

# Automated motion planning for robotic assembly of discrete architectural structures

Yijiang Huang<sup>1</sup> · Caelan R. Garrett<sup>2</sup> · Caitlin T. Mueller<sup>1</sup>

Received: date / Accepted: date

**Abstract** While robotics for architectural-scale construction has made significant progress in recent years, a major challenge remains in automatically planning robotic motion for the assembly of complex structures. This paper proposes a new hierarchical planning framework to solve the assembly planning problem for architectural discrete structures, which usually have a long planning horizon and 3D configuration complexity. By decoupling sequence and motion planning, the planning framework is able to efficiently solve the assembly sequence, end-effector poses, joint configurations, and transition trajectories for assembly of spatial structures with nonstandard topologies, which hasn't been demonstrated in previous literature. Together with the algorithmic results, this paper also presents an open-source and modularized software implementation called *Choreo* that is machine and application-agnostic. To demonstrate the power of this algorithmic framework, three case studies, including real fabrication and simulation results, are presented to show *Choreo*'s application on spatial extrusion.

**Keywords** robotic assembly planning · task and motion planning · digital fabrication

Yijiang Huang  
E-mail: yijiangh@mit.edu

Caelan R. Garrett  
E-mail: caelan@csail.mit.edu

Caitlin T. Mueller  
E-mail: caitlinm@mit.edu

<sup>1</sup>Building Technology Program, Department of Architecture, Massachusetts Institute of Technology, Cambridge, MA, 02139 USA

<sup>2</sup>Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, 02139 USA

## 1 Introduction

Architectural robotics has proven a promising technique for assembling nonstandard configurations of building components at the scale of the built environment, complementing the earlier revolution in generative digital design. In recent years, the sharp reduction of industrial robotics' cost has made investment in these advanced manufacturing machines increasingly accessible, converting the industrial robot into a cost-efficient tool to materialize bespoke design [72].

However, despite the advantages of the decreasing hardware cost, dexterity, and precision of these multi-axis machines, the time investment in solving the construction sequence and associated robotic motion grows increasingly with the topological complexity of the target design. The level of automation in this design-assemble workflow is still comparably low, due to the technical challenge of finding a feasible assembly sequence and generating trajectories for the multi-axis robots. While transitioning between a digital design model and machine code for a 3-axis gantry machine is easy and direct, for multi-axis robots, gaining fine levels of control and bypassing the complexity of generating collision-free robotic trajectories is much more nuanced and subtle, which requires significant effort.

Existing investigations in the field of architectural robotics often involve manual planning of a path guidance for the robot's end effector, followed by tedious diagnosis for potential problems in a trial-and-error manner. This slow and convoluted workflow deviates from the initial purpose of having such a digital design-assemble workflow: to forge a smooth and direct transition from digital design to real-world machine materialization; instead, the current process often requires a complete re-program for the robot whenever the target geometry has a small change. This technical challenge in the assembly planning and programming of the robots congests the overall digital design/production pro-

cess and often confines designers to geometries with standard topology with repetitive patterns. In order to close this gap and enable more possibilities for discrete architectural robotic assembly, an automated assembly planning system is needed, which calls for a more systematic and explicit computational exploration of assembly constraints and robotic motion planning.

This work presents a new algorithmic framework for robotic assembly planning, which embodies an hierarchical algorithm to integrate assembly sequence and motion planning. The planning framework is implemented as a flexible assembly planning tool, called Choreo, that allows users to input unconstrained spatial structures, and receive an automatically generated feasible assembly sequence and robotic trajectory. Case studies are presented to show the computational planning system's power in enabling automated planning for robotic assembly of complex structures with non-standard topologies, which hasn't been shown possible before.

### 1.1 Scope: Discrete spatial structures

Discrete spatial structure's definition broadly includes all 3D structures that consist of individual *elements*, which are connected to each other via structural joints and behave as a system when load is applied. The assembly planning problem is defined as: given a discrete spatial structure's design model, the robot needs to be assigned a coordinated sequence of transition and assembly actions, to manipulate raw or sorted individual elements in a specific order to construct the designated design. There are essentially two main classes of robotic assembly applications: (1) *spatial extrusion* (also called spatial 3D printing) and (2) *spatial positioning* (also called pick-and-place). In this work, for demonstration purposes, all the algorithmic framework description and case studies use spatial extrusion of 3D trusses with a fixed-base industrial robotic arm as a concrete problem instance. However, the presented planning framework can be generalized to multiple robotic manipulators or mobile manipulation with a straightforward extension.

### 1.2 Contributions and organization of paper

The contributions of this paper are summarized as follows:

- This paper embodies the first attempt in the field of architectural robotics to formally characterize, formulate, and algorithmically solve the architectural assembly planning problem. A new hierarchical assembly planning algorithmic framework is proposed to harness the assembly planning problems that have long planning horizon and three-dimensional complexity.

- An open-source, modularized, and highly customizable implementation of the proposed planning framework is presented. The planning software, called Choreo, is adaptable to various assembly applications and hardware setups, and can be smoothly fitted into existing digital design workflow. Case studies on applying the planning system to spatial extrusion of 3D trusses with irregular topologies are presented to show Choreo's computational power and efficiency.

This paper starts with a review of existing efforts in the field of automatic assembly. Section 3 presents the assembly planning algorithm, starting from model input (section 3.2), and goes through layers of its planning hierarchy: first sequence planning layer (section 3.3) and then motion planning layer (section 3.4). A post-processing module is presented in section 3.5 to increase usability and adaptability of the computed results. Section 3.6 presents the engineering ideas behind the implementation of the assembly planning tool Choreo. Section 4 shows three case studies with computation statistics and fabrication results to demonstrate Choreo's efficiency and power. Finally, section 5 concludes the paper by noting out limitations and suggesting areas of future research.

## 2 Related work

In response to the need for automated planning in architectural robotic assembly, this section summarizes previous efforts in the area of automatic assembly. Key research from five distinct fields, (1) robotic assembly for architecture, (2) classic assembly planning, (3) computer graphics, (4) manipulation planning, and (5) task and motion planning is presented, with contributions and drawbacks highlighted. The aim of this section is to demonstrate why an integrated planning system, which combines features from all of the above fields, is needed for robotic assembly to be fully accessible to architectural designers.

### 2.1 Robotic assembly for architecture

The exploration of robotic assembly in different architecture-scale application contexts, such as spatial positioning and extrusion [20], has focused on the design of application-specific processes and associated hardware systems. In all of these applications, researchers have encountered a similar problem: the generation of feasible robotic trajectories that do not collide with objects in the workspace [50][13][64]. Current solutions to this problem typically involve an intuition-based trial-and-error method. For a given robot configuration during assembly, designers manually specify end-effector poses on the assembly geometry to achieve linear end-effector

movement. For transition trajectories, designers manually generate guiding curves for the end-effector to follow, which hover over the workspace within a safety distance. Utilizing industrial robot's built-in commands like Linear movement (LINE) or Point-To-Point (PTP) [73], users rely on the built-in interpolation method to translate end-effector assignment to joint trajectories that are free of collisions, respect joint limits, and avoid singularities. As a result, this requires significant effort to diagnose the planning failure in a trial-and-error manner. Software packages exist to support this trial-and-error procedure by simulating robotic motion (such as HAL [60] and KUKA|PRC [2]), but these tools can only simulate/test a robotic trajectory based on TCP planes and joint configurations input, without the ability to automatically plan a collision- and kinematics-aware trajectory. Because of this, these tools currently support a sub-optimal manual planning manual process.

A recent attempt on harnessing this problem from Sondergaard et al. uses an incremental search algorithm to find a construction sequence for a large-scale topology-optimized space frame, while guaranteeing the existence of node-specific, collision-free motion of the assembly parts [64]. However, the reachable robotic configuration space is not considered during construction sequence searching, and robots trajectories are later found by inserting custom unwinding positions. While this geometry and machine-specific approach is feasible for designs with simple and sparse topologies, the construction sequence and robotic motion planning is much more nuanced for designs with denser material distribution and non-standard topologies.

In recent years, there has been some success in tackling this planning problem by using a single-query robotic motion planning algorithm [50] or an online control strategy [19] to compute transition trajectory between pre-programmed assembly primitives. However, the construction sequence in the existing work was still assigned manually, taking advantage of either the sparse or the repetitive topological nature of the target geometry. Recent progress in the field has moved towards sensor-enabled online robotic control [32][19]. However, a global planning tool, which combines assembly sequence planning and associated robotic motion planning, has not been developed. As pointed out in [19], the combination of an autonomous control scheme with a "higher-level planner" that is "able to negotiate cluttered environment" is a key step to enable robotic assembly systems to operate safely in densely populated workspaces [13].

## 2.2 Classic assembly planning

There is a large body of work in classic assembly planning, also called mechanical assembly planning, dating back to the 1980s and 1990s with the influential work by Homem de Mello [7], Wilson [74], and others [5][75]. The focus of

this line of research was generating of sequences that allow robots in an industrial assembly line setting to assemble a product based on design CAD files. The primary concerns were to satisfy low-level constraints such as mutual blocking during assembly, mating constraints, and tolerances. These methods focused on the product itself but not the robots that perform the assembly. This might be a safe assumption for this domain since the environment and the assembly robots can be specifically engineered to the task at hand. However, given a specific robotic setup and a set of mechanical parts to be assembled, the robot's configuration space significantly constrains the reachable end-effector poses [44]. Addressing this requires considering the robot and the assembly object simultaneously in the assembly planning (see section 2.4).

The key contributions in the area of classic assembly planning include (1) the mathematical formulation of the assembly planning problem (2) the proposal of several compact and efficient representations of intermediate assembly states and (3) the theoretical characterization of computational hardness for different classes of these problems. The reader is referred to section 2.3.2 of Heger's thesis [25] for a more extensive overview of the problems explored and methods used in classic assembly planning. Nearly all work in classic assembly planning dates back to the 1990s and earlier. Still, many problems are left unsolved, such as how to integrate the constraints of the assembler, the robots, into a single planning scheme.

## 2.3 Computer graphics

Existing assembly-related research in computer graphics can be summarized into two categories: (1) computational design methods with assembly sequence as a physical constraint (2) fabrication techniques that require resolving assembly sequence for input objects.

In contrast to classical assembly planning, where assembly planning works like a technical assessment tool for arbitrarily input mechanical parts, computational assembly design focuses on the generation of interesting objects with the constraint that there exists an assembly sequence. Existing work considers objects with specific features, such as 3D polynomino puzzles [43], 3D burr puzzles [77], voxelized recursive interlocking puzzles [65], furniture with interlocking joints [14], and planar interlocking pieces [61].

The other line of research invents new hardware systems with assembly sequencing algorithms to enable new fabrication and assembly opportunities. Existing efforts focus on adding more degrees of freedom to the 3D printing technology. WirePrint [49] proposes an efficient way to print wireframe meshes, where edges in the mesh are directly extruded in 3D space. A wireframe of a model is generated by slicing it horizontally and filling each slice with zigzagging filaments. This approach is limited in the types of meshes that

can be printed. To improve flexibility, Peng et al. introduce a 5-DOF printer that modifies a standard delta 3D printer by adding two rotation axes to the print bed [51]. Following up this work, Wu et al. present a printing sequence planning algorithm for this 5-DOF printer [76]. Huang et al. and Yu et al. present a constrained graph cut algorithm to tackle the printing sequence planning problem for robotic spatial printing [30][78], which marks the first algorithmic attempt in attacking the sequence planning for printing frames with arbitrary topologies. However, their method abstracts away the robot's kinematics and instead uses an ad-hoc method to generate feasible guiding curves for the robot's end effector to follow. This results in slow computation and lacks any trajectory feasibility guarantees. Recently, Dai et al. augmented the degrees-of-freedom of the print bed of layer-based 3D printing technology by using an industrial robot to hold the bed. They also presented shape decomposition algorithms with support and robotics constraints [4].

In summary, existing work in computer graphics takes the existence of an assembly sequence as a constraint for designing objects with specific features and customized 3D printing technology. A general system that integrates assembly sequence searching and robotic motion planning has not been presented in the literature.

*Related problems* There is other work that is not produced in computer graphics community, but also addresses the assembly sequence planning problem to connect the digital design and the physical assembly process. Tai presents a computational design framework to design interlocking wooden frames while considering the assembly sequence [69]. In the context of 3D printing in bio-engineering, Gelber et al. presented a heuristic backtrack searching algorithm to generate printing sequence to enable micro-scale freeform 3D printing on a purpose-built isomalt 3D printer [17]. They were the first to identify that joint positioning errors are caused by beam compliance and include it as a cantilever constraint in the sequence searching process [17][18]. This finding influenced the nodal printing orders routing part of the sequence planning module presented in this work (section 3.3.2).

## 2.4 Manipulation planning

The robotic planning community has developed many approaches for motion planning that identify trajectories by searching in the continuous space of robot joint angles. Recent approaches perform this search using either sampling [41] or optimization [55][33][59]. In manipulation planning, the goal is not only to move robot without colliding with objects in the environment, as in classical motion planning, but also to contact, operate, and interact with objects in the world. Early treatment of this problem uses a *manipulation graph* to decompose planning for one robot to one ob-

ject into several problems that each require moving between connected components of the combined configuration space [1][62]. This work observes that solutions are alternating sequences of *transit* and *transfer* paths, which corresponds the robot moving with its hands empty and while holding an object. Hauser et al. identify a generalization of manipulation planning problem as *multi-modal motion planning*, i.e. motion planning for systems with multiple modes, representing different sub-manifolds of the configuration space subject to different constraints [23][24].

Rearrangement planning is a special instance of pick-and-place planning where all objects have explicit goal poses. These problems are very similar to the robotic assembly planning problems addressed in this work, where object goal poses are specified in the input design model. Stilman et al. first introduced a version of the rearrangement problems called *navigation among movable obstacles* (NAMO), where the robot must reach a specified location among a field of movable obstacles [66][67]. They provide a greedy backchaining algorithm for solving *monotone* problem instances, where each object need only be moved once. Extending this work to non-monotone problem instances, Krontiris and Bekris provided an algorithm that constructs a probabilistic roadmap (PRM) [35] in the combined configuration space, using the algorithm of Stilman et al. as connection primitive [36][37].

Dogar et al. propose a formulation of multi-robot grasp planning as a constraint satisfaction problem (CSP) [10]. They attempt to find short plans that requires few regrasps. However, they assume an assembly sequence is given and does not consider reachability constraint between assembly configurations.

## 2.5 Task and motion planning

While motion planners deal with geometric constraints in high-dimensional configuration spaces, they do not consider abstract features of the domain, i.e. they can plan how to move the robot's joints to pick up an object but cannot decide the order of tasks to satisfy certain constraints. In contrast, the artificial intelligence (AI) planning community considers problems that are discrete but require many types of actions to be performed over long horizons [47][27]. Recent work in task and motion planning (TAMP) [11][71][39][15] combines AI and motion planning to simultaneously plan for discrete objectives as well as robot motions. This work aims to enable robots to operate in applications such as cooking, which require discrete choices of which objects to grasp or cook as well as continuous choices of which joint angles and object poses can physically perform each task. A key challenge is that often physical constraints such as collision, kinematic, and visibility constraints can restrict which high-level actions are feasible. Readers are referred to [16] for a more complete review of the work in this area.

Lagriffoul et al. propose a constraint-satisfaction approach to interleave the symbolic and geometric searches. They focus on limiting the amount of geometric backtracking [40]. Lozano-Pérez and Kaelbling take a similar approach but leverage CSPs operating on discretized variable domains to bind free variables [45]. The sequence planning module (section 3.3) proposed in this work adopts a similar technique by using CSP to bind free geometric variables on a plan skeleton. However, it relaxes the requirement on feasible whole paths' existence and trades the algorithm's completeness for scalability.

## 2.6 Architectural robotic assembly planning: Unique challenges and unmet needs

Architectural assembly planning is a subclass of high dimensional robot manipulation problems, or more generally, task and motion planning (TAMP) problems, which requires planning a coordinated sequence of motions that involve extrusion, picking, placing or manipulating specific type of construction materials, as well as moving through free space. Architectural robotic assembly problems differ from typically studied TAMP problems in three key aspects. First, the discrete horizon of the assembly problems is much longer than many TAMP benchmarks [38], which often only require manipulating a couple of objects. Because each element must be assembled once and the goal object poses are specified by the input design geometry, the assembly horizon is known in advance. Thus, assembly planning requires identifying an order for object manipulation, fitting this order to a fixed plan skeleton, and binding the required geometric parameters. In contrast, TAMP problems generally have unsettled action plans - it is not initially clear which actions are needed and in which order to perform these actions to complete a task, and thus the planning horizon can be arbitrarily long.

Second, assembly problems involve physical constraints such as stiffness and stability that are not typically found in TAMP benchmarks. These constraints impact many state variables at once, making them challenging to effectively incorporate in many discrete task planning algorithms. Rather than directly using existing TAMP algorithms, a specialized system is developed in this research that incorporates several existing ideas but, because of its specialization to assembly planning, can scale to complex models.

Third, common task specification languages for planning systems, such as planning domain Planning Domain Definition Language (PDDL) [47] are not intuitive for architects and designers. The requirement of specifying task domains, predicates, action's preconditions and effects departs from the architectural language of shape and geometry, and thus creates a gap between an architect's geometric model and robotic task specification for planning. This gap in the

modeling interface inhibits these algorithms from being easily adapted to architectural robotic assembly applications.

In summary, there is a rich literature of work related to robotic assembly for architecture, ranging from theoretical research in robotic task and motion planning to examples of built work of considerable intricacy. However, the field is nevertheless lacking an integrated, general-purpose method that can be applied systematically across many assembly project types while also handling the geometric and topological complexity of contemporary architectural design. This work addresses this gap by presenting a new assembly planning algorithm framework and a modularized implementation that is adaptable to various assembly applications and hardware setups. Although the algorithm described in this paper is more specialized than most of the TAMP approaches, the ability to scale to problems with much longer planning horizon and a larger branching factor is the key focus of this research.

## 3 Assembly planning framework

This section introduces a new computation framework that can efficiently handle the problem of robotic assembly planning. First, section 3.1 gives a conceptual overview of the entire framework's hierarchy and introduces its three main modules. Then, detailed problem formulations and associated solution strategies are described for the *sequence planning module* (section 3.3), the *motion planning module* (section 3.4) and the *post-processing module* (section 3.5).

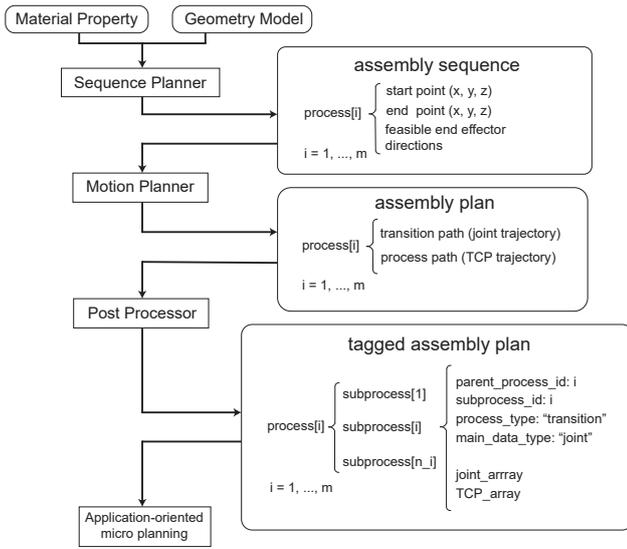
### 3.1 Conceptual overview

A general robotic assembly planning system should have following capabilities:

1. Take general discrete structures as input, with minimal possible restriction on the geometry
2. Generate an assembly sequence and associated poses for the assembly operation, while satisfying geometric, kinematic, and structural constraints
3. Solve for collision-free transition trajectories between assembly operations
4. Be robot and hardware-agnostic
5. Provide an interface to integrate generated trajectory into hardware control scheme

Creating an assembly planning system that meets the above capabilities is a challenge because of (1) the computational complexity inherent from assembly planning problem (2) the engineering complexity for creating an interface bridging design to robotic planning.

To address the computational and engineering challenges posed by the assembly planning problem, this work proposes a planning framework that uses a hierarchical task and



**Fig. 1** Overview of the assembly planning framework.

motion planning approach. The proposed planning framework incorporates three key modules as shown in figure 1. Instead of searching for a solution considering all parts of the searching tree at once, the proposed approach identifies and breaks the problem into two isolated sub-problems, sequence planning and motion planning. This separation cuts the sequence-dependent ties between the sequence and motion planning subproblems, narrowing down the search space. First, the sequence planner (section 3.3) takes a discrete structure as input and outputs the assembly sequence and associated feasible end-effector poses. Next, given the fixed assembly sequence and focused end-effector directions, the motion planner (section 3.4) chooses the end-effector pose for each assembly and plans for the robot’s entire joint trajectory during and between assemblies. Finally, the post processor (section 3.5) tags the computed trajectory plan with associated assembly information and outputs a complete assembly plan. After this is completed, the user can optionally use the post processor’s tagging system to insert tool path modification or control commands to fine-tune the hardware control.

These modules, along with the framework inputs and outputs, are described in greater detail in the following sections. An example problem of using a fixed-base 6-axis robot to spatially extrude a discretized linear frame structure is used to illustrate the details of each module, but the system is general and can also apply to other robotic assembly tasks, for example, spatial positioning of discretized surfaces or volumetric elements.

**Assumptions** In this paper, the robot is assumed to work in a fully observable and deterministic environment. The planning starts with an assembly plan skeleton, or action sequence, that has a pre-defined repetitive pattern on the

actions: for example, *pick element  $o_i$  from material rack - move - place element  $o_i$  at position  $p_i$  - move or extrude element  $o_i$  at position  $p_i$  - move*. The planner needs to assign a correct assignment of object  $o_i$  to each action in the plan skeleton, and bind variables to fully specify robot’s configurations *during* and *between* assembly steps. The generated plan is purely geometric - the computed velocities are not used in the execution. Trajectory’s speed for execution is re-assigned separately by the user after the planning is finished and robot’s position control is carried out by the industrial robot’s controller.

### 3.2 Model input

The assembly planning framework takes as input a 3D model from a designer. The model should represent overall geometry, topology, and discretization for robotic assembly. If started with a continuous form, for example surface or other volumetric structures, discretization can be performed by designer intuition or through an algorithmic meshing or decomposition approach; the framework is agnostic to how this step is carried out. For each discrete element of the input design, the candidate assembly end effector poses on the element need to be specified.

For the discretized linear frame structure, a standard node-member data representation is used. Nodes are described with 3D spatial coordinates in an indexed list. Linear members are described by their start and end node indices. Different cross sections and material properties can be assigned per member index. For spatial extrusion, each linear element specifies a sequence of path points that the end effector’s tip must extrude along. Candidate end effector pose for extrusion is specified by direction sampled in the upper hemisphere in the element local reference frame, together with a rotation angle  $\in [0, 2\pi)$ .

### 3.3 Sequence planning module

A sequence planner takes any discrete geometry as input and solves for the order of the assembly operation and associated feasible end-effector poses. Globally, the sequence planner computes an assignment of objects to each action in the pre-defined plan skeleton, which requires reasoning about the geometric and physical constraints in this discrete search. Locally, in each individual assembly, the planner computes all collision-free end-effector poses, given all the collision objects in the target assembly stage, thus resembling a grasp planner.

This section first identifies the key constraints that arise in the sequence planning problem and formulates the problem as a Constraint Satisfaction Problem (CSP) (Section 3.3.1). Then, a customized solver is proposed (Section 3.3.2)

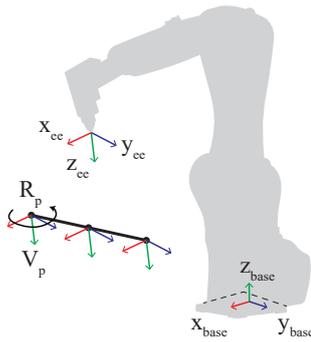


Fig. 2 End effector's frame and trajectory points' frame definition.

that operates using two main techniques: (1) backtracking search with 1-level forward checking and value ordering (2) user-guided model decomposition.

### 3.3.1 Problem formulation

The assembly sequence planning problem requires assigning every assembly action with an element from the model and finding the geometric configurations of a feasible end-effector path for each task. Axis conventions are described in figure 2. The end-effector frame is positioned at the 3D printing extruder's tip. An end-effector's pose is defined by an end-effector frame, which can be uniquely determined by (1) frame's origin, (2) z-axis, and (3) rotation angle around the z-axis. Every trajectory point has a local frame  $\{p\}$  assigned to define the position and orientation of the end effector at that trajectory point. All of these coordinate frames are described in a common reference frame  $\{base\}$ . In the following discussions, let  $n$  denote the total number of elements to be assembled in the model.

The input frame model contains a set of linear elements,  $O_1, \dots, O_n$ . Each element specifies a linear trajectory that the end-effector's tip needs to traverse while extruding material. A discretized representation of the linear trajectory is used, which divides the trajectory into a sequence of points under certain discretization resolution. These points only specify the end-effector poses' origins, which admit a possibly infinite amount of feasible end-effector orientations. To avoid the twisting force that the end effector might exert on the molten plastic beam during extrusion [18], the end effector is required to maintain its orientation when extruding to obtain a straight printing result. Thus, the robot's path for printing an element is determined by (1) the point origins specified by the element's linear path and (2) the orientation of the end effector. For spatial extrusion of 3D trusses, only one assembly action type is considered: extrude. The sequence of assembly actions follows an alternating pattern: *extruder-move-...-move-extrude*. An important simplification is made to eliminate the *move* action concatenating adjacent *extrude* actions in the sequence planning module, which

differs from the general plan skeleton binding approach that enforces full path-existence [45]. In this way, evaluating transition path feasibility is reduced to testing kinematic feasibility and checking collision during assembly. This simplification is equivalent to assuming that if robot has a collision-free kinematic configuration at the start and the end of each assembly step, the transition motion planner (section 3.4.3) can always find a feasible transition trajectory. This assumption can be found in some work in TAMP [40] and is generally valid through all of the performed experiments in this paper. However, in some extreme cases, this assumption may not hold true, resulting in planning failure. Future work involves including a mechanism to recover from such a planning failure.

Each action in the predefined assembly plan skeleton is specified with a constraint variable and a set of geometric parameters. Each constraint variable is a symbol that names an assembly element. Each geometric parameter defines an end-effector pose. To bind these variables, a CSP planner is called to verify if the assembly plan skeleton is satisfiable. The correctness of an assembly plan skeleton is enforced by the constraints, which are expressed as relationships between the assembled elements at each assembly step and the end effector's pose.

To formulate a discrete CSP, it is necessary to specify a set of constraint variables, a discrete domain of values for each variable, and a set of constraints. Constraints are specified by a set of variables to which they apply and a test predicate that maps an assignment of variable values to true or false [8].

*Constraint variables and geometric variables* The CSP is encoded using constraint variable  $O_i, i \in \{1, \dots, n\}$ , which represents the assembly element assignment for  $i$ -th assembly action in the assembly action skeleton. Its value domain is  $1, \dots, n$ , representing the indices of elements in the input model.

Though not explicitly expressed as constraint variables, the geometric variables are pruned by the CSP solver and are used to guide the solver's search. The pruned geometric domains will be outputted as a part of the solution. Geometric variables used in this problem are  $V_i, i \in \{1, \dots, n\}$ , representing the end effector's direction for  $i$ -th assembly action. Its value domain is  $1, \dots, m$ , which represents the indices of directions. The indices of those directions refers to an ordered list of unit vectors sampled on a semi-sphere. The sampling size  $m$  is set according to desired discretization granularity. This meta-parameter balances the completeness and tractability of the computation and could be iteratively increased upon failing to find a solution.

An assigned value  $v$  of  $V_i$  alone cannot uniquely determine the pose of an end effector. One needs to determine the rotation angle  $r$  around the assigned direction value  $v$  to de-

termine the end effector's pose for assembly (see figure 2). This degree of freedom remains undetermined during the entire sequence planning process. The selection is postponed until the motion planning process (Section 3.4). Notice that the domain definition of this rotation angle  $r$  is application-dependent. All the assembly tasks share a continuous rotation angle domain in interval  $[0, 2\pi)$  for spatial extrusion due to the application and end effector's z-axis symmetric nature. However, general assembly tasks, for example spatial positioning, might need different rotation angle domain  $R_i$  to be assigned to each assembly element, depending on the grasp relationship between the end effector and the target assembly element.

**Constraints** Constraints relate the variables to one another and limit the set of valid assignments. If all the constraints are collectively satisfiable, then an assembly plan skeleton is valid, and the pruned geometric variable domains specify the geometric details for subsequent motion planning. In the spatial printing domain, the following types of constraints are used:

**AllDiff**( $O_1, \dots, O_n$ ): Each assembly element is used only once by an assembly action. No disassembling and reassembling is allowed. Thus, all assembly element assignment  $O_i$ 's values are distinct.

**Connectivity**( $O_1, \dots, O_k$ ),  $k = 1, \dots, n$ : At each assembly step, the newly added element must be either connected to the existing structure or connected to the ground. Let Boolean matrix  $A \in \{0, 1\}^{m \times m}$  denotes the adjacency matrix of the input spatial truss design model:

$$A[i][j] = \begin{cases} 1, & \text{if element } O_i \text{ and } O_j \text{ share a node;} \\ 0, & \text{otherwise.} \end{cases}$$

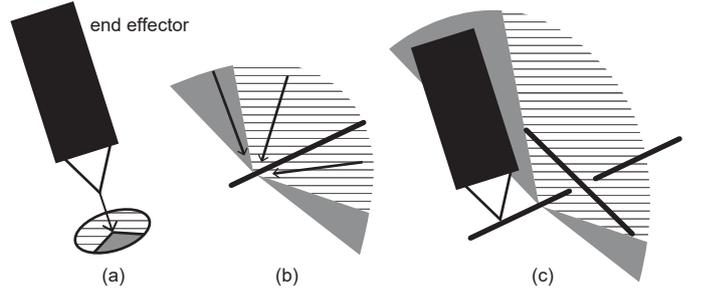
And ground connectivity matrix  $G \in \{0, 1\}^{m \times 1}$ :

$$G[i] = \begin{cases} 1, & \text{if element } O_i \text{ has a grounded node.} \\ 0, & \text{otherwise.} \end{cases}$$

Then the connectivity constraint can be expressed as:

$$\text{FORALL } 1 \leq i \leq m, \text{ EXIST } 1 \leq j < i, \\ A[O_i][O_j] = 1 \text{ OR } G(O_i) = 1$$

**ExistValidEndEffectorPose**( $O_1, \dots, O_k$ ),  $k = 1, \dots, n$ : This constraint checks if there exists a valid end effector pose for each assembly action in the assembly plan skeleton. At each assembly step, existing assembly elements  $O_1, \dots, O_{k-1}$  are considered as collision objects. These collision objects may collide with the end effector for some of the poses specified by direction  $V_i$ 's value and rotation angle around the direction.  $V_i$ 's domain is pruned by the collision objects, eliminating the values that have no valid rotation angle. For spatial printing, a symmetric cone that encloses the end effector is used to avoid explicitly checking or sampling all



**Fig. 3** Illustration of the geometric pruning. The existence of already assembled element (the element on the top in (c)) restricts the collision-free end-effector pose in current assembly task, which prunes out values in associated end-effector direction  $V_i$ 's domain in (b). Around a chosen direction, the end effector's rotation angle  $R_i$ 's domain is further pruned as in (a). The solid gray regions are valid while the line hatched regions are invalid.

rotational values around the chosen direction. A graphical demonstration of this geometric pruning is shown in figure 3. This constraint can be expressed as:

$$\text{FORALL } 1 \leq i \leq n, \\ \text{EXIST } a, 1 \leq a \leq m, \\ (\text{FORALL } 1 \leq j < i, T[O_i][O_j][a] = 1) \\ \text{AND } (\text{ExistValidKinematics}(a, O_1, \dots, O_{i-1}, O_{\text{static}}))$$

where the three-dimensional Boolean matrix  $T \in \{0, 1\}^{n \times n \times m}$ :

$$T[i][j][a] = \begin{cases} 1, & \text{if printing element } i \text{ with direction } a \\ & \text{does not collide with element } j. \\ 0, & \text{otherwise.} \end{cases}$$

and *ExistValidKinematics* is a function that returns true if and only if there exists one rotation angle around the chosen direction  $a$  that admits whole-body kinematic solutions for the robot to traverse the path points of the current element, without colliding into already assembled objects  $O_1, \dots, O_{i-1}$  and static world collision objects  $O_{\text{static}}$ . The kinematics checking function samples rotation angles in  $[0, 2\pi)$  around direction  $a$  and checks the existence of a feasible joint solution until it finds a solution. It immediately returns true if it succeeds and returns false if it fails to find a feasible rotation angle within a specified sampling timeout. This function does not guarantee the existence of feasible kinematic solution for all the rotation angles - it is used only to eliminate the case where collision-free end-effector poses exist but without associated feasible kinematic solutions. Note that the computation involved in checking the end effector's collision ( $\text{FORALL } 1 \leq j < i, T[O_i][O_j][a] = 1$ ) is much lighter than checking the existence of a feasible kinematic solution, enabling faster pruning in the search.

**Stiffness**( $O_1, \dots, O_k$ ),  $k = 1, \dots, n$ : The stiffness constraint ensures that the partial assembly at each assembly step is stiff and the maximal deformation due to gravity (or other

constantly presented load) is bounded by a predefined tolerance. In the case of spatial 3D printing, the deformation of all the nodes under gravity can be calculated using finite element analysis [48]. The constraint test function returns true if and only if the maximal node deformation is smaller than the tolerance.

**Stability**( $O_1, \dots, O_k$ ),  $k = 1, \dots, n$ : The stability constraint checker returns true if the gravitational center's projection on the supporting plane lies in the convex hull of all the grounded nodes, and returns false otherwise. It guarantees that the rigid, partially-assembled structure meets moment equilibrium and does not require a tension connection at the support to remain upright [52].

For different types of discrete structure assembly, such as masonry vault assembly [9][42], a different evaluation scheme for checking the stability constraint can be added to check the static equilibrium of the partially assembled structure, in response to the difference in the mechanics involved.

The evaluation of the stability constraint commonly induces a large amount of overhead as it will be called many times by the CSP planner. Finding an efficient constraint encoding to enable more efficient pruning and faster computation is in the authors' investigation for future work.

### 3.3.2 Solving the CSP

A key advantage of a CSP formulation is that if a user provides a description of their problem in this representation, a generic solver can perform the search. In this paper, a simple backtracking search with 1-level forward checking and dynamic variable ordering is used as a baseline solver (chapter 5.3, [8]). A domain-dependent heuristic is proposed to assist the variable ordering. In addition, to limit the computation in a reasonable amount of time, a user-guided model decomposition is introduced before running the search algorithm. Integrating the CSP encoding with generic, blackbox CSP solvers is left as future work.

*Backtracking search with dynamic variable ordering and 1-level forward checking* The heuristic used in the dynamic variable ordering includes two types of costs:

**Collision cost**  $E_c$ : Although  $o$  is printable, it might cause the remaining unprinted elements to have no feasible end effector orientation in the following stage. Thus, a collision cost is added for tie-breaking by prioritizing the successor that roughly admits the most future orientations.

**Distance-to-base cost**  $E_d$ : A grounded element  $o$  that is further from the robot's base should be printed first. The Distance-to-base cost is defined as:

$$E_d(o) = \begin{cases} 1 - \frac{\text{dist}(\text{base}, o)}{\max_o(\text{dist}(\text{base}, o))}, & o \text{ is grounded} \\ 0, & \text{otherwise.} \end{cases}$$

Notice that the cost  $E_d$  is only used for grounded elements as a tie-breaker, while the collision cost  $E_c$  is used as a heuristic to guide the search for all the elements.

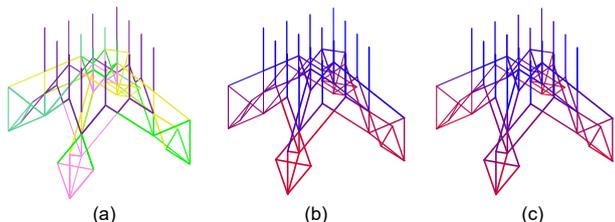
*User-guided model decomposition* Model decomposition involves grouping the discrete input model into several connected components. Taking advantage of a user's intuition on the geometric relationship, the decomposition breaks the whole assembly sequencing problem into several smaller ones, and the search is confined to each of these small sub-problems. This decreases the size of the search space and leads to more efficient CSP solving overall. When the input model has a large number of elements, the heuristic described above might not be informative enough to guide the search efficiently and result into many backtrackings. For example, for the topology optimized vault (figure 4), model decomposition can eliminate backtracking to zero and lead to 40% of decrease in computation time (table 1). However, for some models, e.g. the 3D Voronoi (section 4.1), the heuristic searching itself is powerful enough to find a solution without any backtracking (table 1) and thus model decomposition is not necessary in this case.

Thus, in order to accelerate the computation for model with a large number of elements, the planning framework offers users the choice to manually group the elements to guide the search in CSP, based on their intuition on the geometric occlusion between the decomposed groups. The resulting decomposition has been proven to be effective in handling the sequence planning for many complex geometry instances that exceed the 3D printing capabilities of robots in existing literature (see section 4). However, it is possible for the user to provide a bad decomposition that leads to longer runtime or even failure of finding a feasible solution. Based on the authors' experiments, it usually takes several iterations before one can find one decomposition that works. Nevertheless, searching with decomposition provides users a quicker way to have a sense of whether the model has a sequence solution or not, while a direct heuristic search can only assert the inexistence of a solution after searching all possible partial states. A more general automatic model decomposition, along with the relationship between decomposition and completeness, is left as future work.

*Routing nodal printing orders* After the CSP planner finishes its search and produces an assembly order, the nodal printing orders can be further optimized to increase empirical printing success. For assembly steps that connect two existing nodes, the assignment of start node and end node can be chosen without affecting the sequence planning result's feasibility. This assignment has recently proven to be critical for the physical execution of spatial extrusion due to the molten joint's incapability to resist bending moment and elastic recoil effect [18]. Gelber et al. introduce a cantilever

model	decomposition	search time [s]	# of backtrack	# of partial states visited
TopOpt Vault (fig 4)	without	878	26	158
	with	518	0	132
3D Voronoi (sec 4.1)	without	2311	0	306
	with	2299	0	306

**Table 1** Sequence planning computation statistics with and without decomposition.

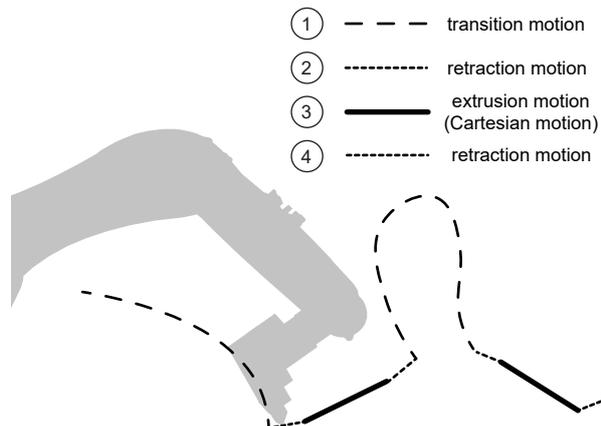


**Fig. 4** Model decomposition and assembly sequence results for a topology optimized vault [28]. (a) shows that the model is decomposed into 11 layers, where six colors are used cyclically to depict layers. (b) and (c) show the assembly sequence results that are generated with decomposition and without decomposition respectively. In (b) and (c), elements are colored from red to blue, corresponding to the first to the last in the assembly sequence.

constraint to their assembly planning algorithm to address this problem: new elements cannot be connected to node  $p$ , if any previously printed element connected to  $p$  is cantilevered [17]. A relaxed version of this constraint is used here to route the nodal printing order: starting from the node with larger degree (number of connected elements) is preferred. Based on the authors' experiments, the introduction of this direction routing process dramatically increases the rate of empirical printing success while not affecting the geometric planning.

### 3.4 Motion planning module

The plan skeleton obtained from the sequence planner specifies the order  $O_1, O_2, \dots, O_n$  and a range of collision-free end-effector directions  $V_j, j \in \{1, \dots, m_i\}$  for each assembly task. To obtain a full kinematic motion for the robot, the motion planner needs to (1) determine the robot's trajectory during each assembly task and (2) plan for the robot's trajectory between assembly tasks. This is a dual motion planning problem due to the Cartesian motion planning with constraints on end-effector's poses during assembly task and free motion planning without constraints on end effector's pose in transition. In the proposed planning framework, this dual motion planning problem is solved in two phases: semi-constrained *Cartesian planning* (section 3.4.1) to resolve the redundancy in end effector poses and associated robot kinematic during each assembly task. Then, *retraction planning* (section 3.4.2) is added between the Cartesian motion and transition motion to enable a safer robot trajectory. Fi-



**Fig. 5** Illustration of a transition motion, retraction motion, and Cartesian motion sequence for two adjacent extrusion processes.

nally, *transition planning* (section 3.4.3) is used to compute robot's trajectory in between adjacent assembly tasks. The sequential layout of transition motion, retraction motion, and Cartesian motion is illustrated in figure 5.

#### 3.4.1 Semi-constrained Cartesian planning

In many robotic assembly applications, the robot's end effector is required to move linearly, where the end effector's tip must follow designated path points. However, its orientation may still have some degrees of freedom [6]. For example, spatial extrusion requires that the tip of the printing nozzle traverse the points on the linear path formed by the element but has freedom in choosing the end effector's orientations. In addition, even when the end-effector's poses are fully specified, there is still kinematic redundancy in choosing corresponding robot configurations. Planning for this type of motion is called semi-constrained Cartesian planning.

In this section, a graph-based semi-constrained Cartesian planner is proposed to resolve the redundancy in the end effector's orientation and robot's kinematics to fully determine robot's joint configuration during each assembly process. For spatial extrusion of a single element, the robot's end effector needs to traverse a linear path with a fixed end effector direction and rotation angle. In order to fully determine the robot's configuration in each individual extrusion task, the planner has three variables to assign for each assembly task: (1) the end-effector direction  $\mathbf{v}_k$ , (2) the end-

effector's rotation angle  $r_k$  around its z-axis direction, and (3) joint configurations corresponding to each of the end effector poses:

$$\begin{aligned} \mathbf{p}_k^i &= (x_k^i, y_k^i, z_k^i), i \in \{1, \dots, \text{number of path points in task } k\} \\ \mathbf{v}_k &\in \text{direction domain } V_k \\ r_k &\in [0, 2\pi) \end{aligned}$$

Solving for robot's kinematic solution problem requires finding feasible joint positions  $\theta_{k_i}$  for each pose  $\mathbf{p}_{k_i}$  in task  $k$ 's path:

$$\begin{aligned} \theta_{k_i} &= (\theta_{k_i}^1, \dots, \theta_{k_i}^d), d = \text{robot's degrees of freedom} \\ \mathbf{p}_{k_i} &= f(\theta_{k_i}) \end{aligned}$$

Here,  $f$  represents the robot's kinematics. Notice that the inverse kinematics solution  $\theta_{k_i}$  for target end-effector pose  $\mathbf{p}_{k_i}$  is not unique and needs to be determined by the planning algorithm. Meanwhile, the computed joint solutions have to be collision-free with respect to objects in the environment during the assembly stage. In addition, the motion between consecutive joint solutions should respect the robot's maximum velocity and acceleration limitations so that the joint solution sequence is physically executable.

Existing work addresses semi-constrained Cartesian planning problem using an approach that discretizes the end effector's candidate poses and kinematic solutions and performs a discrete search on a planning graph [6][56]. This algorithm starts with a list of given end effector poses for the robot to traverse and each end effector pose is assigned with parameters with tolerance ranges. With the tolerance, each given path pose represents a family of parameterized end effector poses and each pose in this family corresponds to a family of robot's joint configurations by performing analytical inverse kinematics with ikfast [31]. These joint configurations can be organized as vertices in a planning graph where edges only exist between vertices that belongs to the same or adjacent end effector pose families. Vertices that represent joint configurations in collision are pruned and edges that represents sharp turns of adjacent joint configurations will not be added to the planning graph. A cost is assigned to each edge in the graph as the  $L_1$  norm of the difference of the two adjacent joint configurations. In this way, the semi-constrained Cartesian path planning problem is converted to a shortest path searching problem on a directed *ladder graph*, which is a multi-partite graph with edge connections between only independent set  $k$  and  $k+1$ ,  $k \in \{1, \dots, n-1\}$ . Each rung in the ladder graph consists of joint configurations that belong to the same end effector pose's parameterized family. The rung's index can be viewed as a time index on path points. The resulting path represents a sequence of joint configurations with minimal joint difference between adjacent joint configurations [6].

However, the planning problems encountered in architectural robotic assembly usually involve longer planning horizons and two dimensions on end effector choice per assembly. These features make a direct application of the ladder graph search algorithm described above impractical. For example, for spatial extrusion of a truss model with 300 elements, the storage of the corresponding planning graph will take 362 Gigabytes, which exceeds the RAM capacity of a common desktop computer. To address this memory issue, a sampling-based optimization algorithm is proposed to first search on a sparse representation of the planning graph and then expand this representation into a significantly reduced full graph to perform a shortest path search.

*Extracting sparse ladder graph* This section first introduces a sparse representation of the planning graph, called *sparse ladder graph*, to help compress and locate the region on the planning graph that contains a close-to-optimal solution for the semi-constrained Cartesian planning problem. A sampling-based planning algorithm is used to extract this sparse representation that is asymptotically optimal locally in this module, which means that the probability of converging asymptotically to the optimum approaches 1.00 with an infinite number of samples [34].

There are two reasons for the memory overhead in the original planning ladder graph: (1) the entire ladder graph needs to be expanded and stored and (2) time indices are assigned to workspace path points that leads to massive number of edges connecting rungs that have adjacent time indices. This observation leads to the idea of leveraging the assembly task's sequence  $\{1, \dots, n\}$  as a sparser time index for rungs and incrementally building a *sparse ladder graph* to first find end effector poses for each assembly task and later recover a reduced ladder graph to search for joint configurations. A *sparse ladder graph* is a compressed version of the original ladder graph, where joint configurations are grouped under a compact data structure called *capsule* and directed edges are constructed between capsules. A capsule is an element, direction, rotation triplet  $(i, \mathbf{v}, r)$  that determines end effector's poses during an element's assembly. In addition, the first and last path point's corresponding joint configurations are recorded in the capsule (figure 6).

Thus, a capsule uses end effector pose to represent the corresponding joint configurations, while enabling (1) the definition of cost (or distance) on a directed edge between two capsules based on first and last pose's joint configurations and (2) the later expansion to a full planning graph of joint configurations. Graph edges in the sparse ladder graph are directed and limited to capsules that have adjacent time index, i.e. between  $(i, \mathbf{v}, r)$  and  $(i+1, \mathbf{v}', r')$ . The cost of such an edge is defined as the minimal  $L_1$  norm of joint pose difference between the last path point's kinematic solutions of source capsule  $(i, \mathbf{v}, r)$  and the first path point's kinematic

solutions of target capsule  $(i + 1, \mathbf{v}', r')$ . By searching on the sparse ladder graph, one can locate close-to-optimal end effector poses for all the assembly task, without encountering the memory overhead caused the expansion of joint configurations for all the path points.

*Computing an optimal capsule path on the sparse ladder graph* The sparse ladder graph is used to find a path of capsules to traverse all the assembly tasks that is close-to-optimal locally in this module. This capsule path can be expanded to a fraction of the original planning graph that contains the close-to-optimal path of joint configurations. Sampling-based algorithms are well suited for this problem because they allow an incremental construction of the sparse graph and provide almost-sure convergence to the optimal solutions locally for this module [34].

In order to apply these algorithms to the problem here, special initialization, sampling, feasibility checking, and connecting functions are provided. These procedures adapt planning to the hybrid discrete-continuous state space and the sequential layout of the sparse ladder graph.

Let  $X \subseteq [n] \times [m] \times [0, 2\pi)$  be the state space of the sparse planning problem, where  $[m] \times [0, 2\pi)$  parameterizes the end-effector's pose by assigning end effector's direction with index  $j \in [m] = \{1, \dots, m\}$  in a precomputed list of directions and rotation angle  $\theta \in [0, 2\pi)$ .  $n$  is the number of elements to be assembled and represents the time index in the assembly process. Each state  $(i, \mathbf{v}_j, \theta)$  corresponds to a capsule. Let  $X_{obs} \subseteq X$  be the set of states where the capsule does not have feasible joint poses for some of the path points for the corresponding task. Let  $X_{free}[i] = X - X_{obs}[i]$  be the resulting set of permissible states in assembly step  $i$ . Let  $\delta : [n] \mapsto X$  be a sequence of states and  $\Sigma$  be the set of all paths. The optimal path planning problem on a sparse ladder graph can be defined as the search for the path  $\delta^*$  that minimizes the accumulated cost of the path while traversing each assembly task in a chronological order:

$$\delta^* = \underset{\delta \in \Sigma}{\operatorname{argmin}} \{c(\delta) \mid \delta[i] = x(i, \cdot, \cdot), \forall i \in [n], \delta[i] \in X_{free}[i]\}$$

We use the following cost function:

$$c : X \times X \mapsto R_+$$

$$c(x, x') = \min_{\mathbf{J}_1, \mathbf{J}_2} \|\mathbf{J}_1 - \mathbf{J}_2\|_{L_1}$$

s.t.  $\mathbf{J}_1 \in \operatorname{InvKm}(\operatorname{ExtractPose}(x)[\text{last path point}])$

$\mathbf{J}_2 \in \operatorname{InvKm}(\operatorname{ExtractPose}(x')[\text{first path point}])$

where *InvKm* denotes an analytical inverse kinematic solver that returns all collision-free robot's joint configurations corresponding to a given end effector pose. The function *ExtractPose*:  $(X \mapsto \text{end effector poses})$  returns all the end effector poses that state  $x$  (capsule) encodes.

In this paper, the Rapidly-exploring Randomize Tree\* (RRT\*) algorithm is applied to the sparse ladder graph (figure 7). Other asymptotically optimal sampling-based algorithms, e.g. Probabilistic Roadmap\* (PRM\*), could also be used. The complete description of these algorithms can be found in [34]. Key modifications enabling these algorithms to operate on the sparse ladder graph are highlighted below:

**Sample:** The sampler operates on a hybrid discrete-continuous state space, which returns a state  $x \in X(\cdot, \cdot, \cdot)$  that is generated from three different and stand-alone samplers. Each one of these three samplers generates independent samples by randomly sampling uniformly the corresponding state space. The generated samples uniquely determine (1) assembly task's time index  $i$  (2) end-effector direction index  $j$  in assembly task  $i$ 's direction list, and (3) rotation angle  $\theta \in [0, 2\pi)$ , which all together determine end-effector's poses along the path points in assembly task  $i$ .

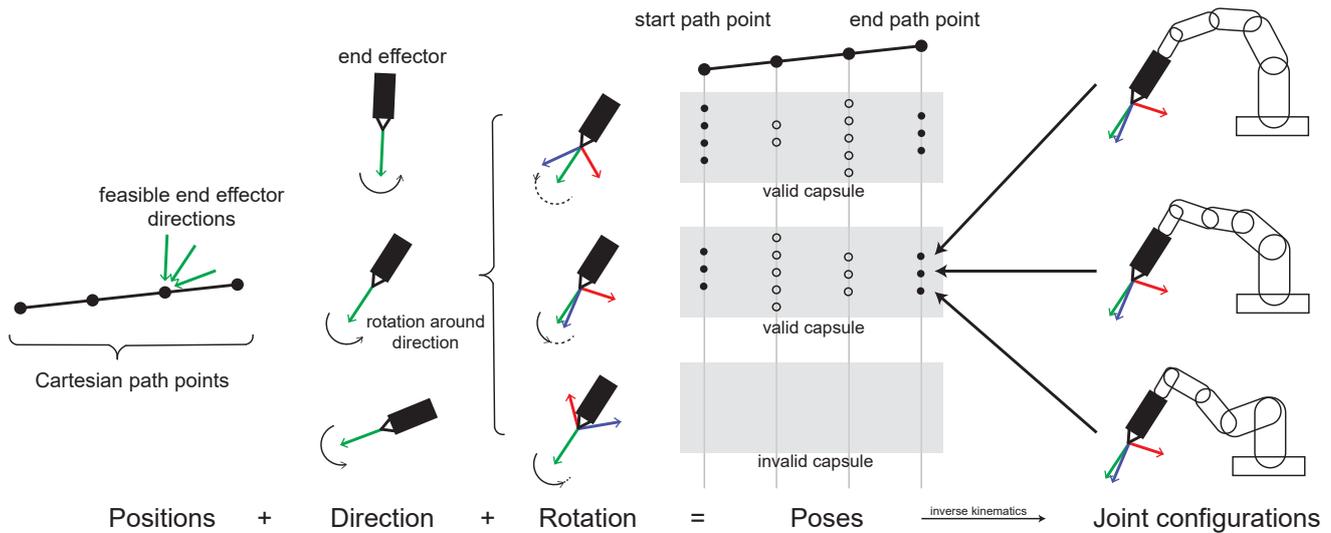
**CheckFeasibility:** State  $x$ 's validity is verified by checking if all the encoded path points have feasible robot kinematics solutions. For state  $x \in X(i, \cdot, \cdot)$  with task index  $i$ , a kinematic solution for a given end effector pose is pruned if it results in a collision. Each task has a different set of collision objects, as elements assembled in previous tasks become collision objects in subsequent assembly tasks.

**Nearest and Rewiring:** Given a state  $x \in X(i, \cdot, \cdot)$ ,  $1 < i \leq n$ , the function *Nearest* returns the vertices with smallest cost to  $x \in G \cap X(i - 1, \cdot, \cdot)$ , where  $G$  is the current sparse ladder graph. Edge connections are restricted to only vertices in adjacent assembly tasks, as skipping assembly tasks is not allowed.

*Extracting a trajectory solution* The sampling-based algorithm returns a path  $\delta$  in the sparse ladder graph. The path is then expanded as a subgraph of the original planning graph to enable the use of standard shortest path search algorithms to find the sequence of joint poses with minimal cost. Each state (capsule) in the returned path  $\sigma$  is expanded by adding the intermediate path points' kinematic solutions as vertices on the corresponding rungs and then constructing edges between all vertices on adjacent rungs, which corresponds to two successive path points (figure 7).

The expanded graph is a directed acyclic graph (DAG). By topologically sorting its vertices, a shortest path can be identified in time linear in the size of the graph (chapter 24.2, [3]). The resulting path gives a discretized joint trajectory for each assembly task in the assembly action sequence, which fully determines the robot's configurations during each individual assembly task.

Notice that when applied to semi-constrained Cartesian planning problems, this sparse graph hierarchical approach preserves local optimality in this module, compared to directly applying shortest-path search on a full ladder graph of joint configurations. In the original ladder graph, edge



**Fig. 6** Demonstration of capsules in the sparse ladder graph. Inside each capsule, only the start and end pose's kinematic families are stored (black dots), while all other joint poses (white dots) are abstracted away.

connections are limited between joint configurations that belongs to the same or adjacent end effector pose parametrization to satisfy the end effector's orientation constraint. Viewed in the sparse ladder graph, this disallowance of edge connection across pose families is enforced by putting all the capsules that correspond to the same element's assembly in the same rung (independent set). Thus, no potential decrease in path is lost by applying this hierarchical sparse ladder graph approach locally in this module. However, if viewed globally on the entire assembly planning system, the joint configurations generated by this module might result in sub-optimal or even infeasible transition trajectories. In general, trading the entire system's completeness and global optimality for tractability is common among hierarchical planning approaches.

### 3.4.2 Retraction planning

Retraction motion is a short segment of slow linear motion that is inserted between each transition motion and Cartesian motion as a buffer to allow the robot to safely change from high speeds to low speeds when it's approaching (or departing) the workpiece (figure 5). In this work, the retraction planner constructs the linear segment by sampling in the set of feasible end-effector's directions produced by the sequence planner and generates a line along this vector with a user-defined length. The same feasibility checking strategy used in the sampling-based algorithm in the last section is applied here to verify the sampled direction's feasibility. Using the Cartesian extrusion motion's orientation, the end effector's orientation during this retraction motion is kept unchanged. For more general assembly problems, for example spatial positioning, the direction of this retraction motion

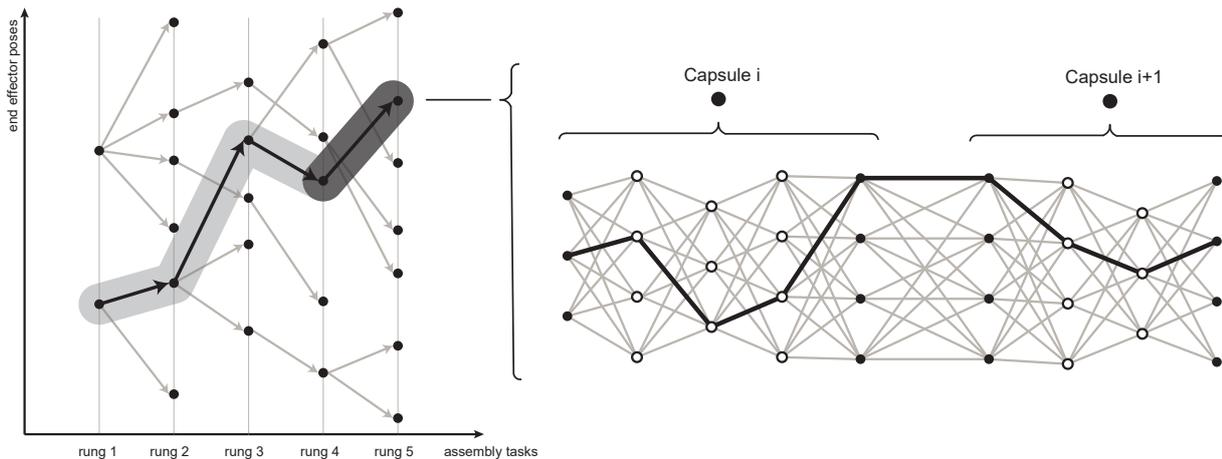
is related to the assembly and the end effector's geometry or the structural joint's geometry (e.g. interlocking joint between two wood elements).

### 3.4.3 Transition planning

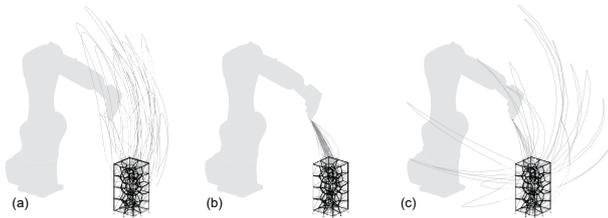
Following semi-constrained Cartesian planning and retraction planning in the last two sections, transition planning computes a collision-free joint trajectory connecting the last joint pose in the departing retraction motion in assembly task  $i$  and the first joint pose in the approaching retraction motion in assembly task  $i + 1$ . This is solved using a standard single-query motion planner, which takes into account of the present collision objects in assembly task  $i$  (figure 8). The transition planner first tries to call the motion planner for directly connecting the target start and goal configurations. Upon failure, it replans by inserting a reset home waypoint between the start and goal configurations. This guides the planner to find a feasible solution as the configuration space near the home waypoint is less constrained. The transition trajectories generated from three state-of-the-art motion planners are shown in figure 8. The result in figure 8 (b) shows that the CHOMP planner [55] frequently fails to generate a feasible transition plan on its first attempt and thus requires resetting itself to the home waypoint quite often. Based on the authors' experience, the STOMP planner works the best, producing smooth and feasible trajectories with less excessive joint movement.

### 3.5 Post processing module

In this work, post processing includes (1) the reassignment of velocities to the computed trajectories and (2) the in-



**Fig. 7** Demonstration on applying RRT\* on sparse ladder graph. The optimal capsule path is highlighted, and the expansion of two adjacent capsules is depicted.



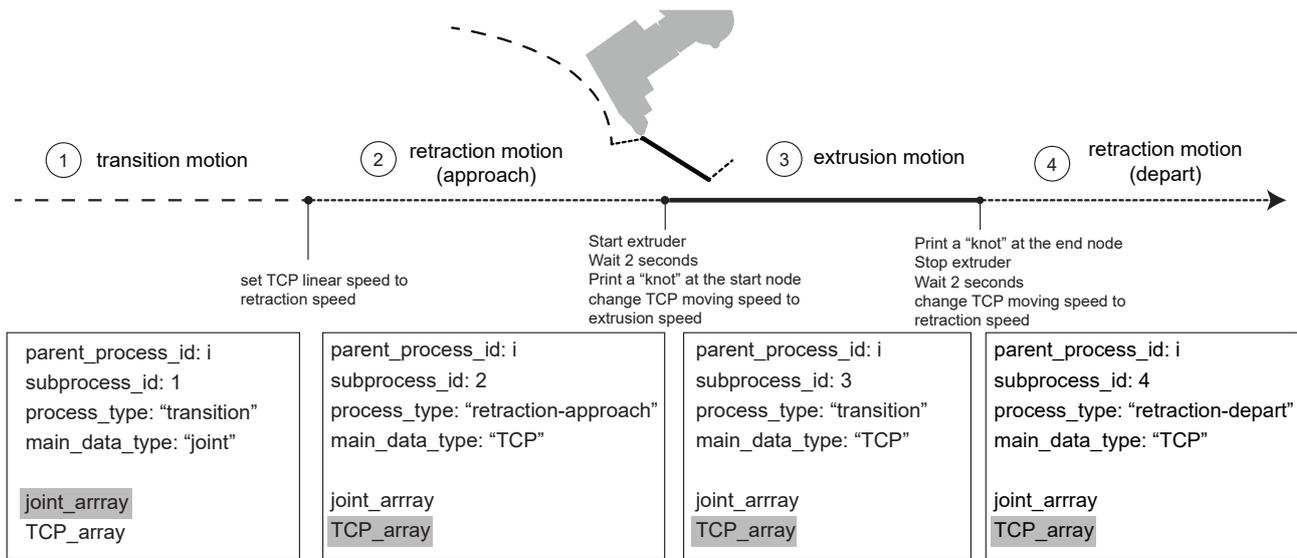
**Fig. 8** Transition planning with different planners: (a) STOMP [33] (b) CHOMP [55] (c) RRT\* [34]

sertion of end effector control between trajectories. After post processing, the generated commands can be converted into an executable robot code that is specific to an industrial robot’s brand. This “translation” step is left to external robot’s softwares. The post processing module proposed here uses a tagging system to group and tag the trajectories with meta-information that describes the containing process’s name and time index. This tagging process enables an easier importing and parsing of the results into various programming systems for application- and hardware-specific adjustment and fine-tuning. This allows the planning framework to be used in various robotic assembly applications with different hardware setups. Two specific ways that the tagging process can be used are described in this section.

*The reassignment of control velocities and synthesis of IO commands* The generated robot trajectories is entirely geometric and its inherent timestamp information only provides the order of joint configurations. Meaningful timestamps need to be reassigned by the users to the computed trajectories after the planning is finished. In addition, in order to generate instructions for the robot to interact with the physical world, the users need to weave IO commands to synthesize the robot’s motion and end effector’s behavior. Many existing architectural robotic assembly projects in-

volve an offline programming process. In these projects, the insertion of IO commands is usually carried out in a graphical programming platform, for example, Grasshopper [21], to have a visually friendly way to insert IO commands in the trajectory command list with live simulation playback. This process, however, can be very tedious when working with robotic assembly applications with a long planning horizon and a massive number of configurations. To increase the computed trajectory’s compatibility to these visual programming platforms for trajectory post processing, the generated trajectory is formatted in a customized JSON format, which contains a hierarchical information structure to maximize its readability and usability. Each element’s assembly process contains several subprocesses, each of which is tagged with a subprocess type: transition, retraction-approach, extrusion, or retraction-depart (figure 9).

Many robotic assembly projects require the robot to have different end effector speed (also called workspace speed) in different phases of its motion. Users need to produce control velocity subject to the constraint or need of their applications. For example, for spatial extrusion, the robot must to extrude material with its end effector following a straight linear movement in a constant speed. Most of the industrial robots provide linear movement commands that take a tool center point (TCP) plane to generate linear movement with a user-defined constant end effector speed. This requires that the result produced by the tagging system contains both robot’s joint trajectory and the associated TCP planes to allow users to choose according to the subprocess’s definition. To support this feature, when exporting computed trajectories, the planning system performs forward kinematics to every joint configuration to compute corresponding TCP planes. Both of these joint array and TCP array are packed with assembly task id, subprocess id, and subprocess type. In addition, the data type can be specified to indi-



**Fig. 9** Illustration of the meta-information generated by the tagging system. An element’s assembly process consists of four subprocesses: (1) transition, (2) retraction-approach, (3) extrusion, and (4) retraction-depart. Insertion of end effector control commands and path modifications are shown between processes.

cate what kind of motion the subprocess is using. TCP data should be used if end effector linear movement with constant speed is desired, and joint data should be used if there is no constraint on the end effector’s speed.

On the other hand, control commands for the end effector need to be synthesized into the robotic trajectories. These commands are usually application- or hardware-specific, which involves digital IO, analog IO, and wait times, to enable industrial robot’s controller to send commands to activate/stop external customized devices’ behavior. For example, spatial extrusion needs the end effector to start extruding material between retraction-approach and extrusion motion, and stop extruding right after the robot finishes the extrusion. This is done by inserting a digital ON/OFF command between the designated processes.

To form a smooth transition into the established method of weaving IO commands in a graphical programming environment, the formatted trajectories produced by the post processor can be imported into any such platform, such as Grasshopper [21], with a simple customized parser, to decode the JSON file. Users can insert robot commands, such as digital IO, analog IO, and wait time, based on the assembly element’s index and process context, without having to find the index of a specific joint configuration themselves. Then, existing robot simulation packages can be used to simulate the robot’s trajectory to verify the correctness and safety of the trajectories and export brand-specific robot instruction code.

*Application-oriented path modification* For many robot assembly processes, especially spatial extrusion, the variety of end effector designs and material properties requires the

incorporation of ad-hoc fabrication logic to achieve the desired visual results [22][26] or increase the product’s structural performance [70]. These fabrication logics, which are derived from physical extrusion experiments, usually involve local modification of an end effector’s pose, such as pressing or extruding following small circular movements at structural joints to create local “knots”.

The metadata associated with the computed trajectories allows users to easily insert these micro path modifications. These path modifications usually require users to iterate on the parameters controlling robot and its end effector’s behavior, until they find the best parameter setting based on experimental observations. For spatial extrusion, one needs to perform many experiments to find the delicate balance between robot’s linear moving speed while extruding, cooling air’s pressure, and extrusion rate. Because of the tagging system, the fabrication parameter calibration process repeats between the fine-tuning programming platform and physical tests, while keeping the overall planned robotic trajectory unchanged.

### 3.6 Implementation

The proposed hierarchical assembly planning framework has been implemented in a proof-of-concept planning tool called *Choreo*. This tool allows users to compute feasible robotic assembly trajectories using unconstrained target assembly geometry, and it supports customized hardware work environments. In this paper, *Choreo* is configured to work with spatial extrusion applications. This section first presents the general system architecture (section 3.6.1) and then presents

an overview of the user experience of Choreo along each of its computation state (section 3.6.2 - 3.6.5).

### 3.6.1 System architecture overview

Choreo is implemented in C++ on the Robot Operating System's (ROS) Kinetic Release on Ubuntu 16.04 [54]. The C++ code is open-source and available online<sup>1</sup>. Drawing inspiration of the Godel system from ROS industrial [57], Choreo's system architecture is designed to be modular and flexible: graphical user interface (GUI) module, data IO module, visualization module, and core planning engine modules are all implemented as standalone ROS nodes. Instead of directly communicating to each other, the communication between these modules is coordinated by a central core node using formatted ROS messages and services (figure 10). This enables a clean decoupling between modules that offers users the flexibility to plug in and experiment with their customized sequence or motion planner without changing the rest of the codebase. The GUI is implemented as a simple Qt plugin for the Rviz visualization platform to provide buttons, sliders, and data IO to help users import and export their data and navigate them through the planning process.

### 3.6.2 Assembly problem setup

Robots and end effectors are specified using a Unified Robot Description Format (URDF) file<sup>2</sup> in Choreo, which is an XML format data that contains robot's link geometry, joint limitation, and other related data. To specify customized end effector, users need to have the STL mesh for the end effector (used for collision checking) and create a URDF file to link the imported geometry to a desired robot link. Static collision objects in the work environment are imported as STL meshes and linked to the robot's URDF file.

The geometry of spatial trusses can be specified using the node-connectivity format described in section 3.2. The diameter of the truss element is defined by the extruder's nozzle. A decomposition can be added to the geometry model by simply assigning a layer index to each element. The authors develop a simple parser based on the graphical programming platform Grasshopper [21], to have a visually friendly layer tagging workflow. The relative position between the the build platform and the robot's base can be calibrated from the robot and input into the system by a 3D vector using the GUI widget.

### 3.6.3 Sequence planning

Currently, Choreo's sequence planner is powered by a customized backtracking search engine (section 3.3.2). The an-

alytical kinematics computation is performed through the ikfast kinematics plugin [31]. The collision check between robot and the environment is implemented using the collision checking interface provided by Moveit! [68].

### 3.6.4 Motion planning

The semi-constrained Cartesian planner is implemented based on the Descartes planning package from ROS-Industrial [56]. The sparse ladder graph and the RRT\* algorithm is implemented by the author using the Descartes package's ladder graph data structure. The retraction planner is a direct application of the Descartes package with direction vector sampling.

The transition planner utilizes the motion planner plugin interface of the Moveit! motion planning framework [68]. Choreo uses the STOMP planner from ROS-industrial [33][58] as the main single-query motion planner, but can be easily configured to work other motion planners.

### 3.6.5 Post-processing and execution

After the motion planning is finished, the computed trajectories are tagged with meta-data associated to the assembly tasks and can be exported as a JSON file. The core module coordinates with the simulation module to display the chosen assembly tasks' trajectories in Rviz (figure 11).

Next, extra post processing and fine-tuning can be performed in other programming platforms. In all the case studies in this work, a customized C# JSON file parser is implemented in Grasshopper [21]. The KUKA|PRC package [2] and the Robots plugin [63] are used to post-process the trajectory into a KUKA Robot Language (KRL) file and ABB RAPID file for simulation and execution. The exported trajectory can be configured easily to work in other parametric design platforms and be adapted to other robotic simulation packages such as HAL [60] and Jeneratiff [12]. As described in section 2.1, such simulation tools are useful for visualizing a generated robotic motion plan and generates robotic brand-specific instruction code within the Grasshopper environment.

Hardware-wise, a customized extrusion system was designed and assembled by Archi-Solution Workshop<sup>3</sup>. A detailed description of the end effector, extrusion system, and cooling system can be found in [78] as well as the online supplementary materials of [30].

## 4 Case Studies

To illustrate the capabilities of Choreo, this section describes three case studies that utilize Choreo's power to automati-

<sup>1</sup> <https://github.com/yijiangh/Choreo>

<sup>2</sup> <http://wiki.ros.org/urdf>

<sup>3</sup> <http://www.asworkshop.cn/>

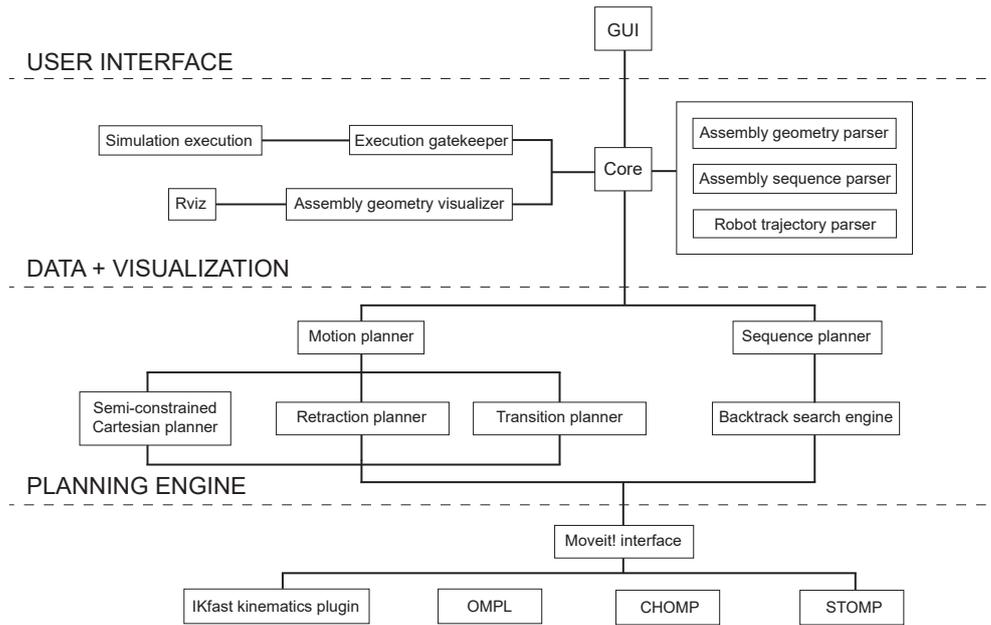


Fig. 10 Choreo's system architecture.

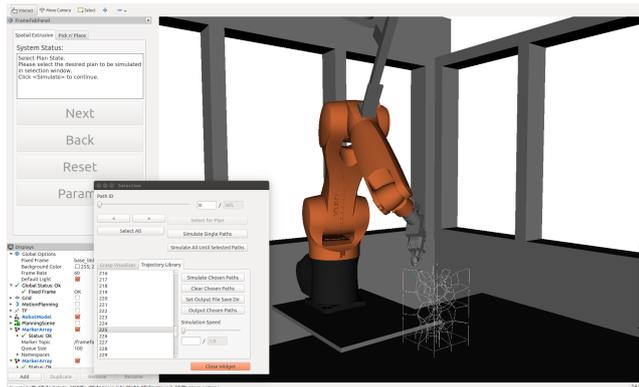


Fig. 11 Screenshot of Choreo at trajectory simulation stage.

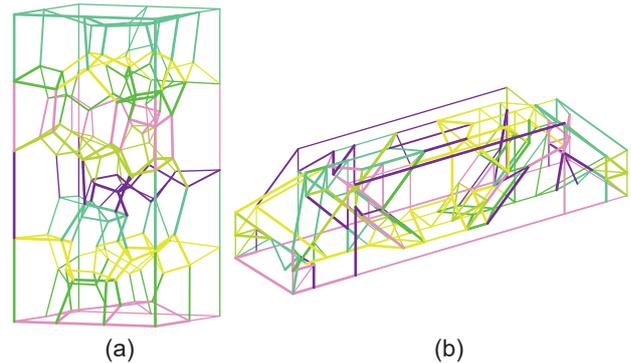


Fig. 12 The user-specified decomposition. (a) 3D Voronoi has 10 layers (b) Topopt beam has 53 layers. In the image, six colors are used cyclically to depict layers.

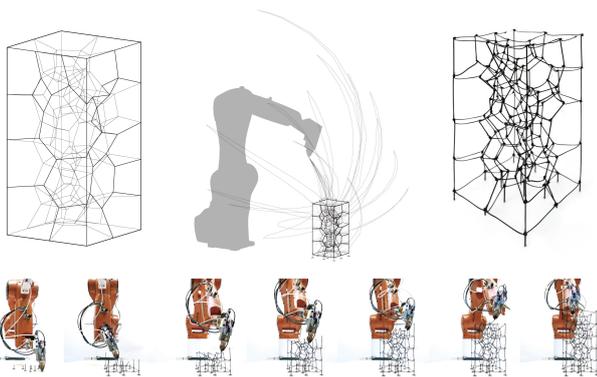
cally plan for feasible robotic trajectories for spatial extrusion of complex spatial trusses with non-standard topologies. The presented case studies have fundamentally different topologies and scales: 3D Voronoi (section 4.1) and topology optimized simply-supported beam (small and large scale, section 4.2). Model-related data, together with statistics on assembly planning and fabrication time are presented in table 2. The user-guided decomposition of the models are shown in figure 12. All computational experiments were performed on a Linux virtual machine with 4 processors and 16 GB of RAM on a desktop PC with a quad-core Intel Xeon CPU. Additional case studies on robotic extrusion of non-standard topologies can be found in [29][28].

#### 4.1 3D Voronoi

The 3D Voronoi design was generated by randomly sampling points within a rectangular solid, and then using the 3D Voronoi component in Grasshopper [21] together with Kangaroo2 [53] to initiate the 3D Voronoi pattern. A sphere collision algorithm was used to force the element lengths to have low variance. A KUKA KR6-R900 robot was used to execute the extrusion. Figure 13 shows the design and fabrication of this structure. Because of the Voronoi-generating algorithm, there is low variation in node valence, and most nodes only connect four elements. Elements are well supported during each construction step, and there are few very long elements. The internal topology does not have a trivial layer-based pattern. Thus, it is unintuitive for humans to

Model	Node count	Element count	Layer count	Sequence planning time [s]	Extrusion planning time [s]	Transition planning time [s]	Fabrication time [hr]	Size [mm]
3D Voronoi (sec 4.1)	148	292	10	2299	1200	1286 (RRT*)	3.2	150 * 150 * 320
Topopt beam (small) (sec 4.2.1)	121	271	53	2170	1271	893 (STOMP)	3.6	400 * 100 * 100
Topopt beam (large) (sec 4.2.2)	121	271	53	1577	1809	553 (STOMP)	-	2800 * 700 * 700

**Table 2** Input model information, computational statistics, and fabrication time of the case studies. Layer count is the number of layers used in the user-generated decomposition (figure 12).



**Fig. 13** 3D Voronoi design, robotic trajectories with RRT\*, and final extruded result. A fixed-base KUKA KR6-R900 robotic arm is used.

find a sequence manually, and the Choreo platform proves helpful.

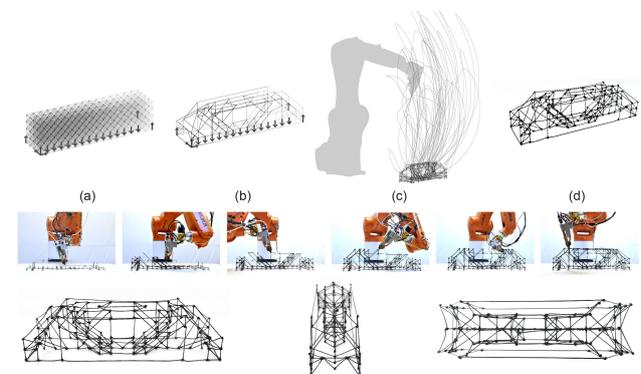
However, elements at the boundary have smaller node valences and very long length. Even though the geometrically planned trajectory is feasible, the extruded element deviates from its ideal position because of the material’s thermal wrapping. This deviation is sometimes large enough that the robot is not able to connect to these elements in the subsequent extrusion processes by following the computed trajectories. This issue is resolved by adding micro-path modifications to the computed Cartesian extrusion path in the post processing stage to extrude a “knot” at the node to compensate for the inaccuracy brought by the thermal behavior of the material.

## 4.2 Topology optimized simply-supported beam

Using the ground structure topology optimization method described in [28], a simply-supported beam was designed for the loads and boundary conditions shown in figure 14 (a), (b). The resulting topology is fairly irregular when compared to a standardized mesh topology. The beam is scaled to a small size and a large size and two different machine setups are used for assembly planning. The large-scale example is presented to demonstrate the potential of applying

Choreo at the scale of a real building component, which in particular fits into the context of construction robotics.

### 4.2.1 Small scale



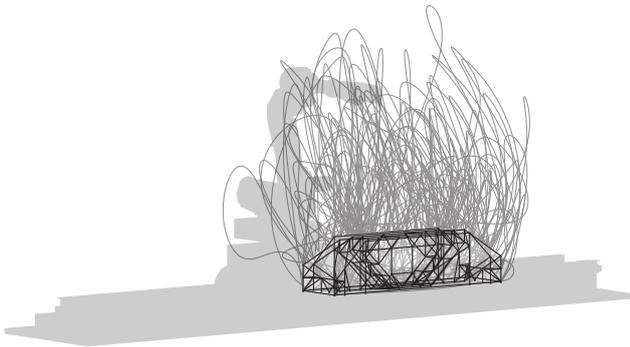
**Fig. 14** Topology optimized simply-supported beam, with (a-b) topology optimization input and result, (c) robot trajectories with STOMP, and (d) final extruded result. A fixed-base KUKA KR6-R900 robot is used.

The small-scale beam spans 400 millimeters. A fixed-base KUKA KR6-R900 (maximal reach 0.9m) robot is used to execute the extrusion. The average element length of the model is fairly long, and element length variation is low because the design is generated from a regular base mesh. However, the geometric configurations generated from these elements is not trivial. The trajectory highlighted in figure 14 shows the corresponding tool center point traveling trajectory from the transition planning result, indicating that the robot’s configuration changes significantly between many pairs of adjacent extrusions. As a result, trajectories that respect joint limits and avoid collisions are long and unintuitive to humans.

### 4.2.2 Large scale

The large-scale beam has a span of 2.8 meters, with 0.7 meters in thickness and height. A 5.4-meter linear track is added to an ABB IRB6640-180-280 robotic arm (maximal

reach 2.8m) to accommodate the scale of the beam. The additional degree of freedom from the linear track expands the feasible workspace of the robot. Specifically, this extra dimension increases the set of kinematic solutions, making it easier to find a feasible joint configuration. In practice, the movement of the six revolute axes of the robotic arm is preferred over the track's movement for accuracy and energy consumption reason. In this paper, in order to penalize extra movement of the prismatic linear joint, an extra penalty weighting factor is added to linear track's joint movement when computing the  $L_2$  distance between joint configurations for constructing ladder graph in the semi-constrained Cartesian planning module (section 3.4.1). The analytical inverse kinematics of the robot is done by discretizing the prismatic joint and attempting 6-dof IK. This discretization resolution also balances the completeness and RAM overhead of the computation and could be iteratively increased to find a feasible solution or limit excessive base movement. In this experiment, the prismatic joint is discretized every 0.01 m over a full length 5.4 m. A more in-depth investigation of the robot base placement, using reachability analysis [46], is under investigation for future work. The resulting simulated assembly trajectory is shown in figure 15.



**Fig. 15** Simulated end-effector trajectory of an ABB IRB6640-185-280 mounted on a 5.4-meter linear track to spatially extrude of the large-scale topology optimized beam.

## 5 Conclusion

### 5.1 Summary of contributions

This paper presents the first attempt to rigorously formulate the architectural robotic assembly planning problem and provide an integrated algorithmic solution. A new hierarchical planning algorithm is proposed to solve assembly problems with long planning horizons and complex geometry.

This paper also presents a proof-of-concept software implementation, called Choreo, which is a hardware- and application agnostic. Choreo can be easily configured to work

with industrial robots across brands, sizes, and joint spaces. Three case studies involving spatial trusses with non-standard topologies, including two real fabrication experiments and one large-scale assembly simulation, are presented to demonstrate the new fabrication possibilities enabled by applying the proposed planning framework.

#### 5.1.1 Potential impact

The case studies presented in this work have demonstrated how the proposed assembly planning framework's integration into existing digital design workflow can support topology as a fundamental design variable on designers' palette for robotic assembly. The emergence of this automated planning system can provide a better way for designers to interact with robots, shifting the machine programming experience back to high-level tasks in the architectural language of shape and topology.

On the other hand, the flexibility and the efficiency of Choreo creates a testbed for educators, researchers, and practitioners to explore novel robotic fabrication and assembly applications more boldly. It provides a general common ground for future research in assembly-related sequence and motion planner, and creates a bridge between architectural robotics research community and task and motion planning research community.

#### 5.1.2 Limitation and future work

One key limitation of the current work is on the need of human intervention for model decomposition to accelerate the sequence planner. An automatic decomposition algorithm will eliminate this last bit of human intervention and fully automate the planning process.

Potential directions of future investigation are summarized as follows:

*Backtrack between planning layers* When the planning system encounters a planning failure in any layer in the hierarchy, there is no backtracking mechanism provided to allow it to backtrack across planning layers. Thus, the hierarchical planning algorithm is not complete for the entire system, i.e. guaranteed to find a solution if one exists, although the algorithm proposed in each level of the hierarchy is complete. Existing work in TAMP has devised various way to allow this geometric backtracking across planning layers. The integration of some of this research is left as future work.

*Extension to other robotic assembly applications* All the algorithmic descriptions and case studies presented in this work are performed in the context of the specific application of robotic spatial extrusion. Generalizing the proposed planning framework to other assembly applications, such as spatial positioning, requires little modification of the algorithm.

These small modifications include different predefined plan skeletons, different constraints on the end effector's orientation in the semi-constrained Cartesian planner, and extensions to the assembly sequence data format.

### 5.1.3 Concluding remark

Automatic assembly planning has been a key missing link in the established digital design-robotic assembly workflow. Architectural robotic assembly problems have posted unique technical challenges, including (1) long planning horizon and (2) structural stiffness and stability constraints for assembly sequence planning. This paper presents a new algorithmic framework and a software tool called Choreo to fill in this missing link and thus clears away the technical barrier of assembly planning that has been congesting the workflow and limiting design-build freedom. Although Choreo is still in its early stage, its flexibility and speed have already suggested an exciting future possibility: fabrication and assembly logic related to robotic constructibility could be integrated as a driver in iterative conceptual design, pushing the role of technical assessment from checking a nearly finalized design to an early-stage design aid.

**Acknowledgements** The authors want to acknowledge Thomas Cook, Khanh Nguyen, and Kodiak Brush at MIT for their work on constructing the early-stage prototype of the software and hardware presented in this work. The authors would also like to thank Jonathan Meyer from ROS-Industrial for his insightful discussion on github and Zhongyuan Liu from University of Science and Technology of China for his help on generating the 3D Voronoi shape. The authors want to thank Archi-Solution Workshop (<http://www.asworkshop.cn/>) for their support on the designing and assembling of the mechanical extrusion system used in the case studies. Caelan Garrett acknowledges the support from NSF grants 1420316, 1523767 and 1723381, from AFOSR FA9550-17-1-0165, from ONR grant N00014-14-1-0486, and an NSF GRFP fellowship with primary award number 1122374. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

## References

- Alami, R., Laumond, J.P., Siméon, T.: Two manipulation planning algorithms. In: WAFR Proceedings of the workshop on Algorithmic foundations of robotics, pp. 109–125. A. K. Peters, Ltd., Natick, MA, USA (1994)
- Braumann, J., Brell-Cokcan, S.: Parametric robot control: integrated cad/cam for architectural design. In: Proceedings of the 31st Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) (2011)
- Cormen, T.H.: Introduction to algorithms. MIT press (2009)
- Dai, C., Wang, C.C., Wu, C., Lefebvre, S., Fang, G., Liu, Y.: Support-free volume printing by multi-axis motion. *ACM Transactions on Graphics (TOG)* (2018). (in press)
- De Fazio, T., Whitney, D.: Simplified generation of all mechanical assembly sequences. *IEEE Journal on Robotics and Automation* **3**(6), 640–658 (1987)
- De Maeyer, J., Moyaers, B., Demeester, E.: Cartesian path planning for welding robots: Evaluation of the descartes algorithm. In: Proceedings of the 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (2017)
- De Mello, L.H., Sanderson, A.C.: And/or graph representation of assembly plans. *IEEE Transactions on robotics and automation* **6**(2), 188–199 (1990)
- Dechter, R.: Constraint processing. Morgan Kaufmann (2003)
- Deuss, M., Panozzo, D., Whiting, E., Liu, Y., Block, P., Sorkine-Hornung, O., Pauly, M.: Assembling self-supporting structures. *ACM Transactions on Graphics (TOG)* **33**(6), 214 (2014)
- Dogar, M., Spielberg, A., Baker, S., Rus, D.: Multi-robot grasp planning for sequential assembly operations. In: Robotics and Automation (ICRA), 2015 IEEE International Conference on, pp. 193–200. IEEE (2015)
- Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., Nebel, B.: Semantic Attachments for Domain-Independent Planning Systems, p. 99115. Springer Tracts in Advanced Robotics. Springer (2012)
- Dritsas, S.: A digital design and fabrication library. In: Proceedings of the Symposium on Simulation for Architecture & Urban Design, pp. 75–80 (2015)
- Eversmann, P., Gramazio, F., Kohler, M.: Robotic prefabrication of timber structures: towards automated large-scale spatial assembly. *Construction Robotics* **1**(1-4), 49–60 (2017)
- Fu, C.W., Song, P., Yan, X., Yang, L.W., Jayaraman, P.K., Cohen-Or, D.: Computational interlocking furniture assembly. *ACM Transactions on Graphics (TOG)* **34**(4), 91 (2015)
- Garrett, C.R., Lozano-Pérez, T., Kaelbling, L.P.: Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research* **37**(1), 104–136 (2018)
- Garrett, C.R., Lozano-Pérez, T., Kaelbling, L.P.: Sampling-based methods for factored task and motion planning. *arXiv preprint arXiv:1801.00680* (2018)
- Gelber, M.K., Hurst, G., Bhargava, R.: Freeform assembly planning. *arXiv:1801.00527* (2018)
- Gelber, M.K., Hurst, G., Comi, T.J., Bhargava, R.: Model-guided design and characterization of a high-precision 3d printing process for carbohydrate glass. *Additive Manufacturing* **22**, 38–50 (2018)
- Gifftthaler, M., Sandy, T., Dörfler, K., Brooks, I., Buckingham, M., Rey, G., Kohler, M., Gramazio, F., Buchli, J.: Mobile robotic fabrication at 1: 1 scale: the in situ fabricator. *Construction Robotics* **1**(1-4), 3–14 (2017)
- Gramazio, F., Matthias, K., Willmann, J.: The robotic touch. Park Books (2014)
- Grasshopper, R.: Grasshopper rhinoceros. <http://www.grasshopper3d.com/> (2018). Last accessed July 10 2018
- Hack, N., Lauer, W.V.: Mesh-mould: Robotically fabricated spatial meshes as reinforced concrete formwork. *Architectural Design* **84**(3), 44–53 (2014)
- Hauser, K., Latombe, J.C.: Multi-modal motion planning in non-expansive spaces. *The International Journal of Robotics Research* **29**(7), 897–915 (2010)
- Hauser, K., Ng-Thow-Hing, V.: Randomized multi-modal motion planning for a humanoid robot manipulation task. *The International Journal of Robotics Research* **30**(6), 678–698 (2011)
- Heger, F.W.: Assembly planning in constrained environments: Building structures with multiple mobile robots. Ph.D. thesis, Carnegie Mellon University (2010)
- Helm, V., Willmann, J., Thoma, A., Piškorec, L., Hack, N., Gramazio, F., Kohler, M.: Iridescence print: Robotically printed lightweight mesh structures. *3D Printing and Additive Manufacturing* **2**(3), 117–122 (2015)
- Helmert, M.: The fast downward planning system. *Journal of Artificial Intelligence Research* **26**, 191–246 (2006)

28. Huang, Y., Carstensen, J., Mueller, C.: 3d truss topology optimization for automated robotic spatial extrusion. In: Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium 2018 (2018)
29. Huang, Y., Carstensen, J., Tessmer, L., Mueller, C.: Robotic extrusion of architectural structures with nonstandard topology. In: Robotic Fabrication in Architecture, Art and Design 2018. Springer (2018)
30. Huang, Y., Zhang, J., Hu, X., Song, G., Liu, Z., Yu, L., Liu, L.: Framefab: Robotic fabrication of frame shapes. *ACM Transactions on Graphics (TOG)* **35**(6), 224 (2016)
31. IKFast: Ikinfast: The robot kinematics compiler. <http://openrave.org/docs/0.8.2/openravepy/ikfast/> (2018). Last accessed July 22 2018
32. Jeffers, M.: Autonomous robotic assembly with variable material properties. In: Robotic Fabrication in Architecture, Art and Design 2016, pp. 48–61. Springer (2016)
33. Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., Schaal, S.: Stomp: Stochastic trajectory optimization for motion planning. In: 2011 IEEE International Conference on Robotics and Automation, pp. 4569–4574 (2011)
34. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. *arXiv:1105.1186* (2011)
35. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* **12**(4), 566–580 (1996)
36. Krontiris, A., Bekris, K.: Dealing with difficult instances of object rearrangement. In: Robotics: Science and Systems (RSS) (2015). DOI 10.15607/RSS.2015.XI.045
37. Krontiris, A., Bekris, K.E.: Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner. In: International Conference on Robotics and Automation (ICRA), pp. 3924–3931. IEEE (2016)
38. Lagriffoul, F.: On benchmarks for combined task and motion planning. In: Robotics: Science and Systems (RSS) 2016 Workshop on Task and Motion Planning (2016)
39. Lagriffoul, F., Andres, B.: Combining task and motion planning: A culprit detection problem. *The International Journal of Robotics Research* **35**(8), 890–927 (2016)
40. Lagriffoul, F., Dimitrov, D., Bidot, J., Saffiotti, A., Karlsson, L.: Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research* **33**(14), 1726–1747 (2014)
41. LaValle, S.M.: Planning algorithms. Cambridge university press (2006)
42. Livesley, R.K.: A computational model for the limit analysis of three-dimensional masonry structures. *Meccanica* **27**(3), 161–172 (1992)
43. Lo, K.Y., Fu, C.W., Li, H.: 3d polyomino puzzle. In: *ACM Transactions on Graphics (TOG)*, vol. 28, p. 157. ACM (2009)
44. Lozano-Pérez, T.: Spatial planning: A configuration space approach. *IEEE transactions on computers* (2), 108–120 (1983)
45. Lozano-Pérez, T., Kaelbling, L.P.: A constraint-based method for solving sequential manipulation planning problems. In: Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on, pp. 3684–3691. IEEE (2014)
46. Makhal, A., Goins, A.K.: Reuleaux: Robot base placement by reachability analysis. *arXiv:1710.01328* (2017)
47. McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: Pddl—the planning domain definition language. *Tech. rep.*, Yale Center for Computational Vision and Control (1998)
48. McGuire, W., Gallagher, R., Ziemian, R.: Matrix Structural Analysis. Wiley (1999)
49. Mueller, S., Im, S., Gurevich, S., Teibrich, A., Pfisterer, L., Guimbretière, F., Baudisch, P.: Wireprint: 3d printed previews for fast prototyping. In: Proceedings of the 27th annual ACM symposium on User interface software and technology, pp. 273–280. ACM (2014)
50. Parascho, S., Gandia, A., Mirjan, A., Gramazio, F., Kohler, M.: Cooperative fabrication of spatial metal structures. In: *Fabricate 2017*, pp. 24–29. UCL Press (2017)
51. Peng, H., Wu, R., Marschner, S., Guimbretière, F.: On-the-fly print: Incremental printing while modelling. In: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, pp. 887–896. ACM (2016)
52. Phear, J.B.: Elementary Mechanics. MacMillan, Cambridge (1850)
53. Piker, D.: Kangaroo physics. <https://www.food4rhino.com/app/kangaroo-physics> (2018). Last accessed July 26 2018
54. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA workshop on open source software, vol. 3, p. 5. Kobe, Japan (2009)
55. Ratliff, N., Zucker, M., Bagnell, J.A., Srinivasa, S.: Chomp: Gradient optimization techniques for efficient motion planning. In: International Conference on Robotics and Automation, pp. 489–494. IEEE (2009)
56. ROS-I: ROS Industrial - Descartes. <https://github.com/ros-industrial-consortium/descartes> (2018). Last accessed March 14 2018
57. ROS-I: ROS Industrial - Godel. <https://github.com/ros-industrial-consortium/godel> (2018). Last accessed March 14 2018
58. ROS-I: ROS Industrial - Industrial Moveit. [https://github.com/ros-industrial/industrial\\_moveit](https://github.com/ros-industrial/industrial_moveit) (2018). Last accessed March 14 2018
59. Schulman, J., Duan, Y., Ho, J., Lee, A., Awwal, I., Bradlow, H., Pan, J., Patil, S., Goldberg, K., Abbeel, P.: Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research* **33**(9), 1251–1270 (2014)
60. Schwartz, T.: Hal: Extension of a visual programming language to support teaching and research on robotics applied to construction. In: Robotic Fabrication in Architecture, Art and Design 2012, pp. 92–101. Springer (2012)
61. Schwartzburg, Y., Pauly, M.: Fabrication-aware design with intersecting planar pieces. In: *Computer Graphics Forum*, vol. 32, pp. 317–326. Wiley Online Library (2013)
62. Siméon, T., Laumond, J.P., Corts, J., Sahbani, A.: Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research* **23**(7-8), 729–746 (2004)
63. Soler, V.: Robots plugin for Grasshopper. <https://github.com/visose/Robots> (2018). Last accessed June 20 2018
64. Søndergaard, A., Amir, O., Eversmann, P., Piškorec, L., Stan, F., Gramazio, F., Kohler, M.: Topology optimization and robotic fabrication of advanced timber space-frame structures. In: Robotic Fabrication in Architecture, Art and Design 2016, pp. 190–203. Springer (2016)
65. Song, P., Fu, C.W., Cohen-Or, D.: Recursive interlocking puzzles. *ACM Transactions on Graphics (TOG)* **31**(6), 128 (2012)
66. Stilman, M., Kuffner, J.J.: Planning among movable obstacles with artificial constraints. *The International Journal of Robotics Research* **27**(11-12), 1295–1307 (2008)
67. Stilman, M., Schamburek, J.U., Kuffner, J., Asfour, T.: Manipulation planning among movable obstacles. In: Proceedings 2007 IEEE International Conference on Robotics and Automation, pp. 3327–3332 (2007)
68. Sucan, I.A., Chitta, S.: Moveit! <http://moveit.ros.org> (2018). Last accessed March 14 2018

69. Tai, A.S.C.: Design for assembly: a computational approach to construct interlocking wooden frames. Master's thesis, Massachusetts Institute of Technology (2012)
70. Tam, K.M., Marshall, D.J., Gu, M., Kim, J., Huang, Y., Lavallee, J.A., Mueller, C.T.: Fabrication-aware structural optimisation of lattice additive-manufactured with robot-arm. *International Journal of Rapid Manufacturing* **7**(2-3) (2018)
71. Toussaint, M.: Logic-geometric programming: An optimization-based approach to combined task and motion planning. In: *Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15*, pp. 1930–1936. AAAI Press (2015)
72. UNESCO: Press release ece/stat/05/p03, geneva, world robotics survey (october 11, 2005). <http://www.unesco.org/fileadmin/DAM/press/pr2005/05stat.p03e.pdf> (2005). Accessed March 14 2018
73. Wikipedia: Kuka robot language - wikipedia, the free encyclopedia. [https://en.wikipedia.org/wiki/KUKA\\_Robot\\_Language](https://en.wikipedia.org/wiki/KUKA_Robot_Language) (2018). Last accessed April 16 2018
74. Wilson, R.H.: On geometric assembly planning. Ph.D. thesis, Stanford University (1992)
75. Woltery, J.D.: On the automatic generation of assembly plans. In: *Proceedings, 1989 International Conference on Robotics and Automation*, pp. 62–68 (1989)
76. Wu, R., Peng, H., Guimbretière, F., Marschner, S.: Printing arbitrary meshes with a 5dof wireframe printer. *ACM Transactions on Graphics (TOG)* **35**(4), 101 (2016)
77. Xin, S., Lai, C.F., Fu, C.W., Wong, T.T., He, Y., Cohen-Or, D.: Making burr puzzles from 3d models. In: *ACM Transactions on Graphics (TOG)*, vol. 30, p. 97. ACM (2011)
78. Yu, L., Huang, Y., Liu, Z., Xiao, S., Liu, L., Song, G., Wang, Y.: Highly informed robotic 3d printed polygon mesh: A novel strategy of 3d spatial printing. In: *Proceedings of the 36st Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, pp. 298–307 (2016)