Abstract Planning for Reactive Robots

Saket Joshi, Paul Schermerhorn, Roni Khardon, Matthias Scheutz

Abstract— **Hybrid reactive-deliberative architectures in robotics combine reactive sub-policies for fast action execution with goal sequencing and deliberation. The need for replanning, however, presents a challenge for reactivity and hinders the potential for guarantees about the plan quality. In this paper, we argue that one can integrate abstract planning provided by symbolic dynamic programming in first order logic into a reactive robotic architecture, and that such an integration is in fact natural and has advantages over traditional approaches. In particular, it allows the integrated system to spend off-line time planning for a policy, and then use the policy reactively in open worlds, in situations with unexpected outcomes, and even in new environments, all by simply reacting to a state change executing a new action proposed by the policy. We demonstrate the viability of the approach by integrating the FODD-Planner with the robotic DIARC architecture showing how an appropriate interface can be defined and that this integration can yield robust goal-based action execution on robots in open worlds.**

I. Introduction

Research in reasoning and planning for high-level goals has progressed largely independently of research in robotic architectures. While stochastic and open-world extensions to classical deterministic closed-world planning are currently active areas of research in the planning community ([18], [10]), the aim in robotics has always been to produce systems that can perform robustly in noisy environments not fully known in advance, thus handling both real-world situated control and open world dynamics. Various reactive architectures, in particular, have been proposed for mapping perceptions to actions in a way that allows for real-time action selection and execution (e.g., [15]). However, since goals and actions are not explicitly represented in those architectures (but rather, are implicit in their structures), it is difficult to adapt reactive architectures to new tasks. Hybrid reactive-deliberative architectures were proposed to address these limitations, adding two important extensions to reactive architectures: a task or navigation planner with explicit goal and action representations as well as rules connecting actions to goals, and a link between the deliberative extension and the reactive execution engine, usually a plan sequencer. The extensions, however, present significant integration challenges, e.g., to allow for typically closed-world planners to work in open-world environments while retaining the reactivity necessary for real-time plan execution [16]. Given the real-time constraints it is usually difficult to provide any guarantees on the quality of replanning.

In this paper, we propose a new integration of a stochastic planner into a hybrid robotic architecture which overcomes these two difficulties: (1) it provides a general policy for action selection that fully preserves the reactivity of the robotic architecture while being applicable in unknown environments; and (2) it allows for guarantees about the quality of the selected actions. Concretely, we show how the FODD-Planner [8] can be integrated into the DIARC robotic architecture [14] and solve tasks requiring high-level goals demonstrated in two application domains: a *search and report* task capturing a rescue operation [16] and a *search and manipulation* task where various manipulation activities related to rescue operations have to be performed [3].

In the first task, illustrated in Fig. 1, the robot starts at one end of the hall and has to deliver a package to the rescue team at the end. As an additional goal it is told to report any injured people (represented by red boxes) that may be in the rooms. The exact maps and location of rooms are not known in advance, and it is not known which rooms have injured people. Hence, these facts have to be sensed for, and appropriate actions must be performed to achieve both goals.

In the second task, illustrated in Fig. 2, the agent is to get all yellow blocks from green boxes and deliver them to a pink box. There is at least one pink box, any number of green boxes, and possibly other irrelevant objects (such as the brown boxes in Fig. 2). The number and location of green boxes, and which boxes contain yellow blocks, is unknown. Again, these facts have to be sensed for, and appropriate actions must be performed as the robot learns of them.

Novel contributions: First, we devise a novel formulation of sensing actions that with some probability "bring about" existence of objects and their properties. In this way, planners can treat sensing actions as regular actions and seamlessly integrate them into their policies, making the distinction between open and closed worlds less severe.

Our second contribution comes from observing that combining sensing actions with abstract planning can be used to yield a reactive policy that is a solution to the open-world planning problem. In particular, we propose using symbolic dynamic programming (SDP) [2] and its implementation in the FODD-Planner [8]. The SDP algorithm plans without knowledge of the initial state or even number of objects in the environment, so the abstract policy it calculates is valid in any instance of the domain. Therefore, if the situated system notices that the world state is not as expected (e.g., a new door has been detected) it can simply present the new state

S. Joshi is with the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331, USA joshi@eecs.oregonstate.edu

P. Schermerhorn is with the Cognitive Science Program, Indiana University, Bloomington, IN 47406, USA pscherme@indiana.edu

R. Khardon and M. Scheutz are with the Department of Computer Science, Tufts University, Medford, MA 02144, USA {roni|mscheutz}@cs.tufts.edu

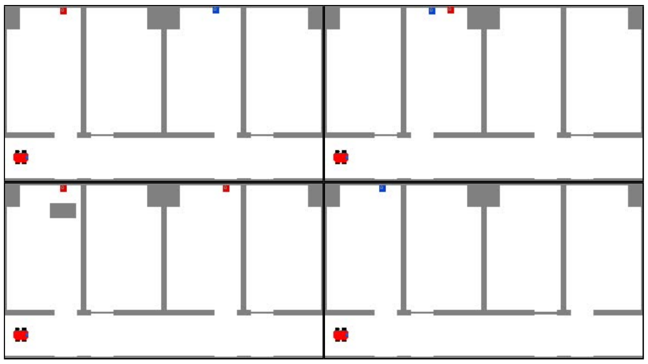

Fig. 1. Example test configurations for the search and report domain.

to the policy and get a new action for the same goal without any additional reasoning. This cannot be done with a planner that requires a grounded domain for planning.

Third, combining SDP with a formulation for sensing actions puts some requirements on the robotic architecture. In particular, the system must track and appropriately represent a notion of state to be communicated to the policy (e.g., when executing a scan for objects the system must invent a representation for the new objects identified and add them to the state). Additionally, when given an abstract action the system must interpret and execute it appropriately (e.g., moving to a location might require going through a door and/or avoiding obstacles). We show how such an interface can be defined by encapsulating robot actions using action schemas as common in planning, using these to define the predicates that are needed for state representation, and updating these predicates using the robotic system.

Fourth, we demonstrate the integration in two robotic domains as described above, discussing advantages and limitations of our approach.

Finally, to apply the FODD-PLANNER in these domains we have to support goals specified non-logically as a sum of rewards per unknown potential objects. We show how this can be approximated in a useful and flexible manner.

To summarize, our integrated architecture encapsulates capabilities of the robotic system through action schemas, handles open worlds using a formulation of probabilistic sensing actions, defines an appropriate interface in order to plan and achieve goals defined at the abstract level, and uses the power of SDP to derive reactive policies for such tasks.

II. BACKGROUND

A Markov decision process (MDP) is a mathematical model of the interaction between an agent and its environment [11]. A MDP over states S and actions A is given by the transition distributions $P(s'|s,a)$ and reward function $R(s,a)$. In this paper we assume for simplicity that the reward is independent of a (i.e. $R(s,a) = R(s)$). The objective of solving an MDP is to generate a policy that maximizes the agent's total, expected, discounted reward, for a discount factor $\gamma \leq 1$. The well known Bellman equation $V(s) = Max_a[R(s) + \gamma\Sigma_{s'}P(s'|s,a)V(s')]$ captures optimality, and the value iteration algorithm (VI) treats the Bellman

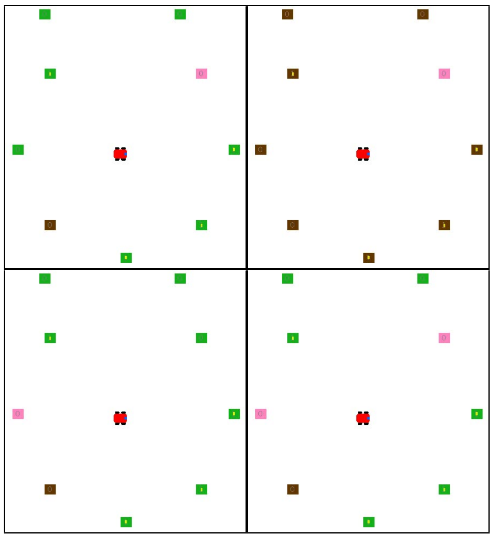

Fig. 2. Example test configurations for the search and manipulate domain.

equation as an update rule and iteratively updates the value of every state until convergence. Once the optimal value function is known, a policy is generated by assigning to each state the action that maximizes the expected value.

VI and other early algorithms require one to enumerate the state space, which is clearly not possible in large problems. As a result, work on factored and relational models aims to obtain faster solutions by taking advantage of structure in the domain. In particular, by employing compact representations for $R(s)$, $P(s'|s,a)$ and $V(s)$, this structure can be exploited to back up the values of many states at the same time. These batch backups obviate the need to enumerate the state space, thereby giving rise to a more efficient VI algorithm.

Rich structure in many planning domains comes from the observation that the world consists of objects and relations among them. Taking advantage of such relational structure, [2] developed the SDP algorithm that was later elaborated in several formalisms and systems [9], [5], [13], [17]. One of the important ideas in SDP was to represent stochastic actions as deterministic alternatives under nature's control. This helps separate regression over deterministic action alternatives from the probabilities of action effects. This simplification leads to the following compact implementation of one iteration of the VI algorithm:

1) **Regression:** The n step-to-go value function V_n is regressed over every deterministic variant $A_j(\vec{x})$ of every action $A(\vec{x})$ to produce $Regr(V_n, A_j(\vec{x}))$.
2) **Add Action Variants:** Generate the Q-function $Q_{V_n}^{A(\vec{x})} = R \oplus [\gamma \otimes \oplus_j(prob(A_j(\vec{x})) \otimes Regr(V_n, A_j(\vec{x})))]$ for each action $A(\vec{x})$.
3) **Object Maximization:** Maximize over the action parameters of $Q_{V_n}^{A(\vec{x})}$ to produce $Q_{V_n}^{A}$ for each action $A(\vec{x})$, thus obtaining the value achievable by the best ground instantiation of $A(\vec{x})$.
4) **Maximize over Actions:** Generate the $n+1$ step-to-go value function $V_{n+1} = \max_A Q_{V_n}^{A}$.

In this algorithm all intermediate constructs are captured us-

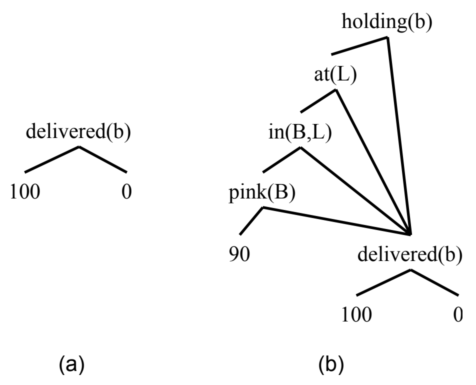

Fig. 3. FODD representations of (a) Reward function for the search and manipulate domain. (b) Corresponding 1-step-to-go value function generated by the FODD-PLANNER

ing some compact representation. The FODD-PLANNER is an SDP system that employs First Order Decision Diagrams (FODD) [17] to represent $R(s)$, $P(s' \mid s, a)$ and $V(s)$ and to perform all the calculations. We next give a brief overview of FODDs, and their use to solve SDP.

FODDs provide a graph-based representation for real-valued functions over structured domains. Thus a FODD can be used to represent a mapping from states in a planning problem to values. FODDs generalize the successful algebraic decision diagrams [1] to first order logic so as to handle objects and relations among them. Space constraints preclude a complete exposition of FODDs; instead we illustrate the main ideas using examples.

A FODD is a labeled directed acyclic graph. Each non-leaf node has exactly 2 outgoing edges. Left going edges represent the `true` branches and right going edges represent the `false` branches. Non-leaf nodes are labeled by relational atoms and leaves have non-negative numeric values.

Two examples of FODDs are given in Fig. 3. Consider a state in the search and manipulation task where the agent is holding block b in location l_1 with a pink box bx_1 in it. Such a state can be expressed as $[at(l_1), in(bx_1, l_1), pink(bx_1), holding(b)]$. To evaluate the FODD in Fig. 3(b) on this state, we must identify every path in the FODD that matches (is satisfiable in) the state. Then the final function output is obtained by maximizing over the set of leaves reached by these paths. In this example, the only paths that match the state are the ones reaching the leaves 90 and 0. Thus our final output is $max\{90, 0\} = 90$.

To implement the SDP algorithm, FODDs must support operations such as adding and multiplying functions represented by diagrams, and goal regression by diagrams; these have been provided by [17]. However, these operations introduce redundant structure in the resultant FODD and removal of these redundancies is essential for practical implementations. In this paper we use FODD-PLANNER with model checking reductions [6] to remove such structure. This requires a set of "focus states" in addition to the description of the domain when planning.

Returning to the SDP algorithm, Fig. 3(a) shows a representation of $R(s)$ for a simple ground goal in the search and manipulate domain. A reward of 100 is obtained when a specific block b is delivered. Fig. 3(b) represents the corresponding 1-step-to-go value function produced by FODD-PLANNER after one iteration. The value of the state where the block b has been delivered is 100, whereas the value of a state where the agent is holding b in a location with a pink box in it, is 90. In this description the box B and location L are quantified and the value function is valid in any instance of the domain. This simple example already demonstrates that an existentially quantified policy can be developed using SDP and that it is applicable in any instance of the domain. This is what we mean by abstract planning.

III. SENSING ACTIONS AND SDP

Closed world planners including SDP require complete knowledge of the state. We propose the following simple encoding to handle unknown facts through probabilistic actions (encoded here in a simple STRIPS-like representation, but using FODDs in our system):

```
operator lookfor(Box,Loc),
preconds([at(Loc), not_searched(Loc),
       room(Loc), potential-object(Box,Loc)])
outcome(0.5,  addList([searched(Loc)]),
              delList([]))
outcome(0.25, addList([blue(Box),
         in(Box,Loc),searched(Loc)]),
              delList([]))
outcome(0.25, addList([red(Box),
         in(Box,Loc),searched(Loc)]),
              delList([]))
```

In this formalization, taken from the search and report task described above, when a room has not been searched it has a potential object (which may not actually exist) that is discovered when looked for. Thus the robot can *lookfor* objects unless it has already searched that location before. When it does, it might find nothing (50% of the time), or find a blue box (uninjured person, 25% of the time) or find a red box (injured person, 25% of the time). The probabilities can be set appropriately with prior expectation about the frequency of red boxes/injured people; otherwise we can set it arbitrarily using an "uninformative prior". With such an action, the goal of reporting red boxes becomes just a standard goal and the planner can plan directly for it.

However, this operation on its own will not avoid the need for replanning with a standard closed world planner. When no rooms are known, the only plan is to go to the end of the hall and deliver the package. Thus, when a room is discovered, replanning is necessary. Here, the use of SDP is crucial, as SDP plans without knowledge of the initial world configuration; in this domain that means planning without knowledge of the explicit map to be traversed, that is, *planning for all possible maps*. Thus the SDP policy will be optimal for any map and any location on that map. As a result when a new room is discovered, we only need to update the state and use the reactive policy to select the action for that state. In this way the combination of SDP with the encoding of sensing actions through probabilistic operators provides a significant qualitative advantage over traditional approaches.

Considering the search and manipulate task, the agent must first search for green boxes and then peek inside the discovered boxes to check if they contain blocks. In this case we have two sensing actions. The first discovers (with some probability) boxes that exist in the same room as the robot, and the second peeks into the box and discovers (with some probability) a block inside the box. As above, using this formulation, the objects and their properties come into existence as a result of the sensing actions.

This technique can be used for the off-line planning portion that produces the policy. On the other hand the execution routine using the policy must add "potential objects" to the state description before action selection. This can be done mechanically in the planner's action selection routine.

IV. Additive Goals: Approximation and Flexibility

The SDP approach [2] was formulated with the intention of handling logically defined goals. However, as noted by many authors (e.g., [4]), rewards in structured worlds often have an additive structure. In our domains the agent obtains a constant reward per person reported in the search and report domain or per block delivered in the search and manipulate domain. While extensions to handle these types of rewards have been developed ([12], [7]), efficient algorithms and implementations do not exist. We therefore use a heuristic approximation to handle additive rewards. Our approach naturally follows from previous applications of SDP ([13], [8]), where the value obtained for a conjunctive goal is approximated by the sum of values obtained for individual atoms. Here we approximate the value of an action for an additive goal by the sum of values provided by each of the individual ground goals. Space constraints preclude a detailed explanation, but as a short example, consider an additive reward, say $\sum_{loc} reported(loc)$ from the search and report domain. The 1-step-to-go value function produced by the algorithm of [7] has the form $\exists_{box} \sum_{loc}[in(box, loc) \wedge \neg reported(loc) \wedge \ldots]$ in which the corresponding action parameters (the box in this example) must be quantified outside the sum. Our approximation effectively swaps the aggregators and produces a value function of the form $\sum_{loc} \exists_{box}[in(box, loc) \wedge \neg reported(loc) \wedge \ldots]$. Therefore, our approximation can be seen to be an optimistic (and impossible) interpretation where action parameters are selected for each location separately. Despite the obvious potential problems, this approach has been shown to work well in a number of problems involving conjunctive goals and, as we show here, it can be used for additive goals as well.

The practical implication of this approximation is that we use SDP to plan to achieve generic forms of individual ground atoms, where each atom is handled separately. Once we have the separate policies for each generic atom, we can perform on-line execution by approximating the true value using the formula above where each atom is appropriately substituted by its generic form.

As a result, our system has an added level of flexibility, because we can use a different discount factor (in the MDP sense) for each goal atom. In this way, we can also handle prioritization of sub-goals, by using a smaller value for the discount factor γ on urgent goals. If this is done then the loss in value from one step delay for the urgent goal is larger than the loss in value for the other goals and therefore actions that push the former will be preferred. This can sometimes be useful to compensate for the approximation. We demonstrate this ability in our experiments.

V. Integration

The FODD-Planner was integrated into the DIARC robot architecture, which uses the Java-based *Agent Development Environment* (ADE) [14] as a framework for implementing architectural components (e.g., perceptual processing, navigation, action planning and natural language processing). Such an integration presents challenges at multiple levels. The most obvious challenge is *syntactic* integration, establishing the mechanics of information exchange (i.e., programming API, synchronization of potentially different cycle times, etc.). However, of greater interest is the *semantic* integration of the two systems. Because the robot architecture and planner represent the world at different levels, understanding how each represents the world (i.e., entities, characteristics, locations, etc.) is key to successful information exchange. In this section we illustrate how both challenges are met by defining the interface syntactically using logical constructs, in a way that is robust despite mismatches in the semantics of these predicates.

Syntactic integration: Interaction between FODD-Planner and DIARC is via a new ADE component created to perform the mapping between DIARC representations and FODD-Planner representations. State updates and goal requests received from the robot architecture are submitted to the planner process and actions returned from policy consultation are routed back to the goal manager by the ADE planner component. Communication between architectural components using predicates is well-established in ADE [14], so at the syntactic level, the interface is straightforward once the vocabulary is established. The search and report task was captured using five deterministic actions and one sensing action: *movehall* from one location to another, *enter* a room, *exit* a room, *report* a red box, *deliver* at a certain location, and the sensing action *lookfor* as described above. Two goals, *reported* (for injured people) and *delivered* (for package to destination) are needed. The search and manipulation domain was modeled using three deterministic actions and two sensing actions: *move* from one location to another, *get* a block from a box, *deliver* a block into a pink box, and the probabilistic sensing actions *lookat* that potentially discovers a green box in a location, and *peek* that potentially discovers a block in a box. This domain has only one goal, *delivered*, requiring that a block be delivered into the pink box.

Updates are sent to the planner when actions complete or when new world states are sensed. Action-generated state updates are simply the action's postconditions, depending on the outcome. Perceptual updates can come from "background" sensing processes or from explicitly scheduled sensing actions. Background sensing is performed via the goal manager's *attend mechanism*, which allows domain-specific monitors (e.g., "attend DOORS" watches for doorways when traversing a hall) to be instantiated that continually query perceptual resources. Although implicit background sensing removes the need to schedule sensing actions, limited computational resources make explicit sensing more practical when detection events are sporadic but predictable. Regardless of the source, when a state change is detected, the goal manager updates the planner's state and the policy is consulted. Action descriptions generated by the planner are returned as DIARC script objects that can be executed directly.

Semantic integration: Ideally, the ADE and FODD-PLANNER representations would match up perfectly. However, in reality several mismatches exist between the two systems' interpretations of predicates and actions. First, as described above, the planner must invent "potential objects" with temporary names to enable planning over entities that may or may not exist. The robot, however, adds to its state objects that are actually detected, and the names it generates correspond with actual entities. Hence, arguments to sensing actions are treated as search types and unknown names are ignored. Similarly, the semantics of sensing actions are not always perfectly aligned. Obviously, the robot does not implement the planner's probabilistic schema for sensing actions—actual sensing is performed. This can lead to unexpected outcomes from the perspective of the model. For example, when asked to *lookfor* a red box, the robot might detect two or more. Likewise, in the second task, the planner tacitly assumes that the robot searches for a single green box in a specific location, when in fact the robot scans the room and returns all visible green boxes. Despite mismatched expectations, the policy can be applied without problems in both cases.

Mismatches can also be found in the interpretation of syntactically identical predicates. The robot architecture does not distinguish between entities in the environment and their locations, so any object can be treated as a location. On the other hand, the planner expects to be informed of the locations of objects presented in state updates (e.g., *in*(box_1,loc_1)). In this case the solution is simple; an entity can be specified as its own location (*in*(box_1,box_1)) and the planner can use the entity name in both roles without adverse effects. However, these examples demonstrate how subtle interpretation differences can arise when interfacing robotic architectures and planners. The crux of designing a successful interface is to provide a qualitative correspondence guaranteeing robust behavior despite this mismatch.

VI. EVALUATION

Our approach uses off-line planning time to provide an abstract policy applicable in any world instance and usable in a reactive manner in open worlds. Previous approaches are either nonreactive, require replanning or (for ground MDP solvers) require prior knowledge of problem size and structure. Therefore, one cannot perform a direct comparison to previous work. Instead our evaluation focuses on demonstrating properties of our system: real-time response and robustness of the policy across multiple scenarios in both simulated and robotic platforms. For both domains we used the FODD-PLANNER system using three approximations: model checking reductions, non-standardizing apart, and the additive goal decomposition described above [6]. We used off-line planning to produce a policy and then employed the policy in the integrated architecture, both on a physical robot and in simulation in the ADE simulator.

Fig. 4. The robot performing the search and report task.

Search and report task: For this domain we used off-line planning to derive policies for the *reported* and *delivered* goals separately. We used edge-removal model-checking reductions using a hand constructed example set that traces an agent's actions in a small hypothetical environment. The policy was generated by running 9 iterations of VI for each goal. The total off-line planning time for the two goals (on a 1.86 GHz Intel Core 2 Duo CPU with 2Gb RAM) was approximately 4 hours.

This domain was previously studied with a focus on temporal aspects using the forward state space planner `Sapa Replan` [16]. Although the goals of the current paper are different, we have explored methods for manipulating the order in which goals are pursued. For example, we can use the flexibility of planning for each goal separately to prioritize one goal over the other. The idea is to use a small γ on a high priority goal and a large γ (almost 1) on other goals, which results in a one step delay for the high-priority goal being more costly than a long delay of the other goal. This mechanism can also be used to compensate for inaccuracies in the policy that are caused by the approximation. The values for γ were chosen by performing a parameter search with respect to a suite of test problems. For the scenario of [16] our results show that using $\gamma = 0.999$ for the *delivered* goal and $\gamma = 0.45$ for the *reported* goal, the agent will search all rooms on its way to the delivery location (at the end of the hallway). Setting $\gamma = 0.45$ for both goals, the agent first makes the delivery and then goes back to search rooms for injured people.

To investigate robustness and real-time response we generated 10 different problem instances with different maps including the number of accessible rooms and boxes and their locations. The performance measures of interest are task success and planner response time. The integrated architecture was tested in the ADE simulator and on a physical robot. The simulation tests ran on a dual quad-core 2.33 GHz Xeon E5345 workstation with 64 Gb of RAM. The robot tests used a Videre robot equipped with a Hokuyo laser range finder, a USB camera and a quad-core 2.53 GHz Core 2 Extreme laptop with 4 Gb RAM (see Fig. 4). We ran the simulated system multiple times in all 10 scenarios and the physical robot multiple times in the real-world environment.

In terms of task completion the policy had 100% success; all goals are achieved as expected across all attempts in simulation and physical system. In terms of response time the evaluation shows that policy queries are efficient: the average response time (for any particular run) for goal submission, including all overhead writing and reading pipes and processing the planner's response, was about 80 msec for the simulation and about 90 msec for the robot runs.

Search and manipulation task: For this domain we used off-line planning for the single goal *delivered*. We used node-removal and edge-removal model-checking reductions using an example set constructed automatically using random walks [6] to produce 112 examples. The policy was generated by running 7 iterations of VI. This domain is simple enough that 7 iterations should include all the information needed for optimal behavior. The total off-line planning time (on a 2.8 GHz Intel Core 2 Quad CPU with 8Gb RAM) was 32 minutes. The integrated architecture evaluations used only the ADE simulator because the robot did not have any manipulation capabilities. The ADE Simulator, which includes a physics engine for realistic navigation, was extended by a simple object manipulation mechanism. Here, too, we generated 10 scenarios that differ in the location and number of boxes and in existence of blocks within the boxes. The results demonstrate 100% success in terms of goal achievement, although the policy is sub-optimal due to the additive goal decomposition. The run-time for policy evaluation (including all inter-process communication and associated processing) averaged 90 msec.

VII. Limitations of our approach

Our work demonstrated that one can successfully integrate open world planning with a reactive robot architecture. However, defining the right interface for a given task can be non-trivial due to both engineering and semantic issues as discussed above. In addition, algorithmic aspects are still a limiting factor. For example, in our evaluation we limited the number of iterations in VI to avoid unreasonable planning time. Similarly our test scenarios use moderate size instances and larger instances will require longer time for action selection which is an NP-Hard problem. These semantic, engineering, and algorithmic problems offer important challenges for future work.

VIII. Conclusion

We demonstrated that two very different technologies, a reactive real-time robotic architecture and a high-level abstract planning system, can be integrated in a novel way through a formulation using sensing actions. This integration yields robust, real-time, goal-based execution of policies generated off-line on robots in open worlds. Our results show that developing planning and reasoning independently from robotics is viable and that results from the two research areas can be usefully integrated using appropriate carefully-designed architectural mechanisms. We believe that the proposed integration is important for developing robust intelligent systems and is therefore a fruitful direction for further work.

IX. Acknowledgments

This work was partly supported by NSF grant IIS 0964457. Saket Joshi was additionally supported by a Computing Innovation Postdoctoral Fellowship. Some of the experiments reported in this paper were performed on the Tufts Linux Research Cluster supported by Tufts UIT Research Computing. This work was also supported in part by ONR MURI grant #N00014-07-1-1049 to the fourth author.

References

[1] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *IEEE /ACM ICCAD*, pages 188–191, 1993.
[2] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In *Proc. of IJCAI*, pages 690–700, 2001.
[3] K. Eberhard, H. Nicholson, S. Kübler, S. Gundersen, and M. Scheutz. The Indiana cooperative remote search task (CReST) corpus. In *Proc. of LREC*, 2010.
[4] C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalizing plans to new environments in relational MDPs. In *Proc. of IJCAI*, 2003.
[5] S. Hölldobler, E. Karabaev, and O. Skvortsova. FluCaP: a heuristic search planner for first-order MDPs. *JAIR*, 27:419–439, 2006.
[6] S. Joshi, K. Kersting, and R. Khardon. Self-taught decision theoretic planning with first order decision diagrams. In *Proc. of ICAPS*, pages 89–96, 2010.
[7] S. Joshi, K. Kersting, and R. Khardon. Decision theoretic planning with generalized first order decision diagrams. *AIJ*, 175:2198–2222, 2011.
[8] S. Joshi and R. Khardon. Probabilistic relational planning with first order decision diagrams. *JAIR*, pages 231–266, 2011.
[9] K. Kersting, M. Van Otterlo, and L. De Raedt. Bellman goes relational. In *Proc. of ICML*, 2004.
[10] N. Onder, G. Whelan, and L. Li. Engineering a conformant probabilistic planner. *JAIR*, 25:1–15, 2006.
[11] M. L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. Wiley, 1994.
[12] S. Sanner and C. Boutilier. Approximate solution techniques for factored first-order MDPs. In *Proc. of ICAPS*, pages 288–295, 2007.
[13] S. Sanner and C. Boutilier. Practical solution techniques for first order MDPs. *AIJ*, 173:748–788, 2009.
[14] Paul Schermerhorn and Matthias Scheutz. Using logic to handle conflicts between system, component, and infrastructure goals in complex robotic architectures. In *Proc. of ICRA*, 2010.
[15] Matthias Scheutz and Virgil Andronache. Architectural mechanisms for dynamic changes of behavior selection strategies in behavior-based systems. *IEEE Transactions of System, Man, and Cybernetics Part B*, 34(6):2377–2395, 2004.
[16] Kartik Talamadupula, J. Benton, Paul Schermerhorn, Rao Kambhampati, and Matthias Scheutz. Integrating a closed world planner with an open world robot: A case study. In *Proc. of AAAI*, July 2010.
[17] C. Wang, S. Joshi, and R. Khardon. First order decision diagrams for relational MDPs. *JAIR*, 31:431–472, 2008.
[18] H. Younes, M. Littman, D. Weissman, and J. Asmuth. The first probabilistic track of the international planning competition. *JAIR*, 24:851–887, 2005.