with the agent's current knowledge (lines 1–2). Next, for each currently applicable action a, we consider the resulting state $s' = a(s)$ (line 4). Then, each possible world w_i is initialized with the state s', generating a fully-known closed-world deterministic planning problem. Recall that, to incorporate sensing, the determinized problem requires the agent to sense before interacting with hypothesized features of a sampled world. Solving this problem provides an optimistic estimate of the cost from s'. The mean cost across the set of samples (line 5) along with the cost of the action $C(s, a)$ is used as the Q-value for each applicable action a in the original state s (line 6). Finally, we return the action with the minimum Q-value (line 7), the agent takes the action, possibly observing new facts and objects in the world, yielding a new current state, and the cycles begins anew.

Related Work

Open world planning is a broad problem that has been attacked from many angles. One issue is how to represent knowledge and goals related to open-ended sets; Etzioni and Weld (1994) and Babaian and Schmolze (2006) have addressed this. We do not address this issue in this paper, except to point out that the underlying planners used in our approach are closed-world and do not require a particularly expressive (and expensive) representation language. We do require that the agent tracks what is currently known about the world and that the world generator respects this knowledge when sampling possible worlds.

In conformant planning (Cimatti, Roveri, and Bertoli 2004, inter alia), one requires plans that are guaranteed to work without sensing. For most robotics domains, this is overly restrictive and renders problems unsolvable. Contingent planning (Meuleau and Smith 2003, inter alia) allows for sensing, but computes a plan before beginning execution. In addition to handling open-worlds, we aim to scale to domains in which the number of contingencies may be very large (e.g., the number of possible floor plans), making synthesis of branching plans prohibitively expensive.

In the POMDP literature, computing actions on-line is recognized to provide increased scalability (Ross et al. 2008). However, many POMDP algorithms attempt to compute future belief states of the agent, which can be expensive and cumbersome. Optimization in hindsight represents an extreme approach, disregarding future belief uncertainty and assuming that the agent can achieve the cost accrued by the plans for the fully-observed sampled worlds. Our work is perhaps most closely related to work on sampling techniques for POMDPs, where a particle filter approximates the belief space during sampling (Silver and Veness 2010). Open world planning goes beyond traditional factored POMDP representations (Boutilier, Dean, and Hanks 2011) because the structure of the world state requires representing a logically infinite domain of discourse; the universe of objects that exist and the possible relationships between them remain unknown to the agent (Doshi 2009).

There has been sustained interest from roboticists in open-world planning. One way of handling open-world planning in practice is to force the robot to move in one direction simply to explore without a concept of cost or reward. Such simple ad hoc approaches cannot exploit the agent's expectations about goals (e.g., people are likely in offices) or take sensed information into account (e.g., a hallway implies new rooms to explore). Talamadupula et al. (2010) present an approach where the planner assumes objects exist in order to instantiate goals and motivate a search and rescue robot to collect reward by discovering and reporting victims. As new information arrives about the environment, the planner replans. This can be seen as a degenerate form of our hindsight approach, where the robot operates on a single optimistic "sample". While it is simpler, it cannot generalize to domains where uncertainty is a major component.

Joshi et al. (2012) use offline symbolic dynamic programming with known goals but unknown numbers or locations of objects, which does allow for reusable policies on any instance of the domain. However, in their experiments the number of possible objects was severely limited to retain feasible computation times (they require 4 hours for their 3 room example), which makes the resulting policies suboptimal. They also do not handle temporal constraints, action costs, or goal rewards.

Evaluation

We evaluate OH-WOW by applying it in two domains: the classic omelette benchmark for planning under uncertainty, and urban search-and-rescue, which we investigate both in simulation and using a physical robot.

Omelettes

In the omelette benchmark, introduced by Levesque (1996), the agent is attempting to make a three-egg omelette with ingredients of unknown freshness. The agent has four available actions. The agent can *break* an egg into a bowl, *pour* the contents of a bowl into another bowl or the trash, *wash* a bowl, or *sniff* whether the eggs in a bowl are good. All actions are deterministic except for the sniff action. The goal is to have exactly three good eggs in a specific bowl with no trace of bad eggs. To make the domain more challenging, we extended it to have both regular white eggs, which are bad with a probability of 0.5, and local brown eggs, which are bad with a probability of 0.1. The agent is able to observe the color of the next available egg without requiring a sensing action.

We compared OH-WOW to a perfectly omniscient oracle and also to a hand-coded controller. The controller puts eggs into the goal bowl, sniffing after each addition and cleaning out bad eggs until it finds a good one. Then it does the same routine using an extra bowl, pouring good eggs into the goal bowl from the extra bowl until the goal is reached. We generated three sets of 100 random instances, each set with a different probability of the next egg being brown. OH-WOW used a domain-dependent deterministic planner based on uniform-cost search.

Figure 2 (left) shows the distribution of the resulting plan costs using box and whisker plots. Each box surrounds the middle 50% of the data, with a horizontal line indicating the median and whiskers indicating the range (values beyond $1.5\times$ the inter-quartile range are shown as circles). The gray vertical stripes inside each box show 95% confidence intervals on the mean. The plot shows the increase in cost over the optimal solution found by the oracle, of the hindsight planner with 32 and 256 samples, and the hand-coded controller (ctlr). The boxes are grouped by the probability of an egg being brown (0.0, 0.5, and 1.0). We can see that, when all eggs were white, the hindsight planner with 256 samples had a median cost that was less than the hand-coded controller (significant with $p < 0.05$ via the Wilcoxon signed-rank test). As the probability of a brown egg increased, the hindsight planner performed better, nearly dominating the controller when all eggs were brown. This is likely because the hindsight planner could recognize that brown eggs tend to be good, and put multiple into a bowl before bothering to smell, saving redundant sniff actions.

The average total planning time on a 3.1 GHz Core2 PC for OH-WOW to reach the goal using 256 samples on a

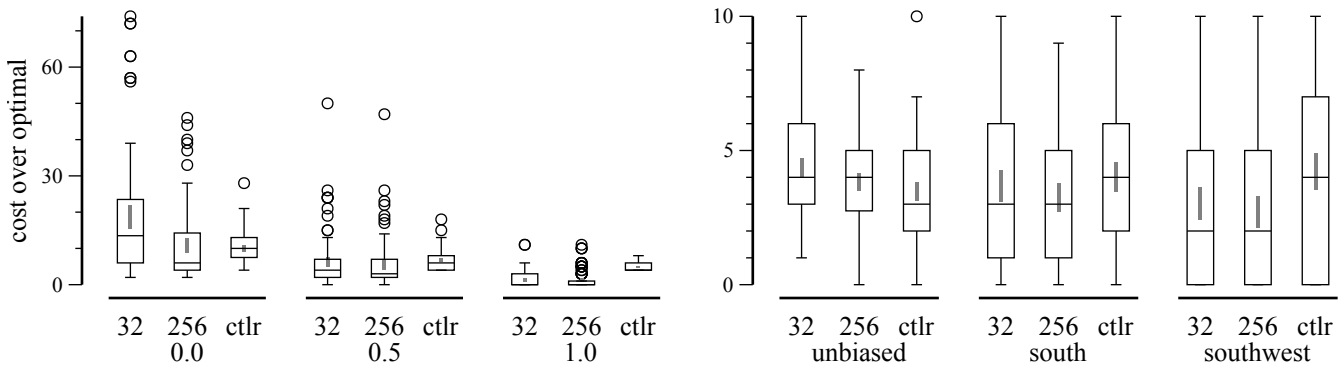

Figure 2: Plan cost in the three-egg omelette domain (left), and the search and rescue domain (right).

problem without brown eggs was 12.9 seconds (standard deviation 8.0 seconds). Each plan was an average of 24.9 actions long (standard deviation 13.7 actions) and each action in the plan took an average of 0.52 seconds to select (standard deviation 0.31 seconds) before executing it. This compares favorably with the 185 seconds of offline planning reported for approximate RTDP (CPU unspecified) by Bonet and Geffner (2001). Levesque (2005) also generates full plans offline to solve the three-egg omelette in 1.4 seconds but requires 1,681 seconds if the omelette is scaled to four eggs. When using four eggs, OH-wOW's costs relative to optimal were similar to the three-egg case, and total computation time averaged only 76.7 seconds (standard deviation 43.6). Each plan was an average of 49 actions long (standard deviation 27.3 actions) and each action in the plan took an average of 1.57 seconds to select (standard deviation 0.99 seconds) before executing. The plans found by Levesque's planner also contain strictly more actions than our hand-coded controller (which in turn finds more costly plans than OH-wOW on the median), as Levesque's solution always uses the auxiliary bowl for staging and requires an additional pour action to move the first good egg into the goal bowl.

Search and Rescue

Now, we return to the motivating example of search and rescue robotics. The robot's objective is to maximize the number of injured people it reports while still returning to its starting location by a given hard deadline.

To generate possible worlds for OH-wOW, we need to generate building layouts consistent with the robot's current map and hypothesize the possible locations of injured people. We represent building layouts as rough topological maps. We assume that undiscovered nodes will lie on a uniform four-connected grid, and that a known node can be extended if it has an adjacent grid cell that can be reached without going through an obstacle or crossing an existing edge in the map. We iteratively choose an extendable node, generate a valid neighbor and connect them. We use a bias toward extending the most recently added node, and toward generating the neighbor that forms a straight line from the

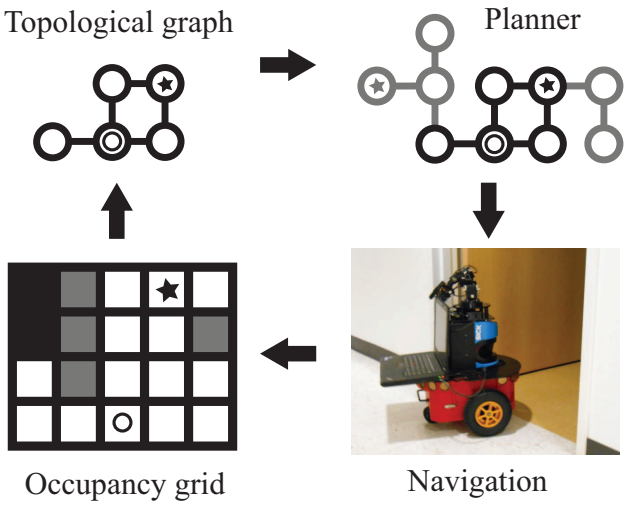

Figure 3: Architecture diagram.

chosen node's parent. This was sufficient to yield plausible building layouts with hallways. Victims are generated independently with fixed probability per hypothesized node. The upper right panel of Figure 3 shows a very small example map with hypothesized extensions shown in gray.

The base planner used by OH-wOW precomputes all-pairs shortest-paths among nodes containing people and the start location. It then uses depth-first search, considering at each step to visit each unreported person or return home. The available actions depend on the remaining time. For efficiency, we avoid considering time as a separate state variable by incorporating it into the cost function (Phillips and Likhachev 2011).

Simulation To test the planner in simulation, we created 100 random worlds with 100 nodes each. We considered three victim distributions: *unbiased*, uniform probability of 0.1 per node; *south*, nodes south of the start location contains a person with probability 0.2 and nodes north of the start location 0; and *southwest*, southwest of the start 0.4

	victims found			
deadline	0	1	2	3
1 minute	4	6	0	0
5 minutes	0	7	3	0
10 minutes	0	3	4	3

Figure 4: Number of injured victims found and reported over 10 runs using a physical robot.

and 0 elsewhere. These distributions are representative of helpful domain knowledge that can be leveraged when generating possible worlds. Skewing the probability of a victim's existence to one side of the building could be used to to represent the knowledge of a closed wing of the building or a scheduled company-wide event. We limit the total number of victims to 10. The cost of a plan is the number of unreported people remaining when the agent returns home and performs a dummy *finish* action. We compared OH-wOW to two different algorithms. The first is an oracle that knows the exact configuration of the building and location of all victims. The second is a hand-coded controller that performed a depth-first exploration of the building, reporting people that it encountered and returning to the start location when it had no more time to explore.

To gauge the complexity of these instances, we must consider the number of possible configurations of maps and victims. Considering only $n \times n$ grids, there are $2(n-1)n$ possible places for edges; our generator is limited to trees, so it must pick $n^2 - 1$. For $n = 10$, this is $\binom{180}{99} \approx 10^{52}$ maps. For each possible map, we must choose locations for victims; for 10 victims, there are $\binom{100}{10} \approx 10^{13}$ possible configurations on the map. Maintaining a belief over so many possible worlds would be challenging. Thankfully, it also seems unnecessary if we merely wish to estimate the expected value of actions.

Figure 2 (right) shows results, grouped by the victim distributions. For the unbiased case, the hand-coded controller gave the best performance, but OH-wOW was quite competitive. With a biased distribution, OH-wOW was superior as it was easily able to leverage prior knowledge about possible worlds. The average maximum per-action planning time for OH-wOW with 256 samples was 2.7 seconds (standard deviation 0.85 seconds). In order to compare with Joshi et al. (2012), we also ran smaller instances with at most three victims. The average maximum per-action time for 256 samples was 0.18 seconds (standard deviation 0.035 seconds), which is negligible compared to typical mobile robot latencies.

Physical Robot We also integrated OH-wOW with the Robot Operating System (ROS, `www.ros.org`) on a 3.7 GHz quad-core i7 laptop on-board a Pioneer 3dx equipped with a SICK LIDAR shown in Figure 3. We use the ROS Gmapping SLAM stack to generate a fine-grained occupancy grid, from which we extract a topological map with edge lengths of 1 meter to provide to OH-wOW. Figure 5 shows a final topological map overlayed on the corresponding SLAM map created during an experiment run of the search and rescue application. In the topological map, nodes are marked as either black, green, or pink. Pink nodes indicate an area of the building where a victim was found and reported. Green nodes are points in the map that can be extended when creating possible building layouts. The black nodes indicate that the layout of the building can not be extended from this area.

The ROS Navigation stack is used to execute movement actions, which are specified as the topological node to visit next. These topological nodes are then mapped to a two dimensional point in the map built by SLAM before issuing the move action to the robot. In some cases, the rough topological graph places a node very near to an obstacle and the planner can not find a safe way to achieve the requested action. We supplemented the navigation component in these instances by issuing a set of perturbed points around the initial point before returning failure to the planner. This set was simply four points, one in each cardinal direction, one half of a discretization away.

We performed experiments in a hallway of approximately 20 meters with between 2 and 5 open doors to offices and 3 victims. We simulated detection of a victim using the range capability of the laser rangefinder. When the laser is able to collect data and populate a portion of the map corresponding to certain pre-selected locations (that were unknown to the planner), we pass that detection information along to the planner. In order to report a victim the robot must navigate to the containing topological node.

We used three different deadlines, one minute, five minutes and ten minutes. As shown in Table 4, the performance of the robot improves as it is given more time to search for victims. In all experiments the robot returned within the hard deadline we provided. At first, only given a short deadline of one minute, the robot is able to find one out of the three victims in six of the ten trials before returning home. When the deadline is increased to five minutes, the robot takes advantage of this and performs more exploration and is able to find two out of the three victims in two trials and one victim in the remaining eight. When this deadline is further increased to ten minutes, the robot is able to find all three victims in two trials, two victims in four trials, and one victim in the remaining four trials.

These results demonstrate that generating possible worlds consistent with experience is feasible in practice, even as the robot's knowledge is being updated during exploration. It also shows that under realistic conditions, OH-wOW correctly trades off soft goals under temporal constraints, but without the ad hoc goal handling of Talamadupula et al. (2010) or the hours of preprocessing required by Joshi et al. (2012).

Discussion

OH-wOW requires a generative model of plausible worlds. We assume such expectations can be developed either manually or through experience. When the world contradicts the agent's expectations, this can be interpreted as surprise, which might naturally lead to increased learning. The fundamental vulnerability of sampling-based planners is when unlikely worlds play a large role in determining action

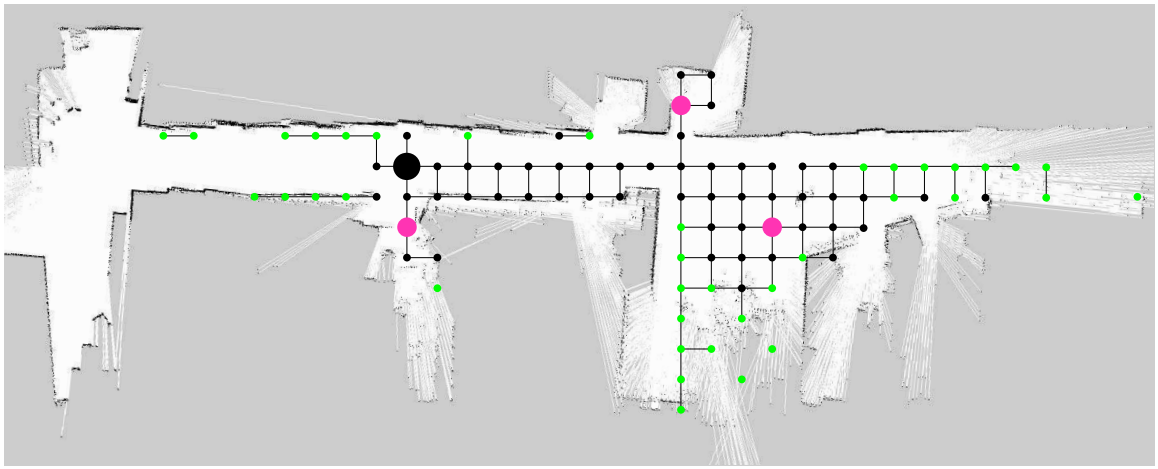

Figure 5: Example SLAM and topological map.

value; importance sampling may help here. For example, in the "Bombs in Toilets" domain (McDermott 1987; Smith and Weld 1998), OH-wOW may never sample a world in which a certain undunked package contains the bomb. The probability of this, however, is small ($7 \cdot 10^{-11}$ for 6 packages and 128 samples). In any case, optimal behavior is unattainable if one insists on fast response times in dynamic domains.

While faster than many POMDP algorithms, OH-wOW is much slower than a classical planner, as it must solve one classical planning problem for each sampled world. In our implementation, during each step, all planning problems were solved serially. These problems are entirely independent though and could trivially be solved in parallel to take advantage of multiple processor cores. OH-wOW is more general than standard off-line techniques as it can be used on-line, as shown in the experiments, and also off-line by simulating the domain to construct a branching plan. It is possible to improve the performance of OH-wOW by applying some of the enhancements of Yoon et al. (2010). One such technique is called *probabilistically helpful actions*. To find probabilistically helpful actions, the planner evaluates all samples from the current state of the agent instead of the one step lookahead states. Actions that lead to optimal plans starting from the current state are considered to be helpful while the others are not. The samples are solved as normal from the one step lookahead states, but the only actions that are considered are the ones that were deemed helpful. Another improvement presented by Yoon et al. (2010) is to save samples and plan prefixes that remain consistent with the outcome of a selected and then executed action. In domains with large amounts of determinism, this enhancement can greatly reduce the amount of planning required by saving work across deterministic transitions.

In this paper, we assume that the world remains static as we explore it and that non-sensing actions are deterministic. OH-wOW however, is very general and immediately applies to dynamic worlds, stochastic actions, and on-line goal arrival; this remains an exciting area for future work.

Conclusion

Open world planning is essential for many real-world agents. We have shown how optimization in hindsight yields a simple and general approach to open-world planning with temporal constraints, decision-theoretic reasoning, and soft goals. While the technique is approximate, it is easy to implement and our results suggest that it can be successful in practice.

Acknowledgments

This work was supported in part by NSF (grant 0812141) and the DARPA CSSG program (grant D11AP00242).

References

Babaian, T., and Schmolze, J. G. 2006. Efficient open world reasoning for planning. *Logical Methods in Computer Science* 2(3).

Bonet, B., and Geffner, H. 2001. GPT: a tool for planning with uncertainty and partial information. In *Proc. IJCAI-01 Workshop on Planning with Uncertainty and Partial Information*, 82–87.

Boutilier, C.; Dean, T. L.; and Hanks, S. 2011. Decision-theoretic planning: Structural assumptions and computational leverage. *CoRR* abs/1105.5460.

Chong, E.; Givan, R.; and Chang, H. 2000. A framework for simulation-based network control via hindsight optimization. In *IEEE Conference on Decision and Control*.

Cimatti, A.; Roveri, M.; and Bertoli, P. 2004. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence* 159(1–2):127–206.

Doshi, F. 2009. The infinite partially observable markov decision process. In *Neural Information Processing Systems*, volume 22, 477–485.

Etzioni, O., and Weld, D. S. 1994. A softbot-based interface to the internet. *Communications of the ACM* 37(7):72–76.

Joshi, S.; Schermerhorn, P. W.; Khardon, R.; and Scheutz, M. 2012. Abstract planning for reactive robots. In *Proceedings of IEEE ICRA*, 4379–4384.

Levesque, H. 1996. What is planning in the presence of sensing? In *Proceedings of AAAI*.

Levesque, H. J. 2005. Planning with loops. In *Proceedings of IJCAI*.

McDermott, D. 1987. A critique of pure reason. *Computational Intelligence* 3(1):151–160.

Mercier, L., and van Hentenryck, P. 2007. Performance analysis of online anticipatory algorithms for large multistage stochastic programs. In *Proceedings of IJCAI*.

Meuleau, N., and Smith, D. E. 2003. Optimal limited contingency planning. In *Proceedings of UAI*.

Phillips, M., and Likhachev, M. 2011. Planning in domains with cost function dependent actions. In *Proceedings of the fourth international symposium on combinatorial search (SoCS-11)*.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-draa, B. 2008. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research* 32:663–704.

Sanner, S. 2011. Relational dynamic influence diagram language (rddl): Language description. *NICTA, Australia*.

Silver, D., and Veness, J. 2010. Monte-carlo planning in large POMDPs. In *In Advances in Neural Information Processing Systems 23*, 2164–2172.

Smith, D. E., and Weld, D. S. 1998. Conformant graphplan. In *Proceedings of AAAI*, 889–896.

Talamadupula, K.; Benton, J.; Schermerhorn, P.; Kambhampati, S.; and Scheutz, M. 2010. Integrating a closed world planner with an open world robot: A case study. In *Proceedings of AAAI*.

Wu, G.; Chong, E.; and Givan, R. 2002. Burst-level congestion control using hindsight optimization. *IEEE Transactions on Automatic Control*.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *Proceedings of Conference on Artificial Intelligence (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 Tenth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

Younes, H. L., and Littman, M. L. 2004. Ppddl1. 0: The language for the probabilistic part of ipc-4. In *Proceedings of the international planning competition*, 46.