Modular Robot Design Synthesis with Deep Reinforcement Learning

Julian Whitman[1], **Raunaq Bhirangi**[2], **Matthew Travers**[2], **Howie Choset**[2]
[1]Department of Mechanical Engineering, Carnegie Mellon University
[2]The Robotics Institute, Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, Pennsylvania 15213
jwhitman@cmu.edu

Abstract

Modular robots hold the promise of versatility in that their components can be re-arranged to adapt the robot design to a task at deployment time. Even for the simplest designs, determining the optimal design is exponentially complex due to the number of permutations of ways the modules can be connected. Further, when selecting the design for a given task, there is an additional computational burden in evaluating the capability of each robot, e.g., whether it can reach certain points in the workspace. This work uses deep reinforcement learning to create a search heuristic that allows us to efficiently search the space of modular serial manipulator designs. We show that our algorithm is more computationally efficient in determining robot designs for given tasks in comparison to the current state-of-the-art.

1 Introduction

Modular robots offer the potential to create customized robots that can be readily deployed to perform a variety of tasks. Synthesizing the design of a modular robot for a given task involves a number of challenges, one being that the space of possible module *arrangements* (ordered sequences of discrete robotic modules) grows exponentially in the number of types of modules and ways they can be connected. When searching this exponentially large space, we have to evaluate whether each candidate robot can complete the task. This evaluation requires planning for and comparing the relative costs of motions for each candidate, which is computationally intractable at scale. We address this intractability by learning a search heuristic which implicitly encodes robot performance under the evaluation metric. The main contribution of this work is an algorithm which uses deep reinforcement learning to create this heuristic, enabling us to efficiently search the space of arrangements in the context of each robot's inherent capabilities for a given task. We limit the scope of the problem to synthesizing the arrangements of modular serial manipulators, for tasks in which the manipulator must reach a set of quasi-static workspace positions and orientations.

We build on prior modular design synthesis methods (Desai, Yuan, and Coros 2017; Ha et al. 2018) which incrementally construct and search a tree of different modular arrangements. Specifically, each node added as a child to a current leaf node represents adding a module to the distal end of the manipulator, as shown in Figure 1. We view the construction of this tree as a series of states (arrangements) and actions (adding modules), and treat assembly of an arrangement as a Markov Decision Process (Sutton 1988). Under this formulation, we learn a state-action value function which approximates the benefit of adding each module type to an arrangement given the task. We train a deep Q-network

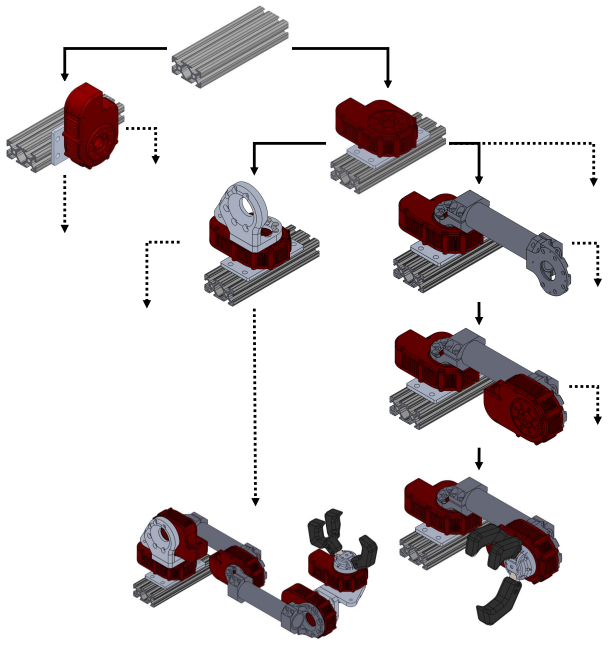

Figure 1: Our approach searches for modular manipulator designs by viewing the space of arrangements as a tree, where modules are sequentially added to the end of the robot. The arrangement at the root of the tree is a base mounting location. Solid arrows represent module additions, and dashed arrows indicate that the tree continues but is not shown. We use deep reinforcement learning to create a data-driven search heuristic which guides search on this tree. We apply our algorithm to modular components produced by Hebi Robotics (Hebi Robotics 2019).

PREPRINT: To appear in the proceedings of the 2020 AAAI Conference on Artificial Intelligence.

(DQN) to approximate this value function using reinforcement learning (Mnih et al. 2015). The DQN is used within a search heuristic for a best-first graph search (Bhardwaj, Choudhury, and Scherer 2017). The heuristic estimates the potential for a branch of the search tree to contain a low-cost robot which completes the task.

We compare our approach to two related methods which search for modular arrangements: a best-first search (Ha et al. 2018) and an evolutionary search (Icer et al. 2017). After training the DQN, our algorithm finds lower-cost solutions more efficiently than these related methods.

The rest of this paper is organized as follows: Section 2 discusses related work on robot design synthesis, work that motivated our approach, and a brief review of deep Q-learning. Section 3 describes our methodology for training the DQN and searching the space of modular arrangements. Section 4 presents our results and benchmarks them against existing approaches. Sections 5 and 6 discuss the limitations of our approach, future work, and concluding remarks.

2 Background

Our approach draws inspiration both from recent discrete modular design search work and from literature on the use of deep reinforcement learning for design and search.

2.1 Related Work

The most closely related methods for manipulator arrangement synthesis are best-first graph searches (Desai, Yuan, and Coros 2017; Desai et al. 2018; Ha et al. 2018). In these works, a heuristic was crafted that estimated the ability of each partially complete arrangement in fulfilling the task, and was used to guide a search over a tree of different arrangements. The evaluation of their heuristics involved solving an optimization subproblem, which becomes computationally burdensome as the number of possible module types and connections grows. Further, the heuristics did not consider obstacles, self-collisions, or torque constraints.

Another arrangement search method is a pruned exhaustive search. Althoff et al. (2019) performed increasingly computationally expensive checks on arrangements, eliminating candidates based on criteria such as total length or static torques. This method requires the evaluation criteria be manually specified for each task and module set, and could become computationally expensive *en masse* given an exponentially large search space.

Evolutionary algorithms have been used for design synthesis (Chen 1996; Leger 2012; Icer et al. 2017), searching the design space by randomly varying arrangements while selecting for those with high fitness. These methods allow the evaluation of many candidates in parallel, and work with discrete spaces. But, any domain-specific knowledge must be encoded by the user, the results of these algorithms vary substantially between runs, and they are computationally expensive because many candidate robots must be evaluated.

Deep RL has recently been used for robot design (Schaff et al. 2018; Ha 2018) to simultaneously learn a design and a control policy. Deep RL requires only a sparse reward function be formulated for each task. RL has also been used to design a deep neural network for image recognition (Baker et al. 2016). We similarly learn the value of each sequential discrete choice. These works use RL as the optimizer for a single task; that is, they fix the task and environment then use RL to search for a design. They suffer from the time it takes to optimize each design, thereby limiting their true potential, especially when rapidly prototyping designs or when the task may change frequently.

Our work is also inspired by recent work on learning-based motion planning. Chen, Murali, and Gupta (2018) learned a single control policy to control multiple robot designs, by training with a variety of manipulator designs on reaching and inserting a peg into a hole. As in our work, their policy was conditioned on both the workspace target and the robot design. Bhardwaj, Choudhury, and Scherer (2017) learned a search heuristic for a best-first search, used as a path planner in a grid world; we also learn a best-first search heuristic, but in the context of design rather than planning.

2.2 Deep Q-learning for Modular Robot Design

We formulate the robot design problem as a finite Markov Decision Process, in which we construct a robot by adding one module at a time. We define a *complete* arrangement as one that ends with an end-effector module, and a *partial* arrangement as one that does not. At each time step t, the agent selects an action a_t that adds a module to a partial robot. The state s_t contains the arrangement, so the next state depends deterministically on only the previous state and the module added. This results in a new robot, s_{t+1}, and a reward, r_t, from the environment. In this context the set of all robot modules defines the action space $\mathcal{A}$, while the set of partial and complete robots defines the state space, $\mathcal{S}$.

We define the return at time t, $R_t = \sum_{t'=t}^{T} \gamma^{t-t'} r_{t'}$, with a discount factor $0 \leq \gamma \leq 1$. The state-action value function $Q_\pi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is then defined as the expected return given the action a_t is taken in state s_t following policy $\pi : \mathcal{S} \mapsto \mathcal{A}$. Our approach uses reinforcement learning to estimate the optimal state-action value function Q^*, which can be defined in terms of the Bellman equation,

$$Q^*(s_t, a_t) = \max_\pi \mathbb{E}\left[r_t + \gamma \max_{a' \in \mathcal{A}} Q^*(s_{t+1}, a')\right]. \quad (1)$$

Tabular Q-learning (Watkins 1989), a temporal difference learning method (Sutton 1988), can be used to compute an estimate of the state-action value ("Q-value") corresponding to every possible state-action pair. This approach becomes intractable for large state and action spaces, so Deep Q-networks (DQN) use a deep neural network as a function approximator $Q(s, a; \theta)$ with network parameters θ to approximate $Q^*(s, a)$ (Mnih et al. 2015). We train this network with experience replay (Riedmiller 2005) and a target network (Van Hasselt, Guez, and Silver 2016).

Our method also uses additions to the original DQN framework. Universal value function approximators (UVFA) are learned value functions conditioned on the task goal (Schaul et al. 2015). We use a UVFA to enable our DQN to apply to a range of goals. Hindsight experience replay (HER) is a data augmentation technique employed for

RL problems with sparse reward signals (Andrychowicz et al. 2017). In HER, episodes are replayed with a different goal than the one used during the original episode.

3 Methods

In contrast to recent work (Ha 2018; Schaff et al. 2018) that used RL to solve an optimization problem to build a robot for each task, we use RL to learn a UVFA for a class of tasks (Schaul et al. 2015). Specifically we use a DQN as a UVFA to learn the expected state-action value of adding each module type to an arrangement given the goal of reaching a workspace target. The modules are chosen from a set of N_m types with indices $m \in 1, 2, ...N_m$. Each module could include any number of actuators and links, and may be able only to connect to some subset of other module types. The modular design synthesis problem is then to select a sequence of modules which form an arrangement A that can complete a given task.

In this work we limit the space of tasks to a set of N_T workspace targets which a serial manipulator should reach. A workspace target $T = [p, \hat{n}]$ consists of a position in space $p \in \mathbb{R}^3$ and tip axis orientation $\hat{n} \in \mathbb{R}^3, ||\hat{n}|| = 1$. This representation can include manipulation tasks including peg-in-hole-insertion, positioning a camera, or screw insertion.

Let $N_J(A)$ represent the number of actuated joints in a given arrangement A. The forward kinematics (FK) of A with joint angles $\vartheta \in \mathbb{R}^{N_J(A)}$

$$[p_{EE}, \hat{n}_{EE}] = \text{FK}(A, \vartheta), \tag{2}$$

outputs p_{EE}, the end-effector tip position, and $\hat{n}_{EE}$, the tip axis. To evaluate whether an arrangement can reach a target, we define the inverse kinematics (IK) of an arrangement as the joint angles that minimize the difference between the FK and a target,

$$\begin{aligned} \vartheta &= \text{IK}(A, p, \hat{n}) \\ &= \underset{\vartheta}{\text{argmin}}\, ||p - p_{EE}|| + (1 - \hat{n} \cdot \hat{n}_{EE}) \\ &\text{s.t.} \quad f(A, \vartheta) \leq 0 \end{aligned} \tag{3}$$

where f represents a set of constraints including self-collision avoidance, obstacle-collision avoidance, and joint limits. We use the interpenetration distance between colliding rigid bodies as the collision constraint metric. We solve IK numerically using gradient descent with multiple random initial seed restarts. In a slight abuse of notation, we will use $p_{EE}(A, p, \hat{n})$ and $\hat{n}_{EE}(A, p, \hat{n})$ to denote the forward kinematics output of the inverse kinematics solution for a given target. To evaluate whether an arrangement can reach a given target, we set tolerances ϵ_p and ϵ_n, and define a "reachability" function for the arrangement as

$$\text{reach}(A, T) = \begin{cases} 1 & ||p - p_{EE}(A, p, \hat{n})|| \leq \epsilon_p \quad \text{and} \\ & 1 - \hat{n} \cdot \hat{n}_{EE}(A, p, \hat{n}) \leq \epsilon_n \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Our goal is to find an arrangement of modules that is capable of reaching the targets. At the same time, we desire robots with fewer actuators (lower complexity) and lower mass. However, we must recognize that for arbitrary environments and module type sets, not every target may be reachable. Therefore, we pose this problem as a multi-objective optimization to maximize the number of targets reached while minimizing the complexity and mass of the robot, which gives us an objective function F,

$$F(A, T) = -w_J N_J(A) - w_M M(A) + \text{reach}(A, T) \tag{5}$$

where we use $M(A)$ to represent the total mass in arrangement A, and w_J and w_M are user-set weighting factor to trade off between the multiple objectives. We seek an arrangement that maximizes this function,

$$A^* = \underset{A}{\text{argmax}} \sum_{i=1}^{N_T} F(A, T_i). \tag{6}$$

Next we will learn a neural network which approximates the benefit of adding each module to an arrangement to maximize (5) for a single target. Section 3.3 will describe how this function is used to maximize over multiple targets.

3.1 DQN for module selection

Our algorithm assembles a serial-chain manipulator one module at a time, as illustrated in Figure 2. We use the output of a trained DQN to form a search heuristic. To use RL, we must first define the state, actions, and reward signals.

We encode the arrangement A as a list of one-hot vectors, where each index in a single vector indicates a type of module selected, with a user-set maximum number of modules allowed in the arrangement N_{max}. At each time step an action selects a module type m from the set of N_m module types. Each episode is a series of steps where one module is added until either the arrangement is complete (an end-effector is added) or the maximum number of modules in an arrangement has been reached.

We append a single workspace target $T = [p, \hat{n}]$ to the state. This conditions the Q-values on the target, forming a UVFA that can apply to a range of targets (Schaul et al. 2015). We also condition the learned Q-value function on the locations of obstacles in the environment. To make a tractable parameterization of environment obstacles, we voxelize the space into a coarse "grid" and assign a binary occupied/unoccupied value to each voxel, so $O \in \{0, 1\}^{(n_O \times n_O \times n_O)}$, where n_O is the number of voxels on each edge of the grid. The size of the voxels and the range of space over which they span were set by hand; we used $n_O = 5$ with voxel edge length 0.25 m; see Figure 3 for an illustration. The inputs to the DQN are the partial arrangement, the target, and the obstacle grid. Figure 4 depicts the structure of the neural network.

We use a reward signal such that the sum of rewards over an episode matches (5) because we aim to select an arrangement that maximizes that function in (6). The non-terminal rewards are penalties assigned for the mass and complexity of each module m added to the arrangement,

$$r(m) = -w_j N_J(m) - w_M M(m). \tag{7}$$

If the module added is an end-effector (EE), this is considered a terminal action, and the terminal reward is returned.

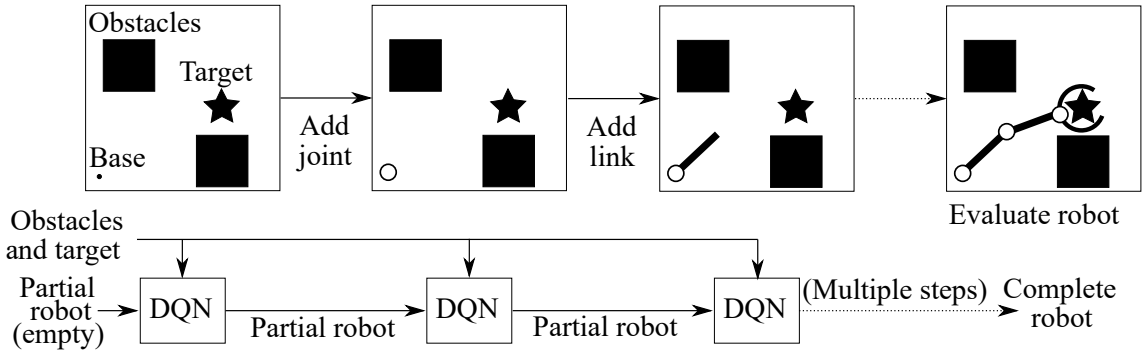

Figure 2: During training, the DQN is used repeatedly to evaluate the contribution each module type would have toward reaching a target. The arrangement is assembled sequentially (top) with modules selections made by the DQN (bottom)

The reachability function (4) is evaluated and added to the reward. If the maximum number of modules is reached without any end-effector added, a penalty of -1 is returned,

$$r_{\text{terminal}} = \begin{cases} -1 & \text{length}(A') == N_{max} \\ & \text{and } m \text{ is not an EE} \\ \text{reach}(A', T) & m \text{ is an EE,} \end{cases} \quad (8)$$

where we define A' as the arrangement resulting from the addition of m to the existing arrangement A. The elements of the Q-value vector $Q \in \mathbb{R}$ output by a forward pass of the DQN represent the expected value of a module type m that could be added to the tip of the arrangement A given a target T and grid O,

$$\begin{aligned} Q(A, T, O, m) &= \mathbb{E}\big[r + \max_{m'} Q(A', T, O, m')\big] \\ &\approx \text{DQN}_m(A, T, O), \end{aligned} \quad (9)$$

where DQN_m is the m^{th} component of the output of the DQN, as shown in Fig. 4.

3.2 Training the DQN

The DQN is trained to approximate the Q-values of each module type for a given arrangement, target, and grid. At the start of each episode during training we randomize the target and grid. Each element of p and $\hat{n}$ is selected from a $[-1, 1]$ range, and $\hat{n}$ is normalized. When we randomize the target and environment occupancy, we ensure that any points that must be occupied by the robot (e.g. the base and target) are unoccupied.

During training we build up an arrangement by sequentially selecting modules. At each step in the episode, the network outputs Q-values for each module type. In our module set, each type of module can connect to only a subset of the other module types. We mask out invalid module connection actions, and only learn Q-values for valid actions. An episode ends when an EE module is chosen or the maximum number of allowable modules has been added. The episodes have a maximum length, enabling us to use a discount factor $\gamma = 1$. We use a Boltzmann exploration strategy (Barto, Bradtke, and Singh 1995), as there are multiple similar module choices with similar values that should be explored, such that we avoid exploiting a single robot arrangement for all tasks. We use curriculum learning (Bengio et al. 2009) on the obstacle grid, mass penalty, and complexity penalty. We begin training with no obstacles or penalties, and periodically increase the maximum number of randomly selected obstacles and the penalty value during the early stages of training.

To learn from the sparse reward signal, we use HER (Andrychowicz et al. 2017). Each time a complete arrangement is found which does not reach the target, the episode is replayed with the point that was reached set as the target. We introduce additional data augmentation by randomly sampling joint angles and occupancy grid for the robot found, calculating FK, removing any samples that are in collision, and replaying the episode with the pose reached by each sample's FK set as the target. We found this results in higher quality solutions to our full graph search procedure by training the network to better predict the potential value of lower mass/complexity arrangements. While training, we periodically test the DQN on a small set of randomly generated test points. The performance of the graph search procedure on these test sets is used as an evalution metric to decide when to end training.

3.3 Using the DQN as a search heuristic

To search for task-specific arrangements, we use the DQN module value approximator to guide a best-first search. The forward pass of the DQN outputs the Q-value for each module type conditioned on a single target and grid. This Q-value encodes the expected future value of the objective function F defined in (5).

Different tasks may involve reaching different numbers of targets; as per (6), we seek to maximize return over multiple targets. But, for a single neural network to operate on multiple points at once, the value function would need to be conditioned on all permutations of those points, and would be constrained to a fixed maximum number of points. It would be significantly more computationally expensive to train if each arrangement selection were to be conditioned on a set of targets than if it were conditioned on one target. To address this challenge, we create a search heuristic from the output of one forward pass for each target.

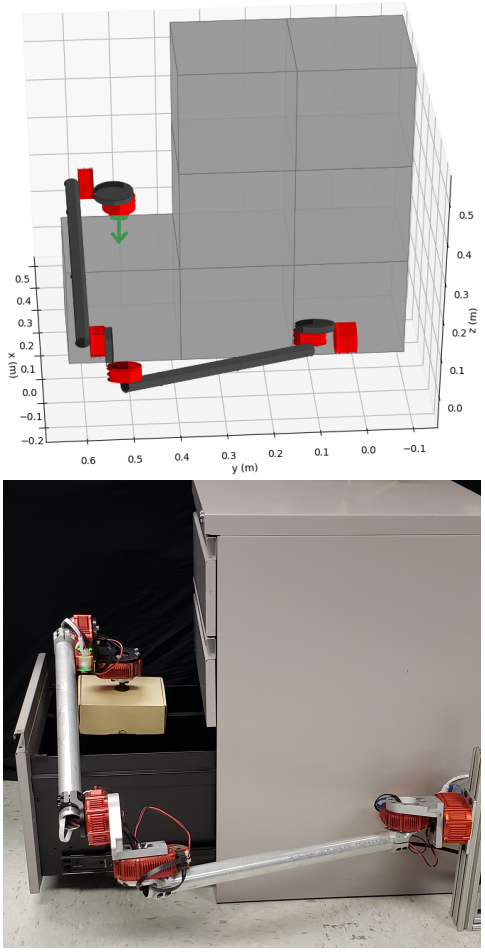

Figure 3: Top: An arrangement of modules (dark grey and red) with base located at the origin reaches a single workspace target position and tip axis (green point with arrow) without colliding with voxelized obstacles (grey cubes). Bottom: The physical modular robot matches the arrangement and environment.

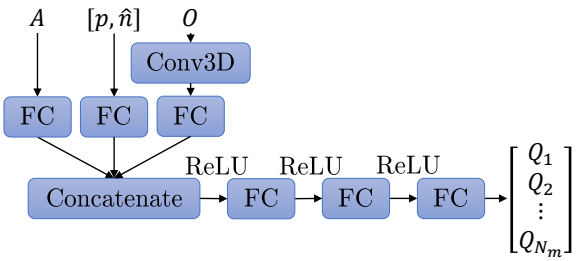

Figure 4: The neural network architecture we used for our DQN consists of fully connected (FC) layers with rectified linear unit (ReLU) activation, and a 3D convolution (Conv3D) over the grid of obstacles. The inputs to the DQN are the current arrangement A, target $T = [p, \hat{n}]$, and obstacle grid O. The outputs are the state-action values Q for each type of module.

Algorithm 1: Manipulator arrangement search, a best-first search guided by the output of a DQN.

Input: A set of N_T targets and an occupancy grid O
Result: Arrangement A
openset = [Empty arrangement]
while *time* $<$ *time limit* **do**
  Pop node with highest h value from openset;
  Expand the node;
  if *node contains complete robot* **then**
    Evaluate IK at all targets;
    if *all targets reached* **then**
      Store return for the arrangement
    end
  else
    Forward pass of DQN and sum output for each target as in (11);
    Add each child A' to the openset with value h
  end
end
Return arrangement with highest return (lowest cost)

First we observe that at terminal actions, the state-action value summed over all targets matches the desired maximization in (6). That is, for actions that result in terminal states (when the selected action m is an end-effector),

$$\sum_{i=1}^{N_T} Q(A, T_i, O, m) = \sum_{i=1}^{N_T} F(A', T_i). \tag{10}$$

Even though this equation is not exact for non-terminal actions, we find that the summation over Q-values is a good search heuristic to maximize objective F. Therefore we form the search heuristic $h \in \mathbb{R}$ from a summation of forward passes of the DQN for each target,

$$h(A, T_1 ... T_{N_T}, O, m) = \sum_{i=1}^{N_T} \text{DQN}_m(A, T_i, O). \tag{11}$$

This search heuristic prioritizes modules selected based on their potential to reach the targets with fewer additional modules.

Our DQN-best-first search algorithm is outlined in Algorithm 1. At each iteration, the arrangement with the highest heuristic value is popped from the open set. If it is a complete robot, it is evaluated. Otherwise it is expanded, passed through the DQN to create new h values for its children, and those children are added to the open set.

The Q-value is the expected return from the current arrangement onward. We penalize the addition of modules, so the DQN outputs from arrangements with more modules are usually higher than the outputs from arrangements with fewer modules. As a result, the search tends to act more like a depth-first search than a breadth-first search. A neural network forward pass is computationally inexpensive, so computation of this heuristic scales linearly with the number of targets, keeping computation for each node expansion low.

3.4 Comparisons to related work

We implemented two methods from prior work, a genetic and a best-first search, as bases of comparison. Here we describe these implementations and the experiments we ran.

Genetic algorithm Each individual A in the population was represented with a gene $g \in [0, 1)^{N_{max}}$. To convert each gene to an arrangement, each element was interpreted sequentially as the next valid module to attach. For example, if there are two possible children module types for the module at $j-1$, and element j of the gene is $0 \leq g_j < 0.5$, then the first of the two types would be selected, but if $0.5 \leq g_j < 1$ then the second of the two types would be selected. Each individual in the population was evaluated with a score combining their IK error, weighted complexity and mass, and whether they are complete. The population was resampled with elite selection, crossover, and mutation.

Best-first search algorithm We implemented the algorithm of Ha et al. (2018), in which the tree of possible designs is explored with a best-first search. At each step, partial robots are evaluated with a heuristic function based on an IK-like subproblem. The candidate with the lowest heuristic cost is expanded, and any complete robots are evaluated for the specified task. We removed velocity constraints from the IK and heuristic subproblem evaluations, which speeds up these functions which are evaluated many times.

Comparison tests We conducted a comparison test between the different methods: a genetic algorithm, best-first search, and our DQN-best-first search. We used modular components produced by Hebi Robotics (Hebi Robotics 2019) with a set of 11 types of modules: three base mount orientations, one actuated joint, six different links/brackets, and one end-effector. We limit the maximum number of modules in an arrangement to $N_{max} = 16$, a sufficient length for complete robots with a maximum of seven actuated joints given these modules. During training and all tests, we set the objective weights $w_J = 0.025$, $w_M = 0.1$. In the comparison tests, we generated 50 sets of 10 random targets, each set with a randomized obstacle grid with up to 10 obstacles. For each method, we measured:

- the time until the first feasible arrangement (one which reaches all targets) was found for each set,
- the standard deviation of the time until the first feasible robot was found was found for each set,
- the penalty $w_J N_J(A) + w_M M(A)$ from the complexity and mass of the first feasible robot,
- the number of complete arrangements evaluated before a feasible robot was found,
- the feasible arrangement with the lowest cost found after five minutes, and
- the number of target sets for which no feasible arrangement was found after five minutes.

When no feasible arrangement was found for a given method and set within the time limit, that set was not included in the averages or times for that method. We selected these criteria because we are interested in rapid prototyping and field applications, where we may need to trade off between speed and solution quality. As such both the first arrangement found (fastest solution) and the solution found after a fixed amount of time are relevant. The IK evaluation of complete robots is the most computationally expensive step. We trained the DQN and conducted all tests on a desktop computer with Ubuntu 16.04, Intel i5 four-core processor at 3.5 GHz, and an NVIDIA GTX 1050 graphics card. We trained the DQN for 450,000 episodes (about 33 hours) before using it within our algorithm.

Searching with torque constraints In addition to the DQN network above, we trained a network for a more difficult variant of the problem, with more module types and a constraint on the actuator torque limits. We added five more module types (four links and one rotary actuator), for 16 total module types. One actuator module type had lower mass and lower maximum torque, and the other had higher mass and higher maximum torque. When evaluating the reachability function, if any actuator torque limit was exceeded, then a terminal reward of 0 was returned. As a basis of comparison, we modified the genetic algorithm to include a penalty on arrangements that overload the actuator torques. We were unable to compare this extension to the method of (Ha et al. 2018) as their method does not consider torques. The test set used in this test was the same as those described above. We trained this DQN for 700,000 episodes (about 57 hours).

4 Results

The results of the comparison tests are shown in Table 1. We found that our method produces the best results in all categories. For one of the tests, none of the three algorithms were able to find a feasible robot within five minutes.

The genetic algorithm finds costly feasible arrangements in few iterations by randomly sampling arrangements, and then refines those results over further iterations to less costly arrangements. Qualitatively we found it tends to do well when there are many feasible robots for the task, for example when there are few targets and few obstacles, because the initial sampling may include costly arrangements that complete the task. However, the genetic algorithm requires many complete robot planning evaluations. If the computational cost of evaluating planning for complete robots were to increase, we expect this method to correspondingly become more expensive.

The best-first search does not include obstacles in its search heuristic, so its performance tends to degrade in the presence of many obstacles. It evaluates robots in order of increasing complexity, but must solve an nonlinear program to evaluate each node. Due to this computationally expensive subproblem, this algorithm was not able to find solutions for a third of the test cases within the five minute time limit. In the cases where it did find a solution, it was not usually able to improve upon that solution within the remaining time.

We observed that our method acts depth-first initially, evaluating a complete robot after only a few DQN forward passes. The reward structure during training guides the search toward less costly arrangements. In contrast to the heuristic of (Ha et al. 2018), our heuristic considers obstacle

Table 1: Results of the comparison tests described in Section 3.4 (lower values are better for all metrics).

	11 modules			16 modules, torque constraint	
Method	**DQN**	Best-first	Genetic	**DQN**	Genetic
Avg. runtime to first (min.)	**0.26**	3.04	0.64	**0.20**	3.08
Std. dev. runtime to first (min.)	**0.14**	1.00	0.69	**0.38**	1.69
Avg. num. complete robot evaluations to find first	**10.31**	29.03	138.12	**39.76**	311.41
Avg. cost for first found	**0.57**	0.59	0.62	**0.63**	0.64
Avg. best cost after five min.	**0.52**	0.58	0.53	**0.60**	0.63
Num. trials none found after five min.	**1/50**	16/50	1/50	**1/50**	28/50

locations. We found this improves average solution quality and run time over an ablated variant that did not condition the heuristic on obstacles.

In the variant with a torque constraint and additional module types, our method still searched the space of arrangements efficiently, and output feasible designs quickly, albeit after a longer training time. The higher-mass actuator module was frequently needed to create arrangements capable of extending to the farthest targets without exceeding the maximum torques, resulting in solutions with higher cost than in the previous experiments. Even with the larger set of modules and additional constraint, a feasible design was still consistently returned within one minute. In contrast, the genetic algorithm was unable to find a feasible arrangement within five minutes in the majority of the test cases.

In the most directly related work (Ha et al. 2018) the search suffers from the curse of dimensionality at runtime. When the branching factor (from number of types of modules available) increases, the number of heuristic function evaluations increases exponentially. In contrast, when more modules are added, we must train the DQN for additional time, but still use DQN forward passes to assign a heuristic to all children of the expanded node at once. Where our method is strongest, compared to related methods, is the low computation needed before finding a feasible arrangement, arising both from the computational efficiency with which the search heuristic is computed (forward passes from the DQN) and in the lower number of complete robot evaluations. As the task becomes more complex, we expect that the number of complete robot motion planning evaluations will dominate the search time, resulting in decreased performance of related methods, but only increasing training time for our method.

We have included the code to train the network, pre-trained network weights, and the code and results for our experiments in the supplementary material.

5 Limitations and future work

One limitation of our work is the need to retrain the neural network if the set of module types changes; future work will consider using a trained network to warm-start training with small differences in module set. Another limitation is that our formulation does not include costs on velocity/motion smoothness. In future work we will to move toward dynamic motion plans rather than quasi-static IK. Further, rather than rely on conventional motion planning algorithms for evaluation of each arrangement at the task, future work will involve learning control policies conditioned on the robot design, task, and environment (Chen, Murali, and Gupta 2018) end-to-end with the module selection policy.

The module arrangement input representation in this work is a list of one-hot vectors, each vector representing a module in the sequence, and padded with zeros up to the maximum number of allowed modules in the arrangement. A limitation of this encoding, which we will address in future work, is that it limits the arrangement to serial topologies. Similarly, we restricted the design to be composed of discrete selection of components. A more general, but more complex, case of robot designs composed of both continuous design parameters and discrete components is an area of ongoing research (Whitman and Choset 2018).

6 Conclusions

In this paper we presented an algorithm that uses a data-driven graph search heuristic to synthesize task-specific modular robot designs. We showed that our method returned lower-cost solutions more computationally efficiently than similar state-of-the-art methods. In the arrangement search, the "curse of dimensionality" appears from the high branching factor in the series of discrete module selection choices. Search efficiency is needed to mitigate the computational burden of creating a motion plan for each candidate arrangement. Our method addresses these challenge by using a deep neural network forward pass to approximate the value of all options at once, moving the vast majority of the computation into off-line training. Although we limit our focus to serial manipulators, a similar method could be applied to more complex body designs. We envision our method could be used in applications where there is a finite mass or monetary budget for parts, and a need to apply the same modules to applications that change frequently, but where the set of module types remains fixed, for instance, in space, low-volume manufacturing, or military applications. As modular robots become less expensive to produce, we hope that automated design tools like ours will allow non-experts to easily create customized robots.

Acknowledgments

This work was supported by NASA Space Technology Research Fellowship NNX16AM81H.

References

Althoff, M.; Giusti, A.; Liu, S.; and Pereira, A. 2019. Effortless creation of safe robots from modules through self-programming and self-verification. *Science Robotics* 4(31):eaaw1924.

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, O. P.; and Zaremba, W. 2017. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, 5048–5058.

Baker, B.; Gupta, O.; Naik, N.; and Raskar, R. 2016. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*.

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial intelligence* 72(1-2):81–138.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 41–48. ACM.

Bhardwaj, M.; Choudhury, S.; and Scherer, S. 2017. Learning heuristic search via imitation. In *Conference on Robot Learning*, 271–280.

Chen, T.; Murali, A.; and Gupta, A. 2018. Hardware conditioned policies for multi-robot transfer learning. In *Advances in Neural Information Processing Systems*, 9333–9344.

Chen, I. M. 1996. On optimal configuration of modular reconfigurable robots. In *Proceedings of the 4th International Conference on Control, Automation, Robotics, and Vision*.

Desai, R.; Safonova, M.; Muelling, K.; and Coros, S. 2018. Automatic design of task-specific robotic arms. *arXiv preprint arXiv:1806.07419*.

Desai, R.; Yuan, Y.; and Coros, S. 2017. Computational abstractions for interactive design of robotic devices. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 1196–1203. IEEE.

Ha, S.; Coros, S.; Alspach, A.; Bern, J. M.; Kim, J.; and Yamane, K. 2018. Computational design of robotic devices from high-level motion specifications. *IEEE Transactions on Robotics* 34(5):1240–1251.

Ha, D. 2018. Reinforcement learning for improving agent design. *arXiv preprint arXiv:1810.03779*.

Hebi Robotics. 2019. *[Online]*. www.hebirobotics.com. Accessed Aug. 8, 2019.

Icer, E.; Hassan, H. A.; El-Ayat, K.; and Althoff, M. 2017. Evolutionary cost-optimal composition synthesis of modular robots considering a given task. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3562–3568. IEEE.

Leger, C. 2012. *Darwin2K: An evolutionary approach to automated design for robotics*, volume 574. Springer Science & Business Media.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Riedmiller, M. 2005. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, 317–328. Springer.

Schaff, C.; Yunis, D.; Chakrabarti, A.; and Walter, M. R. 2018. Jointly learning to construct and control agents using deep reinforcement learning. *arXiv preprint arXiv:1801.01432*.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal value function approximators. In *Proceedings of the 1st Annual Conference on Robot Learning*, 1312–1320.

Sutton, R. S. 1988. Learning to predict by the methods of temporal differences. *Machine learning* 3(1):9–44.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.

Watkins, C. J. C. H. 1989. *Learning from delayed rewards*. Ph.D. Dissertation, King's College, Cambridge.

Whitman, J., and Choset, H. 2018. Task-specific manipulator design and trajectory synthesis. *IEEE Robotics and Automation Letters* 4(2):301–308.