Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 2334236, 16 pages
https://doi.org/10.1155/2021/2334236

Research Article

Constraint Satisfaction for Motion Feasibility Checking

Seokjun Lee and Incheol Kim

Department of Computer Science, Kyonggi University, San 94-6, Yiui-Dong, Youngtong-Gu, Suwon-Si 443-760, Republic of Korea

Correspondence should be addressed to Incheol Kim; kic@kyonggi.ac.kr

Received 9 April 2021; Revised 9 May 2021; Accepted 18 May 2021; Published 27 May 2021

Academic Editor: Zain Anwar Ali

Task and motion planning (TAMP) is a key research field for robotic manipulation tasks. The goal of TAMP is to generate motion-feasible task plan automatically. Existing methods for checking motion feasibility of task plan skeletons have some limitations of semantic-free pose candidate sampling, weak search heuristics, and early value commitment. In order to overcome these limitations, we propose a novel constraint satisfaction framework for checking motion feasibility of task plan skeletons. Our framework provides (1) a semantic pose candidate sampling method, (2) novel variable and constraint ordering heuristics based on intra- and inter-action dependencies in a task plan skeleton, and (3) an efficient search strategy using constraint propagation. Based upon these techniques, our framework can improve the efficiency of motion feasibility checking for TAMP. From experiments using the humanoid robot PR2, we show that the motion feasibility checking in our framework is 1.4x to 6.0x faster than previous ones.

1. Introduction

In recent years, Artificial Intelligence (AI) has become an increasingly common presence in robotic solutions, introducing flexibility and learning capabilities in previously rigid applications. In this study, we address the issues in applying AI constraint satisfaction problem (CSP) solving techniques to task and motion planning (TAMP) [1–37], which is a key research field of robotic manipulation tasks. A typical TAMP involves the combination of task planning [38–40] that generates a sequence of actions for satisfying a goal condition in the current state based on a symbolic abstract action model and motion planning [41–43] that checks the motion feasibility for determining if the motions can be executed in a physical space. The ultimate goal of TAMP is to combine these to generate a motion-feasible task plan.

Task planning determines the logical order of actions. It knows symbolic knowledge such as type of action, parameters, preconditions, and effects. However, in task planning, the goal pose (or configuration), path (or trajectory), etc., to execute each action are unknown. This disadvantage can be supplemented through motion planning. While abstract symbolic knowledge is unknown in motion planning, it can generate collision-free paths for executing an action in a physical space. To combine the two planning methods, which have different advantages and disadvantages, an intermediate (interface) layer is required. Figure 1 shows an example of combining task and motion planning to generate a motion-feasible task plan. The left side of Figure 1 shows the task layer including the task planning process.

The right side of Figure 1 shows the motion layer including the motion planning process. With the help of the task layer, the interface layer generates a task plan, that is, a task plan skeleton that includes unbound pose parameters. Once the task plan skeleton is generated, the interface layer generates candidates of pose parameters of each action. As motion planning cannot generate the goal pose on its own, the interface layer must play a role in generating candidates of pose parameters. The interface layer then checks if there is a collision-free path between candidates of neighboring pose parameters with the help of the motion layer. If all of collision-free paths exist, then the candidates are assigned to the pose parameters of the task plan skeleton to generate a motion-feasible task plan. Otherwise, it creates a new task planning problem from motion-related errors (e.g., obstacles), and repeats the same interfacing process.

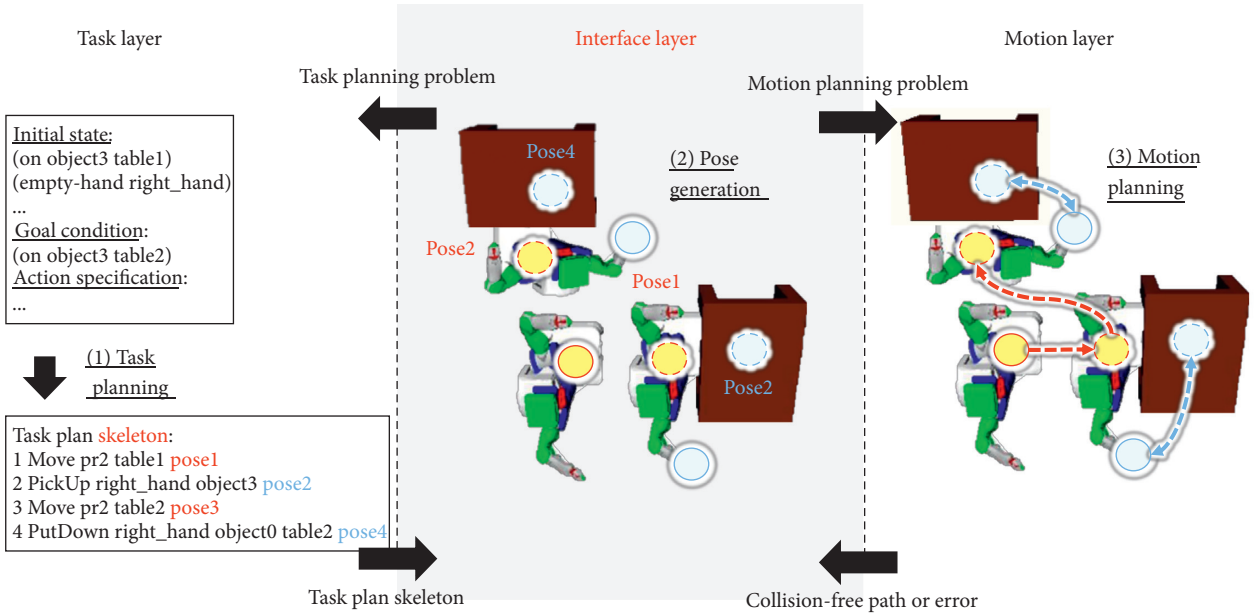


Figure 1: An example of combining task and motion planning using the interface layer.

This paper focuses on checking the motion feasibility of the task plan skeleton in the interface layer. Motion feasibility checking involves finding motion-feasible values of unbound pose parameters included in the task plan skeleton. This is a challenging problem with a very complex search space. It expands the motion planning problem that finds the collision-free path of a single behavior to the problem of validating the pose candidates and collision-free paths of multiple actions. To solve this problem effectively, the followings are required: (1) theoretical modelling of the search problem to find motion-feasible values of pose parameters, (2) a method for generating candidates of the value of pose parameters in continuous space, and (3) search strategies and heuristics (e.g., using constraints that the pose parameters must satisfy).

The previous study of Lozano-Pérez and Kaelbling [18] attempted to model and to solve the motion feasibility checking problem as a traditional CSP (Constraint Satisfaction Problem) [44, 45]. Garrett et al. [37] attempted to model and to solve the motion feasibility checking problem as on-the-fly CSP. Lagriffoul et al. [17] attempted to model and to solve the motion feasibility checking problem as a general search tree. These studies have limitations, including generating pose candidates with a high probability of failure due to simple sampling methods, or developing search strategies based on only general-purpose heuristics that is not suitable for motion feasibility checking. Furthermore, they attempted to determine values in advance that must be redetermined at the time of execution, such as IK (Inverse Kinematics) and collision-free path with high variability. It complicates the search problem and makes planning and execution inefficient in the long run.

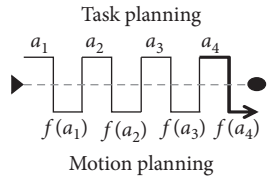
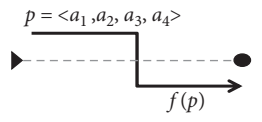
This paper proposes an efficient constraint satisfaction framework to address the motion feasibility checking problem. Motion feasibility checking is the problem that finds a valid (or motion-feasible) value of pose parameters. The valid value means that it satisfies constraints such as kinematics and collision-free path. In our framework, the motion feasibility checking is modelled as a constraint satisfaction problem (CSP). The CSP formulation has the advantage that the various types of constraints can be represented equally in a concise form. In addition, effective general-purpose or special-purpose search strategies and heuristics can be used to solve both discrete and continuous constraint satisfaction problems. (1) Our framework proposes a method of generating pose candidates guided by the semantics of unbound pose parameters and already bound task parameters. This method reduces the search space substantially. Meanwhile, actions belonging to the task plan skeleton have inter-action dependencies because of logical order. In addition, parameters and preconditions within an action have intra-action dependencies. (2) Based on these inter- and intra-action dependencies, novel, variable, and constraint ordering heuristics are proposed in our framework. (3) Moreover, our framework employs an efficient search strategy with constraint propagation [44] to detect failure early or speed up the search substantially. To verify the practical applicability and performance of the proposed framework, this study performs various experiments using humanoid robots that can perform mobile manipulation.

The rest of the paper is organized as follows. Section 2 introduces the related works in the field of TAMP. Section 3 provides the problem statement. In Section 4, the proposed framework with CSP modelling and heuristic search methods is presented in detail. In Section 5, experimental results are reported. Finally, Section 6 summarizes our work and discusses some limitations and future works.

2. State of Art

We categorize previous works according to their motion feasibility checking methods, as shown in Table 1. There are two different methods for checking motion feasibility. The

Table 1: Categorization of previous works.

Type	Illustration
Eager [1–14, 37]	
Lazy [15–37]	

eager method [1–14] alternately performs task planning and motion-feasibility checking step-by-step. On the contrary, the lazy method [15–37] postpones its motion-feasibility checking until a complete skeleton of task plan is built. The eager method has the advantage of being able to identify in advance future motion-unfeasible actions that the task planner cannot identify during the task planning process. However, the cost of generating the motion plan is high because motion planning is continuously interleaved during task planning. Although the lazy method cannot identify in advance the motion feasibility of the actions during task planning, it limits the search space of the motion planning to a small range since the task plan to achieve the goal condition is already defined. In addition, it examines different knowledge contained in the task plan skeleton and applies a wide range of search strategies and heuristics. This paper utilizes the lazy method.

Especially, Lozano-Pérez and Kaelbling [18], Garrett et al. [37], and Lagriffoul et al. [17] are closely related to our study, because they focus on motion feasibility checking of task plan skeletons using constraint satisfaction methods. First, Lozano-Pérez and Kaelbling [18] attempted to model the motion feasibility checking problem as a traditional CSP. In [18], the depth-first backtracking search algorithm and constraint propagation considering the dependency between constraints are used. However, [18] used only general-purpose variable ordering heuristics such as MRV (Minimum Remaining Value), which did not consider some dependencies between actions and in an action. Furthermore, [18] generated pose candidates with high probability of failure and dealt with only the parameters and constraints related to the manipulation actions.

Next, Garrett et al. [37] attempted to model the motion feasibility checking problem as on-the-fly CSP. In particular, [37] proposed a conditional sampling method based on the constraint network. However, the CSP modelled by [37] had some variables that cause high cost of motion planning, such as IK and trajectories. It makes the search problem more complex. The method of [37] is also less scalable because it only considers general-purpose CSP heuristics.

Finally, Lagriffoul et al. [17] attempted to model the motion feasibility checking problem as a general search tree. The nodes of the tree represent pose parameters, and the edges represent collision-free trajectories. Based on the depth-first backtracking search, it finds the values of pose parameters with verified validity for the trajectory. In particular, [17] attempted to reduce the number of backtracks by propagating linear constraints before visiting the next nodes. It is the same as forward checking in CSP. However, search tree does not represent all constraints in a unified form, and its simple sampling method generates many samples with a high probability of failure. In addition, [17] dealt only with the parameters and constraints associated with manipulation actions.

In order to overcome the limitations of these existing studies, our framework provides (1) a semantic pose candidate sampling method, (2) novel variable and constraint ordering heuristics based on intra- and inter-action dependencies in a task plan skeleton, and (3) an efficient search strategy using constraint propagation. Based upon these techniques, our framework can improve the efficiency of motion feasibility checking for TAMP.

In addition, there are several recent notable works on TAMP (Task and Motion Planning). The authors of [32] proposed a TAMP framework using a top-k skeleton planner to produce diverse skeletons, guaranteeing that no better solution exists under a current domain description. Moreover, the framework uses a Monte-Carlo Tree Search (MCTS) to solve this stochastic decision-making problem over skeletons and concrete bindings of the action parameters. The authors of [33] proposed a novel online planning and execution system to solve a TAMP problem as a hybrid partially observable Markov decision process (POMDP) and use past plans to constrain the structure of solutions for the current planning problem. On the other hand, both [34, 35] addressed multi-agent or multi-robot TAMP problems. The authors of [34] dealt with a new problem of motion planning feasibility checking for task-agent assignment to perform complex tasks using multi-arm mobile manipulators. The authors of [35] presented a multi-robot integrated task and motion planning method capable of handling sequential subtasks dependencies within multiple decomposable tasks. Interestingly, [36] proposed a TAMP method for efficient and safe autonomous urban driving, different from robotic manipulation tasks.

3. Problem Statement

We propose an efficient constraint satisfaction framework to check the motion feasibility of the task plan skeleton. We assume that a task plan skeleton has already been generated. Then, we focus on checking its motion feasibility.

Figure 2 illustrates an example environment including a humanoid robot with two arms (e.g., PR2) to manipulate objects on tables. In Figure 2, PR2 must move the object from one table to the other.

We remark the task plan skeleton as follows:

Remark 1 (task plan skeleton). A task plan skeleton is a sequence of actions $<a0, \ldots, ak>$ following the states transition $<s0, \ldots, sk+1>$, where $a \in A$ is an action, s0 the

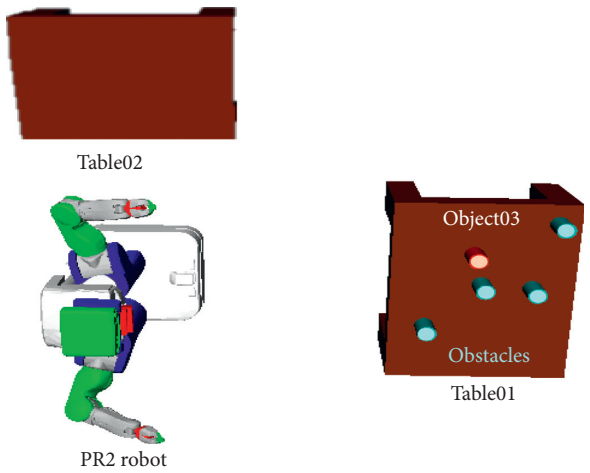


Figure 2: Task environment for mobile manipulation.

initial state, and sk+1 the goal state, which satisfies the goal condition. Meanwhile, each action a has pose references.

The pose references are discrete identifiers for representing the values of the poses still undetermined in the task planning phase. The pose references are regarded to have values that satisfy the preconditions of the actions. To generate the task plan skeleton, the pose references are temporarily bound to the pose parameters.

Table 2 shows an example of a task plan skeleton generated from the task environment in Figure 2. This task plan skeleton consists of a sequence of actions in which the robot moves *object*03 to *table* 02.

The actions that comprise the task plan skeleton are specified according to the schema below.

Remark 2 (action schema). Action schema is a tuple (α (Pt, Pp), pre (Pt, Pp, V), eff (Pt, Pp, V)), where α is an action symbol; Pt is a set of task-level parameters such as object, robot, etc.; Pp is a set of pose parameters; pre (Pt, Pp, V) is a conjunction of predicates representing preconditions of action (where V are additional variables, $V \neq Pt$, Pp); and eff (Pt, Pp, V) is a conjunction of predicates representing effects of action.

Table 3 shows the specifications of the actions for navigation. In the parameters for each action, there are numerical parameters that were not in the previous action specification. For example, *baseLoc0* and baseLoc1 are pose parameters, where *baseLoc0* is the robot's current location and baseLoc1 the goal location. During the generation of the task plan skeleton, these parameters will be bound to pose references. Based on the preconditions of the action, the pose parameters must satisfy the state conditions at the motion level. For example, the baseLoc1 parameter of *MoveBasePickUp* must satisfy state conditions such as *reachableBasePose* and *validBasePath*. Section 4 details the specific meanings of these state conditions.

Table 4 shows the specifications of the actions for manipulation. First, *PickUpGeneral* is the action of moving a hand to grasp an object and then returning the hand back to its initial pose. This action contains many motions, thus requiring a relatively large number of pose parameters. This action includes the initial hand pose (*handPose0*), the pregrasping hand pose (*handPose*1), the grasping hand pose (*handPose*2), and the post-grasping hand pose (*handPose*3). *PutDownGeneral* is the action of putting the grasped object on a support plane such as a table and returning the hand to the initial pose. This action has pose parameters similar to the *PickUpGeneral* action. However, the *PickUpGeneral* action includes a pose parameter for the placement position of the object. Section 4 details the specific meanings of these state conditions.

Under the assumptions described above, we summarize the main problem for checking the motion feasibility of the task plan skeleton as follows.

Remark 3 (motion feasibility checking). Given a task plan skeleton S, the problem is to find a sequence of valid values <v1, . . ., vk> of pose references in S. The valid values <v1, . . ., vk> satisfy preconditions of corresponding action.

We attempt to model and solve this motion feasibility checking problem by converting it to a CSP [32, 33]. CSP refers to the problem of finding a value that satisfies a plurality of constraints within a domain. The valid values of the undetermined pose parameters must satisfy the state conditions presented in the task-planning problem so that they can be fully formulated as a CSP. As poses exist in a continuous space, discrete domains are created to find a solution in a reasonable amount of time. Thus, the CSP considered in this paper is a discrete CSP. Now, we summarize subproblems as follows:

Remark 4 (constructing discrete CSP). Given a task plan skeleton S, the problem is to construct a discrete CSP csp = (V, D, C) from the task plan skeleton, where V is a set of variables, D is a set of the respective domains of values, and C is a set of constraints.

Remark 5 (solving CSP). Given a discrete CSP csp, the problem is to search valid values of each variable in V.

4. Proposed Solution

The task plan skeleton contains unbound pose references. Values for the pose references must satisfy the motion-related constraints, such as IK and the collision-free path. If any value satisfying all constraints exists, then we can consider that the task plan skeleton is motion feasible. This is a constraint-based search problem. So, we model the problem for checking the motion feasibility as a CSP.

Figure 3 describes the process of motion feasibility checking. First, it generates CSP statements from a task plan skeleton. Second, it reduces the size of domain through preprocessing before solving CSP. Last, it finds a solution and decides if the task plan skeleton is motion feasible or not.

4.1. Formulating Constraint Satisfaction Problem from Plan Skeleton

4.1.1. Variables. First, we have to formulate the motion feasibility checking problem into a constraint satisfaction

Table 2: An example of a task plan skeleton to move an object to *table02* from *table01*.

1. (MOVEBASEFORPICKCUP pr2 object03 B_{init} B_{reach})
2. (PICKUPGENERAL rgirpper object03 H_{init} H_{pre} H_{grasp} H_{post})
3. (MOVEBASEFORPUTDOWN pr2 rgripper object03 table02 B_{reach} B_{place})
4. (PUTDOWNGENERAL rgirpper object03 table02 O_{place} H_{init} H_{low} H_{place} H_{ret})

Table 3: Action specifications for navigation.

action MoveBaseForPickUp
param robot, obj, baseLoc0, baseLoc1
precond robotAt (baseLoc0), objectAt (obj, objLoc), reachableBasePose (robot, obj, objLoc, baseLoc1), validBasePath (robot, baseLoc0, baseLoc1)
effect—robotAt (baseLoc0), robotAt (baseLoc1)
action MoveBaseForPutDown
param robot, hand, obj, plane, baseLoc0, baseLoc1
precond robotAt (baseLoc0), graspedBy (obj, hand), placeableBasePose (robot, obj, plane, baseLoc1), validBasePath (robot, baseLoc0, baseLoc1)
effect—robotAt (baseLoc0), robotAt (baseLoc1)

Table 4: Action specifications for manipulation.

action PickUpGeneral
param obj, hand, handPose0, handPose1, handPose2, handPose3
precond empty (hand), robotAt (baseLoc), objectAt (obj, objLoc), handAt (hand, handPose0), graspableHandPose (hand, obj, objLoc, baseLoc, handPose2), preGraspableHandPose (hand, obj, objLoc, baseLoc, handPose2, handPose1), poseGraspableHandPose (hand, obj, objLoc, baseLoc, handPose2, handPose3), validHandPath (hand, baseLoc, handPose0, handPose1), validHandPath (hand, baseLoc, handPose1, handPose2), validHandPath (hand, baseLoc, handPose2, handPose3), validHandPath (hand, baseLoc, handPose3, handPose0)
effect graspedBy (obj, gripper), —empty (gripper), —objectAt (obj, objLoc)
action PutDownGeneral
param obj, hand, plane, objLoc, handPose0, handPose1, handPose2, handPose3
precond graspedBy (obj, hand), robotAt (baseLoc), handAt (hand, handPose0), placeableObjectLocation (obj, plane, objLoc), lowerableHandPose (hand, obj, objLoc, baseLoc, handPose2), placeableHandPose (hand, obj, objLoc, baseLoc, handPose2, handPose1), retractableHandPose (hand, obj, objLoc, baseLoc, handPose2, handPose3), validHandPath (hand, baseLoc, handPose0, handPose1), validHandPath (hand, baseLoc, handPose1, handPose2), validHandPath (hand, baseLoc, handPose2, handPose3), validHandPath (hand, baseLoc, handPose3, handPose0)
effect—graspedBy (obj, gripper), objectAt (obj, objLoc)

problem (CSP). Usually, goal pose, grasp, IK, and path were modelled as CSP variables in a way a similar to that in [17, 18, 37]. But, we only model the goal pose to the CSP variable, that is, modelling the path to the CSP variable. The reason for this is as follows: (1) If too many variables are modelled, it increases the complexity of search. In addition, the domain of variable (for path especially) is very difficult and expensive. (2) To bind the value of anything other than the pose is not required necessarily in the planning phase. Considering changes of the environment during planning time, even if the IK, paths, etc., are determined at the planning phase, rechecking is required at the execution phase. It is appropriate that these are assigned once at the execution phase.

We categorize the poses into three types. The types include position of mobile base B, pose of hand H, and position of object O. The variables are extracted from the pose references of the actions the task plan skeleton. Table 2 is a mapping table of the abovementioned variables. As shown in Table 5, the pose references bound to each action are mapped as variables of the CSP. However, since the initial poses of mobile base, hand, and object are constants that can be read directly from the environment, these pose references are not mapped to variables of the CSP. In addition, a pose reference that has already been mapped as B_{reach} is not mapped in duplicate.

4.1.2. Domains. Each variable is represented by a 4×4 homogeneous matrix in continuous space. For execution, a discretized deterministic value must be bound to each variable. For this reason, we generate discrete domains based on the sampling guided by the semantic of the task plan skeleton. To generate the discrete domains, the geometric constants are retrieved from the value of the bounded parameter such as mobile base, hand, object, and support plane. For example, referring the value of bounded parameters, we can know that the domain of B_{reach} should be near the object and the orientation of B_{reach} should be towards the object. This can be represented by the following:

$$B_{\text{reach}} = (r_2 \cdot t_0 \cdot r_1 \cdot r_0) O_{\text{init}}, \tag{1}$$

where r_0 is a matrix representing an axis angle $a = (0, 0, 0)$, r_1 is a matrix representing a quaternion $q = (0, 0, 0, 1)$, t_0 is a matrix representing a pose $p = (1, 0, 0, 0, d, 0, 0)$, and r_2 is a matrix representing an axis angle $a = (0, 0, \theta)$. The details of the equations that derive the matrix from the axis angle, quaternion, etc., are provided in the appendix.

Equation (1) is to calculate the relative position and orientation of the mobile base relative to the coordinate system of the object. This equation has two parameters: d and θ, where d indicates the distance between the center of the robot and the center of the object, whereas θ indicates the direction of the robot towards the object. d can be appropriately determined in consideration of the arm length. θ is calculated by equation (2):

$$\theta = \frac{2i\pi}{n}, \quad \text{where } i = 1, \ldots, n, i, \in N,\ n \in N. \tag{2}$$

If the number of d is m, then the number of domains is mn.

Figure 4 shows the area of the domain of B_{reach}. In Figure 4(a), the circular dotted line that maintains the

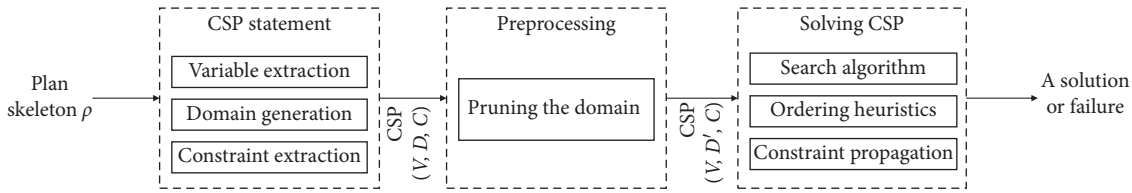


Figure 3: Motion feasibility checking process.

Table 5: Mapping pose references to CSP variables.

Action	Bounded parameters	Pose references	CSP variables
MoveBase *ForPickUp*	pr2, objeect03	O_{init}, B_{init}, B_{reach}	B_{reach}
PickUpGeneral	pr2, rgripper, object03	O_{init}, H_{init}, H_{pre}, H_{grasp},	H_{pre}, H_{grasp}, H_{post}
MoveBase *ForPutDown*	pr2, object03, table02	B_{reach}, B_{place}	B_{place}
PutDownGeneral	pr2, rgripper, object03 table02	O_{place}, H_{init}, H_{low}, H_{place}, H_{ret}	H_{low}, H_{place}, H_{ret}

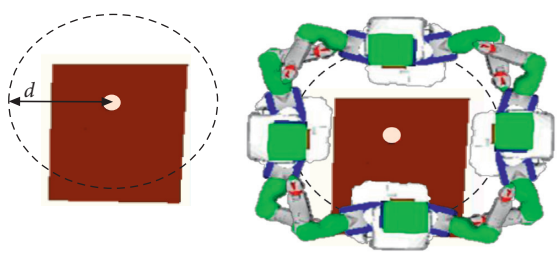


Figure 4: Domain generation of B_{each}. (a) The range of domain and (b) the discrete domain (when $m = 1$, $n = 4$).

distance d with the object is the range of domain. Figure 4(b) shows an example of a domain generated when m is 1 and n is 4. The domain of B_{place} is generated in the similar way as the domain of B_{reach}. However, it calculates the distance d on the basis of the center of the table.

Also, we can know that the domain of H_{grasp} should be located at a location very close to the object and the orientation of B_{reach} should be towards the object. This can be represented in an equation (3) as follows:

$$H_{\text{grasp}} = (t_1 \cdot r_2 \cdot t_0 \cdot r_1 \cdot r_0) O_{\text{init}}, \tag{3}$$

where r_0 is a matrix representing a quaternion $q = (0, 0, 0, 1)$, r_1 is a matrix representing an axis angle $a = (0, -(\pi/2), 0)$, t_0 is a matrix representing a pose $p = (1, 0, 0, 0, -d, 0, 0)$, r_2 is a matrix representing an axis angle $a = (0, 0, \theta)$, and t_1 is a matrix representing a pose $p = (1, 0, 0, 0, 0, 0, h)$.

This equation is to calculate the position and orientation of the hand relative to the coordinate system of the object. This equation has three parameters: d, θ, and h. d is the distance between the center of the hand and the center of the object, h is the height of the hand from the bottom of the object, and θ is the direction of the hand towards the object. d can be appropriately determined in consideration of the finger length. h may be determined in consideration of the volume of the hand, etc. If the number of d is m and the number of h is k, then the number of domains is mkn.

Figure 5 shows the area of the domain of H_{grasp}. In Figure 5(a), the circular dotted line that maintains the distance with the object d and the height h is the domain. Figure 5(b) shows an example of a domain generated when m is 1, k is 1, and n is 6. The domains of H_{pre} and H_{post} are sampled similar to the domain of H_{grasp}. However, d for the domain of H_{pre} is farther than d for the domain of H_{grasp}, and h for the domain of H_{post} is higher than h for the domain of H_{grasp}.

In the case of O_{place}, we can know that the domain of O_{place} should be located on the support plane. To prevent falling, the position should be slightly further inside from each corner of the support plane. When O_{place} is a matrix representing a pose $p = (1, 0, 0, 0, x, y, z)$, x, y, and z are constrained as below.

$$\begin{aligned} P_{\min X} + p < x < P_{\max X} - p, \\ P_{\min Y} + p < y < P_{\max Y}, \\ z = Pz. \end{aligned} \tag{4}$$

In equation (4), the support plane P is assumed to be rectangular. This equation has parameters x, y, z, and p. x and y show the position of the object on the support plane. z represents the height of the object. p represents the space inward from the edge of the support plane, that is, the padding. If the number of x is m, and if the number of y is n, then the number of domains is mn.

Figure 6 shows the area of the domain of O_{place}. The gray area on the table is the area of the domain. The gray area represents the remaining area of the support plane excluding padding. Figure 6(b) shows an example of a domain generated at linear intervals when m is 1 and n is 4.

4.1.3. Constraints. The constraint types are categorized into unary constraints (e.g., *reachableBasePose* (B, o)), binary constraints (e.g., *placeableBasePose* (B, O)), and nonbinary constraints (e.g., *preGraspablaHandPose* (H, H', B, h, o))

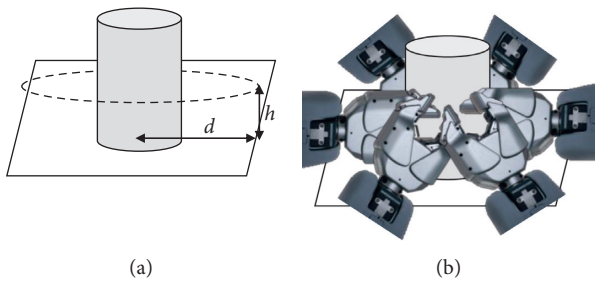


Figure 5: Domain generation of H_{grasp}. (a) The range of domain and (b) the discrete domain (when $m = 1$, $k = 1$, and $n = 6$).

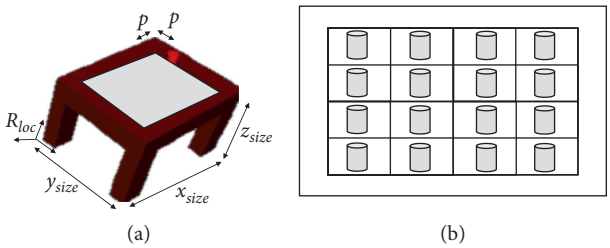


Figure 6: Domain generation of O_{place}. (a) The range of domain and (b) the discrete domain (when $m = 4$ and $n = 4$).

according to arity in the constraints. These constraints are all extracted from the preconditions of each action in the task plan skeleton. Table 6 is a mapping table for this. For convenience, among the parameters in the constraints, constants are excluded from the variables shown.

There is an issue of how to deal with the pose of movable object to be affected by the previous action. There are some ideas for this. In Lozano-Pérez and Garrett, some constraints have a variable of a movable object that was affected by previous action. For example, when they assume that there is a plan skeleton for two pick and place tasks, first *disjoint* (. . ., {*O*1}) constraint is modelled from the first place action and then *disjoint* (. . ., {*O*2, *O*1}) constraint is modelled from the second place action although the second place action does not have a parameter of variable *O*1. However, this is not scalable. In Lagriffoul et al., they avoided this issue by fixing the goal pose of each objects.

In this paper, we model the constraint that only has variables that correspond to self-action parameters. For instance, *placeableHandPose* constraint has only one variable for object pose regardless of the number of place actions. This constraint considers the poses of other movable objects as constraints that are loaded from the environment. This is very scalable to model the CSP from the plan skeleton. But it may assign the wrong pose (that collides with other movable objects) to a variable. To avoid this, we propose a variable ordering in section 4.3.

Meanwhile, we classify the constraints into the pose constraints and path constraints according to their semantics. The path constraints are *validBasePath* and *validHandPath*, and the remainder is the pose constraint. All constraint descriptions are summarized in a table in the appendix. The classification of constraints according to semantic is used in heuristic design.

4.2. Preprocessing. Preprocessing is performed to reduce the search space by reducing the size of domain sets. In the preprocessing step, node consistency is checked using the unary constraints. Through the node consistency, the values that satisfy the unary constraints are left and others are removed. In Table 6, the unary constraints are *reachableBasePose* (*B*) and *placeableObjectPose* (*O*).

Figure 7 shows an example of checking node consistency using *reachableBasePose* (*Breach*). As shown in Figure 7(a), when there are 4 domain values (①–④) for variable Breach, when the node consistency is checked for *reachableBasePose* (*Breach*), domain values ③ and ④ are removed as shown Figure 7(b). Domain value ③ is removed because the target object does not exist in the robot's configuration space accessible by the robot's arm, and domain value④ is removed because it collides with the table.

4.3. Ordering Heuristics. In the CSP, ordering heuristics (the search order of variables, the call order of constraints, etc.) are greatly influenced by the overall search time. We propose two ordering heuristics, variable ordering and constraint ordering. Variable ordering considers the inter-action dependency and the intra-action dependency.

4.3.1. Variable Ordering. First, our variable ordering considers the inter-action dependency. We mentioned the movable object issue of our modelling method in section 4.1.3. This variable ordering solves this issue. This takes into account the order between actions. All variables belonging to the earliest ordered actions are visited first. Then, we can get the priority of variable visits like Table 7. One thing to note is that after bounding values to all the variables of an action, the effect of this action is projected into the environment. As the effect of an action changes the environment, the variables of the next action must be visited. Then, when moving several objects from one table to another, the position of the next object can be determined considering the placement of the previously moved object.

Even if the ordering is conducted considering inter-action dependency, because the manipulation actions such as the *PickUpGeneral* and *PutDownGeneral* include multiple CSP variables, ordering between the variables in these actions is not completely performed. Ordering considering the intra-action dependency between variables orders multiple variables within one action. The intra-action dependency between the variables is as follows. For the *PickUpGeneral* action, H_{grasp} affects H_{pre}, and H_{post}, H_{pre}, and H_{post} do not affect each other. For *PutDownGeneral*, the placement position O_{place} affects H_{place}, H_{low}, and H_{ret}, H_{place} affects H_{low} and H_{ret}, H_{low} and H_{ret} do not affect each other. Figure 8 denotes this as a constraint network. Based on the constraints modelled by these dependencies, this paper determines the search order of the variables using MRV, a general-purpose CSP heuristic.

Table 8 shows the search order of the variables finally determined by this ordering. The variable ordering proposed in this paper is a mixture of generic variable ordering and special-purpose variable ordering.

Table 6: Mapping action preconditions into constraints.

Action	Precondition	CSP constraint
MoveBase *ForPickUp*	robotAt (B_{init}), objectAt (object03, O_{init}), reachableBasePose (pr2, object03, O_{init}, B_{reach}), validBasePath (pr2, B_{init}, B_{reach})	reachablaBasePose (B_{reach}) validBasePath (B_{init}, B_{reach})
PickUp General	empty (hand), robotAt (B_{reach}), objectAt (obj, O_{init}), handAt (rgripper, H_{init}), graspableHandPose (rgripper, object03, O_{init}, B_{reach}, H_{grasp}), preGraspableHandPose (rgripper, object03, O_{init}, H_{grasp}, H_{pre}), postGraspableHandPose (rgripper, object03, O_{init}, H_{grasp}, H_{post}), validHandPath (rgripper, B_{reach}, H_{init}, H_{pre}), validHandPath (rgripper, B_{reach}, H_{pre}, H_{grasp}), validHandPath (rgripper, B_{reach}, H_{grasp}, H_{post}), validHandPath (rgripper, B_{reach}, H_{post}, H_{init})	graspableHandPose (H_{grasp}, B_{reach}) preGraspableHandPose (H_{pre}, H_{grasp}, B_{reach}) postGraspableHandPose (H_{post}, H_{grasp}, B_{reach}) validHandPath (H_{init}, H_{pre}, B_{reach}) validHandPath (H_{pre}, H_{grasp}, B_{reach}) validHandPath (H_{grasp}, H_{post}, B_{reach}) validHandPath (H_{post}, H_{init}, B_{reach})
MoveBase *ForPutDown*	robotAt (B_{reach}), graspedBy (object03, rgripper), placeableBasePose (pr2, object03, table02, B_{place}), validBasePath (pr2, B_{reach}, B_{place})	placeableBasePose (B_{place}) validBasePath (B_{reach}, B_{place})
PutDown General	graspedBy (object03, rgripper), robotAt (B_{place}), handAt (rgripper, H_{init}), placeableObjectLocation (object03, table02, O_{place}), placeableHandPose (rgripper, object03, B_{place}, O_{place}, H_{place}), lowerableHandPose (rgripper, object03, B_{place}, O_{place}, H_{place}, H_{low}), retractableHandPose (rgripper, object03, B_{place}, O_{place}, H_{place}, H_{ret}), validHandPath (rgripper, B_{place}, H_{init}, H_{low}), validHandPath (rgripper, B_{place}, H_{low}, H_{place}), validHandPath (rgripper, B_{place}, H_{place}, H_{ret}), validHandPath (rgripper, B_{place}, H_{ret}, H_{init})	placeableObjectPose (O_{place}) placeableHandPose (B_{place}, O_{place}, H_{place}) lowerableHandPose (B_{place}, O_{place}, H_{place}, H_{low}) retractableHandPose (B_{place}, O_{place}, H_{place}, H_{ret}) validHandPath (B_{place}, H_{init}, H_{low}) validHandPath (B_{place}, H_{low}, H_{place}) validHandPath (B_{place}, H_{place}, H_{ret}) validHandPath (B_{place}, H_{ret}, H_{init})

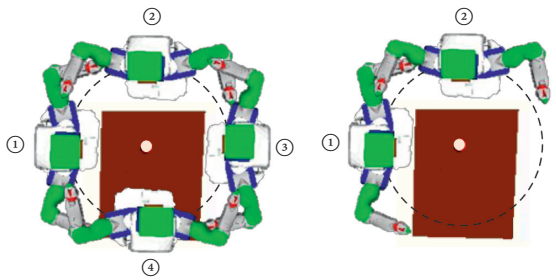


Figure 7: (a) All the domains and (b) domains pruned by *reachableBasePose* constraint.

Table 7: Variable ordering based on inter-action dependency.

Priority	Variable
1	B_{reach}
2	H_{grasp}, H_{pre}, H_{post}
3	B_{place}
4	$O_{place.}$, $H_{place.}$, $H_{low.}$, H_{ret}

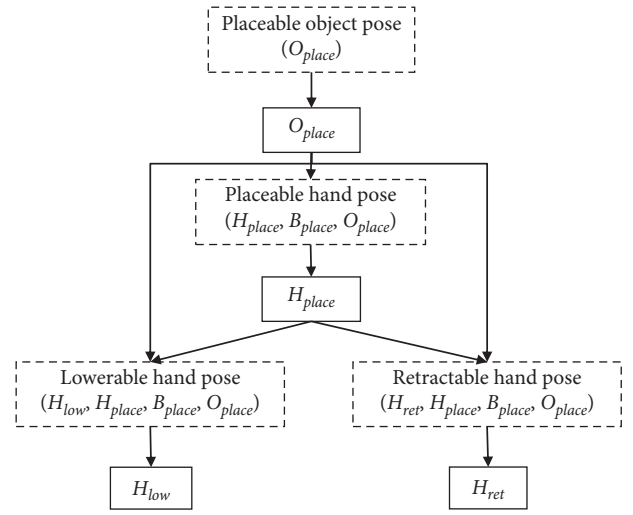


Figure 8: Constraint network for variable ordering in *PutDown-General* action.

4.3.2. Constraint Ordering. As mentioned in Section 4.1.3, in addition to the general classification method of classifying constraints according to the number of parameters, this paper classifies the constraints into pose constraints and path constraints according to the semantics of the constraints. The path constraints are validBasePath and validHandPath, and the remainders are the pose constraints.

Table 8: Variable ordering based on intra-action dependency.

Priority	Variable
1	B_{reach}
2	H_{grasp}
3	H_{pre}, H_{post}
4	O_{place}
5	B_{place}
6	H_{place}
7	H_{low}, H_{ret}

This constraint classification method can be used to determine the calling order of the constraints. The calling order of the constraints is determined as first calling the pose constraint and then the path constraint. Typically, to generate a single motion plan, a valid goal pose that satisfies the constraints is first determined, and then a trajectory to the goal pose is calculated. In practice, the cost of calculating the trajectory is large enough to comprise most of the cost for checking motion feasibility. Therefore, if the trajectory is calculated for an invalid goal pose, the search time is greatly wasted. The constraint calling order has the effect of pre-pruning invalid pose values, thus preventing unnecessary path constraint calls.

4.4. Constraint Propagation. In the search phase, a depth-first backtracking algorithm [44] involving constraint propagation is used to search the solution. Constraint propagation is a technique that tentatively removes the domains of other variables that violate the constraints based on the values bound to the currently visited variables during the backtracking process. Therefore, backtracking with constraint propagation can result in better performance than normal backtracking as it reduces the size of the domains to be visited. GAC (Generalized Arc Consistency) [45] is generally used to propagate nonbinary constraints fully. GAC is an extension of arc consistency [45] used to propagate constraints with more than two variables. However, full propagation of nonbinary constraints not only requires excessive costs for motion planning but also is difficult to see the reduction effect in the domains of the transitive variables. Therefore, we determined that it is more reasonable to use forward checking, which propagates the constraints only to variables neighboring the currently visited variables. In particular, when conducting forward checking, only the pose constraint check is performed to the exclusion of the path constraint check. This is because, first, it is reasonable to check the domains after first reducing them as much as possible because path constraint checking requires considerable computation time. In practice, the cost of path constraint checking accounts for most of the cost of motion feasibility checking. Second, the path constraints are dependent on the pose constraints. Thus, a valid goal pose must be determined first to enable the search for a collision-free path to the goal pose.

The proposed framework solves the constraint satisfaction problem by using the search method shown in Algorithm 1. Algorithm 1 describes a depth-first backtracking search that involves forward checking. In Algorithm 1, one of the domain's values is bound to the visited variable, and then forward checking is performed on the neighboring variables of the current variables among those still unbound. As mentioned above, only the pose constraint check is performed in forward checking. The path constraint check is applied when the values are bound to the visited variables, in which constraint checking is performed with the remaining variables.

Algorithm 2 describes the constraint propagation function. This algorithm propagates pose constraints to variables that are still unbound and neighboring the currently visited variable. When propagating a pose constraint, the size of the domain of the neighboring variables may be partially reduced by the pose constraint. If all domain values violate the constraint, then this is treated as failure. The part of the domain in which values are removed is in the revise function. The revise function conducts a pose constraint check and removes all domain values that violate the pose constraints.

5. Implementation and Evaluation

5.1. Implementation. Our framework was implemented in Ubuntu 16.04, Python 2.7. The FD (Fast Downward) library [46] was used for task planning, and the TrajOpt library [47] was used for motion planning. The robot task environment was implemented using the OpenRAVE [48] simulator. The modelling and solution methods of the CSP were implemented by extending the python-constraints open-source library [49]. The computing environment was an Intel (R) Core (TM) i7-7700 CPU @ 3.60 GHz, 16 GB memory.

5.2. Evaluation. We performed experiments to evaluate the generality, scalability, efficiency, and optimality of the proposed CSP framework for TAMP. The length of the task plan skeleton was set as shown in Table 9. P1 is a task plan skeleton that takes pick and place once, and P2 is a task plan skeleton that takes pick and place twice. The size of the domain set was set as shown in Table 10. Table 10 shows the size of the domain set generated for each variable. The domain set D_2 is twice as large as the domain set D_1. The sizes of the obstacles were set as shown in Table 11. The obstacles were placed at table T_{pick} where the target object is located, as well as at table T_{place} for placing the target object, at the same sizes. For the obstacle sizes, none were O_1, 3 were O_2, and 6 were O_3.

First, an evaluation was performed to confirm comprehensiveness of constraints which the proposed CSP framework dealt with. For this evaluation, the constraints of our framework were compared with those of previous studies. Table 12 shows the evaluation results. The constraints were largely divided into pose and path constraints, and more specifically, constraints on mobile base, hand, and object. The experimental results show that out of 16

```
function BWP (assignment = {}, csp= (V, D, C)) returns a solution or failure
if assignment is complete then return assignment end if
var = ordered_unassigned_variable (V)
for each val in D (var)
   if val is consistent with assignment then
      add {var = val} to assignment
      result = forward_checking (csp, var, assignment)
      if result is failure then return failure end if
      add neighbors to assignment
      result = BWP (assignment, csp)
      if result is not failure then return result end if
      remove {var = value} from assignment
   end if
end for
return failure
```

Algorithm 1: Backtracking with propagation.

```
function LFC (csp = (V, D, C), var, assignment) returns false if inconsistency is found and true otherwise, and update domain of
   neighbors
neighbors = all_neighbors (csp, var)
while neighbors is not empty do
   N_i = neighbors.pop
   if revise (N_i, csp, assignment)
      if D (N_i) is empty then
         return false
      end if
   end if
end while
return true
function revise (N_i, csp, assignment) return true if the domain of N_i is changed is Revised = false
   for each val in D (N_i)
      if val is inconsistent in pose constraints with assignment then remove val from D (N_i) is Revised = true
   end if
end for
```

Algorithm 2: Limited forward checking.

Table 9: Experiment setting: task plan skeleton.

Task plan skeleton	Action sequence
P_1	1 MoveBaseForPickUp (...) 2 PickUpGeneral (...) 3 MoveBaseForPutDown (...) 4 PutDownGeneral (...)
P_2	1 MoveBaseForPickUp (...) 2 PickUpGeneral (...) 3 MoveBaseForPutDown (...) 4 PutDownGeneral (...) 5 MoveBaseForPickUp (...) 6 PickUpGeneral (...) 7 MoveBaseForPutDown (...) 8 PutDownGeneral (...)

Table 10: Experiment setting: domain.

Variable	Domain	
	D_1	D_2
B_{reach}	16	32
H_{grasp}	8	16
H_{pre}	24	48
H_{post}	24	48
O_{place}	10	20
B_{place}	16	32
H_{place}	8	16
H_{low}	24	48
H_{ret}	24	48

Table 11: Experiment setting: obstacle.

Table	Obstacle		
	O_1	O_2	O_3
T_{pick}	0	3	6
T_{place}	0	3	6

constraints, 13 checks are possible in ours, 8 in Garrett, 7 in Lagriffoul, and 4 in Lozano-Pérez. Our framework showed more comprehensive coverage of constraints than existing works because it includes additional constraints on both

Table 12: Constraint list.

Constraint			Ours	Lagriffoul	Lozano-Pérez	Garrett
Pose	Base	Picking up	Yes	No	No	Yes
		Putting down	Yes	No	No	Yes
	Hand	Grasp	Yes	No	No	No
		Pre-grasp	Yes	No	No	No
		Post-grasp	Yes	No	No	No
		Re-grasp	No	Yes	No	No
		Placing	Yes	Yes	Yes	Yes
		Lowering	Yes	No	No	No
		Retracting	Yes	No	No	No
		Hand over	No	Yes	No	No
	Object	Placing	Yes	Yes	Yes	Yes
		Stacking	No	Yes	No	No
Path	Base	Collision-free path	Yes	No	No	Yes
		Collision-free path with object	Yes	No	No	Yes
	Hand	Collision-free path	Yes	Yes	Yes	Yes
		Collision-free path with object	Yes	Yes	Yes	Yes

base and hand poses for mobile manipulation tasks. On the contrary, our framework includes neither hand over constraint nor stacking constraint since it mainly focuses on pick and place tasks. However, because we focus on pick and place, it does not cover constraints such as stack and hand over.

Second, experiments were conducted to investigate the effect of the proposed preprocessing. Figure 9 shows the measurements of the search time with and without preprocessing. The horizontal axis is a problem instance with different values of the experimental parameters, and the vertical axis is the search time. The experimental results demonstrate that the search with the proposed preprocessing is faster than that without preprocessing. The set of constraints listed in Table 9 includes many unary constraints. Therefore, the proposed preprocessing was much effective to reduce the search space.

Third, experiments were conducted to analyze the performance of the proposed variable ordering method. In these experiments, our variable ordering method was compared with generic ordering methods such as MRV, DH (Degree Heuristic), and random ordering. Figure 10 shows the experimental results. The proposed variable ordering method shows the fastest search time, followed by MRV, DH, and random ordering. These results demonstrate that the proposed region-dependent ordering method considering the order between the actions has a positive effect on the search time.

Fourth, experiments were conducted to analyze effect of the proposed constraint ordering. Figure 11 shows the performance with and without the proposed constraint ordering. The experimental results demonstrate that the search time with constraint ordering is faster than the search time without constraint ordering. These results indicate that the proposed constraint ordering to pre-prune unnecessary path constraint checks positively impacts the search time.

Fifth, experiments were conducted to analyze effect of the proposed constraint propagation. Figure 12 shows the experimental results to compare search time of the proposed constraint propagation with those of generic constraint propagation methods such as forward checking (FC) and arc consistency (AC). The experimental results demonstrate that the proposed constraint propagation method has the shortest search time, followed by forward checking, arc consistency, and the method without constraint propagation. These results indicate that forward checking limited to the pose constraints as proposed in this paper is the most effective. Meanwhile, the results of Figure 12 demonstrate that forward checking (FC) is more effective than arc consistency (AC) for motion feasibility checking.

Sixth, experiments were conducted to compare the performance of our TAMP framework with existing constraint-based methods, namely, Lozano-Pérez [18] and Lagriffoul [17]. In Lozano-Pérez, both the MRV variable ordering heuristics and arc consistency checking with pose constraints are applied. In case of Lagriffoul, both an action-dependent variable ordering heuristics and forward checking with pose constraints are applied. Because their CSP modellings are much different from ours, we reimplement their heuristics and search strategies within our framework. The experimental results in Figure 13 demonstrate that the proposed framework used the shortest search time. Lozano-Pérez's method showed the lowest performance because its general-purpose heuristic, MRV, and arc consistency are not effective for multi-hop propagation. Lagriffoul's method is faster than that of Lozano-Pérez because of its unique variable ordering heuristics and forward checking. However, Lagriffoul's method only considers the inter-action dependency for variable ordering, while our framework considers both inter- and intra-action dependencies.

6. Discussion and Conclusion

This paper proposed an efficient constraint satisfaction framework for checking motion feasibility of task plan skeletons as a TAMP solution. The proposed framework not only presents mapping rules that can automatically generate a constraint satisfaction problem (CSP) from the task plan skeleton, but it also provides useful methods to solve the CSP such as pre-processing, unique variable ordering heuristics based on intra- and inter-action dependencies in a task plan skeleton, and constraint propagation to improve efficiency of motion feasibility checking. Through numerous experiments using the humanoid robot PR2, we confirmed high performance and efficiency of the proposed framework.

However, the current framework has some technical limitations. The proposed framework was applied to relatively simple task environments based on the assumption that the environments are fully observable, static, deterministic, and have a single robot. However, the real-world

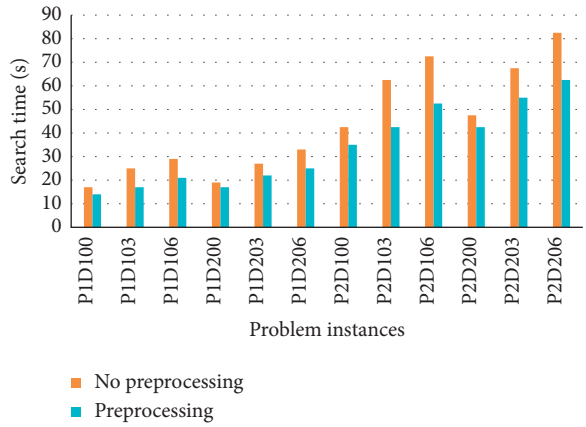


Figure 9: Experimental results of preprocessing.

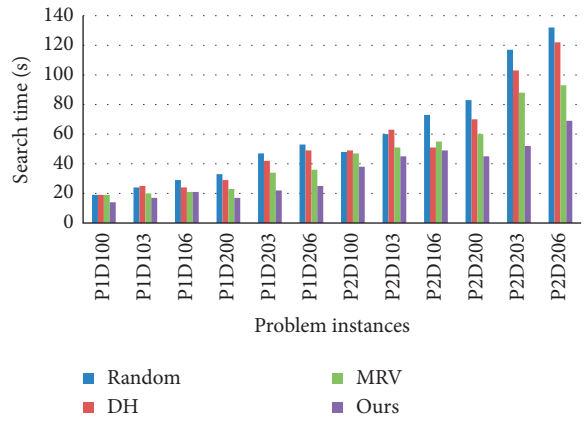


Figure 10: Experimental results of variable ordering.

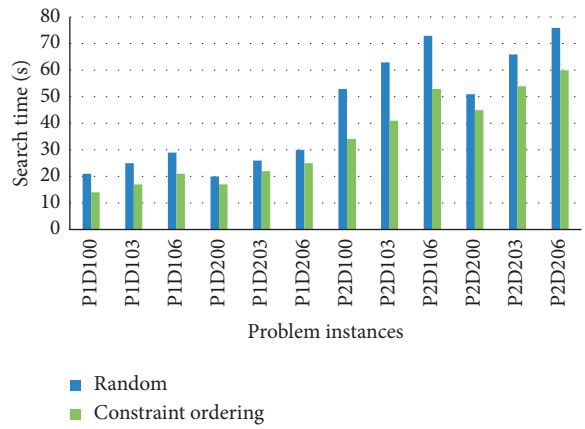


Figure 11: Experimental results of constraint ordering.

physical environments do not meet the assumption. They are partially observable, dynamic, stochastic, and have multiple robots or manipulators. Therefore, we plan to extend our framework to deal with dynamics of real-world environments by designing advanced action model and replanning capability. Furthermore, to address uncertainty in state

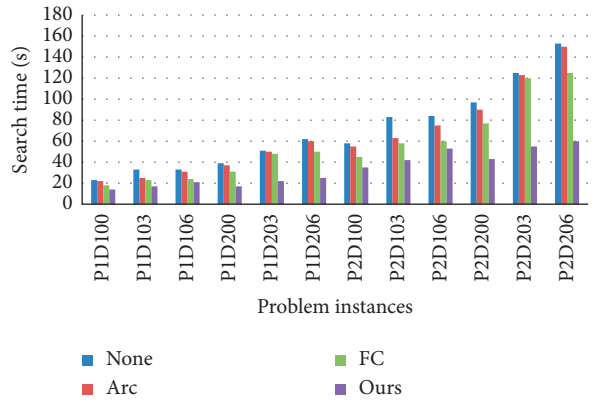


Figure 12: Experimental results of constraint propagation.

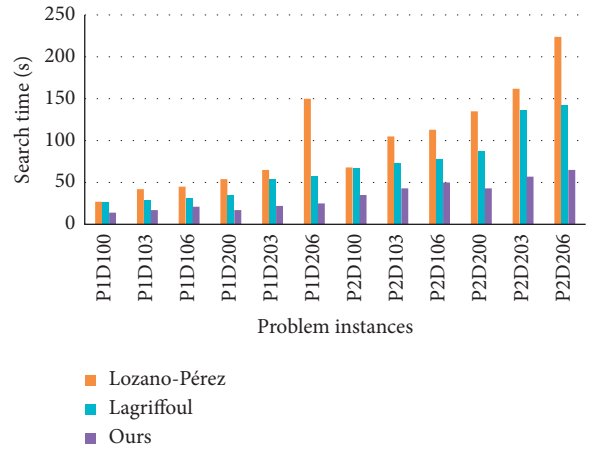


Figure 13: Performance comparison with previous works.

Table 13: Constraint descriptions.

Constraint	Description
reachablaBasePose (B)	When robot is located on base pose B, there must be no collision and the target object must be within radius of robot's arm.
placeableBasePose (B)	When robot is located on base pose B, there must be no collision and the center of table must be within radius of robot's arm.
graspableHandPose (H, B)	When robot's hand is located at hand pose H, robot should be able to grasp the target object by closing fingers.
preGraspableHandPose (H, H', B)	When robot is located on base pose B, there is at least one inverse kinematics for the hand pose H. When robot's hand is located at hand pose H, the hand must be a little farther than hand pose H' which is a graspableHandPose for target object.
postGraspableHandPose (H, H', B)	When robot is located on base pose B, there is at least one inverse kinematics for the hand pose H When robot's hand is located at hand pose H, the hand must be a little higher than hand pose H' which is a graspableHandPose for target object.
placeableHandPose (B, O, H)	When robot is located on base pose B, there is at least one inverse kinematics for the hand pose H. When robot's hand is located at hand pose H, robot should be able to place the target object on the target location O by opening fingers.
lowerableHandPose (B, O, H', H)	When robot is located on base pose B, there is at least one inverse kinematics for the hand pose H. When robot's hand is located at hand pose H, the hand must be a little higher than hand pose H' is a placeableHandPose for target location O.
retractableHandPose (B, O, H', H)	When robot is located on base pose B, there is at least one inverse kinematics for the hand pose H. When robot's hand is located at hand pose H, the hand must be a little farther than hand pose H' is a placeableHandPose for target location O.
placeableObjectPose (O)	When robot is located on base pose B, there is at least one inverse kinematics for the hand pose H. When target object is located on target location O, there must be no collision.
validBasePath (B, B')	There must be at least one collision-free path from base pose B to base pose B'.
validHandPath (H, H', B)	There must be at least one collision-free path from hand pose H to hand pose H' at base pose B.

recognition, the current framework will be extended to solve TAMP over belief state space. It will be also interesting to apply the proposed framework to multi-agent or multi-robot environments.

Appendix

Appendices A–D describe the functions that appear in Section 4.1.2. Table 13 describes the definition of constraints Section 4.1.3.

A. Matrix from Pose

Given a pose p, this function converts p to rotation matrix R. R is obtained as follows:

$$R = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_2 + q_0q_2) & x \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) & y \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{A.1}$$

where

$$p = (q_0, q_1, q_2, q_3, x, y, z). \tag{A.2}$$

B. Matrix from Quaternion

Given a quaternion q, this function converts q to rotation matrix R. R is obtained as follows:

$$R = \begin{bmatrix} 1 - Q2 - Q3 & \dfrac{2(q_1q_2 - q_0q_3)}{L} & \dfrac{2(q_1q_3 - q_0q_2)}{L} \\ \dfrac{2(q_1q_2 - q_0q_3)}{L} & 1 - Q1 - Q3 & \dfrac{2(q_2q_3 - q_0q_1)}{L} \\ \dfrac{2(q_1q_3 - q_0q_2)}{L} & \dfrac{2(q_2q_3 - q_0q_1)}{L} & 1 - Q1 - Q2 \end{bmatrix}, \tag{A.3}$$

where

$$\begin{aligned} q &= (q_0, q_1, q_2, q_3), \\ L &= a_0^2 + a_1^2 + a_2^2 + a_3^2, \\ Q1 &= \frac{2a_1^2}{L}, \\ Q2 &= \frac{2a_2^2}{L}, \\ Q3 &= \frac{2a_3^2}{L}. \end{aligned} \tag{A.4}$$

C. Quaternion from Axis Angle

Given an axis angle, this function converts the axis angle to quaternion q. q is obtained as follows:

$$q = \begin{cases} (1, 0, 0, 0), & \text{if } L = 0, \\ \left(\dfrac{L}{2}, sa_x, sa_y, sa_z\right), & \text{otherwise}, \end{cases} \tag{A.5}$$

where

$$\begin{aligned} a &= (a_x, a_y, a_z), \\ L &= a_x^2 + a_y^2 + a_z^2, \\ s &= \frac{\sin(L/2)}{L}. \end{aligned} \tag{A.6}$$

D. Matrix from Axis Angle

Given an axis angle, this function converts the axis angle to rotation matrix R. This function obtains a quaternion from the function of Appendix C and obtains R from the function of Appendix B.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the Technology Innovation Program or Industrial Strategic Technology Development Program (Grant No. 10077538, Development of manipulation technologies in social contexts for human-care service robots) funded by the Ministry of Trade, Industry and Energy (MOTIE, Korea).

References

[1] K. Hauser and J. C. Latombe, "Integrating task and PRM motion planning: dealing with many infeasible motion planning queries," in *Proceedings of ICAPS Workshop on Bridging the Gap between Task and Motion Planning*, Thessaloniki, Greece, July 2009.

[2] E. Plaku and G. D. Hager, "Sampling-based motion and symbolic action planning with geometric and differential constraints," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA'10)*, pp. 5002–5008, Anchorage, Alaska, May 2010.

[3] J. Guitton and J. L. Farges, "Taking into account geometric constraints for task-oriented motion planning," in *Proceedings of AAAI Workshop on Bridging the Gap between Task and Motion Planning*, pp. 26–33, Atlanta, GA, USA, July 2010.

[4] S. Alili, A. K. Pandey, E. A. Sisbot, and R. Alami, "Interleaving symbolic and geometric reasoning for a robotic assistant," in *Proceedings of ICAPS Workshop on Combining Action and Motion Planning*, Toronto, Canada, March 2010.

[5] L. De Silva, A. K. Pandey, M. Gharbi, and R. Alami, "Towards combining HTN planning and geometric task planning," in *Proceedings of RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*, Berlin, Germany, June 2013.

[6] L. De Silva, A. K. Pandey, and R. Alami, "An interface for interleaved symbolic-geometric planning and backtracking," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'13)*, pp. 232–239, Tokyo, Japan, November 2013.

[7] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, "Semantic attachments for domain-independent planning systems," in *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'09)*, pp. 114–121, Thessaloniki, Greece, September 2009.

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

[9] C. Dornhege, P. Eyerich, T. Keller, M. Brenner, and B. Nebel, "Integrating task and motion planning using semantic attachments," in *Proceedings of AAAI Workshop on Bridging the Gap between Task and Motion Planning*, pp. 10–17, Atlanta, GA, USA, July 2010.

[10] A. Hertle, C. Dornhege, T. Keller, and B. Nebel, "Planning with semantic attachments: an object-oriented view," in *Proceedings of 20th European Conference on Artificial Intelligence (ECAI'12)*, pp. 402–407, Montpellier, France, August 2012.

[11] L. P. Kaelbling LP and T. Lozano-Pérez, "Hierarchical planning in the now," in *Proceedings of AAAI Workshop on Bridging the Gap between Task and Motion Planning*, pp. 33–42, Atlanta, GA, USA, July 2010.

[12] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA'11)*, pp. 1470–1477, Shanghai, China, May 2011.

[13] A. Gaschler, R. P. Petrick, M. Giuliani, M. Rickert, and A. Knoll, "KVP: a knowledge of volumes approach to robot task planning," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'13)*, pp. 202–208, Tokyo, Japan, November 2013.

[14] J. Bidot, L. Karlsson, F. Lagriffoul, and A. Saffiotti, "Geometric backtracking for combined task and motion planning in robotic systems," *Artificial Intelligence*, vol. 247, pp. 229–265, 2017.

[15] K. Okada, A. Haneda, H. Nakai, M. Inaba, and H. Inoue, "Environment manipulation planner for humanoid robots using task graph that generates action sequence," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS'04)*, pp. 1174–1179, Sendai, Japan, September 2004.

[16] 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.

[17] 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*, vol. 33, no. 14, pp. 1726–1747, 2014.

[18] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'14)*, pp. 3684–3691, Chicago, IL, USA, September 2014.

[19] R. Dearden and C. Burbridge, "An approach for efficient planning of robotic manipulation tasks," in *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'13)*, pp. 55–63, Rome, Italy, June 2013.

[20] R. Dearden and C. Burbridge, "Manipulation planning using learned symbolic state abstractions," *Robotics and Autonomous Systems*, vol. 62, no. 3, pp. 355–365, 2014.

[21] D. Leidner and C. Borst, "Hybrid reasoning for mobile manipulation based on object knowledge," in *Proceedings of IROS Workshop on AI-based robotics*, Tokyo, Japan, November 2013.

[22] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA'11)*, pp. 4575–4581, Shanghai, China, May 2011.

[23] G. Havur, K. Haspalamutgil, C. Palaz, E. Erdem, and V. Patoglu, "A case study on the tower of hanoi challenge: representation, reasoning and execution," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA'13)*, pp. 4552–4559, Karlsruhe, Germany, March 2013.

[24] 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 *Proceedings of IEEE International Conference on Robotics and Automation (ICRA'14)*, pp. 639–646, Hong Kong, China, May 2014.

[25] C. Phiquepal and M. Toussaint, "Combined task and motion planning under partial observability: an optimizationbased approach," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA'19)*, pp. 9000–9006, Montreal, Canada, May 2019.

[26] R. Shome and K. E. Bekris, "Anytime multi-arm task and motion planning for pick-and-place of individual objects via handoffs," in *Proceedings of IEEE International Symposium on Multi-Robot and Multi-Agent Systems (MRS'19)*, pp. 37–43, New Brunswick, NB, USA, August 2019.

[27] T. Migimatsu and J. Bohg, "Object-centric task and motion planning in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 844–851, 2020.

[28] V. N. Hartmann, O. S. Oguz, D. Driess, M. Toussaint, and A. Menges, "Robust task and motion planning for long-horizon architectural construction planning," 2020, https://arxiv.org/abs/2003.13649.

[29] A. Akbari, Muhayyuddin, and J. Rosell, "Knowledge-oriented task and motion planning for multiple mobile robots," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 31, no. 1, pp. 137–162, 2019.

[30] A. Akbari, F. Lagriffoul, and J. Rosell, "Combined heuristic task and motion planning for bi-manual robots," *Autonomous Robots*, vol. 43, no. 6, pp. 1575–1590, 2019.

[31] S. Thakar, A. Kabir, P. M. Bhatt et al., "Task assignment and motion planning for bi-manual mobile manipulation," in *Proceedings of IEEE International Conference on Automation Science and Engineering (CASE'19)*, pp. 910–915, Vancouver, Canada, June 2019.

[32] T. Ren, G. Chalvatzaki, and J. Peters, "Extended task and motion planning of long-horizon robot manipulation," 2021, https://arxiv.org/abs/2102.09066.

[33] C. R. Garrett, C. Paxton, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox, "Online replanning in belief space for partially observable task and motion problems," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA'20)*, pp. 5678–5684, Xi'an, China, May 2020.

[34] A. M. Kabir, S. Thakar, P. M. Bhatt et al., "Incorporating motion planning feasibility considerations during task-agent assignment to perform complex tasks using mobile manipulators," in *Proceeding of IEEE International Conference on Robotics and Automation (ICRA '20)*, pp. 5663–5670, Paris, France, September 2020.

[35] J. Motes, R. Sandström, H. Lee, S. Thomas, and N. M. Amato, "Multi-robot task and motion planning with subtask dependencies," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3338–3345, 2020.

[36] Y. Ding, X. Zhang, X. Zhan, and S. Zhang, "Task-motion planning for safe and efficient urban driving," 2020, https://arxiv.org/pdf/2003.03807.

[37] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Sample-based methods for factored task and motion planning," in *Proceedings of Robotics: Science and Systems*, Cambridge, MA, USA, April 2017.

[38] R. E. Fikes and N. J. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[39] M. Ghallab, A. Howe, C. Knoblock et al., "PDDL—the planning domain definition language," in *Proceedings of International Conference on Artificial Intelligence Planning Systems (AIPS'98)*, Pittsburgh, PA, USA, December 1998.

[40] K. Erol, J. Hendler, and D. S. Nau, "HTN planning: complexity and expressivity," in *Proceedings of 12th National Conference on Artificial Intelligence (AAAI'94)*, pp. 1123–1128, Seattle, WA, USA, August 1994.

[41] J. C. Latombe, *Robot Motion Planning*, Springer Science and Business Media, Boston, MA, USA, 2012.

[42] S. M. LaValle, "Rapidly-exploring random trees: a new tool for path planning," Technical Report TR 98-11, Iowa State University, Ames, IA, USA, 1998.

[43] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[44] S. C. Brailsford, C. N. Potts, and B. M. Smith, "Constraint satisfaction problems: algorithms and applications," *European Journal of Operational Research*, vol. 119, no. 3, pp. 557–581, 1999.

[45] V. Kumar, "Algorithms for constraint-satisfaction problems: a survey," *Artificial Intelligence Magazine*, vol. 13, no. 1, pp. 32–44, 1992.

[46] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[47] J. Schulman, J. Ho, and A. Lee, "Finding locally optimal, collision-free trajectories with sequential convex optimization," *Robotics: Science and Systems*, vol. 9, no. 1, pp. 1–10, 2013.

[48] R. Diankov and J. J. Kuffner, "OpenRAVE: a Planning architecture for autonomous robotics," Tech. Rep. CMU-RI-TR-08-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2008.

[49] G. Niemeyer, "Python-constraint: solving constraint satisfaction problems in Python," 2017, https://pypi.org/project/python-constraint/.