Compositional Diffusion-Based Continuous Constraint Solvers

Anonymous Author(s)
Affiliation
Address
email

Abstract: This paper introduces an approach for learning to solve continuous constraint satisfaction problems (CCSP) in robotic reasoning and planning. Previous methods primarily rely on hand-engineering or learning generators for specific constraint types and then rejecting the value assignments when other constraints are violated. By contrast, our model, the compositional *diffusion continuous constraint solver* (*Diffusion-CCSP*) derives global solutions to CCSPs by representing them as factor graphs and combining the energies of diffusion models trained to sample for individual constraint types. Diffusion-CCSP exhibits strong generalization to novel combinations of known constraints, and it can be integrated into a task and motion planner to devise long-horizon plans that include actions with both discrete and continuous parameters.

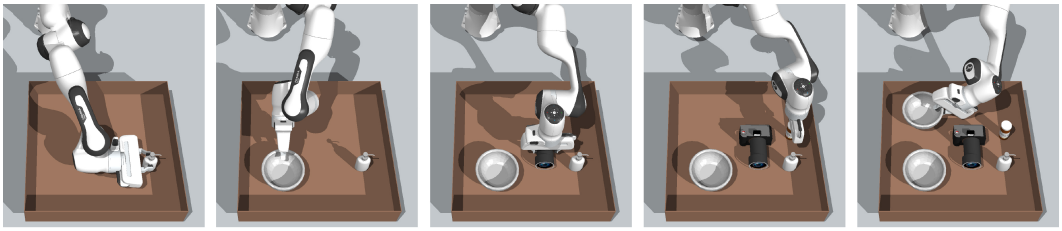
Keywords: Diffusion Models, Constraint Satisfaction Problems, Task and Motion Planning

1 Introduction

Reasoning and planning based on geometric and physical relationships among objects are pivotal in robotic manipulation planning, including generating feasible grasps, physically stable object placements, and poses for dense packing tasks. Existing approaches have used *separate* samplers, obtained through learning or optimization, for each constraint type. For instance, GraspNet [1] focuses on generating valid grasps and StructFormer [2] specializes in generating semantically meaningful object placements. However, complex problems require the solver to conform to multiple constraints simultaneously. Consider the task of 3D object packing with a robot arm, as depicted in Figure 1. Given the geometric and physical properties of objects, our objective is to predict the target pose and motion trajectories that fulfill three essential criteria. First, all objects must be contained within the target container and satisfy qualitative user requirements (e.g., bowls should be placed next to each other). Second, collisions among objects and the robot, during their movement and when stationary, must be avoided. Finally, the final configuration should be physically stable.

In practice, constructing or training a monolithic model capable of solving *every possible combination* of goals and constraints can be challenging due to the poverty of data. Therefore, two key requirements for generally capable robot planners are the ability to *reuse* algorithms for specific goals and constraints and to *compose* individual solvers for overall tasks. A common strategy is to combine specialized methods for solving individual problems in a *sequential* manner. This sequential approach, widely employed by state-of-the-art planners for object rearrangement and general task and motion planning (TAMP) problems [3, 4, 5], rests on rejection sampling. The set of solvers is applied sequentially following a predetermined order guided by heuristics (e.g., first sampling object poses in the box and then sampling grasps and trajectories). If a previously sampled value violates a later constraint (e.g., the sampled pose in the box does not leave enough space for the gripper during object placement), backtracking is performed. This rejection-sampling-based approach can be highly inefficient when faced with many constraints. An alternative approach is to apply optimization-based methods to find solutions to these optimization problems, where the objective

Submitted to the 7th Conference on Robot Learning (CoRL 2023). Do not distribute.



(a) The dispenser is constrained to be grasped only from the sides.

(b) The gripper pose when grasping the bowl should be reachable.

(c) The gripper at grasp poses are constrained be collision-free.

(d) All arm trajectories are constrained to be collision-free.

(e) The two bowls are constrained to be next to each other.

Figure 1: **Solving Continuous Constraint Satisfaction Problems by Composing Diffusion Models.** Our approach combines diffusion models, representing individual constraints, to generate object placement poses that are reachable and allow collision-free grasps. The choice of an object placement depends on both semantic constraints about object placement and collision avoidance constraints on the object and the robot gripper.

function quantifies the geometrical and physical constraints [6, 7]. However, these methods usually require manually-specified differentiable constraint formulas and, therefore do not generalize well to qualitative constraints such as "next-to" which depend on situational context and thus should be learned from data. Furthermore, they are usually sensitive to hyperparameter choices like the smoothness of approximations to discrete constraints such as object contacts.

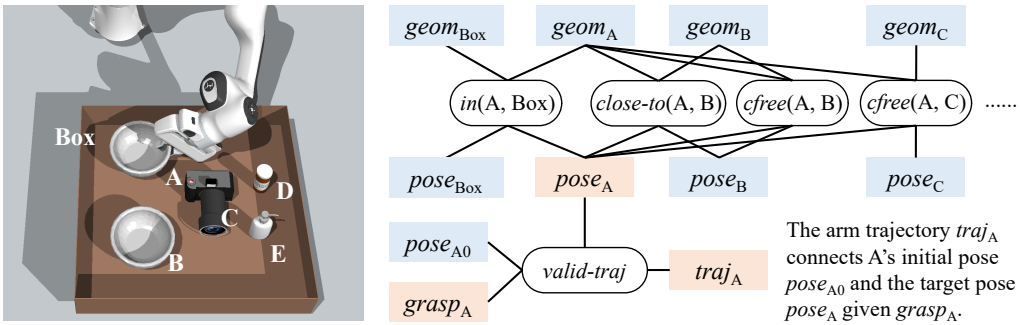
This paper proposes to use constraint graphs as a unified view for all these types of tasks and apply diffusion-model-based constraint solvers to find solutions that jointly satisfy the constraints. A constraint graph consists of nodes representing decision variables (e.g., grasping poses, placement poses, and robot trajectories) and nodes representing constraints among these decision variables. Our model, the compositional diffusion constraint solver (Diffusion-CCSP), learns a collection of modular diffusion models for individual types of constraints and recombines them using an energy-based model (EBM) formulation [8]. In particular, each modular diffusion model learns to generate a distribution of feasible solutions for one particular type of constraint (e.g., collision-free placements and physical stability). Since the diffusion models are generative models of the set of solutions, at inference time we can condition on an arbitrary subset of the variables and solve for the rest. Furthermore, the diffusion process is compositional: since each diffusion model is trained to minimize an implicit energy function, the task of global constraint satisfaction can be cast as minimizing the global energy of solutions (in this case, simply the summation of individual energy functions). These two features introduce notable flexibility in both training and inference. Component diffusion models can be trained independently or simultaneously based on paired compositional problems and solutions. At performance time, Diffusion-CCSP generalizes to novel combinations of known constraint types, even when the constraint graph involves more variables than those seen during training.

We evaluate Diffusion-CCSP on four challenging domains: 2D triangle dense-packing, 2D shape arrangement with qualitative constraints, 3D shape stacking with stability constraints, and 3D object packing with robots. Compared with baselines, our method exhibits efficient inference time and strong generalization to novel combinations of constraints and problems with more constraints.

2 Related Work

Geometric rearrangement. A large body of work has studied geometric rearrangement of objects in robotics subject to conditions specified in natural language [9, 10, 2, 11, 12, 13] or via formal specification or demonstration [14, 15, 16, 17, 18, 19]. Most similar to our work, Liu et al. [13] also use diffusion models to solve for geometric rearrangements of objects. However, their approach is specific to rearrangements, while our work provides a framework for solving arbitrary CCSPs.

Compositional generative modeling. Our work relates to existing work on compositional generative modeling, where individual generative models are combined to generate joint outputs [20, 21, 22, 23, 24, 25, 11, 26], primarily in the setting of image generation. Most similar to our work, Gkanatsios



(a) Visualization of the environment while placing object A.

(b) Visualization of the constraint graphs associated with the object placement. There are three decision variables.

Figure 2: **Continuous Constraint Satisfaction Problem (CCSP) in Robot Planning.** It unifies geometric, physical, and qualitative constraints. To place A into the tray, we need to generate the grasping pose $grasp_A$, placement pose $pose_A$, and the robot arm trajectory. We omit collision-free constraints with robots for brevity.

et al. [11] compose EBMs to represent different subgoals in compositions of object scenes. Like Liu et al. [13], their domain of application is specific to rearrangement.

Task and motion planning (TAMP). There are special-purpose 3D object packing algorithms based on heuristic-search or simulation [27, 28, 29] for finding object poses that satisfy collision and stability constraints. They are again not directly comparable to Diffusion-CCSP, because they cannot address the same range and combinations of constraints.

3 Compositional Diffusion Constraint Solvers

Our goal is to develop a general algorithm capable of learning and solving diverse *continuous constraint satisfaction problems* (CCSPs) encountered in robotic manipulation tasks, which involve both discrete and continuous parameters. To achieve this, we adopt a generic graph-based representation of CCSPs unifying various geometric, physical, and qualitative constraints*.

3.1 Formulation and Representation of Constraint Satisfaction Problems

Formally, a *continuous constraint satisfaction problem* (CCSP) can be represented as a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{U}, \mathcal{C} \rangle$. Each $v \in \mathcal{V}$ is a decision variable (such as the pose of an object), while each $u \in \mathcal{U}$ is a conditioning variable (such as the geometry of an object, which will be constants at performance time). Each $c \in \mathcal{C}$ is a constraint, formally a tuple of $\langle t_c, n_c \rangle$, where t_c is the type of the constraint (e.g., collision-free), and $n_c = (\mathcal{V}^c, \mathcal{U}^c)$ contains two sets of variables in $\mathcal{V}$ and $\mathcal{U}$, which correspond to the argument to this constraint. For example, a collision-free constraint between object A and B can be represented as $\langle \text{cfree}, (pose_{\text{A}}, pose_{\text{B}}, geom_{\text{A}}, geom_{\text{B}}) \rangle$. Here, $pose_{\text{A}}$ and $pose_{\text{B}}$ are decision variables that represent the target pose of objects A and B, while $geom_{\text{A}}$ and $geom_{\text{B}}$ are conditioning variables that represent the geometry of objects A and B.

Let V and U be assignments of values to the decision and conditioning variables $\mathcal{V}$ and $\mathcal{U}$, respectively, and let V^c and U^c be the subsets of assigned values to variables in constraint c. For example, in the 3D packing task shown in Figure 2a, the assignment V to variables $\mathcal{V}$ is a mapping from the variables $pose_i$ to an $SE(3)$ pose of object i. The solution should satisfy that for all $c = \langle t_c, (\mathcal{V}^c, \mathcal{U}^c) \rangle \in C$, $test(t_c, (V^c, U^c)) = 1$, where *test* takes the constraint type and the assignments to each variable and returns 1 if the constraint c is satisfied and 0 otherwise. The function *test* can be implemented as human-specified rules, learned classifiers, or the outcome of a physics simulator.

Example. Figure 2 illustrates the constraint graph associated with the pick-and-place of a bowl (A) into a box close to another bowl (B), while avoiding it and other obstacles. It includes three

*Throughout the paper, we use the word qualitative constraint to refer to constraints that can not be easily and accurately described analytically; in particular, spatial relationships such as "left," "close-to," etc. These constraints depend on contexts and would ideally be learned from human annotations.

Algorithm 1 Training

repeat
  $\mathcal{G}, V_0 \sim \mathcal{D}$
  $t \sim \text{Uniform}(\{1, \ldots, T\})$; $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
    $\nabla_\theta \left\| \epsilon - \sum_{c \in \mathcal{C}} \epsilon_{t_c}(\sqrt{\bar{\alpha}_t} V_0^c + \sqrt{1-\bar{\alpha}_t}\epsilon, t)) \right\|^2$
until converged

Algorithm 2 Sampling

$V_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \ldots, 1$ **do**
  if $t > 1$ **then** $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ **else** $\mathbf{z} = \mathbf{0}$
  $V_{t-1} = \frac{1}{\sqrt{\alpha_t}} \Big(V_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \sum_{c \in \mathcal{C}} \epsilon_{t_c} \left(V_t^c, U^c, t \right) \Big) + \sigma_t \mathbf{z}$
return V_0

decision variables: $\textit{grasp}_{\text{A}}$ (transform between object and hand), $\textit{pose}_{\text{A}}$ (target pose), and $\textit{traj}_{\text{A}}$ (robot trajectory). The scenario involves three constraint groups: 1) qualitative constraints, including $\textit{in}(\text{A}, \text{Box})$ ensuring the bowl's placement within the box and $\textit{close-to}(\text{A}, \text{B})$ asserting proximity between two bowls; 2) collision avoidance constraints such as $\textit{cfree}(\text{A}, \text{B})$, which asserts that objects A and B do not collide in the final placement; 3) trajectory constraints, encapsulated by *valid-traj*, which assert that the trajectory $\textit{traj}_{\text{A}}$ is a feasible robot path that connects the bowl's initial position $\textit{pose}_{\text{A0}}$ and the target $\textit{pose}_{\text{A}}$, given $\textit{grasp}_{\text{A}}$. It is important to note that these constraints are interdependent: the choice of $\textit{pose}_A$ requires the consideration of the robot trajectory to ensure the existence of a valid grasping pose and trajectory.

3.2 Compositional Diffusion Models

Given a set of constraints $\mathcal{C}$, decision variables $\mathcal{V}$, and conditioning values $\mathcal{U} = U$, we wish to find an assignment of V_0 that satisfies $\mathcal{C}$. We represent the conditional distribution of variable assignments given an individual constraint c using a diffusion model $p_c(\mathcal{V}^c = V^c \mid \mathcal{U}^c = U^c) \propto \mathbb{1}[\textit{test}(t_c, (V^c, U^c))]$, which is effectively uniform over the (bounded) region of $\mathcal{V}^c$ that satisfies the constraint. Finding a satisfactory assignment of variables V_0 then corresponds to finding an assignment that maximizes the likelihood in the joint distribution constructed from the product of the individual constraint models. Each constraint diffusion model represents $p(\mathcal{V}^c \mid \mathcal{U}^c)$ in the form of an energy-based model, $p(\mathcal{V}^c \mid \mathcal{U}^c) \propto e^{-E(\mathcal{V}^c | \mathcal{U}^c)}$ through its score [23, 25], so the probability maximization problem corresponds to the energy minimization problem:

$$V_0 = \arg\max_V \prod_{c \in \mathcal{C}} p_c(\mathcal{V}^c \mid \mathcal{U}^c) = \arg\min_V \sum_{c \in \mathcal{C}} E(\mathcal{V}^c \mid \mathcal{U}^c) \ .$$

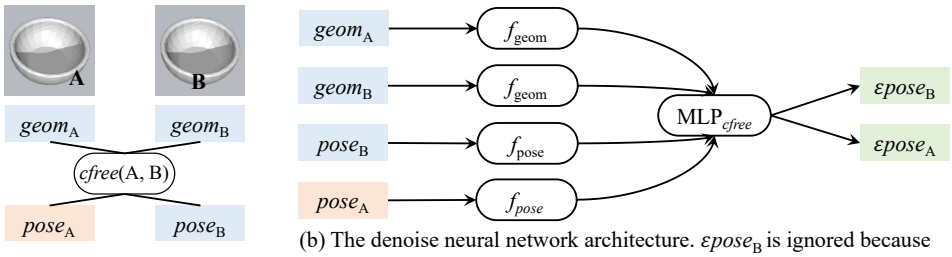
We solve this energy minimization problem by using a sampler based on the annealed unadjusted Langevin algorithm (ULA) discussed by Du et al. [25].

During training, we optimize a noise prediction model ϵ_t for each constraint type t. Each ϵ_t takes conditioning variables U^c and a noisy version of V^c as input, and predicts the noise applied on V^c. Most simply, we could train the diffusion models independently for each constraint, and combine them at prediction time. Instead, since we do not have data from the individual marginals for each constraint, we jointly train all constraint diffusion models.

Specifically, our training dataset $\mathcal{D}$ contains a set of constraint graphs, each with conditioning values and a solution to the decision variables: $\langle \mathcal{G} = (\mathcal{V}, \mathcal{U}, \mathcal{C}), U, V_0 \rangle$. For each type of constraint, we randomly initialize a denoising function ϵ_t, and sum the denoising predictions over the constraints in the graph to denoise a noisy version of V_0:

$$\mathcal{L}_{\text{MSE}} = \mathop{\mathbb{E}}_{\substack{\langle \mathcal{G}=(\mathcal{V},\mathcal{U},\mathcal{C}), U, V_0 \rangle \in \mathcal{D} \\ \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I}) \\ t \sim \text{Uniform}(1, \cdots, T)}} \left[\left\| \epsilon - \sum_{c \in \mathcal{C}} \epsilon_{t_c} \left(\sqrt{\bar{\alpha}_t} V_0^c + \sqrt{1-\bar{\alpha}_t}\epsilon, U^c, t \right) \right\|^2 \right],$$

where $\boldsymbol{\epsilon}$ is random Gaussian noise applied to V_0, α_t are predefined noise scheduling for training and sampling following Nichol and Dhariwal [30], $\bar{\alpha}_t := \prod_{i=1}^t \alpha_t$, t is a randomly sampled diffusion step, and T is the total number of diffusion steps. The full pseudocode for training and sampling from our conditional diffusion models for constraints can be found in Algorithm 1 and Algorithm 2.



(a) The *cfree* constraint.

(b) The denoise neural network architecture. $\varepsilon pose_{\mathrm{B}}$ is ignored because $pose_{\mathrm{B}}$ is fixed in this case.

Figure 3: **Illustration of the denoising neural network** for satisfying constraint *cfree* (collision-free) and the corresponding gradient map learned by Diffusion-CCSP.

3.3 Geometric and Physical Constraint Solving with Diffusion-CCSP

Given the general formulation of Diffusion-CCSP, we describe its concrete implementation in our robot planning setting. Recall that for each type of constraint, we have defined a "denoise" function ϵ_t that takes the conditioning values and (the noisy version of) the decision values as inputs, and predicts the noise applied on the decision values. In our robotic domains, we focus on two variable types: geometry variables and pose/transformation variables. We illustrate the neural network architecture for collision-free constraints though the construction easily generalizes to other constraint types.

Recall that the argument to a collision-free constraint c has four elements: $\mathcal{U}^c = (geom_{\mathrm{A}}, geom_{\mathrm{B}})$, $\mathcal{V}^c = (pose_{\mathrm{A}}, pose_{\mathrm{B}})$. We use $pose^t_{\mathrm{A}}$ to denote the value of this variable at diffusion time step t. The output of ϵ_{cfree} is a tuple $(\epsilon pose^t_{\mathrm{A}}, \epsilon pose^t_{\mathrm{B}})$ which corresponds to the noise applied on $pose_{\mathrm{A}}$ and $pose_{\mathrm{B}}$ at diffusion time step t. In our implementation, object shapes are represented as a 3D axis-aligned bounding boxes, and poses are represented as a 5-dimensional feature vectors composed of the 3-DoF translation and rotation γ about $+z$ axis, encoded using $\sin(\gamma)$ and $\cos(\gamma)$. We use a multi-layer perceptron (MLP) f_{geom} to encode the object shapes (shared across all constraints) and a constraint-specific MLP f_{pose} to encode poses. Both geometric features and pose encoders map input representations into a fixed-length vector embedding. The function ϵ_{cfree} is implemented as

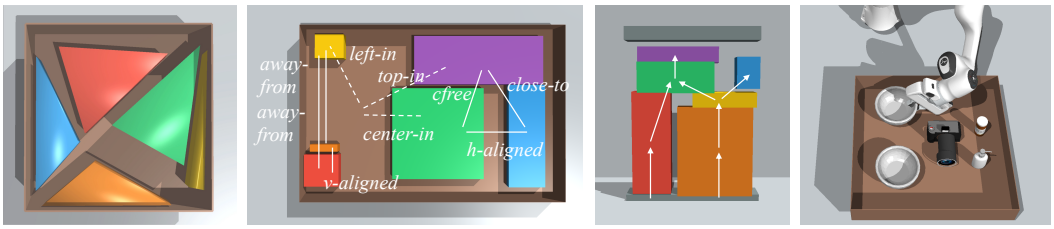
$$\epsilon^t_{\text{cfree}}(geom_{\mathrm{A}}, geom_{\mathrm{B}}, pose_{\mathrm{A}}, pose_{\mathrm{B}}) = g_{\text{cfree}}\left(f_{\text{geom}}(geom_{\mathrm{A}}) \oplus f_{\text{geom}}(geom_{\mathrm{B}}) \oplus f_{\text{pose}}(pose_{\mathrm{A}}) \oplus f_{\text{pose}}(pose_{\mathrm{B}}) \oplus t\right),$$

where $\oplus$ denotes vector concatenation, and g_{cfree} is another MLP model that takes the concatenated features of inputs and outputs a vector of length 10. The first 5 entries correspond to the prediction of $\epsilon pose^t_{\mathrm{A}}$, and the rest corresponds to $\epsilon pose^t_{\mathrm{B}}$. t is the decoding time step.

Remark. The generative nature of diffusion models allows us to perform training and inference over different combinations of conditioning variables and decision variables. For example, during both training and inference time, depending on the actual split of conditioning variables and decision variables in the constraint graph, we may have $pose_{\mathrm{A}}$ given and fixed, and only optimize for $pose_{\mathrm{B}}$. Second, although we have been using a simplified representation of object geometries (i.e., bounding boxes) and poses (considering only translation and a 1 DoF rotation), our framework can be naturally extended to support point clouds as inputs and the full $SE(3)$ poses, possibly through the exponential maps of $SE(3)$ poses [31].

Example. Figure 3 illustrates the neural network architecture for the specific constraint and the corresponding gradient map learned by Diffusion-CCSP. In this example, our objective is to generate a pose for object A that avoids collisions with object B in its current position. The neural network encodes the four input features and predicts the noise for $pose_{\mathrm{A}}$ and $pose_{\mathrm{B}}$. As object B is fixed, the noise (gradient) predicted by ϵ_{cfree} for its pose is disregarded. Figure 3c visualizes the energy function and the gradient map for the X and Y components of $pose_{\mathrm{A}}$, where the gradient points approximately outward from the current location of object B. It is important to note that the gradient over the pose variable $pose_{\mathrm{A}}$ will have contributions from multiple constraints in the constraint graph.

Planning. So far, we have presented a generic solution to solving constraint satisfaction problems that involve geometric and physical constraints, assuming that the constraint graph is given as input to the algorithm. This approach can be directly used to solve particular tasks such as the pose prediction



(a) 2D Triangle Packing. **(b) 2D Shape Arrangement with Qualitative Constraints** **(c) 3D Object Stacking with Stability Constraints.** **(d) 3D Object Packing with Robots.**

Figure 4: **Illustration of four domains.** They contain geometric, physical, and qualitative constraints. (a) Triangle packing. (b) Dense 2D packing with qualitative constraints. The figure shows a subset of 45 constraints of 13 types. (c) 3D object stacking. The arrows show the support relationships. (d) 3D object packing with a panda robot.

task in object rearrangements. Our primary motivation is to solve subproblems that arise during general robot manipulation planning. In Appendix A we illustrate how Diffusion-CCSP can be integrated with a search algorithm to solve general task and motion planning (TAMP) problems, where the constraint graphs are automatically constructed based on the sequence of actions that has been applied and the goal specification of the task.

4 Experiments

We evaluate Diffusion-CCSP through a sequence of continuous CSP tasks incorporating geometric, physical, and qualitative constraints. We include all environmental details in Appendix B.

We compare our full model to a strong sequential-sampling baseline and another model variant. ***Sequential Sampling***: We sequentially sample each decision variable according to a generic sampler (e.g., for 2D poses, we randomly sample a location that is within the tray and a random rotation) and check all geometric constraints (e.g., *in* and *cfree* associated with the decision variable. For each variable, we sample 50 samples, and report failure when no successful samples can be produced within 50 samples. ***Diffusion-CCSP w. Reverse Sampling***: In this variant, we directly use the standard reverse diffusion process to sample from composed diffusion models [23]. This method is faster at sampling than ULA, but is generally found to have worse generalization performance. We include a detailed comparison between these two in Appendix C.

4.1 2D Triangle Packing

The first domain considers packing a set of random triangles into a square tray, ensuring no overlap. Triangles can be rotated, and their shapes are encoded by their vertex coordinates in their zero poses. The model's output includes a 2D translation vector and a 1D rotation. Training data consists of 30,000 solutions to problems with 2-4 triangles, and testing includes 100 problems with 2-5 triangles. We generate training solutions by randomly splitting the tray.

Fig. 5a shows the result, and Fig. 4a shows a concrete testing problem and its solution found by Diffusion-CCSP. When the number of triangles becomes large, it becomes increasingly difficult for the baselines to succeed. By contrast, Diffusion-CCSP successfully finds solutions to over 60 percent of the problems in just 10 samples for the most challenging 6-triangle packing problems, and ULA outperforms Reverse Sampling in generalization to 5 objects.

4.2 2D Shape Arrangement with Qualitative Constraints

This domain involves packing rectangles into a 2D box, satisfying geometric and qualitative constraints resembling scenarios such as dining table setting and office desk arrangement. Qualitative constraints include *center-in*$(\mathrm{A}, \mathrm{Box})$, *left-in*$(\mathrm{A}, \mathrm{Box})$, *left-of*$(\mathrm{A}, \mathrm{B})$, *close-to*$(\mathrm{A}, \mathrm{B})$, *horizontally-aligned*$(\mathrm{A}, \mathrm{B})$, etc., along with the geometric ones such as containment (*in*) and collision-free. Different problem instances have different tray sizes, box shapes, and constraints. The training dataset consists of 30,000 problems and solutions with 2 to 4 objects, while the testing set contains 100 problems with 2 to 5 objects. Training examples are generated by randomly splitting the tray.

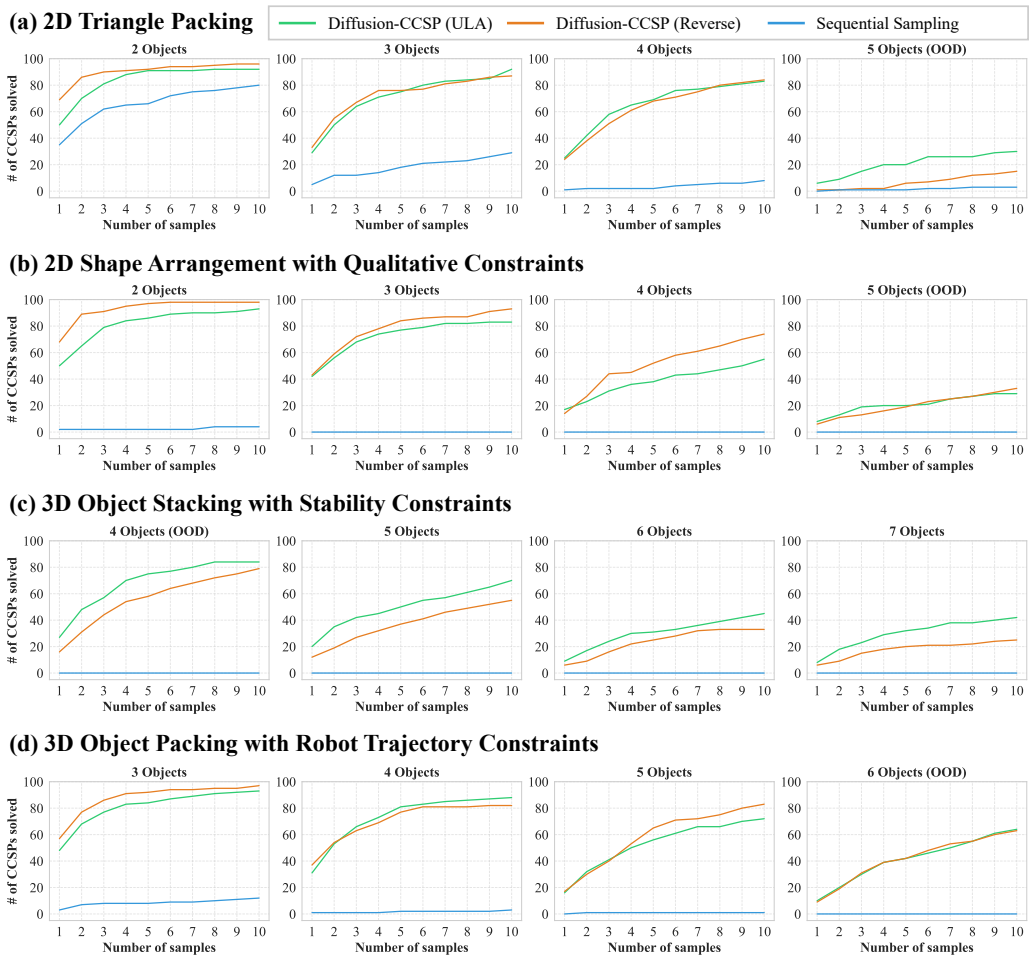


Figure 5: **Quantitative Comparisons of Constraint Solvers.** Accumulated number of problems solved in 10 runs of different models. OOD=Out of Training Distribution. The sequential sampling baseline completely failed for task (c) and hard tasks in (b) and (d).

Fig. 5b shows the result, and Fig. 4b shows a concrete test problem with involves six objects 45 constraints (including qualitative constraints shown in the figure and geometric constraints such as *in* and *collision-free*). The task is very challenging when the number of objects and the number of constraints becomes large. Therefore, the baseline rejection sampling method barely solves any problem instances with 6 objects. Diffusion-CCSP significantly outperforms the baselines. In this environment, we empirically found that ULA did not sample stably and thus is outperformed by Reverse Sampling.

4.3 3D Object Stacking with Stability Constraints

The third domain considers arranging trapezoidal prisms onto a shelf to satisfy a given support structure and stability. The goal is to generate a sequence of object placement actions (including object names and their target poses). This reflects real-life tasks such as organizing storage containers, which often involves tilted boxes due to varying prism heights. All of our training and testing problems involve at least one "bridge-like" structure, i.e., there is at least one object supported by multiple objects simultaneously. Furthermore, since the output of models is a sequence of object placements, a sequence is successful only if the final stage and *all intermediate states* are physically stable. We randomly sample the size of the cabinet and the size of the shapes. We use a physical simulator to generate training examples. The training dataset has 24,000 problem instances and solutions with 5 to 7 objects, and the testing set includes 100 problem instances with 4 to 7 objects.

Objects	N=4	N=5
# of Calls to Diffusion-CCSP	11.9	33.3

(a) 3D Object Stacking.

Objects	N=2	N=3	N=4	N=5	N=6
# of Calls to Diffusion-CCSP	1.5	2.7	5.9	16.1	25.0

(b) 3D Object Packing with Robots.

Table 1: Average number of calls to Diffusion-CCSP in order to solve the full TAMP problem.

An example of the task is illustrated in Fig. 4c. Given that neither our algorithm nor the baselines specifically plan for object placement orders, we integrate them into a very simple task and motion planner, as detailed in Appendix A. This planner randomly samples object placement orders and box "support" structures (i.e., the arrangement of objects supporting other objects), then invokes the Diffusion-CCSP solver to find solutions. We assess two metrics: first, the performance of Diffusion-CCSP with a fixed CSP graph, illustrated in Fig. 5c; second, the number of calls to Diffusion-CCSP required to solve problems involving varying quantities of objects, as shown in Table 1a. For instance, packing 6 boxes into a cabinet necessitates, on average, 25 different object placement and support structure samples. If a CSP is not solvable by Diffusion-CCSP, it might truly be infeasible due to incorrect sampled placement orders or support structures. Therefore, Table 1a shows the integrated system's overall performance. Notably, given that we can process multiple CSPs as a GPU batch and that generating object orders is inexpensive, the system does not have high latency overall. It takes on average 0.01-0.05 seconds for Diffusion-CCSP (Reverse) to solve each CCSP and 0.18-0.28 seconds for Diffusion-CCSP (ULA) to solve each CCSP.

4.4 3D Object Packing with Robot Trajectory Constraints

This domain involves packing 3D objects into a box container in a simulated environment using PyBullet [32], with a Franka Emika Panda arm. The dataset includes 53 objects from the PartNet Mobility dataset and ShapeNet [33, 34], some of which only afford side grasps, while others can be grasped from the top. We pre-generate a set of grasps for each object, and use a motion planner to find robot trajectories. The task and motion planner and Diffusion-CCSP jointly solve for grasping choices, object poses, and arm motions. Although our model doesn't directly generate grasps or trajectories, it reasons about object geometries and has learned to predict poses for which it is likely that there are collision-free trajectories (which is later verified with a motion planner.) The training dataset has 10,000 problem instances and solutions with 3 to 5 objects, and the testing set includes 100 problem instances with 3 to 6 objects.

Fig. 5d illustrates the number of samples needed for tasks of varying sizes. Overall, Diffusion-CCSP significantly outperforms rejection-based samplers, especially for larger numbers of objects. Diffusion-CCSP can even generalize directly to six-object packing problems without additional training. Table 1b showcases the full pipeline's performance, which includes generating placement orders, grasps, poses, and trajectories.

5 Limitation and Conclusion

Limitations. Our current approach is limited by the simple shape encoding for objects, especially in cases that involve concave objects whose bounding boxes overlap, but the objects themselves do not. Additionally, the generation of task labels and solutions now relies on hand-coded rules, which is limited, especially for qualitative constraints such as "setting the dining table." Future improvements could include incorporating sophisticated shape encoders and learning constraints from real-world data, such as online images, for a wider range of applications.

Conclusion. In conclusion, this paper proposes a learning-based approach that leverages the compositionality of continuous constraint satisfaction problems (CCSP) and uses diffusion models as compositional constraint solvers. The proposed algorithm, namely the compositional diffusion constraint solver (Diffusion-CCSP), consists of modular diffusion models for different types of constraints. By conditioning these models on subsets of variables, solutions can be generated for the entire CCSP. The model exhibits strong generalization capabilities and can handle novel combinations of known constraints, even with more objects and constraints than encountered during training. It can also be integrated with search algorithms to solve general task and motion planning problems.

References

[1] A. Mousavian, C. Eppner, and D. Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2901–2910, 2019. 1

[2] W. Liu, C. Paxton, T. Hermans, and D. Fox. Structformer: Learning spatial structure for language-guided semantic rearrangement of novel objects. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6322–6329. IEEE, 2022. 1, 2

[3] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477. IEEE, 2011. 1

[4] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *ICAPS*, 2020. 1, 13

[5] C. Paxton, C. Xie, T. Hermans, and D. Fox. Predicting stable configurations for semantic placement of novel objects. In *Conference on Robot Learning*, pages 806–815. PMLR, 2022. 1

[6] M. Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, pages 1930–1936, 2015. 2

[7] T. Pang, H. Suh, L. Yang, and R. Tedrake. Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models. *arXiv preprint arXiv:2206.10787*, 2022. 2

[8] Y. Du and I. Mordatch. Implicit generation and generalization in energy-based models. *arXiv preprint arXiv:1903.08689*, 2019. 2

[9] M. Shridhar, L. Manuelli, and D. Fox. Cliport: What and where pathways for robotic manipulation. In *CoRL*, 2021. 2

[10] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. In *CoRL*, 2020. 2

[11] N. Gkanatsios, A. Jain, Z. Xian, Y. Zhang, C. Atkeson, and K. Fragkiadaki. Energy-based models as zero-shot planners for compositional scene rearrangement. *arXiv preprint arXiv:2304.14391*, 2023. 2, 3

[12] O. Mees, L. Hermann, and W. Burgard. What matters in language conditioned robotic imitation learning over unstructured data. *IEEE Robotics and Automation Letters (RA-L)*, 2022. 2

[13] W. Liu, Y. Du, T. Hermans, S. Chernova, and C. Paxton. Structdiffusion: Language-guided creation of physically-valid structures using unseen objects. *arXiv preprint arXiv:2211.04604*, 2022. 2, 3

[14] A. Simeonov, Y. Du, B. Kim, F. R. Hogan, J. Tenenbaum, P. Agrawal, and A. Rodriguez. A long horizon planning framework for manipulating rigid pointcloud objects. In *CoRL*, 2020. 2

[15] A. Curtis, X. Fang, L. P. Kaelbling, T. Lozano-Pérez, and C. R. Garrett. Long-horizon manipulation of unknown objects via task and motion planning with estimated affordances. In *ICRA*, 2022. 2

[16] C. Paxton, C. Xie, T. Hermans, and D. Fox. Predicting stable configurations for semantic placement of novel objects. In *CoRL*, 2021. 2

[17] W. Yuan, C. Paxton, K. Desingh, and D. Fox. Sornet: Spatial object-centric representations for sequential manipulation. In *CoRL*, 2021. 2

[18] A. Bobu, C. Paxton, W. Yang, B. Sundaralingam, Y.-W. Chao, M. Cakmak, and D. Fox. Learning perceptual concepts by bootstrapping from human queries. *IEEE Robotics and Automation Letters*, 2022. 2

[19] A. Simeonov, Y. Du, L. Yen-Chen, , A. Rodriguez, , L. P. Kaelbling, T. L. Perez, and P. Agrawal. Se(3)-equivariant relational rearrangement with neural descriptor fields. In *CoRL*, 2022. 2

[20] Y. Du, S. Li, and I. Mordatch. Compositional visual generation with energy based models. *Advances in Neural Information Processing Systems*, 33:6637–6647, 2020. 2

[21] N. Liu, S. Li, Y. Du, J. Tenenbaum, and A. Torralba. Learning to compose visual relations. *Advances in Neural Information Processing Systems*, 34:23166–23178, 2021. 2

[22] W. Nie, A. Vahdat, and A. Anandkumar. Controllable and compositional generation with latent-space energy-based models. *Advances in Neural Information Processing Systems*, 34, 2021. 2

[23] N. Liu, S. Li, Y. Du, A. Torralba, and J. B. Tenenbaum. Compositional visual generation with composable diffusion models. *arXiv preprint arXiv:2206.01714*, 2022. 2, 4, 6

[24] J. Urain, A. Li, P. Liu, C. D'Eramo, and J. Peters. Composable energy policies for reactive motion generation and reinforcement learning. *arXiv preprint arXiv:2105.04962*, 2021. 2

[25] Y. Du, C. Durkan, R. Strudel, J. B. Tenenbaum, S. Dieleman, R. Fergus, J. Sohl-Dickstein, A. Doucet, and W. Grathwohl. Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and mcmc. *arXiv preprint arXiv:2302.11552*, 2023. 2, 4

[26] T. Wu, M. Tjandrasuwita, Z. Wu, X. Yang, K. Liu, R. Sosic, and J. Leskovec. Zeroc: A neuro-symbolic model for zero-shot concept recognition and acquisition at inference time. *Advances in Neural Information Processing Systems*, 35:9828–9840, 2022. 2

[27] J. Egeblad, B. K. Nielsen, and A. Odgaard. Fast neighborhood search for two-and three-dimensional nesting problems. *European Journal of Operational Research*, 183(3):1249–1266, 2007. 3

[28] X. Liu, J.-m. Liu, A.-x. Cao, and Z.-l. Yao. Hape3d—a new constructive algorithm for the 3d irregular packing problem. *Frontiers of Information Technology & Electronic Engineering*, 16 (5):380–390, 2015. 3

[29] F. Wang and K. Hauser. Stable bin packing of non-convex 3d objects with a robot manipulator. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8698–8704. IEEE, 2019. 3

[30] A. Q. Nichol and P. Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021. 4

[31] J. Urain, N. Funk, G. Chalvatzaki, and J. Peters. Se (3)-diffusionfields: Learning cost functions for joint grasp and motion optimization through diffusion. *arXiv preprint arXiv:2209.03855*, 2022. 5

[32] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019. 8

[33] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 8

[34] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11097–11107, 2020. 8

[35] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021. 12

[36] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research*, 37 (10):1134–1151, 2018. 13

Supplementary Material for
Compositional Diffusion-Based Continuous Constraint Solvers

Three videos are included in the supplementary directory:

1. `csp_solve_task1_triangle_packing.mp4` and `csp_solve_task4_robot_packing.mp4` are execution videos of solutions generated by our model Diffusion-CCSP.
2. `data_gen_task3_stability.mp4` shows the data generation simulation for task 3.

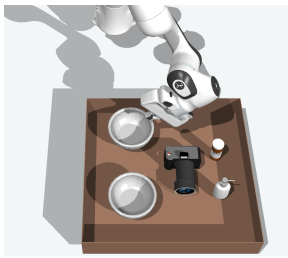
The appendix includes two sections. Appendix A discusses how to integrate Diffusion-CCSP for CCSP solving with task and motion planning (TAMP) algorithms. Appendix B details how our datasets are collected for each task and how the inputs are encoded for diffusion models. Appendix C includes further experiments on the number of samples it take Diffusion-CCSP to solve problems in each task.

A Integration with Task and Motion Planning Algorithms

So far, we have presented a generic solution to solving constraint satisfaction problems that involve geometric and physical constraints. However, assuming that the constraint graph is given as input to the algorithm. This approach can be directly used to solve particular tasks such as the pose prediction task in object rearrangements. Next, we illustrate how the proposed method can be integrated with a search algorithm to solve general task and motion planning (TAMP) problems, where the constraint graphs are automatically constructed based on the sequence of actions that has been applied and the goal specification of the task. For brevity, we will present a simplified formulation of TAMP problems. For more details, please refer to the recent survey [35].

Formally, given a space $\mathcal{S}$ of world states, a problem is a tuple $\langle \mathcal{S}, s_0, \mathcal{G}, \mathcal{A}, \mathcal{T} \rangle$, where $s_0 \in \mathcal{S}$ is the initial state (e.g., the geometry and poses of all objects), $\mathcal{G} \subseteq \mathcal{S}$ is a goal specification (e.g., as a logical expression that can be evaluated based on a state: $\textit{in}(\mathrm{A}, \mathrm{Box})$ *and* $\textit{in}(\mathrm{B}, \mathrm{Box})$), $\mathcal{A}$ is a set of continuously parameterized actions that the agent can execute (e.g., pick-and-place), and $\mathcal{T}$ is a partial environmental transition model $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. Each action a is parameterized by two functions: precondition $\textit{pre}_a$ and effect $\textit{eff}_a$. The semantics of this parameterization is that: $\forall s \in \mathcal{S}. \forall a \in \mathcal{A}. \textit{pre}_a(s) \Rightarrow (\mathcal{T}(s, a) = \textit{eff}_a(s))$.

The goal of task and motion planning is to output a sequence of actions $\bar{a}$ so that the terminal state s_T induced by applying a_i sequentially following $\mathcal{T}$ satisfies $s_T \in \mathcal{G}$. The state space and action space are usually represented as STRIPS-like representations: the state is a collection of state variables and each action is parameterized by a set of arguments. Figure 6 shows a simplified definition of the pick-and-place action in a table-top manipulation domain.



```
;; pick up x and and place x to p.
action pick-place(x, g, p, t)
  pre: valid-grasp(x, pose[x], g)        ;; g is the grasp pose on x
       valid-traj(x, pose[x], g, p, t)   ;; p is the target pose of x
       forall z. cfree(x, z, pose[x], t) ;; t is the robot trajectory
  eff: pose[x] := p                      ;; object x will be moved

goal: in(A, C, pose[A], pose[C]) and in(B, C, pose[B], pose[C])
```

The precondition-effect definition of a continuously parameterized action pick-place: pick and place object x.

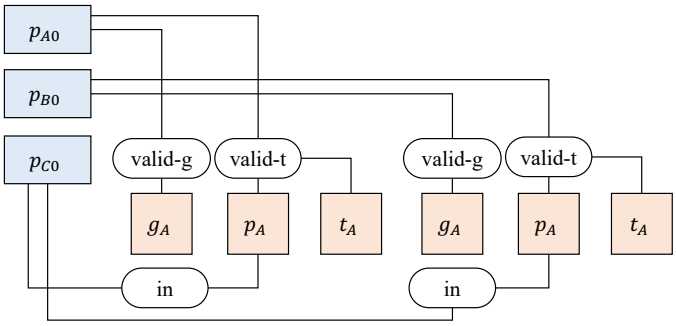
Figure 6: **Illustration of a simple task and motion planning problem.** The domain contains only one action: pick-and-place of objects. It contains three preconditions: g should be a valid grasp pose on object x, t should be a valid robot trajectory that moves x from its current pose to the target pose p, and during the movement, the robot and the object x should not collide with any other objects y. If successful, the object will be moved to the new location. The goal of the task is to pack both objects into the target box without any collisions.

```
pick-place(A, gA, pA, tA)
 valid-grasp(A, pA0, gA)
 valid-traj(A, pA0, gA, pA, tA)
 cfree(A, B, pB0, tA)
 cfree(A, C, pC0, tA)

pick-place(B, gB, pB, tB)
 valid-grasp(B, pB0, gB)
 valid-traj(B, pB0, gB, pB, tB)
 cfree(B, A, pA, tB)
 cfree(B, C, pC0, tB)

Goal
 in(A, C, pA, pC0)
 in(B, C, pB, pC0)
```

(a) A partially specified plan skeleton and its corresponding constraints.



(b) The corresponding TAMP constraint graph derived from the plan skeleton. cfree constraints and geometric features are omitted for brevity.

Figure 7: Example of a partially specified plan skeleton in task and motion planning, together with the corresponding set of constraints. p_{A0}, p_{B0}, and p_{C0} corresponds to the initial pose of objects.

A characteristic feature of TAMP problems is that the decision variables (i.e., the arguments of actions) include both discrete variables (e.g., object names) and continuous variables (e.g., poses and trajectories). Therefore, one commonly used solution is to do a bi-level search [36, 4]: the algorithm first finds a plan that involves only discrete objects (called a *plan skeleton*) and uses a subsequent CSP solver to find assignments of continuous variables, backtracking to try a different high-level plan if the CCSP is found to be infeasible. For example, consider the discrete task plan (also called *plan skeleton*): *pick-and-place*(A), *pick-and-place*(B). There are six parameters to decide: the grasps on A and B, the place locations of A and B, and the robot trajectory while transporting A and B. Based on the preconditions of actions and the goal condition of the task, we can obtain the constraint graph. Therefore, each solution to the CCSP constructed based on the plan skeleton corresponds to a concrete plan of the original planning problem. By combining the high-level search of plan skeletons and our constraint solver Diffusion-CCSP, the integrated algorithm is capable of solving a wide range of task and motion planning problems that involves geometric, physical, and qualitative constraints.

B Problem Domains and Data Generation

Next, we describe how data are generated and how variables are encoded for each task. All datasets are balanced, i.e. the number the examples involving different number of objects are the same. In neural network input encoding, all object dimensions and poses are normalized with regard to that of the container or shelf region.

B.1 2D Triangle Packing

Figure 8: Example collision-free configurations of triangles

Data. The data is generated by randomly sampling points in the square and, connecting them with each other and the edge points using the Bowyer–Watson algorithm, then shrinking each triangle a little to make space among them.

Encoding. We define the resting pose of a triangle as the pose when the vertex A facing the shortest side BC is at origin and its longest side AB is aligned horizontally. Its geometry is encoded three numbers, the length of the longest side $|AB|$ and the vector $(x, y) = \overrightarrow{AC}$. Its pose is encoded as the 2D coordinates of vertex A, along with the *sin* and *cos* values of the rotation θ from the testing pose.

B.2 2D Shape Arrangement with Qualitative Constraints

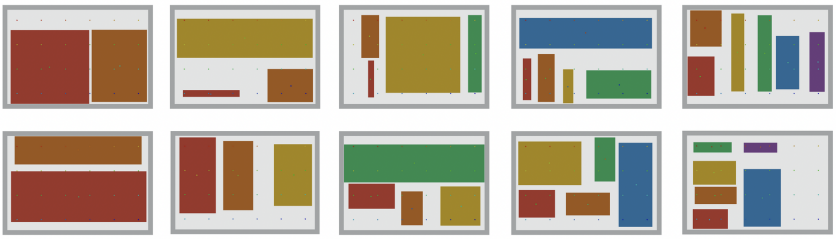


Figure 9: Example rearrangements of rectangles with `in`, `cfree`, and 11 types of qualitative constraints.

Data. The data is generated by recursively splitting the tray at different proportions until depth 3. For each resulting region, a random padding is added to each side. Regions whose area or side is too small are discarded. Labels of qualitative constraints are created by hand-crafted rules, e.g. `'close-to'` means the distance between two objects is smaller than the maximum width of two objects. All qualitative constraints include `'center-in'`, `'left-in'`, `'right-in'`, `'top-in'`, `'bottom-in'`, `'left-of'`, `'top-of'`, `'close-to'`, `'away-from'`, `'h-aligned'`, `'v-aligned'`.

Encoding. The resting pose of a rectangle is when its longer side is oriented horizontally. Its geometry is encoded using its width and length at resting pose. It's pose is the 2D pose of the centroid, along with the sin and cos encoding of the object's yaw rotation. Many problems require some objects to be at vertical positions in order for all the constraints to be satisfied.

B.3 3D Object Stacking with Stability Constraints

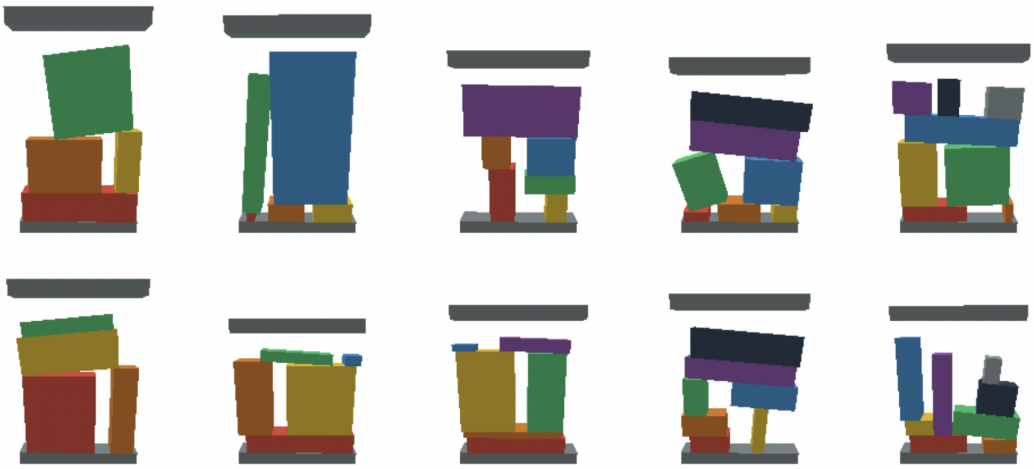


Figure 10: Example stable configurations of rectangles, with `in`, `cfree`, and `supported-by` constraints. There is at least one object that's supported by multiple objects.

Data. The data is generated by randomly splitting a 2D vertical region and shrinking and cutting each box. The resulting small boxes are initiated in PyBullet simulator and letting it drop until rest. We filter for configurations where the final state is stable. Then all objects are removed one by one to test that each intermediate configuration is also stable. An upper shelf is added to be just enough to fit the current configuration.

Encoding. The resting pose of a box is rotated so that the longer side is oriented horizontally. The rectangle's geometry is encoded using its width and length. The rectangle's pose is the 2D pose of the centroid, along with the sin and cos encoding of the object's $roll$ rotation.

B.4 3D Robot Packing with Robots

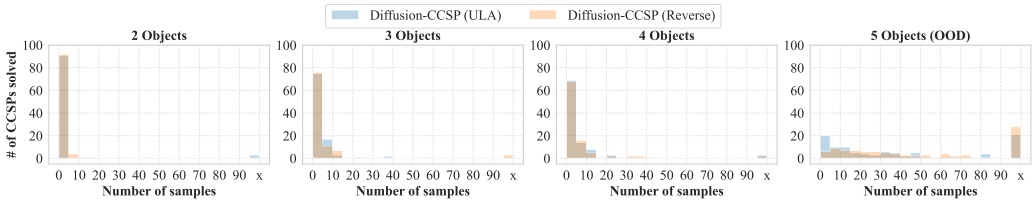
Figure 11: Example configuration of 3D shapes that enables the robot place each object in a given sequence without colliding into other objects already placed. All the bottles, dispensers, and bowls affords only side grasps.

Data. A fixed number of grasps are generated for each object that points directly to one of its five faces, e.g. $+x, -x, +y, -y, +z$. The data is generated by randomly splitting the tray into rectangle regions, then fitting into each region an object with random scale and orientation. The collision-free configuration is then tested by a TAMP planner to see whether a placement order and corresponding grasps can be found (by enumerating all combinations of order and grasps) so that the gripper at each grasp pose won't collide with any objects already in the goal region.

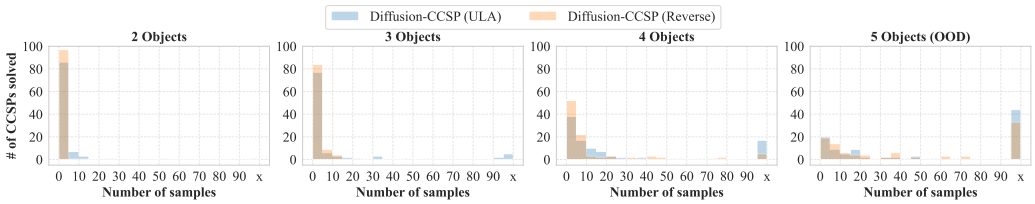
Encoding. The objects' geometry is encoded using axis-aligned bounding box (e.g., width, length, height). An object's grasp is encoded using a five-dimensional vector, indicating the face that the gripper points to. For example, $[0, 0, 0, 0, 1]$ is a top grasp. The object's pose is encoded using its 3D coordinate in the tray frame, along with the sin and cos encoding of the object's yaw rotation.

C Additional Sampler Comparison

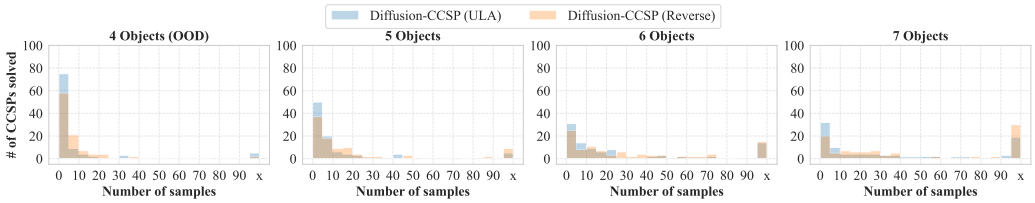
In the main experiment section, we let Diffusion-CCSP and baselines generate 10 samples for each CCSP and check if all constraints are satisfied. Here we let Diffusion-CCSP generate 100 samples on 100 problems, and plot the number of problems it took Diffusion-CCSP to solve for each task. In a batch of 100 CCSPs, it takes on average 0.01-0.05 sec for Diffusion-CCSP (Reverse) to solve each CCSP while 0.18-0.28 for Diffusion-CCSP (ULA) to solve each CCSP.



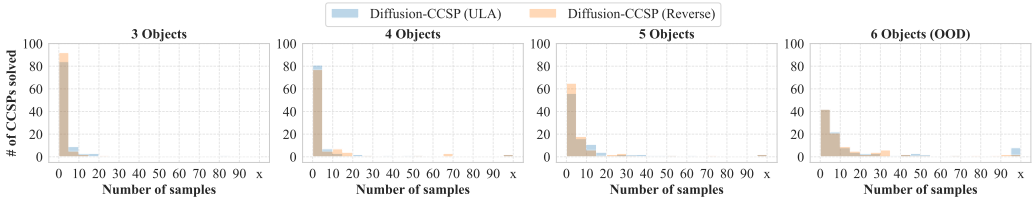
(a) Task 1: 2D Triangle Packing



(b) Task 2: 2D Shape Arrangement with Qualitative Constraints



(c) Task 3: 3D Object Stacking with Stability Constraints



(d) Task 4: 3D Object Packing with Robots

Figure 12: Number of Diffusion-CCSP runs it takes to solve 100 CCSPs. OOD means out-of-distribution problems that include more objects than trained on. x in x ticks means problems not solved within 100 runs.