2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Daejeon Convention Center
October 9-14, 2016, Daejeon, Korea

Co-optimizing Task and Motion Planning

Chongjie Zhang and Julie A. Shah

Abstract— **Solutions to robotic manipulation problems can be substantially improved through integrated task and motion planning. Existing approaches typically focus on satisfaction, finding a feasible solution, instead of optimization. We formulate large-scale robotic manipulation problems as multi-level optimization, incorporating task, action, and motion planning. We develop an integrated planning approach for solving this optimization problem and generating a combined motion plan for a robot to optimize a task-level objective. This approach utilizes a combinatorial search algorithm for task planning and incrementally exploits information from lower-level optimization to improve the high-level task plan. Empirical results show that this integrated approach not only significantly outperforms a traditional top-down approach in solution quality, but also avoids infeasible lower-level motion plans.**

I. INTRODUCTION

In many practical applications, a robot needs to autonomously move around and manipulate objects in the environment. One such application is to deliver parts to a set of target locations in a factory. In this paper, we are particularly interested in the *optimization* problem, i.e., how to achieve its goal at minimum cost (e.g., execution time). In order to solve this optimization problem, the robot must carry out high-level task planning in conjunction with low-level motion planning. It is challenging because decisions at the high level, such as the order of pick-and-place operations for objects, depend on low-level considerations, such as the existence or cost of feasible motions for particular tasks.

Traditionally, such problems have been attacked by a top-down approach. This approach separates task planning from motion planning by providing an estimated cost for each task, and then solves the two components independently by first finding a task-level plan and then solving the motion planning problem for each task. While this approach is usually tractable, it can lead to poor, or even infeasible solutions. This is because the task planner has incomplete knowledge of the cost and dynamics that will be utilized by motion plans when achieving its tasks.

There recently have been very impressive advances on combined task and motion planning (TAMP). For instance, Srivastava et al. [1] have proposed a general approach for interweaving existing symbolic planners (e.g., for tasks describable in PDDL) with existing geometric planners (e.g., RRTs) to check the geometric feasibility of the actions proposed by preliminary symbolic plans. Most existing TAMP approaches focus on the satisfaction problem and backtrack to task planning only when there is no feasible motion plan. These approaches are not sufficient for solving our optimization problems.

Computer Science and Artificial Intelligence Lab, MIT, Cambridge, MA 02139 {chongjie, julie_a_shah}@csail.mit.edu

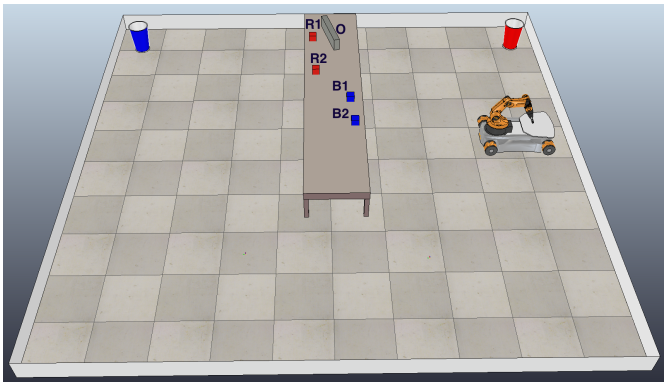

Fig. 1. An example scenario of the pick-and-place problem

For example, in a pick-and-place problem illustrated by Figure 1, a robot picks an object from the table and places it into the container of the same color. The goal is to finish all tasks as soon as possible. A task planner will not uncover that the robot must maneuver to the other side of the table to pick up object R_1 until the motion planner is invoked. The task planner is therefore likely to generate the solution $\langle R_2, R_1, B_2, B_1 \rangle$, which is optimal assuming there is no obstacle O_2. Most existing TAMP approaches will not backtrack and find a real optimal solution (e.g., $\langle R_2, B_2, B_1, R_1 \rangle$), because the sub-optimal sequence $\langle R_1, R_2, B_1, B_2 \rangle$ can refine to a feasible motion plan.

To address this limitation, this paper presents a multi-level *optimization* approach for integrated task and motion planning. This approach formulates large-scale robotic manipulation problems as a three-level optimization problem. At the top level is task planning, which aims to sequence tasks (e.g., pick-and-place operations) in order to optimize a performance metric, such as to minimize time to complete all tasks. As different tasks often have different robot finishing configurations, which affect the completion cost of the next task, we model the task planning optimization problem as a traveling salesman problem (TSP). This efficient representation allows us to utilize a combinatorial search algorithm (e.g., a heuristic TSP solver) instead of generic task planners. At the middle level is action planing, which aims to generate a sequence of primitive actions for implementing a high-level task. To structure and limit the search space, we encode a high-level task as a hierarchical action network (HAN). At the bottom level is motion planning, which produces a

978-1-5090-3762-9/16/$31.00 ©2016 IEEE

motion plan to achieve the primitive actions produced by the action planning level. The optimization at the higher-level depends on solutions of lower-level problems, which are not trivial to compute. In addition, the problems at each level pose computational challenges by themselves.

To exploit this efficient problem representation, we develop an integrated approach for co-optimizing three-level problems, which generates a combined motion plan to optimize a task-level objective. This approach is composed of three interacting planning algorithms, one for each level. The top-level task planning algorithm first generates a task plan without considering lower-level problems and sequentially refines its tasks to executable motion plans by calling the lower-level action planning algorithm. The key novelty for this task planning algorithm is that it incorporates the existence or cost of feasible motion plans returned from lower-level planners and incrementally improves its task plan during the refinement process. We design a recursive method for finding action plans by searching through HANs. Adapting the planner-independent interface for TAMP [1], we develop an iterative, backtracking algorithm that allows to use existing algorithms (e.g., RRT [2], RRT* [3], and PRM [4]) to refine action plans to motion plans. We illustrated our approach in a pick-and-place domain. Empirical results show that our integrated approach significantly outperforms the traditional top-down approach and scales well to large problems (i.e., with more than 50 tasks).

Our co-optimization approach explicitly models the robot transition cost from one task to another. There may be constraints between tasks related to objects. For example, one object may block another object, where the transition cost from the first object to the second one will be infinite. Our backtracking search algorithm will find a feasible task sequence to optimize its objective. In this paper, we assume that the completion cost of a task only depends on its previous task, but not the whole previous task sequence, and also that a robot performs one task per time step.

II. RELATED WORK

There is a wealth of work in combining task and motion planning (TAMP). This section will first review satisfaction-based TAMP approaches and then discuss related optimization-based approaches. Satisfaction-based TAMP approaches are structured into three groups, whose task planning uses PDDL, hierarchical task networks, and constraint satisfaction problems, respectively.

Dornhege et al. [5] describe how to combine low-level motion planning with high-level task planning via semantic attachment to a PDDL planner. In their approach, the lower level planner is used to check action applicability and compute effects whenever certain high level actions are used. Garrett et al. [6] address the TAMP problem in a way that uses symbols (e.g. "CanGrasp" and "Reachable") to represent geometric preconditions for actions also on the symbolic level. These predicates depend on the actual geometric configuration; their approach samples configurations to discretize such continuous variables. Similar to this, Srivastava et al. [1] devise a symbolic description that includes predicates to abstract geometric feasibility conditions and represent action operator preconditions on the symbolic level. For a given task plan, the predicates are evaluated on demand, and an "Obstructs" predicate is added for the situation where the planner does not find a path due to an obstrucing object.

Hierarchical planning approaches have also been proposed for TAMP. Wolfe et al. [7] represent robotic manipulation problems as vertically integrated hierarchical task networks (HTNs) and also develop the SANTH algorithm for finding high-quality solutions. Although this algorithm uses information about subtask-specific irrelevance to speed up the search, it is based on exhaustive search and may not scale to large problems. Kaelbling and Lozano-Pérez [8] present a more flexible technique for combining both task and motion planning called "hierarchical planning in the now." The technique generates a hierarchy dynamically. When refining a transition at one level in the hierarchy, a planner is used where the goal specification is given by the preconditions of the destination node of the transition. Pandey et al. [9] and de Silva et al. [10] also use HTNs and their system can backtrack over choices made by the geometric module, allowing more freedom to the geometric planning than in the approach of Dornhege et al. [5].

Methods for constraint satisfaction problems (CSP) have been employed for TAMP. Lozano-Pérez and Kaelbling [11] reduce the TAMP problem to a constraint satisfaction problem. They introduce the notion of an action *skeleton*, which is a sequence of symbolic actions leaving their specific geometric parameters unspecified. The decisions on the geometric parameterization of actions are deferred to a stage where the skeleton has been fixed. This approach relies on discretizing the geometric parameters of a skeleton. Lagriffoul et al. [12], [13] also employ CSP methods (constraint propagation) to resolve geometric constraints.

In summary, these TAMP approaches to robot manipulation assume a symbolic goal description in their demonstrations and typically introduce predicates to represent geometric conditions or features (including the existence of feasible paths). They focus on the feasibility of the task planning solution instead of optimality.

Recently a few works have started addressing the TAMP optimization problem. Toussaint [14] introduces a logic-geometric programming approach to tackle optimization problems directly on the geometric level, where the role of logic is to control the constraints in the mathematical program. This approach optimizes a final configuration that is evaluated only by an objective function. In contrast, our approach aims to optimize plans to reach this final configuration. Another limitation of this approach is it is not as efficient as other TAMP approaches and specifically does not scale well with the number of tasks.

Kiesel et al. [15] formulates a multi-vehicle routing problem as a multi-level optimization: task sequencing and assignment (i.e., ordering waypoints and assigning them to vehicles), task scheduling, and vehicle routing for each

vehicle. They develop an integrated approach that uses Tabu search for task sequencing and assignment, linear programming for task scheduling, and A* search for routing. This approach aims to optimize the overall system performance and backtracks to the high-level problem when there is no feasible solution at the lower level. However, unlike our approach, it does not exploit the cost of motions computed by the low-level planning.

III. PROBLEM FORMULATION

To illustrate our problem formulation and integrated optimization approach, we take the pick-and-place domain as an example, where a robot with arm(s) and a mobile base operates in an environment consisting of objects on various surfaces. The robot's objective is to move all objects to their goal regions with minimum execution time. States for this problem consist of a robot joint configuration, along with, for each object, a description of its geometry, current position, and goal position. We can find many applications for this problem, such as tidying up a room by putting away objects in a set of boxes and delivering parts in a factory.

To deal with the complexity of this pick-and-place problem, we formulate this problem as a multi-level optimization problem. At the top level is task planning for sequencing pick-and-place operations. At the middle is action planning, which generates a sequence of actions to achieve a single pick-and-place operation. The bottom level is motion planning, which generates a sequence of motions to accomplish an action. The high-level optimization problem depends on solutions of lower-level problems, providing the existence or cost of feasible plans for particular higher-level tasks or actions. The following subsections will discuss each planning problem in greater detail and, in the next section, we will present an integrated approach for solving this multi-level problem.

A. Task Planning

Task planning determines the order of picking objects and placing them to their target position. We assume that every object needs one and only one pick-and-place operation and that not all objects have the same target position (otherwise, the problem becomes trivial). As the robot may end at different locations for different tasks, the cost of a pick-and-place task varies, depending on its previous task (if existing). In this paper, we also assume that the cost of a task only depends on its previous task. To model this dependency, we formulate this task planning as a traveling salesman problem (TSP).

TSP can be modeled as a weighted graph. Objects in our problem are the graph's vertices and there are two special vertices: *start* and *end*. All vertices are connected, forming a complete graph. The weight on an edge from a vertex v to an object vertex o is the cost of the pick-and-place operation for object o starting with the robot configuration ended up at vertex v. The weight on all edges to the special *end* vertex is zero. It is a minimization problem starting at the *start* vertex and finishing at the *end* vertex after having visited each other vertex exactly once. If no feasible pick-and-place operation exists between two vertices, an arbitrarily large weight will be added to their edge. Note that this TSP is asymmetric, because the cost of picking and placing different objects are often different. We employ the technique developed by [16] turning an asymmetric TSP into a symmetric TSP by duplicating vertices .

B. Action Planning

We represent pick-and-place operations for objects as vertically integrated hierarchical action networks (HANs). There are two types of actions in these HANs: primitive and macro. In this paper, the primitive actions move a specific part of the robot (base, arm, or gripper) from its current joint configuration to a target one. Each primitive action has a transition model that takes in a configuration and returns the successor configuration and action cost (or failure). We use a extreme large positive cost to represent the failure of an action. For primitive actions of moving the base or arm(s), their transition models are procedural, and call out to lower-level motion planners. Action costs are set as the costs of motion plans that implement them. A macro action contains a sequence of sub-actions, each of which can be any type of action. A feasible macro action means that all its sub-actions are feasible.

For each pick-and-place operation of object o, we use the following hierarchy. The top-level action `PickAndPlace(`o`)` refines to macro action `GoPick(`o`)`, and `GoPlace(`o`)`. Action `GoPick(`o`)` refines to primitive action `ArmTuck`, action `MoveBaseForPick(`o`,` $bpfg_o$`)`, where $bpfg_o$ is a symbolic reference to possible base poses for grasping object o, and `Pick(`o`)`. Similarly, Action `GoPlace(`o`)` refines to primitive action `ArmTuck`, macro action `MoveBaseForPutDown(`o`,` $bpfpd_o$`)`, where $bpfpd_o$ is a symbolic reference to possible base poses for putting down object o [17], and `Place(`o`)`. Macro action `Pick(`o`)` further refines to primitive actions `OpenGripper(`o`)`, `MoveManipulatorForPick(`o`,` $gpfg_o$`)`, where $gpfg_o$ is a symbolic reference to possible gripper poses for grasping object o, and `CloseGripper(`o`)`. Macro action `Place(`o`)` further refines to primitive actions `MoveManipulatorForPutDown(`o`,` $gpfpd_o$`)`, where $gpfpd_o$ is a symbolic reference to possible gripper poses for putting down object o, and `OpenGripper(`o`)`.

Primitive actions in a HAN are templates, whose variables will be instantiated during the planning process. As the pose space for grasping or putting down an object are continuous, geometric variables in actions for moving the base or the manipulator(s) range over continuous values. Our approach iteratively instantiates them through sampled values and invokes the motion planner to find a feasible plan for achieving an action. Possible poses sampled need to satisfy the pre-conditions of the next action. For example, a robot should move its base to a pose where it can pick up the target object.

C. Motion Planning

Each primitive action for moving the robot base or arm is corresponding to a motion planning problem, the solution of which implements the action. A motion planning problem is a tuple $\langle C, f, p_0, p_t \rangle$, where C is the space of possible configurations or poses of a robot, f is a Boolean function that determines whether or not a pose is in collision and $p_0, p_t \in C$ are the initial and final poses. A collision-free motion plan solving a motion planning problem is a trajectory in C from p_0 to p_t such that f does not hold for any pose in the trajectory. Motion planning algorithms use a variety of approaches for representing C and f efficiently. The cost of a motion plan is set based on the estimated time required, e.g., the length of the returned trajectory multiplied by a weighting constant.

IV. APPROACH

In this section, we will present an integrated approach for solving the multi-level optimization problem, which contains an algorithm for solving the planning problem at each level. The high-level algorithm exploits solutions of lower-level problems to improve its solution. In the following subsections, we will discuss these algorithms and their interactions.

A. Task Planning Algorithm

Our task planning algorithm takes the pick-and-place problem and the robot's initial configuration as input and returns a combined motion plan to achieve its goal. The basic idea is that this algorithm first builds a TSP for the pick-and-place problem by estimating the cost of each edge without invoking lower-level detailed planning, solves this TSP using an existing method to obtain the sequence of pick-and-place operations, and then iteratively refines each operation to motion plans. This refinement requires invoking lower-level planning, which returns a motion plan (if existing) and the associated cost. If the current cost estimate is quite different from the actual cost, a new solution for sequencing tasks will be computed. Therefore, our task planning algorithm not only takes into consideration the existence of a feasible motion plan, but also its actual cost.

Algorithm 1 describes the task planning algorithm. As will be discussed in the following motion planning section, we use pose generators to sample poses and instantiate the geometric parameters for primitive actions in order to refine a task plan to a motion plan. The outer *repeat* loop enables to try different random seeds for pose generators if necessary. The variable p_0 is used to keep the current robot configuration and *plan* stores the computed motion plan for achieving the pick-and-place objective. Line 4 builds an initial TSP. Its edge cost weights can be initialized by some heuristic. In our implementation, we estimate them by simply using the navigation cost without considering obstacles plus a fixed cost for arm manipulation. We use Lin–Kernighan 2-opt heuristic [18] to solve our TSP problem. To speed up the search, we start with an initial solution made by the nearest neighbor algorithm that chooses the nearest unvisited city as the next move. The solution always starts o_0, which is the special *start* vertex.

Algorithm 1 Task Planning Algorithm

Input: problem, initialPose, θ

```
1:  repeat
2:      p_0 ← initialPose
3:      plan ← ∅
4:      TSP ← build initial TSP for input problem
5:      ⟨o_0, o_1, ..., o_n⟩ ← Solve(TSP)
6:      done ← True
7:      repeat
8:          c̄ ← average task cost of solution ⟨o_0, o_1, ..., o_n⟩
9:          for i ← 1, n do
10:             if edge ⟨o_{i-1}, o_i⟩ has plan mp_i then
11:                 plan ← plan + mp_i
12:                 p_0 ← endingPoseOf(mp_i)
13:                 continue
14:             end if
15:             ap ← ActionPlanning(HAN(o_i))
16:             ⟨c_i, mp_i, p_i⟩ ← RefineToMotionPlan(p_0, ap)
17:             plan ← plan + mp_i
18:             p_0 ← p_i
19:             e ← |c_i − cost(o_{i-1}, o_i)| / c̄
20:             TSP ← update the cost for edge ⟨o_{i-1}, o_i⟩ in TSP
21:             Store motion plan mp_i on edge ⟨o_{i-1}, o_i⟩ in TSP
22:             if e > θ then
23:                 done ← False
24:                 ⟨o_0, ..., o_n⟩ ← ImproveSol(TSP, o_{0:n}, ⟨o_{i-1}, o_i⟩)
25:                 p_0 ← initialPose
26:                 plan ← ∅
27:                 break
28:             else
29:                 done ← True
30:             end if
31:         end for
32:     until done or c̄ * n > M
33:     if done then return plan
34:     else
35:         Reset all PoseGenerators with new random seeds
36:     end if
37: until MaxTrials reached
```

The inner *repeat* loop of Algorithm 1 will not terminate until all tasks refine to motion plans or no task sequence is found for such refinements. The average cost of edges in TSP computed by Line 8 is used to evaluate how the estimated cost deviates from the actual cost, which is done by Line 19. The *for* loop of Line 9–31 is to iteratively refine each high-level task to a motion plan and then concatenate them into a single motion plan. Line 15 invokes the lower-level action planning algorithm and its returned action plan will be refined by Line 16 to a motion plan mp_i with its associated cost c_i and the finishing configuration p_i for picking and placing object o_i. We can store computed results to the edge, which can be used later (Line 10–14). Line 22 checks if the estimated cost deviates greater than given threshold θ. If it does, then a new solution is computed with the updated TSP and the process of motion plan refinement is restarted.

Observe that the only difference between the updated TSP and the old TSP is the weight of one edge. We hypothesize that their solutions may not be very different. Therefore, it may not be efficient to use Lin–Kernighan 2-opt algorithm

Algorithm 2 Greedy 2-Opt Algorithm

```
1: procedure IMPROVESOL(TSP, existingSolution, ⟨o_{i−1}, o_i⟩)
2:     affectedList ← {o_i}
3:     repeat
4:         minCost ← calculateTotalCost(existingSolution)
5:         improved ← False
6:         for v ∈ affectedList do
7:             i ← existingSolution.indexOf(v)
8:             for k ← i+1, length(existingSolution) do
9:                 newSolution ← pairSwap(existingSolution, i, k)
10:                newCost ← calculateTotalCost(newSolution)
11:                if newCost < minCost then
12:                    affectedList.add(existingSolution[i+1 : k])
13:                    existingSolution ← newSolution
14:                    improved ← True
15:                end if
16:            end for
17:            if improved then
18:                break
19:            end if
20:        end for
21:    until no improvement is made
22:    return existingSolution
23: end procedure
24: procedure PAIRSWAP(existingSolution, i, k)
25:    n ← length(existingSolution)
26:    newSolution ← existingSolution[1 : i−1]
27:    newSolution.append(reverse(existingSolution[i : k]))
28:    newSolution.append(existingSolution[k+1 : n])
29:    return newSolution
30: end procedure
```

to compute the new solution, which repeatedly searches for one improvement from all pairwise swaps of vertices. We develop a greedy 2-opt algorithm that keeps a list of affected vertices. This list of vertices can be either directly or indirectly (because of swapping operations) affected by the new actual cost updated on the TSP. This greedy algorithm only searches for an improvement by swapping a vertex from this list to any vertex after it in the old task sequence solution.

Algorithm 2 shows in detail our greedy 2-opt algorithm. The affected vertex list is started with the object, the pick-and-place cost of which is just updated. The *for* loop of Line 6–20 searches for an solution improvement by swapping a vertex pair, which goes through from a vertex in the affected list to any vertex after it in the current solution. If such an improvement is found, the current solution is updated, and swapped vertices and all vertices between them are added to the affected list (Line 12). For the Lin–Kernighan 2-opt algorithm, the loop at Line 6 will goes through all vertices in the TSP, which can dramatically increase the search iterations.

B. Action Planning Algorithm

The action planning algorithm is invoked by the higher-level task planner to refine a pick-and-place operation. As shown in Algorithm 3, it takes as input a HAN for picking and placing an object as well as the robot initial configuration and returns the action plan for implementing this high-level operation.

Algorithm 3 Action Planning Algorithm

```
1: procedure ACTIONPLANNING(a)
2:     if a is a macro action then
3:         plan ← ∅
4:         for a′ ∈ REFINEMENT(a) do
5:             plan ← plan+ ActionPlanning(a′)
6:         end for
7:         return plan
8:     else
9:         return a
10:    end if
11: end procedure
```

Algorithm 4 RefineToMotionPlan Procedure

Input: initialPose, actionPlan

```
1: p_1 ← initialPose
2: motionPlan ← ∅
3: c ← M
4: i ← 1
5: while 0 ≤ i < length(actionPlan) − 1 do
6:     action ← actionPlan[i]
7:     nextAction ← actionPlan[i+1]
8:     mpFailed ← True
9:     while PoseGenerator(nextAction).hasNext() and mpFailed
       and p_1 is well defined do
10:        p_2 ← PoseGenerator(nextAction).next()
11:        mp ← MotionPlanning(p_1, p_2)
12:        if mp is valid then
13:            motionPlan ← motionPlan + mp
14:            p_1 ← p_2
15:            i ← i+1
16:            mpFailed ← False
17:        end if
18:    end while
19:    if mpFailed then
20:        i ← i−1
21:        motionPlan.removeMPFor(action)
22:        p_1 ← PoseGenerator(action).next()
23:        PoseGenerator(nextAction).reset()
24:    end if
25: end while
26: if i ≥ 0 then
27:    c ← cost(motionPlan)
28: end if
29: return ⟨c, motionPlan, p_1⟩
```

The hierarchical representation of a pick-and-place operation both structures and potentially restricts the space of solutions. In particular, rather than searching directly over primitive action sequences, an agent can begin with a plan consisting of just `PickAndPlace(o)`, and repeatedly replace the first non-primitive action in this plan with an immediate refinement until a primitive solution is found. We assume WLOG that the hierarchy does not generate primitive non-solutions.

C. Refining Action Plans into Motion Plans

We assume without loss of generality that all action plans are zero-indexed lists ending with a dummy action whose precondition is the final desired pose configuration. The *R*efineToMotionPlan subroutine is invoked to refine a primitive action plan to a motion plan. As shown in Algorithm 4, it

takes as input the initial pose and an action plan, carries out an interative, backtracking search for a feasible refinement, and returns a collision-free motion plan, if found, associated with its cost and the ending pose.

The *R*efineToMotionPlan subroutine iteratively refines actions to motion plans, starting with the input of the initial pose. Similar to the work by Srivastava et al. [1], to refine an primitive action with a geometric parameter, it will invoke a pose generator to get a possible target pose. We define the pose generator on the next action, because the target pose of an action should satisfies the geometric precondition of the next action. If the pose generator has a target pose available, the algorithm will then invoke an existing motion planning algorithm (e.g., RRT [2], RRT* [3], and PRM [4]) to compute a collision-free motion plan for achieving the current action. For efficiency, the motion planner is invoked for a possible target pose only if IK solutions exist. If the pose generator of an action runs out of possible poses before finding a feasible motion plan, the algorithm backtracks (Line 20 – 23). If the algorithm fails to find a feasible refinement, it will return an empty motion plan with an extreme large cost M.

For our pick-and-place domain, we implemented four pose generators for instantiating geometric parameters of actions `MoveBaseForPick(`o`,` $bpfg_o$`)`, `MoveBaseForPutDown(`o`,` $bpfpd_o$`)`, `MoveManipulatorForPick(`o`,` $gpfg_o$`)`, and `MoveManipulatorForPutDown(`o`,` $gpfpd_o$`)`. These pose generators iterate over a finite set of randomly sampled values that are likely to satisfy the preconditions of their next actions. For the target pose $bpfpd_o$ of moving the base to pick up object o, its pose generator samples base poses oriented towards object o within the range where the robot arm can reach the object. For the target pose $gpfpd_o$ of moving the base to put down object 0, its pose generator samples base poses oriented towards the target location of object o within the range where the robot arm can reach the target location. For optimization, these possible poses for moving the base are ordered by the distance from the starting pose. For the target pose $gpfg_o$ of the gripper to grasp object o, its pose generator samples poses at which closing the gripper will result in a stable grasp of the object. As computing effective grasping poses is an independent problem [19], [20], in this paper, we assume such poses are known for each object. For the target pose $gpfpd_o$ of the gripper to put down object o, its pose generator samples possible grasping poses as if the object is at its target location.

V. RESULTS

We evaluate our integrated approach in the pick-and-place domain, an example of which is shown in Figure 1. It is assumed that the robot has a full observability of the environment. We compare our approach with two other algorithms. The top-down approach (TD) solves the task planning problem without taking solutions of lower-problems into consideration. Its task planning algorithm is like Algorithm 1 with Line 20–26 removed. The top-down approach with a heuristic (TD-H) is like TD except that it tries to avoid obstacles on the table and always prefers moving the robot base to a closer point to a target object for picking. We set parameter $\theta = 0.2$ for task planning. In all experiments, the numbers of red and blue cubes are equal.

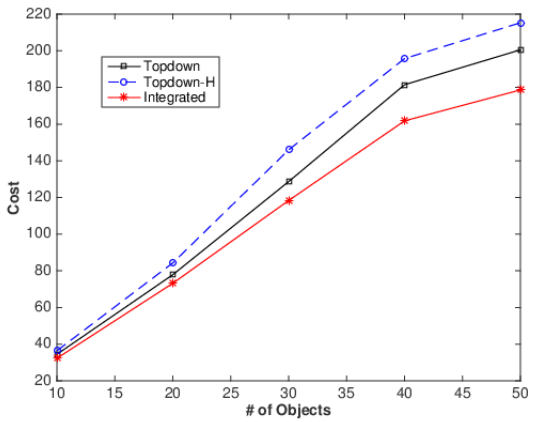

Fig. 2. Solution quality (average over 20 runs) for algorithms with *one* obstacle on the table, as a function of the number of objects

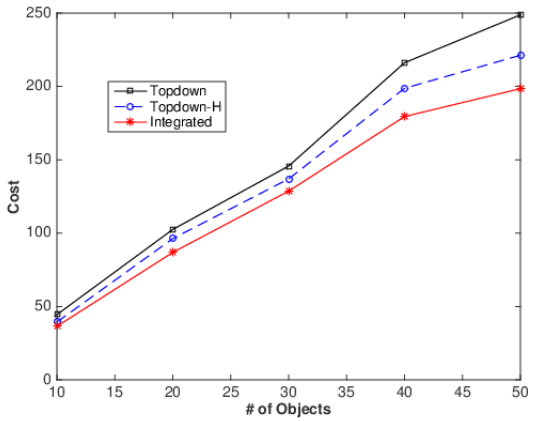

Fig. 3. Solution quality (average over 20 runs) for algorithms with *two* obstacles on the table, as a function of the number of objects

Figure 2 and 3 shows solution quality of algorithms over various objects in two scenarios. The first scenario has only one obstacle block on the table, while the second one has two obstacle blocks. In both cases, our integrated approach significantly outperforms other two approaches. Comparing to the top-down approach, our approach reduces the average cost by 10.2% and 17.8% in the first and second scenarios, respectively. The more the obstacles, the greater the reduction in the solution cost. The reduction percentage varies a lot from one problem instance to another, which can be more than 40% when many cubes are blocked and can also be 0% when no cubes are blocks. We can also observe that TD performs better than TD-H in the first scenario, but worse in the second scenario. This is because, in the first scenario, cubes are less likely to be blocked and can also be grasped

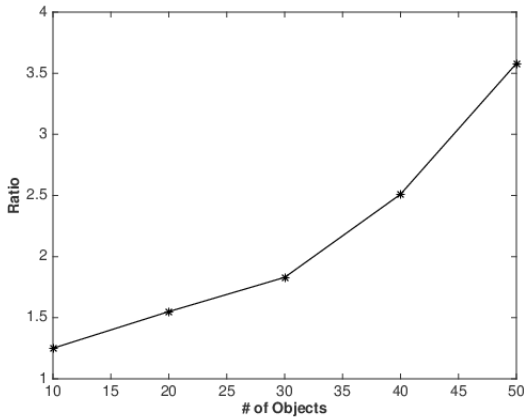

Fig. 4. Ratio of average runtimes from the regular 2-opt algorithm to our greedy 2-opt algorithm, as a function of the number of objects

from the alternate side of the table, while in the second scenario, cubes are more likely to be blocked and must be grasped from one side of the table.

In our experiments discussed above, we set a minimum distance between cubes so that the robot always has a feasible pose for grasping objects on the table. As a result, the TD approach performs like satisfaction-based TAMP approaches [11], [1], always generating feasible motion plans in these experiments. If we do not set such a minimum distance between objects and treat movable objects as obstacles, then top-down approaches are not able to generate a feasible task sequence for some problems (e.g., where a red cube is surrounded by blue cubes closer to the red basket). In contrast, our integrated approach finds an efficient task sequence that refines to motion plan for achieving the goal configuration.

In our experiments, we also evaluate the greedy 2-opt algorithm against the Lin–Kernighan 2-opt algorithm. Our integrated approach yields similar solution costs with both algorithm. Figure 4 shows the average runtime ratio of Lin–Kernighan 2-opt algorithm to our algorithm. The ratio is greater than one, indicating our greedy is faster than the regular 2-opt algorithm. As we can see from the Figure, our algorithm significantly outperforms, more than three times faster than the regular 2-opt algorithm in the case with 50 objects.

VI. CONCLUSIONS

In this paper, we formulate a multi-level optimization problem that integrates task, action, and motion planning and models their dependency. This multi-level formulation structures and restricts the search for executable motion plans in large robotic manipulation problems. We developed an integrated planner for this multi-level optimization problem, which takes into consideration interactions between problems at different levels and exploits both the existence and cost information about lower-level solutions to improve the high-level plan. Empirical results verify the importance of such consideration and exploitation.

REFERENCES

[1] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 639–646.

[2] S. M. LaValle, "Rapidly-exploring random trees a ew tool for path planning," *Technical report, Department of Computer Science, Iowa State University*, 1998.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[4] V. Boor, M. H. Overmars, V. der Stappen, and A. Frank, "The gaussian sampling strategy for probabilistic roadmap planners," in *Robotics and automation, 1999. proceedings. 1999 ieee international conference on*, vol. 2. IEEE, 1999, pp. 1018–1023.

[5] C. Dornhege, M. Gissler, M. Teschner, and B. Nebel, "Integrating symbolic and geometric planning for mobile manipulation," in *Safety, Security & Rescue Robotics (SSRR), 2009 IEEE International Workshop on*. IEEE, 2009, pp. 1–6.

[6] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Ffrob: An efficient heuristic for task and motion planning," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 179–195.

[7] J. Wolfe, B. Marthi, and S. J. Russell, "Combined task and motion planning for mobile manipulation." in *ICAPS*, 2010, pp. 254–258.

[8] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1470–1477.

[9] A. K. Pandey, J.-P. Saut, D. Sidobre, and R. Alami, "Towards planning human-robot interactive manipulation tasks: Task dependent and human oriented autonomous selection of grasp and placement," in *Biomedical Robotics and Biomechatronics (BioRob), 2012 4th IEEE RAS & EMBS International Conference on*. IEEE, 2012, pp. 1371–1376.

[10] L. de Silva, A. K. Pandey, M. Gharbi, and R. Alami, "Towards combining htn planning and geometric task planning," *arXiv preprint arXiv:1307.1482*, 2013.

[11] 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*. IEEE, 2014, pp. 3684–3691.

[12] 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*. IEEE, 2012, pp. 957–964.

[13] 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*, p. 0278364914545811, 2014.

[14] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *IJCAI*, 2015.

[15] S. Kiesel, E. Burns, C. M. Wilt, and W. Ruml, "Integrating vehicle routing and motion planning." in *ICAPS*. AAAI, 2012.

[16] R. Jonker and T. Volgenant, "Transforming asymmetric into symmetric traveling salesman problems," *Operations Research Letters*, vol. 2, no. 4, pp. 161–163, 1983.

[17] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *The International Journal of Robotics Research*, vol. 28, no. 1, pp. 104–126, 2009.

[18] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations research*, vol. 21, no. 2, pp. 498–516, 1973.

[19] C. Goldfeder and P. K. Allen, "Data-driven grasping," *Autonomous Robots*, vol. 31, no. 1, pp. 1–20, 2011.

[20] M. Dogar and S. Srinivasa, "A framework for push-grasping in clutter," *Robotics: Science and systems VII*, 2011.