Combined Task and Motion Planning under Partial Observability: An Optimization based Approach

Camille Phiquepal **Marc Toussaint**
Machine Learning & Robotic Lab, University of Stuttgart
{firstname.surname}@ipvs.uni-stuttgart.de

***Abstract*— We present a new algorithm for Combined Task and Motion Planning (TAMP) under partial Observability. Our approach builds on the Logic-Geometric-Programming framework (LGP) presented in prior work [1, 2]. To represent partial observability, we enable the planner to reason about the agent belief-state. The presented algorithm plans long-term policies that react to the observations that the agent receives. The branching factor due to observations implies that policies are not sequential but tree-like. To handle this specificity, we introduce the notion of "trajectory-tree optimization". The algorithm works in two stages : First, trajectories are optimized piecewise i.e each action is optimized independently. Under this assumption, the decision process is markovian and policies are optimized using Value Iteration and Rmax algorithm. Secondly, the markovian assumption is relaxed, and the trajectory-tree of the best policy obtained in the first step is re-optimized globally. We test our approach on autonomous driving and object manipulation examples. To our knowledge, this is the first TAMP approach that computes long-term policies under partial observability and, hence, computes trajectories that are not sequential but arborescent.**

I. INTRODUCTION

Robots must combine the ability to reason symbolically about discrete actions (Task planning) and geometrically about their realization in the real world (Motion planning). Integrated approaches are referred to in the literature as Task and Motion Planning (TAMP). With the exception of [4], current TAMP research assumes full observability. However partial observability is pervasive in many real world situations, e.g. when objects are hidden or partially hidden. If some objects to manipulate are inside containers, the robot has to explore its environment to perform its task. Self driving cars face the same problem when operating in the presence of other vehicles limiting the field of view of the ego vehicle.

In this paper we extend the Logic-Geometric Programming approach (LGP) presented in prior work to handle Partial Observability. Under partial observability, policies have branching points due to observations. On the geometric level, it means that Motion Planning consists in optimizing a trajectory-tree i.e. trajectories with ramifications. Our TAMP solver works in two stages : First piecewise optimization and then joint optimization. During the first stage, the trajectories of actions are optimized independently (piecewise). This allows the problem to be considered as a Partially Observable Markov Decision Process (POMDP) that we solve from a start belief state. We focus on the case where the true start state is uncertain and partially observable, while the transition model of the underlying MDP is not stochastic. The POMDP is solved in an iterative process. It starts with initial heuristic reward values. A policy is computed using Value Iteration. This candidate policy is given to the motion planner. The resulting trajectory costs are used to replace the initial heuristic reward values of the POMDP. This process is iterated until an equilibrium is reached (no more policy improvement).
The second stage of the algorithm is the Joint Optimization. The trajectory tree of the best policy obtained in the first stage is re-optimized globally: instead of optimizing the sequential motion of each action in isolation, the whole trajectory-tree is optimized all at once.

II. RELATED WORK

Concerning Combined Task and Motion Planning, a number of approaches [5][8][9] rely on the discretization of the configuration spaces or action/skeleton parameters to leverage CSP methods. Our prior work presented in [1][2] states TAMP problems as an optimization problem. These approaches assume full observability, and plans are linear sequences of actions. To our knowledge, the system developed by Lozano-Perez and Kaebling [4] is the only other TAMP planner considering partial observability. A *Look* action is used to actively move the robot sensor to acquire information. This approach (Hierarchical Planning in the Now) interweaves planning with execution (in the now). Sequences of actions are planned by approximating the system dynamics (results of actions and observations). Re-planning is triggered once the robot ends up in a state not covered by the plan. Our approach aims at planning a full policy from the starting state to the final state. In addition, we aim for smooth and locally optimal trajectories.

Planning for autonomous driving also entails a layered decision making process with a combination of symbolic decisions and planning in a continuous space. [6][7] provide a surveys about planning for self-driving cars. To our knowledge, there exists no literature on TAMP approaches for decision making in the automotive context.

III. PROBLEM STATEMENT

Let's consider a decision process defined by a 7-tuple $(S, A, T, \Omega, O, \gamma, G)$, where :

- S is a set of states

Fig. 1: The ego vehicle (cyan) wants to overtake but cannot observe if a vehicle arrives in the opposite direction because of the truck.

- A is a set of actions
- T is a set of conditional transition probabilities between states
- Ω is a set of observations
- O is a set of conditional observation probabilities
- γ is a discount factor
- G is a set of goal conditions defined over the belief state space. A goal condition indicates if a belief state is terminal or not

Although such a 7-tuple is very similar to a POMDP, it differs in several ways : The decision process is not assumed markovian, and no rewards are defined. Moreover, there is the notion of terminal belief state which doesn't exist in the POMDP framework.

After having taken an action a and received an observation o, the relation between the current belief state b and the new b' obeys the same equation than a POMDP.

$$b'(s') = \alpha O(o|s',a) \sum_{s} T(s'|s,a) b(s), \forall s \in S, \forall s' \in S$$

In the above equation, α is a normalization constant.

A. Decision graph

When planning from an initial belief state and observations are discrete, the set of all possible reachable belief states is a graph. Each node is a belief state. There are two kinds of nodes:

- Action nodes: the agent has to choose which action to take. The edges starting from the node are the different possible actions. The reward received after executing the action is associated to each action-edge.
- Observation nodes: the agent receives an observation, each edge starting from an observation node is a possible observation.

1) Example for a decision graph: We consider a car behind a truck, the car wishes to overtake but cannot see the opposite lane because of the truck, see Fig. 1. The car can take three actions: look into the opposite lane, overtake the truck, or continue to follow the truck. After looking into the lane (move slowly toward the center of the road), the car receives an observation (lane free or not). Fig. 2 is the decision graph of this problem. The blue node is the start belief state. The green nodes are terminal belief states.

2) Policy: A policy π is a mapping from belief states to actions. The thick edges in Fig. 2 represent a possible policy. The decision graph potentially contains cycles, however, in the following, we will only consider policies that are trees

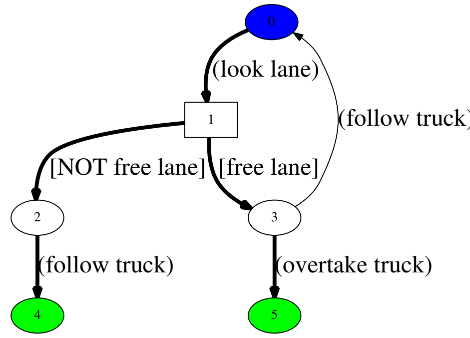


Fig. 2: Decision graph for overtaking, the thick edges represent a possible policy

(no cycles). In the case of full observability, the policy boils down to a sequence of actions. A policy solving the problem, on the symbolic level, is a tree whose leafs are terminal belief states. We call it a candidate policy.

The observation model O and the initial belief state at the root node b_0 define the prior probability of reaching a given node with the belief state b of a policy. We will note this probability $p(b|\pi, b_0)$.

B. Action grounding

To link the symbolic action with trajectories, we associate trajectory cost and constraint functions are associated to each action. These functions implicitly define the action return and its geometric effect.

More formally, let $\mathcal{X}$ be the configuration space of the whole environment, including the robot and all object configurations. Let's assume that the agent takes an action $a \in A$ which takes effect over the interval $[t_k, t_{k+1}]$. The costs and constraints functions f_a, g_a, h_a are associated to the action a. Let x be a trajectory in $\mathcal{X}$ over $[t_k, t_{k+1}]$.

The trajectory x is feasible for the action a if its constraints are satisfied,

$$\begin{aligned} g_a(x(t), \dot{x}(t), \ddot{x(t)}) &<= 0 \\ h_a(x(t), \dot{x}(t), \ddot{x(t)}) &= 0 \ . \end{aligned}$$

If x is feasible under a, its cost is,

$$\begin{aligned} c(a,x) = \int_{t_k}^{t_{k+1}} & f_a(x(t), \dot{x}(t), \ddot{x(t)}) dt \\ s.t \quad & g_a(x(t), \dot{x}(t), \ddot{x(t)}) <= 0 \\ & h_a(x(t), \dot{x}(t), \ddot{x(t)}) = 0 \ . \end{aligned}$$

If x is infeasible under a, we define its cost as $+\infty$.

Implementing a policy geometrically consists in determining trajectories for each actions, this is a mapping from belief state to actions. Taken as a whole, it is a trajectory-tree i.e a trajectory with ramifications. We will note ψ such a trajectory tree.

C. Optimal policy

A TAMP policy is the couple formed by a policy π and its trajectory tree ψ. Its cost is given by,

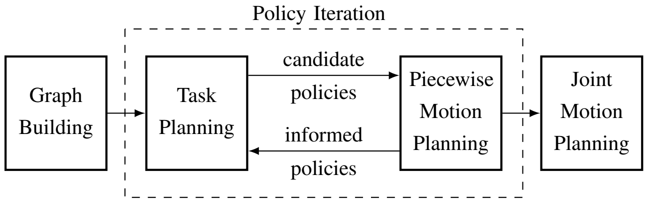


Fig. 3: TAMP algorithm

$$c(\pi,\psi) = \sum_{b\in\pi} p(b|\pi,b_0)c(\pi(b),\psi(b)) \ . \quad (1)$$

Finding the optimal TAMP policy Π^* from an initial belief state b_0, consists in determining a policy π^* and its trajectory-tree ψ^* which minimize the overall trajectory costs,

$$\Pi^* = \{\pi^*,\psi^*\} = \operatorname*{argmin}_{\pi,\psi} \sum_{b\in\pi} p(b|\pi,b_0)c(\pi(b),\psi(b)) \ . \quad (2)$$

IV. TAMP Algorithm

Our approach for optimizing a TAMP policy (solve equation (1)) is schematized on Fig. 3. First, the decision graph is built. In a second stage, the solver optimizes a policy by optimizing motions step by step, we call it the piecewise approach. At this stage, we define the reward of an action as the negative trajectory costs its motion. The reward only depends on the geometric state of agent when starting the action. In other words, we assume the problem markovian. This assumption allows us to leverage classic algorithms from the MDP litterature. In the final stage, joint trajectory optimization is performed on the best policy found. The trajectory-tree as a whole. It is typically slower but gives smoother and more optimal results. Different optimization parameters (more time steps) can be used in this final stage.

The validity of our method relies on the underlying assumption that the markovian version of the problem is a good heuristic for solving the original problem. This is true in practice. However, since this approximation is pessimistic, not optimistic, it compromises the completeness and optimality of the method.

A. Graph Building

The decision graph, is expanded from the start belief state using a breadth first strategy. In the general case, the number of reachable belief states is infinite leading to an infinite decision graph. The algorithm that we present in this paper assumes a graph of finite size. To constraint the graph size to be finite, one solution is to expand the graph up to a certain maximal depth.

B. Value Iteration on the decision graph

Under the optimal policy, the value of each action node obeys the Bellman equation,

$$V^\star(b) = \max_a \left[R(a,b) + \gamma \sum_{o\in\Omega} O(o|b,a)V^\star(T(b,a,o))\right] \ . \quad (3)$$

At each pass, the algorithm goes through each action node and updates its current value based on the values of its children. The update process is iterated until the values are stable i.e. the Bellman equilibrium has been reached,

$$V^\star_{i+1}(b) \leftarrow \max_a \left[R(a,b) + \gamma \sum_{o\in O(b,a)} O(o|b,a)V^\star_i(T(b,a,o))\right] \ . \quad (4)$$

Once the value is know, the optimal policy is retrieved by selecting at each action node the action leading to the children with the highest expected value.

C. Piecewise Motion Planning

Task Planing gives candidate policies to the motion planner. The execution time of this phase is crucial for the overall planning time. This is performed in two steps, a first feasibility check and, then, trajectory optimization. For these two steps, each action is optimized independently in a breadth-first order. Trajectory Optimization is performed using the Logic Geometric Programming framework (LGP)[1][2]. The results (the trajectory and its cost) are saved, so an action edge is only planned one time. There may be a strong overlap between candidate policies (same edge in many candidate policies). This is especially the case in the last iterations of Policy improvement, but motion planning is performed only on edges that haven't been planned yet. Intuitively, as Policy iteration goes along, the decision graph is filled with geometric information. The initial reward is the parameter which controls how large the coverage will be and how fast convergence occurs. Pose level approximation and re-using results of previous optimizations for the planning of new policies are ways to speed-up the optimization.

1) Pose level optimization: To detect quickly infeasible trajectories, we first optimize key-frames only (robot pose at each node). This step is much quicker than optimizing a trajectory. If an action is impossible during the feasibility check, the optimization is not pursued further. This feasibility check is optimistic, it might succeed even if the path itself is infeasible (no possible trajectory without collision between two key-frames for example). Although this gives a feasibility information it doesn't provide with trajectory costs.

2) Trajectory optimization: The second pass of optimization consists in optimizing between key-frames, we consider typically 20 time steps for each action. In addition to the cost and constraint functions defined by the action, the robot dynamic and collision avoidance are considered which results in accurate trajectory costs that will update the reward model of the decision graph.

The initial rewards of decision graph are replaced with the trajectory costs resulting from the fast motion planning. The resulting geometric configuration of the robot at the end of each planned action is also saved. The reward of infeasible actions is set to minus infinity which excludes this edge from next candidate policy.

D. Rewards initialization and graph exploration

The initial reward value is used to control the exploration vs. exploitation trade-off of the policy search behavior. An optimistic initial reward value encourages exploration. In the context of model based reinforcement learning, this is the Rmax algorithm [3]. Rmax is used for learning the rewards values of a MDP or a zero sum stochastic game. In Rmax, the reward model is initialized in an optimistic fashion. The agent acts based on the optimal policy derived from its current reward model and update its model with the observed rewards. Rmax can attain near-optimal average reward in polynomial time. Our case is different, we are not, in a reinforcement learning setting, and the decision process that we consider is not a MDP but a POMDP. However, the same paradigm of "optimism in the face of uncertainty" to explore applies in our case.

On the other hand, with pessimistic initial rewards, the equilibrium is attained after a first policy has been successfully optimized by the motion planner. Higher initial rewards converge to better policies at the expense of the overall planning time. Quantifiable data about the influence of initial reward values on the number of iterations are given in the experimental results section (V).

E. Joint Motion Planning

Optimizing actions independently cannot capture long-term effects on the trajectories. When considering a policy as one single optimization problem, final actions potentially influence motions earlier on the trajectory-tree. The autonomous driving example of the experimental results exemplifies this idea. Moreover, planning parameters (e.g. number of key frames) may be different when planning for informing the search or planning for outputting the final policy. This final stage of optimization allows for reaching a better optimum and smoother motions. Although it takes much longer but this is done on one single policy.

Because of the observation branching, the whole motion is a trajectory-tree and is not straightforward to optimize. We solve it in two phases: First, linear trajectories are optimized from the root graph node to each one of the terminal belief states (see opt_1 and opt_2 on Fig. 4). Secondly, the trajectories are re-optimized with additional constraints enforcing that the common parts between trajectories are identical (see opt_3 and opt_4). The re-optimization is potentially performed multiple times until the equality constraints across trajectories are fully satisfied (in practice, one iteration is often enough).

V. Experimental Results

A. Overtaking behavior

We consider the overtaking problem introduced previously. Fig. 2 is the decision graph and Fig. 6a shows two start configurations. In the first configuration (a), the opposite lane is free enough to overtake. In (b) overtaking is not possible. The Fig. 5 shows the optimal policy. The trajectory cost of the action *Look* is implemented as the distance between the car and the center of the road. It tends to move the car toward

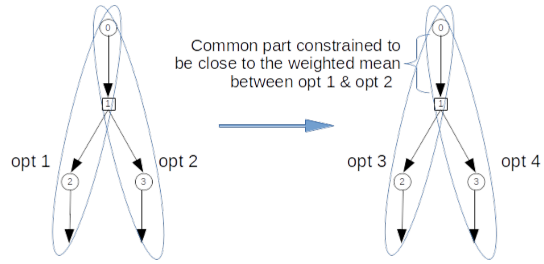


Fig. 4: Joint optimization

the center to get sight of the lane (see Fig. 6b). The action *Follow* is implemented as a constraint which is satisfied if the ego-car is behind the truck (with a safety distance) at the end of the action. On the other hand, the action *Overtake truck* is implemented as a constraint satisfied if the ego-car is in front of the truck.

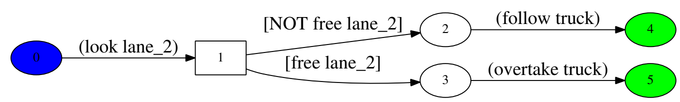
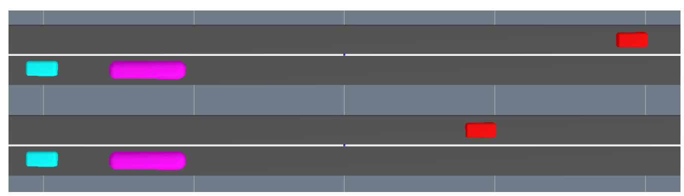
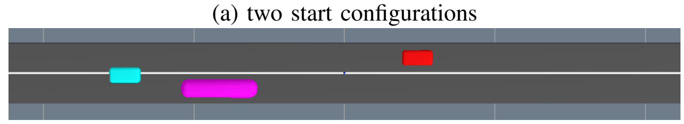


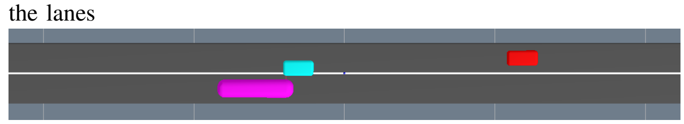
Fig. 5: Overtaking optimal policy



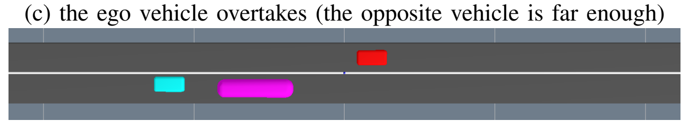
(a) two start configurations



(b) *Look* action, the ego vehicle moved to the road center to observe the lanes



(c) the ego vehicle overtakes (the opposite vehicle is far enough)



(d) the ego vehicle moves back to its lane (opposite lane not free)

This example emphasizes the improvement brought by the joint trajectory optimization. The curves on Fig. 7 represent the longitudinal velocities of the trajectory-tree for different planning configurations. At $t = 6.5s$, the car receives the percept $[lane\ free]$ or $[NOT\ lane\ free]$. This is the branching point of the trajectory-tree. If the lane is free, the car accelerates to overtake and then slows down once the truck is overtaken. Otherwise, the car slows down and move back to follow the truck.

The gray curve results from the fast motion planning. Actions are optimized in isolation, when looking at the lanes,

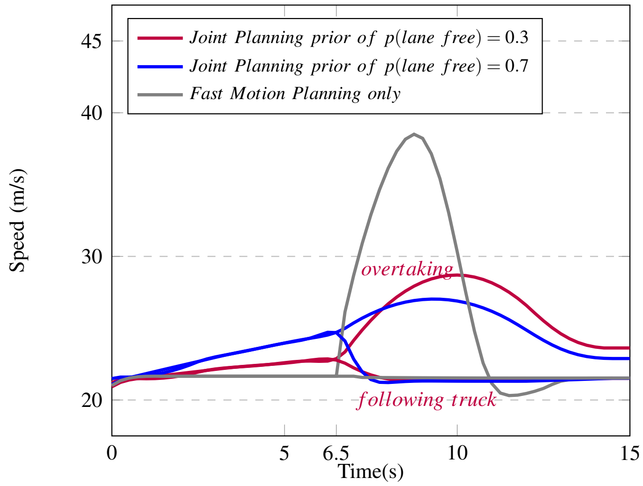
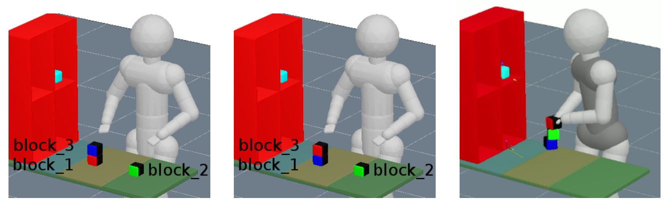
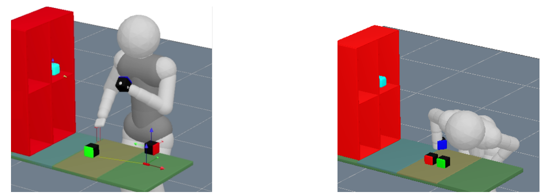


Fig. 7: Longitudinal speed of the overtaking maneuver



(a) first start configuration with model B (b) second start configuration with model B (c) example of goal configuration

Fig. 8: Example of configurations. The robot must stack the blocks in a given color order. The block colors are not visible from behind



(a) *Look* with the block in hand (b) *Look* just after having posed a block

Fig. 9: Example of geometric configurations reached after the *Look* action

the car keeps exactly the same speed (gray curve is flat for $t < 6.5s$). When starting to overtake, the car is, still quite far from the truck and accelerates strongly to overtake. On the other hand, the blue and purple curves (Joint Optimization) are much smoother. To avoid a too strong acceleration, the car anticipates and accelerates slightly when looking. If overtaking is possible, the car pursues its acceleration, otherwise, it slows down and go back in the lane. We think that this mimics what human drivers do in case of "tense" overtaking maneuver. The initial belief state also influences the behavior. If it is likely that overtaking is possible (0.7 likelihood for the blue curve), the car will accelerate more when looking. In practice, the initial belief state could come from a service providing global information about the traffic in the area.

B. Sussman anomaly under partial observability

We consider an humanoid robot (see Fig. 8). There are three blocks a table. The robot has to stack the blocks in a given color order on one of the three table locations (red on the top, green in the middle, blue at the bottom). We compute trajectories for all the robot joints (27 degrees of freedom). Only the robot left hand is assumed to be able to grasp. The blocks are black with one side colored (assumed to be the opposite side). The robot knows where the blocks are (referred as *block*_1, *block*_2, *block*_3). However it cannot see the colored side from behind and has explore to identify the blocks and to build the stack in the correct order. The table is subdivided in 3 different locations (table left, center and right), if a block is already at a location, the location is occupied (no block can be added on the same table part). There are 3 possible actions:

- **Look** a block: the robot seeks to align its sensor (robot head) with the colored side of the box. This will typically lead the robot to both move its head and its hand simultaneously (see Fig. 9a). After this action, the agent receives an observation (color of the block).
- **Grasp** a block: only the left hand can grasp
- **Place** a block at a location: the block is placed at the given table location, or onto another block.

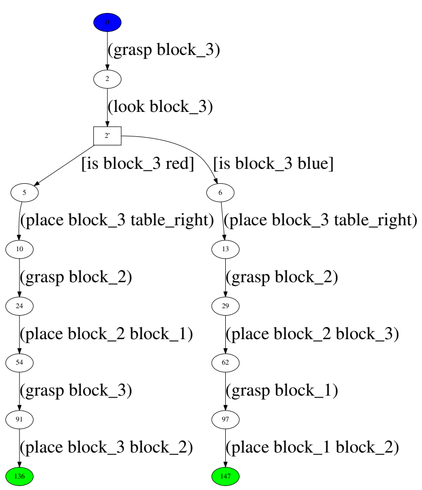
We evaluate two different action models. In the first variation (Variation 1), the action *Look* has a symbolic precondition: the robot should be holding a block before checking it. This precondition is not present in the second variation (Variation 2). The *Look* action is possible more often which increases the branching factor and the size of the decision graph. However, most of the time, when the robot doesn't hold a block, the *Look* action is infeasible geometrically: Indeed the robot has to place place its head far ahead and look backwards which is, in most cases, infeasible given the geometrical constraints of the robot. It is however interesting to note, that this is possible in some cases (if the robot has just previously placed the block close to the table border with some orientation see Fig. 9b). In other words, the grasp-precondition is not absolutely necessary to ensure the feasibility of the *Look* action. This variation is used to analyse how our approach works with a more "free" albeit not invalid symbolic problem description causing a lot of motion planning failures.

To evaluate the scalability, we test with three different initial belief states configurations. In the first configuration (A), the agent has a prior knowledge of the color of each block. This boils down to the fully observable case. To keep the action model unchanged, we still impose that the agent has to look one block to complete its task. In the second configuration (B), 2 blocks are unknown. Fig. 8a and 8b show the two possible start configurations with this model. In the third case (C), there are no prior knowledge which leads to 6 possible initial configurations. The different start configurations are summed up in the table I. In all cases, the initial belief state is uniform (e.g. 1/2 likelihood for each possible start configuration with the model B, 1/6 likelihood for model C).

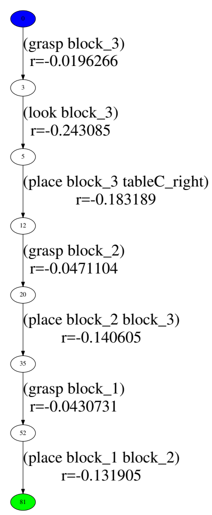
	Belief state size	Blocks known	Grasp *precondition*	Graph size	Graph building time(s)
A1	3/3	1	yes	192	0.20
A2	3/3	1	no	192	0.24
B1	1/3	2	yes	336	0.53
B2	1/3	2	no	480	1.08
C1	0/3	6	yes	2076	8.55
C2	0/3	6	no	4128	34.3

TABLE I: Summary of the considered problem variations

Fig. 10a shows an optimized policy for the model B. The agent first grasps a block and looks it. Once the block is identified, the agent pursues the stacking.



(a) result policy for B2



(b) result policy for A2

	R_0	Iter-ations	N of actions*	Task planning	Fast motion planning	Joint motion planning	Total*
A1	-0.25	1	7	0.007	1.37	11.4	13.0
	-0.1	2	7	0.018	3.14	9.7	13.1
	-0.015	13	7	0.10	18.5	16.3	35.1
A2	-0.25	2	7	0.013	1.81	8.43	10.5
	-0.1	3	7	0.019	2.08	11.6	14.0
	-0.015	16	7	0.10	14.8	11.4	26.6
B1	-0.25	1	12	0.014	2.72	26.2	29.5
	-0.1	1	12	0.016	2.85	21.5	24.9
	-0.015	11	12	0.13	30.6	20.6	51.9
B2	-0.25	7	12	0.089	9.10	56.7	66.9
	-0.1	14	12	0.17	20.4	59.9	81.4
	-0.015	39	12	0.55	69.9	38.6	110.1
C1	-0.25	1	33	0.077	11.6	172.3	192.5
	-0.1	8	33	0.35	56.9	105.5	170.9
	-0.015	41	37	2.07	321.3	112.7	444.1
C2	-0.25	15	33	1.16	42.4	100.8	182.8
	-0.1	48	33	3.38	146.5	250.9	436.6
	-0.015	303	39	26.0	1188.9	261.5	1510.9

* Number of actions of the final policy

** The total planning time also includes the graph building time given in the table I

TABLE II: Number of iterations and planning times

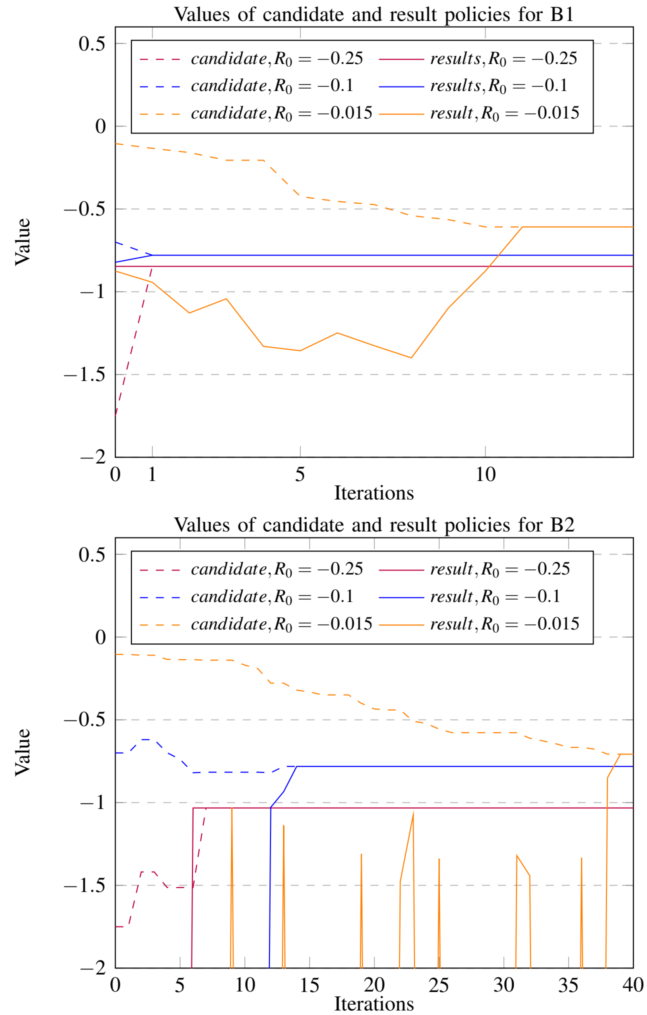


Fig. 11: Policy improvement over iterations

1) Influence of initial rewards: With an initial reward of -0.25 (pessimistic initial reward), the search finishes as soon as a first policy is found. This happens after one single iteration with the B1. With the B2, the search encounters infeasible actions, the first possible policy is found at the 7th iteration. This policy is less optimal than the results found with the other initial rewards.

The highest initial reward (-0.015) is always optimistic and leads to the most exploratory behavior. The value of candidate policies (containing at least one unexplored action) are consequently always higher than the result policies (see the orange curves on Fig. 11. It requires the biggest number of iterations. In particular, task planning also explores deeper policies (with more steps). In some cases a slightly deeper policy results in a better trajectory cost (see B2 with R_0 = -0.015). The search converges to a better policy than with the other initial reward values. The small improvements in the last iterations are due to small rearrangements of the target location when placing blocks (e.g. place a block on table-left instead of table-center).

2) Influence of the action precondition: Removing the precondition increases strongly the decision graph size, more iterations are needed, and a majority of candidate policies are infeasible geometrically. This is visible on Fig. 11 for the model B2 (below), the curves of result policies are very discontinuous because the majority of them are infeasible. Most of the time, Motion planning fails due to one single action. The policy value is minus infinity, but the resulting rewards of each possible actions still inform the decision graph leading to an overall improvement. The search reaches an optimal policy which is as good as the policy obtained with the model with precondition. We think that this is an important quality of the proposed solution. Adding domain specific knowledge in the task planning (to ensure that motion planning will succeed) speeds up the search. However, in the general case, we think that it is not always possible / convenient to incorporate such geometric reasoning (reachability of a view point, reachability of an object) in the logical reasoning.

3) Execution time and scalability: The overall planning time is dominated by the motion planning (see table. II As long as the model is simple (A or B) or the exploration kept low (R_0 = -0.25), motion planning is dominated by the single pass of joint optimization. The execution time of this pass mainly depends on the total number of action steps in the policy and on the belief state size but is independent from the number of iterations occurring before. In the configurations requiring the biggest number of iterations (C1 and C2 with R_0 = -0.015), motion planning and the overall planning time are dominated by the fast motion planning phases of the policy improvement iterations. It also appears clearly that scalability is a crucial problematic here. All the parameters of the problem increase drastically with the size of the belief state (graph size, required number of iterations, size of the resulting policy, computation time). Parameterizing the search with a very exploratory behavior may be feasible for problems of small sizes but suffers from the curse of dimensionality. One way to still enable some exploration while maintaining a bounded planning time is to save the best candidate policy planned so far and interrupt the iterations when a given time limit is reached.

VI. Conclusion & Future Work

We proposed a new, optimization-based approach to TAMP problems. It handles partial observability by reasoning over the agent + environment belief state and by optimizing trajectory-trees that can account for the observation branching. It can plan policies that combine exploratory actions (mostly sensor trajectories) and exploitative actions (e.g grasp, place). The degree of exploration over the vast space of all possible manipulations policies can be controlled by one single parameter (initial heuristic reward). As motion optimization is time consuming, the ability to quickly detect if an an action is infeasible is crucial and we perform it by performing a fast pose-level optimization. Moreover, the policy iteration process naturally copes with motion planning failures that simply inform the decision graph with the real cost (albeit infinitely big) of a given action. Scalability becomes an issue when the number of manipulations and or the size of the belief state increases. An efficient way to speed-up the search is to have a task-level model that is accurately tailored for the problem to solve (example of the grasp precondition), this prevents too many motion planning failures and limits the branching factor. In future work, we intend to explore, how the task-level model can be refined and learned using the results from multiple planning queries. Our current method computes policies that address every possible outcome during the possibility execution. To scale better, we plan to investigate an approach where the policy is planned only for handling the most probable belief state trajectories. As such a policy couldn't handle every outcome at execution time, re-planning would be triggered once the execution layer detects that the system is evolving toward a belief state which is not covered by the current policy.

References

[1] M. Toussaint and M. Lopes, Multi-Bound Tree Search for Logic-Geometric Programming in Cooperative Manipulation Domains. Accepted at ICRA 2017.

[2] M. Toussaint, Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning. In Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI 2015), 2015.

[3] Brafman, Ronen I. and Tennenholtz, Moshe: R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research, Volume 3, 213-231, 2003

[4] Leslie Pack Kaelbling, Tomas Lozano-Perez. Integrated Task and Motion Planning in Belief Space, International Journal of Robotics Research, 2013

[5] T. Lozano-Pérez and L. P. Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on, pages 3684-3691. IEEE, 2014

[6] Katrakazas, C., Quddus, M., Chen, W.-H., Deka, L.: Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. Transp. Res. Part C 60, 416-442, 2015

[7] Paden, B., Cap, M., Yong, S.Z., Yershov, D., Frazzoli, E.: A survey of motion planning and control techniques for self-driving urban vehicles. IEEE Trans. Intell. Veh. 1(1), 33-55, 2016

[8] F. Lagriffoul, D. Dimitrov, A. Saffiotti, and L. Karlsson. Constraint propagation on interval bounds for dealing with geometric backtracking. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pages 957-964. IEEE, 2012

[9] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson. Efficiently combining task and motion planning using geometric constraints. The International Journal of Robotics Research, 2014