Planning Non-repetitive Robotic Assembly Processes with Task and Motion Planning (TAMP)

Pok Yin Victor Leung[1][0000-0003-1536-8636], Yijiang Huang[1,2][0000-0003-1820-2535], Caelan Garret[2][0000-0002-6474-1276], Fabio Gramazio[1][0000-0002-3761-7675], Matthias Kohler[1][0000-0002-4111-4122]

[1] ETH Zurich, Switzerland
[2] MIT, USA [3] NVIDIA Research, USA

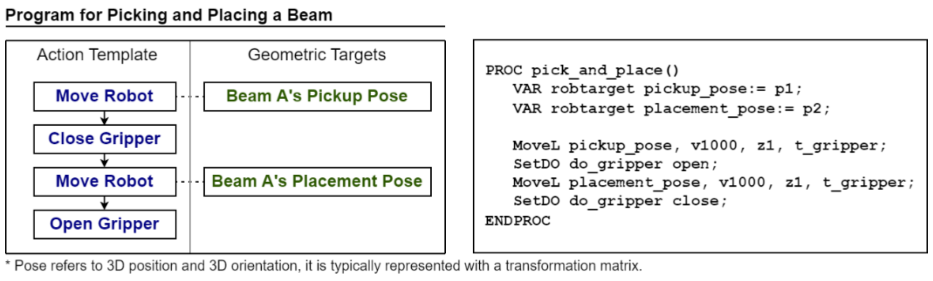
Abstract. Many recent examples of robotic construction are still planned using separate task and motion planning approaches. This separation makes it hard to create efficient action sequences, especially for non-repetitive assembly processes. TAMP is a more advanced planning approach that integrates task and planning to optimize action sequences while ensuring collision-free robotic trajectories.
The paper demonstrates the benefit of TAMP through a real-world case study based on a robotically constructed pavilion. This paper explains the technical details of TAMP and how it can be applied using a TAMP solver, PDDLStream. Six planning experiments with increasing complexity are created with Planning Domain Definition Language (PDDL) using an incremental programming approach that is intuitive to understand. Robotic execution time was collected from the actual construction, and the results show an estimated 16% reduction in execution time using TAMP compared to traditional task planning approaches like flowcharts. The flexibility of PDDLStream to include other planning considerations is also discussed for future research opportunities.

Keywords: Robotic Assembly, Task and Motion Planning, PDDL.

1 Introduction

Traditional paradigm for programming industrial robots is not suitable for architectural fabrication where non-repetitive tasks are often necessary for unique geometries. Instead, automated planning and validation are essential for non-repetitive robotic programs.

When discussing non-repetitive robotic programs, it is necessary to distinguish between the action template and the geometric targets. The action template (also known as a task template) is a list of sequentially executed actions, while geometric targets are descriptions that complement some actions, such as robotic movements. Figure 1 shows an example of an action template with four actions and two geometric targets for a simple construction process. Note that the two movement actions are accompanied by geometric targets that specify the target pose (position and orientation) at which the end effector will reach.


Program for Picking and Placing a Beam
Action Template
Geometric Targets
Move Robot
Beam A's Pickup Pose
Close Gripper
Move Robot
Beam A's Placement Pose
Open Gripper
* Pose refers to 3D position and 3D orientation, it is typically represented with a transformation matrix.

```
PROC pick_and_place()
   VAR robtarget pickup_pose:= p1;
   VAR robtarget placement_pose:= p2;

   MoveL pickup_pose, v1000, z1, t_gripper;
   SetDO do_gripper open;
   MoveL placement_pose, v1000, z1, t_gripper;
   SetDO do_gripper close;
ENDPROC
```

Fig. 1. Example of an action template and geometric targets represented symbolically (left) and in ABB RAPID language (right)

Using a single repetitive action template with a flexible geometric input allows the assembly of mass customized objects. However, structures created with this approach were often limited to a fixed topology. A more advanced customization approach is to use multiple snippets of action templates. For example, to construct a column-beam structure, two template snippets can be used. One snippet can be responsible for the "Column Assembly" while another for the "Beam Assembly". The selection and sequence of template snippets can be automated using computational logic based on the properties of a structure. In the computer science (CS) community, such process of selecting and scheduling actions is called task planning or symbolic planning.

Explicitly scripted conditional logic such as flowcharts (Huang et al. 2021) and behavior trees (Ögrevn and Colledanchise 2018) work well with construction tasks that have limited variations (Wagner et al. 2020). However, these approaches are not flexible enough when dealing with problems with many possible action sequences and interdependency (Garrett et al. 2020). Manually scripted action templates and the flowchart logic are often less than optimal. For these scenarios, new approaches that consider collisions and reachability are necessary (Hartmann et al. 2023).

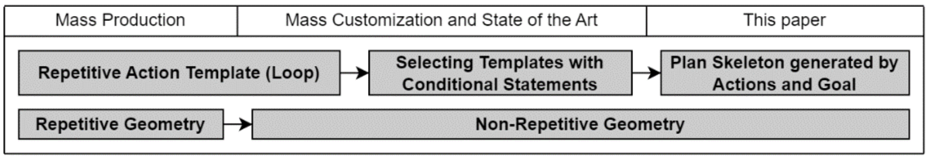

Mass Production
Mass Customization and State of the Art
This paper
Repetitive Action Template (Loop)
Selecting Templates with Conditional Statements
Plan Skeleton generated by Actions and Goal
Repetitive Geometry
Non-Repetitive Geometry

Fig. 2. Diagram showing the introduction of flexibility to the action templates and geometric targets.

The focus of this paper is to explore the application of Task and Motion Planning (TAMP) (Garrett et al. 2021), a state-of-art planning method developed in the CS community to a construction task. It integrates task planners (for deciding action sequences) and robotic motion planners (for finding and certifying collision-free trajectories). The integration allows the detection of robotic constraints by the motion planner to be used as feedback for planning a different and potentially more optimized action sequence (i.e., task plan). Figure 2 highlights the difference of using TAMP vs prior methods.

At this moment TAMP is not widely used in architectural robotics and digital fabrication due to its novelty and limited literature in the design field (Sherkat et al. 2023, Hartmann et al. 2020). Moreover, the Planning Domain Definition Language (PDDL) (Ghallab et al. 1998) is an action-based language for encoding state transitions, and is significantly different from the more common, sequentially-executed languages. In another words, the users model their assembly process with PDDL, after which the TAMP solver will generate the conventional "robotic program" for execution. To make the modelling process easier and more transparent for the user, this paper will introduce an incremental programming approach for writing PDDL. **Specifically, this paper will demonstrate:**

1. How to encode an architectural assembly process for TAMP using Planning Domain Definition Language (PDDL) with an incremental programming approach.
2. How to apply PDDLStream, a state-of-the-art TAMP solver, which combines symbolic PDDL solvers with robotic motion planners.
3. The benefits of TAMP over prior planning techniques, with results showing a more efficient task sequence that can reduce execution times.

2 Case Study

To provide a tangible example, we will use a recently constructed timber pavilion to introduce the incremental programming approach we developed for writing PDDL and demonstrate TAMP's benefits. The robotic process used to assemble the pavilion is called Timber Assembly with Distributed Robotic Tools (DiRT) (Leung et al.). The assembly process is characterized by having a single industrial robotic arm (integrated with a large overhead gantry system) manipulating multiple tools capable of clamping and screwing tasks (see Table 1). One of the unique requirements in this process is that the distributed tools must be attached to and detached from the structure one by one before and after the assembly of each beam. The spatial and temporal complexity of the process is analogous to a chef managing many simultaneously cooking pots on a stove. With many actions at its disposal and many different ways to sequence the actions, the robotic process is uniquely suited to explore the possibilities and limitations the benefit of TAMP.

The design of the demonstration pavilion (see Figure 3a), named CantiBox, consists of three topologically similar boxes assembled by the DiRT Process separately. Each box comprises 20 timber elements connected by integral timber joints machined before assembly. More details of the pavilion's structural and architectural design can be found in (Tanadini et al. 2023). This paper will only focus on the planning of the robotic process.

Table 1. List of robotic tools and their purpose for assembling the case study structure

Tools	Type Name	Amount	Purpose
Robotic Clamp	`CL3`	2	Assemble *clamped joints*. The allowable joint intersection angle for CL3 is 25-90°, and CL3M is 90-155°.
	`CL3M`	2	
Robotic	`SL1`	3	

Screwdriver	`SL1_G200`	1	Assemble *screwed joints*. SL1-G200 can also act as a timber gripper.
Robotic Gripper	`PG500`	1	Grasp timber beams and bring them to the structure for assembly. Three different gripper lengths (500mm to 1500mm) are available to suit different timber lengths.
	`PG1000`	1	
	`PG1500`	1	
Scaffolding Bar	-	4	Stabilize structure during construction. (Ignored in this paper for simplicity)

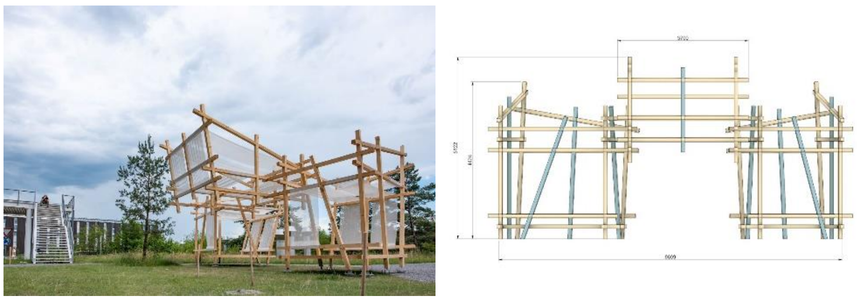


Fig. 3. Photo and front elevation drawing of the CantiBox Pavilion

The planning of tasks and motions are performed separately for each of the three boxes, with a planning horizon[1] of twenty elements every time. Each of the twenty elements is labeled during design time to use one of three assembly methods: Ground Connection, Joint Clamping, and Joint Screwing. The following three paragraphs summarize the assembly methods (see Figure 4 for related images). Note in advance that the Joint Clamping method is the focus of later discussions.

The Ground Connection Method is used for the first two timber elements to create a temporary connection to a ground platform. First, the robotic arm uses a gripper to position the timber element onto the construction platform. Then, a human operator fixes the timber to the platform using a fixture. Finally, the robotic gripper releases the timber, and the robot returns the gripper to storage.

The Joint Clamping Method is used for most of the timber elements (14 per box). First, the robotic arm attaches robotic clamps to the timber structure, one by one, at the location of the mating joints. Then, the robotic arm changes to a gripper tool to pick up a timber element and position it into the clamp jaws. Then, the robotic arm and the clamps move synchronously to close the joints. Finally, the robotic arm returns the gripper to storage and detaches the clamps, one by one, from the timber structure. The number of clamps used depends on the number of mating joints, and the clamp type (CL3 or CL3M) needs to match the joint's intersecting angle.

The Joint Screwing Method is used for the interlocking key elements (4 per box). First, the robotic arm picks up a timber element using a gripper tool. The operator

[1] Planning horizon refers to how far ahead the planning looks, it can refer to the number of actions performed or duration of time. In this work, it refers to the number of discrete timber elements being assembled.

manually attaches screwdrivers to the timber element held by the robot. Then, the robotic arm brings the timber element towards the structure, the screwdriver begins rotation, and the mating joints are pulled together by the screws. Finally, the robotic arm returns the gripper to storage and detaches the screwdrivers from the timber structure.

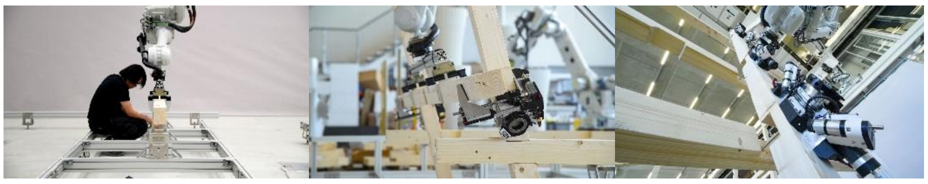


Fig. 4. Photos showing a snapshot of the three assembly methods: Ground Connection (left), Joint Clamping (middle), and Joint Screwing (right)

3 Challenge / Problem

Robotic planning without using TAMP can be generally decomposed into two separate stages. In the first stage, a list of actions is composed manually or automatically using computational logic such as flowcharts. In the second stage, the robotic motions between each target are planned using motion planners.

Using the Joint Clamping Method of our case study as an example, if a flowchart is used to compose the action template, it would consist of three phases: Attach Clamps, Clamping, and Detach Clamps (see Figure 5a).[2] While this logic may be simple to understand, the resulting list of actions could be more efficient. Consider the case when two or more beams are assembled. It would be inefficient to detach the used clamps one by one and bring them back to storage, only to pick them up from storage again and attach them to a new location (see Figure 5b). In terms of saving the number of steps and assembly time, it would be more efficient to detach a clamp from the structure and bring it to the new location without putting it down (see Figure 5c).

Unfortunately, different beams require different amounts of clamps and different clamp types. A direct transfer of clamps from the previous location to the next location is not always possible. Some of the clamps will have to stay on the structure for a longer period before it is used again. However, with the two-stage planning approach, there is no way to ensure that the clamps are still accessible by the robot at a later step.[3] If we attempt to create an action plan that will leave the clamps on the structure and retrieve them later, the motion planner in the second planning step has a high chance of getting stuck. This problem is analogous to a game of chess where a player makes a series of moves that result in a checkmate with no way out. The player must either concede defeat or re-evaluate their strategy and backtrack to avoid the hopeless situation. In TAMP, the ability to backtrack is the critical feature that sets it apart from prior work.

[2] The Attach Clamps and Detach Clamps actions would be considered sub-templates. They would be repeated as many times as the number of clamps needed for that specific beam.

[3] The addition of more timber elements may block the access of the clamps that are left on the structure. The clamps can be considered trapped or inaccessible when the motion planner cannot find any robotic trajectory to reach them.

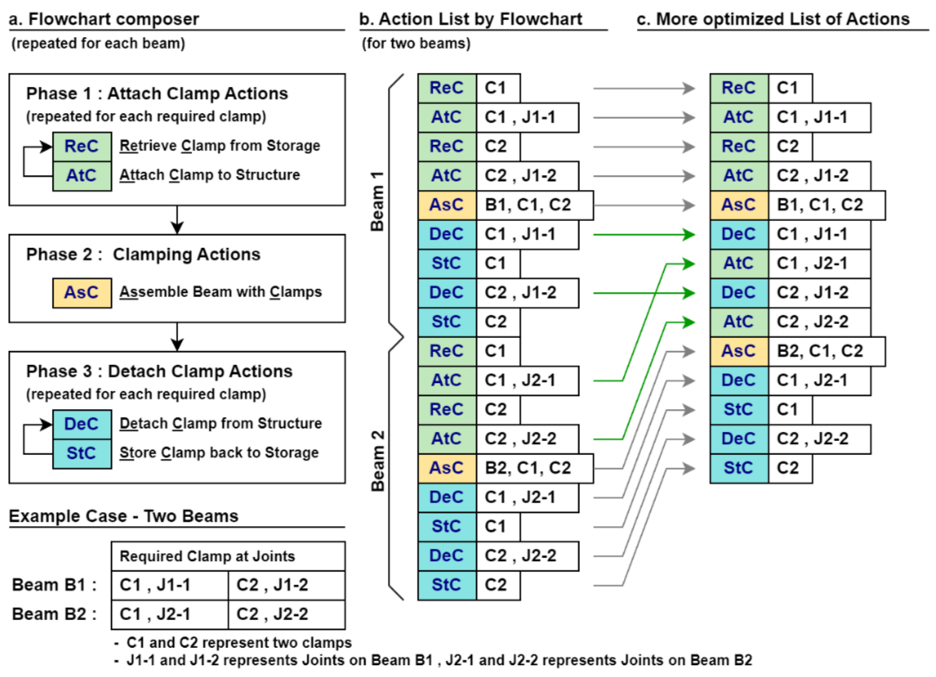

a. Flowchart composer
(repeated for each beam)
Phase 1 : Attach Clamp Actions
(repeated for each required clamp)
ReC Retrieve Clamp from Storage
AtC Attach Clamp to Structure
Phase 2 : Clamping Actions
AsC Assemble Beam with Clamps
Phase 3 : Detach Clamp Actions
(repeated for each required clamp)
DeC Detach Clamp from Structure
StC Store Clamp back to Storage
Example Case - Two Beams
Required Clamp at Joints
Beam B1 : C1 , J1-1 C2 , J1-2
Beam B2 : C1 , J2-1 C2 , J2-2
- C1 and C2 represent two clamps
- J1-1 and J1-2 represents Joints on Beam B1 , J2-1 and J2-2 represents Joints on Beam B2
b. Action List by Flowchart
(for two beams)
Beam 1
ReC C1
AtC C1 , J1-1
ReC C2
AtC C2 , J1-2
AsC B1, C1, C2
DeC C1 , J1-1
StC C1
DeC C2 , J1-2
StC C2
Beam 2
ReC C1
AtC C1 , J2-1
ReC C2
AtC C2 , J2-2
AsC B2, C1, C2
DeC C1 , J2-1
StC C1
DeC C2 , J2-2
StC C2
c. More optimized List of Actions
ReC C1
AtC C1 , J1-1
ReC C2
AtC C2 , J1-2
AsC B1, C1, C2
DeC C1 , J1-1
AtC C1 , J2-1
DeC C2 , J1-2
AtC C2 , J2-2
AsC B2, C1, C2
DeC C1 , J2-1
StC C1
DeC C2 , J2-2
StC C2

Fig. 5. Diagram showing the nonoptimal result of planning actions with a flowchart

4 Task and Motion Planning (TAMP)

Using a TAMP workflow allows the task planner to automatically choose the sequence of actions by verifying the feasibility of its robotic trajectory using a motion planner. If a trajectory cannot be found, the symbolic solver can backtrack and try a different sequence of action, for example, by transferring the clamp earlier or by moving the clamp back to storage after use. The key difference between TAMP versus using flowcharts is that the actions do not have a predefined sequence (see Figure 2).

4.1 Symbolic Planning

TAMP integrates task planning (symbolic) and motion planning. Symbolic planning involves using autonomous techniques to solve symbolic planning and scheduling problems. A *planning problem* is one in which we have some *initial starting state*, which we wish to transform into a desired *goal state* by applying a set of *actions* (Ghallab et al. 1998). In our case study example, the initial state is a list of statements asserting that (1) the timber elements are located at their storage location, (2) each tool is located at its storage location, and (3) the robotic arm is not holding any tool or timber elements. The goal state refers to the same set of statements, except that all the beams are located in their assembled locations.

Predicate - In symbolic planning, states are expressed symbolically, using relationships among objects instead of geometry (such as a transformation matrix). For example, the following PDDL statement `` `(BeamAtStorage b1)` `` can be used to express the fact that a timber element called b1 is currently at the storage location. Another statement, `` `(BeamAtAssembled b1)` `` can be used to describe the same element but located at the assembled location. Each fact-asserting statement is called a *predicate* and expresses a Boolean statement. The initial starting state is simply a list of predicates describing the system at that time. [4]

```
a. (domain.pddl):
(define (domain beam_assembly)
   (:requirements :strips)
   (:predicates (BeamAtStorage ?beam) (BeamAtAssembled ?beam))
   (:action assemble_beam
      :parameters (?beam)
      :precondition (BeamAtStorage ?beam)
      :effect(and (not (BeamAtStorage ?beam)) (BeamAtAssembled
?beam))))
b. (problem.pddl):.
(define (problem minimum_working_example)
   (:domain beam_assembly)
   (:objects b0 b1 b2)
   (:init (BeamAtStorage b0)(BeamAtStorage b1)(BeamAtStorage b2))
   (:goal (and
      (BeamAtAssembled b0)(BeamAtAssembled b1)(BeamAtAssembled b2))))
```

Fig. 6. A minimal working example of a PDDL (a) domain and (b) problem describing beam assembly

Action - In symbolic planning, users do not define a fixed order of actions. The user tells the system what it can do instead of what to do. Actions are composed of two main parts: Precondition and Effect. The *precondition* is a Boolean logical formula that allows the planner to decide if an action can be applied based on the current state. For example, a beam must be at the storage location for the "Assemble Beam" action to begin. The precondition can also prevent illogical actions, such as assembling an already-assembled beam. If the planner decides to apply an action, the *effect* of that action describes the addition and removal of predicates from the current state.[5] For example, the effect of an "Assemble Beam" action should add the predicate `` `BeamAtAssembled b1` `` and remove `` `BeamAtStorage b1` ``. Therefore, the result of applying an action is a new state that is ready to apply another action. The symbolic planner can keep track of the changing states automatically and be smart about applying which actions to reach the goal state. However, it is the user's responsibility to ensure that actions do not create an incoherent state. For example, a beam cannot be located in two places simultaneously.

[4] In planning all unknown values are assumed to be false. That is to say that if we do not know the value of a predicate, then we assume it to be false. This is known as the "closed world" assumption. ("The AI Planning & PDDL Wiki" n.d.)

[5] When a predicate is removed, it means that the statement is now false.

Parameters - PDDL actions are parametrically defined, using variables to express their preconditions and effects.[6] The example in Figure 6a shows a simplified `` `assemble_beam` `` action from the case study with a parameter `` `?beam` ``. The same parameter is used in the precondition and effect.

4.2 Encoding Assembly Problem with PDDL

In PDDL, a problem is represented using a *domain file* and a *problem file*. The *domain file* contains the predicates and actions of the robotic process. Once established, this file remains reusable for different problems as long as the process logic is unchanged. In our case study problem, one domain file is shared for the three problem files corresponding to each timber box. Table 2 shows the complete list of actions used in the case study, note that each action is very human-understandable.

Table 2. List of Actions used in the case study

#	Action Name	Parameters
1	`assemble_beam_by_clamping_method`	`?beam ?gripper ?grippertype`
2	`assemble_beam_by_screwing_method`	`?beam ?gripper ?grippertype`
3	`assemble_beam_by_ground_connection`	`?beam ?gripper ?grippertype`
4	`pick_gripper_from_storage`	`?gripper`
5	`place_gripper_to_storage`	`?gripper`
6	`pick_clamp_from_storage`	`?clamp`
7	`place_clamp_to_storage`	`?clamp`
8	`attach_clamp_to_joint`	`?clamp ?beam1 ?beam2 ?clamptype`
9	`detach_clamp_from_joint`	`?clamp ?beam1 ?beam2`

Each of the three problem files in our case study contains objects, the initial state, and the desired goal specific to each timber box. This file contains a long list of predicates (see associated code repository) to declare the number of beams, the joints between the beams, the type of gripper required for each beam, and the type of clamp required for each joint. For integration with a parametric design workflow, this file was generated computationally via scripting.

4.3 Motion planning with PDDLStream

The second step to achieving TAMP is integrating symbolic planning and motion planning. When used separately, motion planning ensures target reachability and avoids unforeseen collisions. However, as discussed in the problem statement, motion planners alone can get stuck if a trajectory cannot be found. TAMP handles this failure automatically by allowing the symbolic planner to try a different action sequence.

[6] The syntax shown in this paper is described in PDDL v1.2 (Ghallab et al. 1998). This paper uses a popular symbolic planner, FastDownward (Helmert 2006), which supports many common syntaxes.

The TAMP implementation used in this paper is called PDDLStream. It introduces the concept of *stream*, an external function that a symbolic planner can call to certify whether a predicate is true. For robotic assembly processes, the checks include reachability (certified by motion planners) and collision (certified by collision checkers). PDDLStream simply acts as a framework that provides a link between the symbolic planner (such as FastDownward (Helmert 2006)) and the streams (the external functions). The interfaces to the stream samplers are defined in a stream file for PDDLStream (see associated code repository). Figure 7 shows an overview of the TAMP planning process.

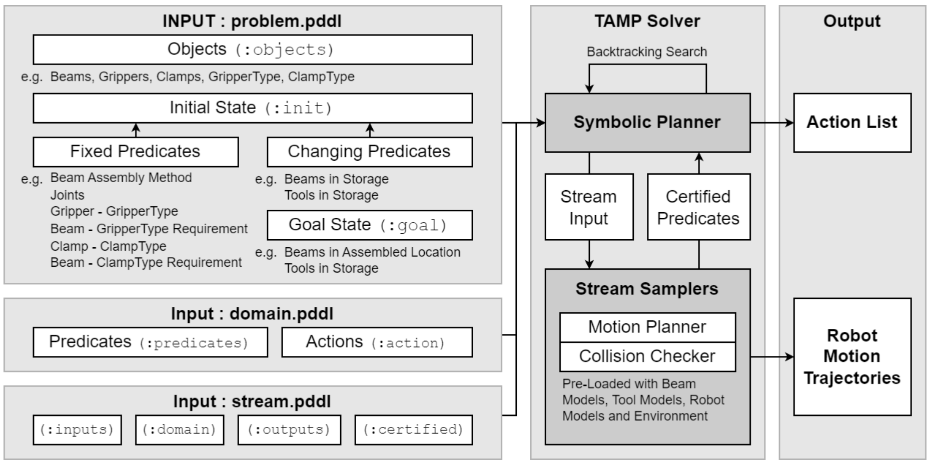


Fig. 7. Overview of the TAMP planning process showing input and output to the solver

Motion Planner (MP) - Separate planners are needed for each action that can get stuck. They are specific to the actions and are preloaded with the necessary geometry for motion planning. For example, a stream for the `BeamAssembly` action should accept the `?beam` parameter as input and check whether a trajectory exists to move that beam to the final assembled location. Only self-collision (the robotic arm, the gripper, and the timber beam) is considered during MP because collisions with other beams and tools are checked later. If the robot can reach the targets, the generated trajectory is stored as a predicate to assert that the corresponding action has a trajectory. For example, `` `(BeamAssemblyTraj ?beam ?traj)` `` can represent the fact that `trajectory ?traj` is available for beam `?beam`.

Collision Checker (CC) - Collisions between the movable beams and tools and the robot are checked separately using a CC stream.[7] Figure 8 shows the `AssembleBeam` action (as seen in Figure 6) where MP and CC streams are added to the precondition.

[7] The movable objects are checked separately from the motion planning steam because the position of these objects is managed by the symbolic planner. More details related to this implementation can be found in (Garrett, Lozano-Pérez, and Kaelbling 2020).

The `` `exists(?otherbeam)` `` clause checks whether any other assembled beams are in collision.

```
:precondition(and
   (BeamAtStorage ?beam)
   (BeamAssemblyTraj ?beam ?traj)
   (not (exists (?otherbeam) (and
      (BeamAtAssembled ?otherbeam)
      (BeamAssemblyInCollision ?traj ?beam ?otherbeam)
   )))
```

Fig. 8. Precondition with motion planning and collision checking streams

5 Experimental Setup

Six experiments with increasing complexity were provided below to demonstrate how a PDDL problem can be formulated in an incremental manner. The first experiment is a minimal working example without considering tool and joint requirements while the final experiment considers all fabrication constraints, and is equivalent to the case study process. Due to page limitations, the complete PDDL code can be found in the code repository associated with this paper. [8]

E01 beam assembly - Minimal working example with only one action to assemble beams. No grippers or clamps. No order is enforced. Symbolic planning only.

E02 joint partial order - The joints are defined as an intersection between two beams. A precondition is added to ensure the joint's first beam is assembled before the second beam can be assembled (a.k.a. partial ordering). Symbolic planning only.

E03 gripper switch - The gripper requirement is added to the assemble beam action. Additional actions for the robot to pick up and release grippers are added. Three gripper types are available. Symbolic planning only. The planning result shows that tool change actions are inserted wherever necessary. The assembly order differs from the previous result as the planner tried to reduce the number of tool changes.

E04 assembly stream - The symbolic logic is the same as E03. However, the `As-sembleBeam` action is validated with MP and CC streams as described in the TAMP section above. This is the first TAMP example. The resulting action sequence is longer than E03. The interpretation is that E03 did not consider MP and CC checks, now that they are considered, the assembly sequence cannot be as optimized. The MP and CC checks can also be considered extra constraints for the TAMP solver, which means the solver has less freedom to choose a shorter action sequence.

E05 clamp transfer - `AssembleBeam` actions are separated according to the three assembly methods of a beam. For the assembly action with *Joint Clamping*, clamps are required at the mating joints. The correct clamp type must be present on these joints before the beam can be assembled. Four clamp manipulation actions are added (the last four rows in Table 2). Symbolic planning only. Note that there are two clamp devices available for each of the two clamp types. The resulting action sequence shows the

[8] https://github.com/yck011522/robarch_pddl

clamps transferred between storage and the joints. However, in many cases, the clamps are transferred directly between the joints.

E06 clamp stream - The actions are similar to E05, but the designer defines the assembly order (manually, by intuition) to improve stability during construction. In addition, MP and CC streams validate the AssembleBeam, the AttachClampToJoint, and the DetachClampFromJoint actions. This experiment is the complete TAMP equivalent to the case study. The resulting action sequence is longer than that of E05. The rationale is similar to the one offered for E04.

6 Results and Discussions

Each experiment (E01 to E06) is performed with three problem files corresponding to the three boxes in the case study. The planner's goal is to minimize the number of actions to reach the assembled state. The results are broken down in Table 3. “Assemble Actions” include all three assembly methods (#1 - #3 in Table 2). “Gripper Actions” include picking and placing grippers (#4, #5 in Table 2). The main highlight is the “Clamp Actions”, which include the four actions that pick and place the clamps (#6 - #9 in Table 2). Although the topology of the three boxes is the same, their geometry is different. Therefore, their geometrical constraints (such as robot reachability) that affect the choice of tasks are also different. This is reflected in the result where the number of planned tasks differs for each of the three boxes.

Table 3. Number of planned actions in the six experiments with three problem sets. The results of the three boxes (left, mid, right) are separated by a + symbol.

Planning Experiment	Assemble Actions (Count)	Gripper Actions (Count)	Clamp Actions (Count)	Planning Time (s)
E01	20 + 20 + 20	-	-	<1 + <1 + <1
E02	20 + 20 + 20	-	-	<1 + <1 + <1
E03	20 + 20 + 20	4 + 8 + 4	-	<1 + <1 + <1
E04 (TAMP)	20 + 20 + 20	4 + 8 + 4	-	113 + 139 + 145
E05	20 + 20 + 20	28 + 28 + 26	56 + 56 + 56	2 + 3 + 3
E06 (TAMP)	20 + 20 + 20	34 + 38 + 32	56 + 56 + 56	54 + 32 + 52
Flowchart	20 + 20 + 20	40 + 40 + 40	96 + 96 + 96	<1 + <1 + <1

The optimization benefit of TAMP can be seen in Table 4, which compares the result of experiment E06 and what would have been required if a flowchart method was used. The execution time of each action was measured during the actual robotic construction. The estimated times listed in the table are calculated using the average value for each action type (calculated separately for each of the three box structures). The significantly reduced number of clamp transfer actions (see Figure 9) resulted in a total time reduction of 16%. Note that a similarly optimized action sequences are almost impossible for a human to create manually because it would be highly prone to error.

On the other hand, we found that the TAMP solver planning time is highly sensitive to the number of beams to be assembled, especially for our process that has a high branching factor (many possible actions at each step). The partial ordering constraint introduced in E02 helps reduce the branching factor. However, in E06 we have to specify the full assembly order to reduce the branching factor for the search to be completed in reasonable time. One explanation is to imagine the number of possible moments for a clamp to be attached to the structure before it is needed by a beam, and after the clamp is used, it can also be detached or relocated at any time after that. Future research can study how to reduce planning time by smarter problem formulation.

Table 4. Time comparison of experiment E06 (TAMP with a fixed order) and the flowchart planning method

	Execution Time for Actions (min)			
Planning Experiment	Assemble Actions	Gripper Actions	Clamp Actions	Total
E06 (TAMP)	179+194+185	93+114+107	194+189+191	1446 (-16.3%)
Flowchart	179+194+185	100+111+119	284+252+303	1729

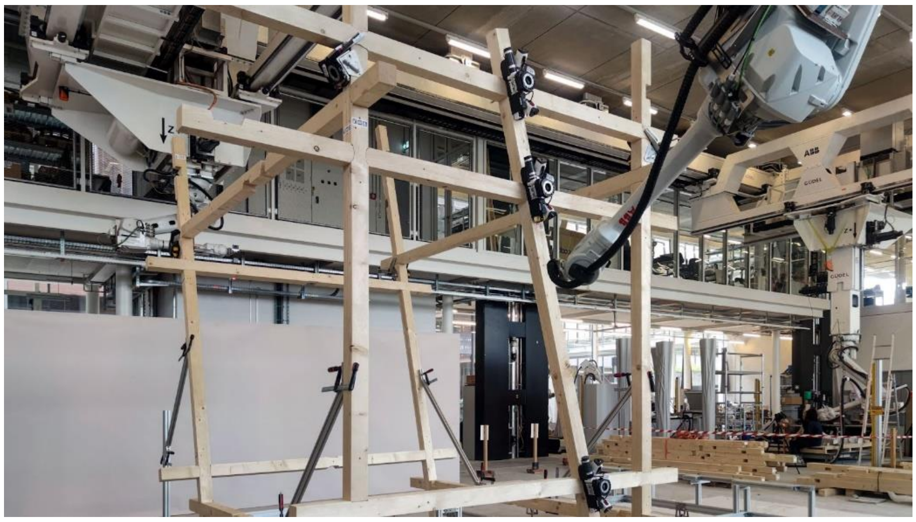


Fig. 9. Image of a beam being assembled by screwdrivers. Note that some clamps remained on the structure from the previous assembly.

7 Conclusion

The integrated Task and Motion Planning (TAMP) workflow using PDDLStream overcomes the limitations of the previous two-stage planning methods such as flowcharts. It allows the planner to explore different task sequences while considering the motion planning constraints. The automatic generation of action plans resulted in a significant execution time reduction without manual programming efforts.

Using our case study, we have introduced an incremental programming approach to formulate complex assembly problems with PDDL. While the incremental addition of actions and constraints makes programming easier, it also creates modular PDDL code snippets that can be reused for different assembly problems. For example, future work can consider adding a stream for the planner to generate and validate grasp poses (i.e., where the workpiece is held), or checking the partially assembled structure to avoid instability.

8 Acknowledgment

The authors would like to acknowledge the contribution of Caitlin Mueller for her insightful discussions and scholarship support for Y. Huang; the design and engineering team behind the CantiBox Pavilion; and the Robotic Fabrication Laboratory team (ETH Zurich). This research was supported by the Swiss National Science Foundation (NCCR Digital Fabrication agreement number 51NF40-141853), and Y. Huang was supported by an ETH Postdoctoral Fellowship.

References

1. Garrett, Caelan Reed, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2021. "Integrated Task and Motion Planning." Annual Review of Control, Robotics, and Autonomous Systems 4 (1): 265–93. https://doi.org/10.1146/annurev-control-091420-084139.
2. Garrett, Caelan Reed, Tomás Lozano-Pérez, and Leslie P. Kaelbling. 2020. "PDDLStream: Integrating Symbolic Planners and Blackbox Samplers." In International Conference on Automated Planning and Scheduling (ICAPS). https://arxiv.org/abs/1802.08705.
3. Ghallab, Malik, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. "PDDL The Planning Domain Definition Language." https://planning.wiki/_citedpapers/pddl1998.pdf.
4. Hartmann, Valentin N., Andreas Orthey, Danny Driess, Ozgur S. Oguz and Marc Toussaint. 2023. "Long-Horizon Multi-Robot Rearrangement Planning for Construction Assembly" IEEE Trans. Robot., vol. 39, no. 1, pp. 239–252. doi: 10.1109/TRO.2022.3198020.
5. Hartmann, Valentin N., Ozgur S. Oguz, Danny Driess, Marc Toussaint, Achim Menges. 2020. "Robust task and motion planning for long-horizon architectural construction planning" IEEE/RSJ 2020 International conference on intelligent robots and systems (IROS), pp. 6886–6893. doi: 10.1109/IROS45743.2020.9341502.
6. Helmert, Malte. 2006. "The Fast Downward Planning System." Journal of Artificial Intelligence Research (JAIR) 26: 191–246.
7. Huang, Yijiang, Victor Pok Yin Leung, Caelan Garrett, Fabio Gramazio, Matthias Kohler, and Caitlin Mueller. 2021. "The New Analog: A Protocol for Linking Design and Construction Intent with Algorithmic Planning for Robotic Assembly of Complex Structures." In Symposium on Computational Fabrication. SCF '21. Association for Computing Machinery. https://doi.org/10.1145/3485114.3485122.
8. Ögren, Petter, and Michele Colledanchise. 2018. Behavior Trees in Robotics and AI: An Introduction. Boca Raton: CRC Press. https://doi.org/10.1201/9780429489105.
9. Sherkat, Shermin, Lior Skoury, Andreas Wortmann and Thomas Wortmann. 2023. "Artificial Intelligence Automated Task Planning for Fabrication" Advances in Architectural Geometry 2023, De Gruyter, pp. 249–260. doi: 10.1515/9783111162683-019.

10. Tanadini, Davide, Giulia Boller, Pok Yin Victor Leung, and Pierluigi D'Acunto. 2023. "Plastic Design of Bespoke Interlocking Timber-to-Timber Connections for Robotic Assembly." In World Conference on Timber Engineering (WCTE 2023), 4399–4408. Oslo, Norway: World Conference on Timber Engineering (WCTE 2023). https://doi.org/10.52202/069179-0573.
11. Wagner, Hans Jakob, Martin Alvarez, Abel Groenewolt and Achim Menges. 2020. "Towards digital automation flexibility in large-scale timber construction: integrative robotic prefabrication and co-design of the BUGA Wood Pavilion," Constr Robot, vol. 4, no. 3, pp. 187–204, doi: 10.1007/s41693-020-00038-5
12. "The AI Planning & PDDL Wiki." n.d. Planning.Wiki - The AI Planning & PDDL Wiki. Accessed October 5, 2023. https://planning.wiki/.