基于深度强化学习的模块化机器人设计综合

Julian Whitman, Raunaq Bhirangi, Matthew Travers, Howie Choset

Department of Mechanical Engineering, Carnegie Mellon University

The Robotics Institute, Carnegie Mellon University

5000 Forbes Ave., Pittsburgh, Pennsylvania 15213

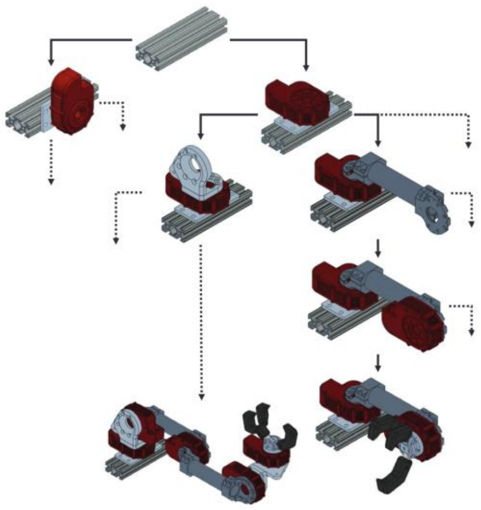
jwhitman@cmu.edu

摘要

模块化机器人具有多功能性，因为它们可以将组件重构，使得机器人能适应部署的任务。即使对于最简单的构型设计，由于模块连接方式的排列数量，确定最佳设计也是指数级复杂的。此外，对给定任务进行设计时，评估每个机器人的性能需要额外的计算量。例如，它是否能够到达工作空间中的某些点。这项研究使用深度强化学习来引导启发式搜索，能够高效地搜索模块化串联机器人的构型空间。我们的研究表明，与当前最先进的算法相比，我们的算法在为给定任务设计机器人时，计算更高效。

1 引言

模块化机器人有着创建定制机器人的潜力，此类机器人可以轻松部署以执行各种任务。为给定任务综合出一种模块化机器人涉及一系列挑战，其一是可能的模块化构型（离散机器人模块的有序序列）空间随着模块类型的数量及其连接方式呈指数增长。



图表 1：我们的方法在构型空间中搜索机械臂的设计，其中模块序列化地添加到机器人末端。在构型树的根节点放置基座。实线代表添加模块，虚线表示构型树继续扩展但未呈现。我们使用深度强化学习建立一种数据驱动的启发式搜索算法，以引导构型树搜索。我们将算法应用到Hebi 机器人（Hebi Robotics 2019）的模块化组件上。

在搜索这个指数级大的空间时，我们必须评估每个候选机器人是否能够完成任务。该评估需要规划和比较每个候选设计的相对运动成本，这样的计算规模过于庞大。我们通过一种启发式学习搜索算法来解决这一难题，该算法在评价指标下隐式编码机器人的性能。这项研究的主要贡献是一种算法，该算法使用深度强化学习来引导启发式搜索，使我们能够在每个机器人的固有能力范围内有效地搜索构型空间。我们将问题的范围限制为综合模块化串联机械臂的构型，针对机械臂必须达到一组准静态工作空间位置和姿态的任务。

我们基于先前的模块化设计综合方法（Desai，Yuan 和 Coros 2017；Ha 等人 2018），该方法逐步构建和搜索不同模块的构型树。具体来说，每个添加到当前叶节点的子节点都表示将模块添加到机械臂的远端，如图 1 所示。我们将构型树的建立视为一系列状态（构型）和动作（添加模块到远端），并将构型的装配视为马尔可夫决策过程（Sutton 1988）。在这个公式下，我们的算法学习一个状态-动作值函数，它近似出将每个模块类型添加到给定任务的构型中的价值。我们训练 Deep Q Network（DQN），使用强化学习来近似价值函数（Mnih 等人，2015）。DQN 用于启发式最佳优先图搜索（Bhardwaj、Choudhury 和 Scherer 2017）。对于搜索树的一个分支，启发式算法评估其包含能完成任务的低成本机器人的潜力。

我们将我们的方法与搜索模块化构型的两种相关方法进行对比：最佳优先搜索（Ha et al. 2018）和进化搜索（Icer et al. 2017）。在训练 DQN 之后，我们的算法比此类算法更有效地找到成本更低的解决方案。

本文的其余部分组织如下：第 2 节讨论了机器人设计综合的相关研究，这是我们研究的动机，以及 Deep Q Learning 的简要回顾。第 3 节描述了我们训练 DQN 和搜索模块化构型空间的方法。第 4 节介绍了我们的成果，并将其与现有方法进行比较。第 5 节和第 6 节讨论了我们方法的局限性、未来研究和总结意见。

2 背景

我们的方法既从最近的离散模块化设计搜索研究中获得灵感，也从使用深度强化学习进行设计和搜索的文献中获得启发。

2.1 文献综述

与机械臂构型综合最密切相关的方法是最佳优先图搜索（Desai，Yuan 和 Coros 2017；Desai 等人 2018；Ha 等人 2018）。在他们的研究中，设计了一种启发式算法，用于评估每个局部构型完成任务的能力，并在不同的构型树上引导搜索。其启发式方法的评估部分涉及求解一个优化子问题，随着模块类型和连接数量的增加，优化子问题在计算上开销变大。此外，这些启发式算法没有考虑障碍物、自碰撞或扭矩约束。

另一种构型搜索方法是删减穷举搜索（pruned exhaustive search）。Althoff 等人（2019）对构型进行了消耗大量计算资源的检查，根据总长度或静态扭矩等标准消除了候选构型。该方法要求为每个任务和模块集手动指定评估标准，并且在指数级大的搜索空间下，整体计算开销会很大。

进化算法已用于设计综合（Chen 1996；Leger 2012；Icer 等人 2017），在设计空间中通过随机改变构型进行搜索，同时选择具有高适应度的设计空间。这些方法允许并行评估多个候选对象，也使用离散空间。但是，任何特定领域的知识都必须由用户编码，多次运行这些算法获得的结果之间有很大的差异，而且它们的计算成本很高，因为必须评估大量候选机器人。

Deep RL 最近被用于机器人设计（Schaff 等人，2018；Ha 2018），以同时学习设计和控制策略。Deep RL 只需要为每个任务提供一个稀疏奖励函数。RL 还用于设计基于图像识别的深度神经网络（Baker 等人，2016）。同样的，我们学习每个连续离散选项的价值。这些工使用 RL 作为单个任务的优化器，即他们固定了任务和环境，然后使用 RL 搜索一种设计方案。受限于优化每个设计所需的时间，从而约束了他们的真正发挥空间，尤其是对于快速原型设计或任务可能频繁更改的情形。

我们的研究也受到最近基于学习的运动规划研究的启发。Chen、Murali 和 Gupta（2018）学习了一种单一的控制策略来控制多个机器人的设计，针对各种机械手的设计来训练如何将桩插入孔中。与我们的研究一样，他们的策略是基于工作空间目标和机器人设计。Bhardwaj、Choudhury 和 Scherer（2017）学习了一种最佳优先搜索的启发函数，在网格世界中用作路径规划器。我们也学习了一种最佳优先搜索的启发函数，但在设计而不是规划的背景下。

2. 2 基于 Deep Q-learning 的模块化机器人设计

我们将机器人设计问题表述为有限马尔可夫决策过程，其中我们通过一次添加一个模块来构造机器人。我们将完整构型定义为以末端执行器模块结束的构型，将局部构型定义为不以末端执行模块结束的构型。在每个时间步（time step）t，agent 选择的动作是将模块添加到机器人远端。状态s_t包含构型，因此下一个状态仅取决于上一个状态和添加的模块。这将产生一个新的机器人s_{t+1}，和一个来自环境的奖励r_t。在这种情况下，所有机器人模块的集合定义了动作空间 A，而局部和完整机器人的集合定义状态空间 S。

我们定义了在t时刻的返回值，$R_t = \sum_{t'=t}^{T} \gamma^{t-t'} r_{t'}$，折扣系数$0 \le \gamma \le 1$。状态-动作值函数$Q_\pi$: $S \times A \to R$，定义为在状态s_t下采取动作a_t的期望 return，该动作服从策略分布π: $S \to A$。我们的方法使用强化学习来估计最优状态-动作值函数$Q^\star$，其能够用贝尔曼方程给出定义。

$$Q^\star(S_t, a_t) = max_\pi E[r_t + \gamma max_{a' \in A} Q^\star(s_{t+1}, a')] \quad (1)$$

Tabular Q-learning（Watkins 1989）是一种时间差分（temporal difference）学习方法（Sutton 1988），可用于计算对应于每个可能状态-动作对的状态-动作值（Q-值）的估计值。这种方法难以处理很大的状态-动作空间，因此深度 Q-网络（deep Q network，DQN）使用深度神经网络作为函数逼近器$Q(s,a;\theta)$，其网络参数为θ，以逼近$Q^{\star}(s,a)$（Mnih 等人，2015）。我们通过经验回放（Riedmiller 2005）和目标网络（Van Hasselt、Guez 和 Silver 2016）来训练我们的网络。

我们的方法还对原始DQN框架进行了扩展。泛值函数逼近器（Universal value function approximators，UVFA）是以任务目标为条件来学习值函数（Schaul 等人，2015）。我们使用 UVFA 使 DQN 适用于一系列目标。后视经验回放（Hindsight experience replay，HER）是一种数据增强技术，用于具有稀疏奖励信号的 RL 问题（Andrychowicz 等人，2017）。在 HER 中，以不同于原 episode 中使用的目标回放 episode。

3 方法

与最近的研究（Ha 2018；Schaff 等人 2018）不同，我们使用 RL 来学习一类任务的 UVFA（Schaul 等人 2015）。特别地，我们使用 DQN 作为 UVFA 来学习在给定工作空间目标的情况下，将每个模块类型添加到机器人构型中的状态-动作估计值。模块是从一组 N_m中选取的，其中索引$m \in 1,2,\ldots N_m$。每个模块可以包括任意数量的驱动器和连杆，并且可能只能连接到模块类型的某个子集。模块化机器人设计综合问题是选择一系列模块，由这些模块组成的构型 A 能够完成给定任务。

在这项研究中，我们将任务空间限制为一组N_T个工作空间目标，串联机械臂应该达到这些目标。工作空间目标（workspace target）$T = [p, \hat{n}]$由空间中的位置坐标$p \in R^3$和末端轴方向 $n \in R^3, ||n|| = 1$组成。能够表示的操纵任务，包括插钉（peg-in-hole-insertion）、定位相机（positioning a camera）或拧螺钉（screw insertion）。

设$N_J(A)$表示给定构型 A 中驱动关节的数量。构型 A 关节角为 $\theta \in R^{N_J(A)}$的正运动学（FK）

$$[p_{EE}, \hat{n}_{EE}] = FK(A, \theta), \quad (2)$$

输出为P_{EE}，末端执行器末端位置和$\hat{n}_{EE}$，末端轴。为了评估构型是否可以到达目标，我们将构型的逆运动学（IK）定义为使 FK 输出和目标之间的差异最小化的关节角度，

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

其中 f 表示一组约束，包括自碰撞规避（self-collision avoidance）、障碍物碰撞规避（obstacle-collision avoidance）和关节限位（joint limits）。我们使用碰撞刚体之间的互穿距离作为碰撞约束度量。我们使用具有多个随机初始种子的重启梯度下降来求 IK 地数值解。在稍微滥用表示法的情况下，我们将使用$p_{EE}(A,p,\hat{n})$和$\hat{n}_{EE}(A,p,\hat{n})$表示给定目标的逆运动学解的正运动学输出。为了评估构型是否能够达到给定目标，我们设置了误差ϵ_p和ϵ_n，并定义构型的“可达性”函数为

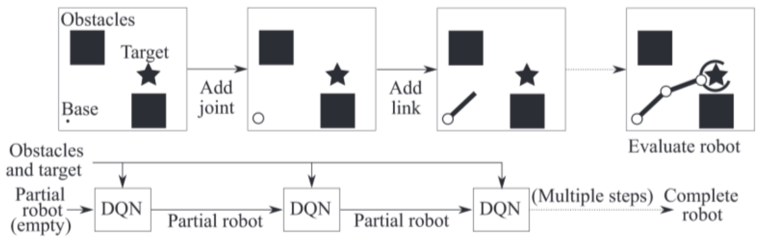
$$reach(A,T)=\begin{cases}1 & ||p-p_{EE}(A,p,\hat{n})||\le\epsilon_p\ and \\ & 1-\hat{n}\cdot\hat{n}_{EE}(A,p,\hat{n})\le\epsilon_n \\ 0 & otherwise.\end{cases} \qquad (4)$$

我们的目标是找到能够达到目标的机器人构型。同时，我们希望机器人具有更少的驱动器（更低的复杂性）和更轻的质量。然而，对于任意环境和模块类型集，并非每个目标都可以到达。因此，我们将此问题作为一个多目标优化，以最大化达到的目标数量，同时最小化机器人的复杂性和质量，由此我们提供了一个目标函数 F，

$$F(A,T)=-\omega_J N_J(A)-\omega_M M(A)+reach(A,T) \qquad (5)$$

其中，我们使用 M(A)表示构型 A 的总质量，ω_J和ω_M是用户设置的权重因子，以在多个目标之间进行权衡。我们寻求一种使$A^{\star}$最大化的构型，

$$A^{\star}=argmax_A\sum_{i=1}^{N_T}F(A,T_i) \qquad (6)$$



图表 2：在训练过程中，重复使用 DQN，以评估在达到目标点时每个模块类型的贡献值。随着 DQN 做出模块选择（图下半部分），构型以序列化的方式进行组装（图上半部分）。

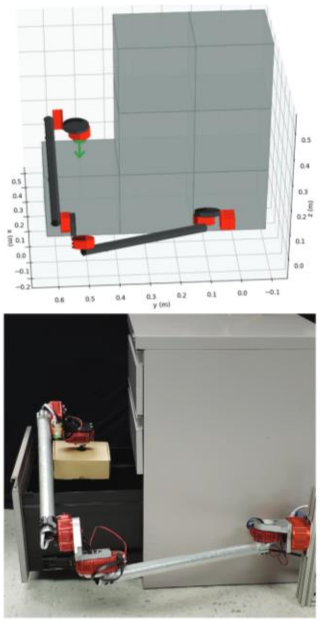
下面，我们将训练一个神经网络，它近似出将每个模块添加到一个构型上的价值，以在单个目标中最大化式（5）。在 3.3 节将描述如何使用该函数在多个目标上实现最大化。

3.1 基于 DQN 的模块选择

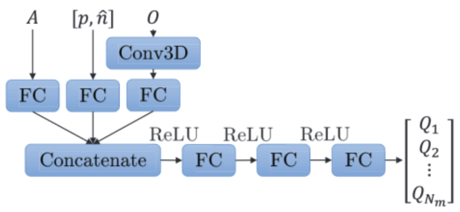
我们的算法一次组装一个串联机械臂模块，如图 2 所示。我们使用训练好的 DQN 的输出引导启发式搜索。在使用 RL 之前，我们必须首先定义状态、动作和奖励信号。

我们将构型 A 编码为独热向量列表，其中单个向量中的每个索引指示所选模块的类型，用户设置构型中允许的最大模块数为N_{max}。在每个时间步中，动作从N_m种模块类型集合中选择模块类型 m。每个 episode 都包含一系列 steps，其中添加一个模块，直到构型完成（安装了末端执行器）或达到构型中模块数量的最大限制。

我们将单个工作空间目标$T = [p, \hat{n}]$添加到状态中。它会影响目标的 Q 值，形成可适用于一系列目标的 UVFA（Schaul 等人，2015）。环境中障碍物的位置也会影响学习到的 Q 值。为了对环境障碍物进行易于处理的参数化，我们将空间体素化（voxelize）为一个“粗网格”，并为每个体素（voxel）分配一个二进制（占用/未占用）值，因此$O \in 0,1^{(n_O \times n_O \times n_O)}$，其中$n_O$是网格每条边上的体素个数。体素的大小和它们占据的范围通过手动设置；我们使用$n_O = 5$，体素边缘长度为 0. 25m；如图 3 所示。DQN 的输入是一部分构型、目标和障碍物网格。图 4 描述了神经网络的结构。



图表 3：上：模块组成的构型（深灰色和红色），其基座位置在原点。构型达到单个工作空间目标点，且末端轴（绿色箭头）没有与体素化的障碍发生碰撞（灰色方块）。下：符合构型以及环境的实体模块化机器人。



图表 4：应用于 DQN 的神经网络结构由以 ReLU 为激活函数的全连接层和对障碍物网格的 3D 卷积层组成。DQN 的输入是当前的构型 A，目标点$T = [p, \hat{n}]$和障碍网格 O。输出是每种模块的状态-行为值 Q。

我们使用奖励信号，使得一个 episode 的奖励总和与式（5）对应，因为我们的目标是选择一种使式（6）中的函数最大化的构型。非终止奖励是针对添加到构型中的每个模块 m 的质量和复杂性而产生的惩罚值，

$$r(m) = -\omega_J N_J(m) - \omega_M M(m) \quad (7)$$

如果添加的模块是末端执行器（end-effector，EE），则这被视为终止动作，并返回终止奖励。

评估“可达性”函数式（4）并将其添加到奖励中。如果达到模块最大数量限制且没有添加任何末端
执行器，则返回-1 的惩罚值，

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

其中我们将 A′ 定义为将 m 添加到现有构型 A 所产生的新构型。通过 DQN 的正向传播输出的 Q 值向量 Q∈R 的表示将模块类型 m 添加到上一个构型 A 以及给定目标 T 和障碍物 O 能获得的期望价值，

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

其中DQN_m是 DQN 正向传播的第 m 个分量，如图 4 所示。

3.2 训练 DQN

DQN 被训练为近似给定构型、目标和障碍物网格时添加的每种模块类型的 Q 值。在训练中每个 episode 的开始，我们随机目标和网格。从，[-1,1]范围中选择 p 和$\hat{n}$的每个元素，$\hat{n}$需要归一化。当我们随机化目标和环境障碍时，我们确保机器人可能/必须占用的任何点（例如，基座和目标）都未被占用。

在训练期间，我们通过顺序选择模块来创建构型。在 episode 的每个 step 中，网络输出每个模块类型的 Q 值。在我们的模块集中，每种类型的模块只能连接到其他模块类型的子集。我们去除了无效的模块连接动作，只学习有效动作的 Q

值。当选择了 EE 模块或添加到许用模块数上限时，一个 episode 结束。由于 episode 有长度上限，使我们能够使用折扣系数 γ=1。我们使用 Boltzmann 探索策略（Barto、Bradtke 和 Singh 1995），因为可供选择的有多个具有相近值的类似模块需要探索，这样，我们避免为所有任务使用单一的机器人构型
。我们在添加障碍网格、质量和复杂性惩罚时使用课程学习（Bengio 等人，2009）。我们开始训练时没有障碍或惩罚，并在训练的早期阶段定期增加随机选择的障碍的最大数量和惩罚值。

为了从稀疏的奖励信号中学习，我们使用 HER（Andrychowicz 等人，2017）。每次发现未达到目标的完成构型时，将此次达到的点为目标，回放（replay）该 episode。我们通过对搜索出的机器人关节角和障碍物网格进行随机采样，计算 FK，移除任何产生碰撞的样本，并以每个样本的 FK 的输出作为目标来回放该 episode，从而引入了额外的数据扩充。我们通过训练网络以更好地预测较低质量/复杂度构型的潜力值，我们发现这一结果使得我们的全图搜索（full graph search）过程更高质量的求解。在训练期间，我们定期在一组随机生成的测试点上测试 DQN。图搜索过程在这些测试集上的性能被用作评估指标，以决定何时结束训练。

3. 3 使用 DQN 引导启发式搜索

为了搜索出指定任务的构型，我们使用 DQN 模块值逼近器来引导最佳优先搜索（best-first search）。DQN 的正向传播输出在单个目标和障碍网格上调节的每个模块类型的 Q 值。该 Q 值表示式（5）中定义的目标函数 F 的期望未来价值。

不同的任务可能涉及达到不同数量的目标；根据式（6），我们寻求在多个目标上实现回报最大化。但是，对于一个神经网络同时对多个点进行操作，值函数将需要以这些点的所有构型为条件，并且会有最大目标点数量的限制。如果每个构型选择都以一组目标为条件，那么训练的计算成本将显著高于以一个目标为条件。为了解决这一挑战，我们从每个目标的一次前向传播的输出中引导启发式搜索。

首先，我们观察到，在终止动作时，所有目标的状态-动作值总和与式（6）中的期望构型的最大值相等。即对于产生终止状态的动作（当所选择的动作 m 是末端执行器时），

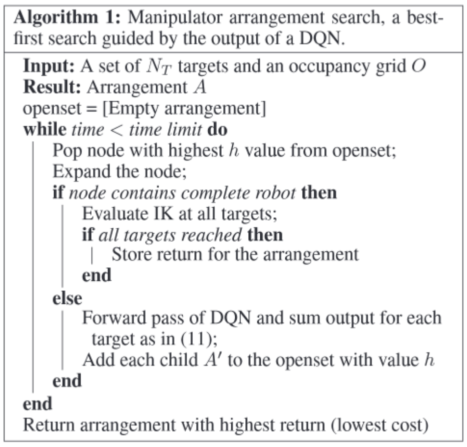
$$\sum_{i=1}^{N_T} Q(A, T_i, O, m) = \sum_{i=1}^{N_T} F(A', T_i) \qquad (10)$$

尽管该方程对于非终止动作并不精确，但我们发现对 Q 值的求和是使目标 F 最大化的一种很好的启发式搜索算法。因此，我们将每个目标的 DQN 的前向传播求和，给出启发式搜索函数$h \in R$，

$$h(A,T_1\dots T_{N_T},O,m)=\sum_{i=1}^{N_T}DQN_m(A,T_i,O) \qquad (11)$$

此启发式搜索算法基于机器人用更少的附加模块达到目标的潜力。

算法 1：机械臂构型搜索，由 DQN 输出引导的最佳优先搜索算法



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)
```

我们的 DQN 最佳优先搜索（DQN-best-first　search）算法在算法 1 中概述。在每次迭代中，把具有最高 h 值的构型从 open　set 中提出（popped）。如果它是一个完整的机器人，则进行评估。否则，它将继续拓展，通过 DQN 为其子集创建新的 h 值，这些子集也被添加到 open　set 中。

Q 值是当前构型的期望 return。我们惩罚额外的模块，因此具有更多模块的构型的 DQN 输出低于具有更少模块的构型输出。从结果来看，搜索倾向更像深度优先搜索而不是广度优先搜索。神经网络前向传播在计算上开销较小，因此这种启发式搜索算法的计算与目标的数量成线性关系，从而保持每个节点扩展的计算量较低。

3. 4 有关研究方法对比

我们从先前的研究中实现了两种方法，一种是遗传搜索，另一种是最佳优先搜索，作为比较的基准。下面，我们将描述这些实现和我们进行的实验。

遗传算法（Genetic　algorithm）

每个 A 用基因$g \in [0,1)^{N_{max}}$表示。为了将每个基因转换为一个构型，每个元素都有序地作为下一个要添加的有效模块。例如，如果在 j-1 处模块有两种可能的子模块类型，并且基因的元素 j 为 $0 \le g_j < 0.5$，则将选择这两个类型中的第一个，但如果$0.5 \le g_j < 1$，则选择这两种类型中的第二个。通过综合他们的 IK 误差、复杂度和质量的加权值以及构型是否完整的分数来评估种群中的每个个体。通过精英选择、交叉和突变对种群进行重新采样。

最佳优先搜索算法（Best-first search algorithm）

我们实现了 Ha 等人（2018）的算法，其中使用最佳优先搜索来探索可能的设计树。在每一步中，基于类似 IK 的子问题，使用启发式函数对机器人进行评估。扩展具有最低启发式成本的候选构型，并针对指定任务评估任何完整的机器人。我们从 IK 和启发式子问题评估中移除速度约束，这可以在多次评估种加快这些函数的运行速度。

对比测试（Comparison tests）

我们对不同的方法进行了比较测试：遗传算法、最佳优先搜索和 DQN 最佳优先搜索。我们使用了 Hebi 机器人公司（Hebi Robotics 2019）生产的模块化组件，包括一组 11 种类型的模块：三个底座安装方向、一个驱动关节、六个不同的连杆/支架和一个末端执行器。我们将构型中的模块的最大数量限制为$N_{max} = 16$，使用给定的这些模块构造最多七个驱动关节的完整机器人，数量是足够的。在训练和所有测试中，我们设定了目标权重$w_J = 0.025, w_M = 0.1$。在比较测试中，我们生成了 50 组 10 个随机目标，每组带有最多 10 个随机障碍物网格。对于每种方法，我们测量：

•每组找到第一个可行构型（达到所有目标的构型）的时间，

•每组找到了第一个可行机器人时间的标准差

•第一个可行的机器人的复杂性和质量的惩罚$w_J N_J(A) + w_M M(A)$

•找到可行机器人之前评估的完整构型数量

•五分钟后找到的成本最低的可行构型，

•以及五分钟之后没有找到可行构型的目标集数量。

如果没有在时限内为给定方法找到可行的构型，则此次搜索不包含在该算法的平均值或时间中。我们选择这样的标准是因为我们对快速原型和现场应用感兴趣，其中我们可能需要在速度和解决方案的质量之间进行权衡。因此，找到的第一个构型（最快的解决方案）和固定时间后找到的解决方案都是有价值的。完整机器人的 IK 评估是计算成本最高的步骤。我们对 DQN 进行训练，并在装有 Ubuntu 16.04、3.5GHz Intel i5 四核处理器和 NVIDIA GTX 1050 图形卡的台式计算机上进行了所有测试。在将 DQN 用于我们的算法之前，我们训练了 450000 个 episodes（约 33 小时）。

使用扭矩约束进行搜索（Searching with torque constraints）

除了上述 DQN 网络之外，我们还训练了一个网络，以解决更困难的问题变体，包括更多的模块类型和对驱动器扭矩的约束。我们增加了另外五种模块类型（四个连杆和一个旋转驱动器），共有 16 种模块类型。一种驱动器模块类型具有较轻的质量以及较低的最大扭矩，另一种具有较高的质量和较高的最大扭矩。在

评估“可达性”能力时，如果超过任意驱动器扭矩限制，则返回 0 的终止奖励。作为比较的基础，我们修改了遗传算法，加入了对驱动器扭矩过载构型的惩罚。我们无法将此扩展算法与（Ha 等人，2018）的方法进行比较，因为他们的方法不考虑扭矩。本测试中使用的测试集与上述测试集相同。我们对这个 DQN 进行了 70 万个 episodes（约 57 小时）的训练。

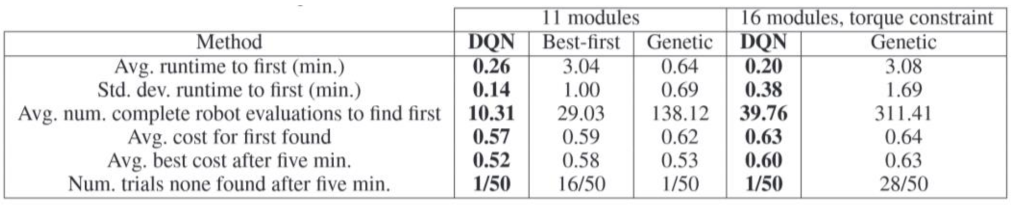
4 结果

比较试验的结果如表 1 所示。我们发现，我们的方法在所有类别中都能产生最佳结果。对于其中一项测试，三种算法都无法在五分钟内找到可行的机器人。

遗传算法通过随机抽样构型，在几次迭代中找到成本较高的可行构型，然后在进一步迭代中将这些结果细化为成本较低的构型。定性地说，我们发现，当有许多可行的机器人执行任务时，比如当目标少、障碍物少时，它往往表现得很好，因为初始采样可能包括完成任务的高成本构型。然而，遗传算法需要许多完整的机器人规划评估。如果评估完整机器人规划的计算量增加，我们预计这种方法会相应地变得更耗时。

最佳优先搜索在其启发式中没有考虑障碍物，因此在存在许多障碍时，其性能往往会降低。它按照增加复杂性的顺序评估机器人，但必须求解一个非线性程序来评估每个节点。由于这一计算成本高昂的子问题，该算法无法在五分钟内找到三分之一测试用例的解决方案。在确实找到解决方案的情况下，它通常无法在剩余时间内改进该解决方案。

表格 1：在 3.4 节中描述的对比测试结果（所有的指标都是越低越好）。



	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

我们观察到，我们的方法最初是深度优先，仅在几个 DQN 前向传播后就可以评估完整的机器人。训练期间的奖励结构引导计算机寻求成本更低的构型。与启发式（Ha 等人，2018）相比，我们的启发式考虑了障碍物的位置。我们发现，与不以障碍物为启发式条件的 ablated 变体相比，我们的算法提高了平均解决方案质量和运行效率。

在具有扭矩约束和附加模块类型的变体中，我们的方法仍然有效地搜索构型空间，并快速输出可行的设计，尽管需要经过更长的训练时间。经常需要更大质量的驱动器模块来创建能够在不超过最大扭矩的情况下延伸到最远目标的构型，从而产生比先前实验中成本更高的解决方案。即使有了更大的模块集和额外的约

束，可行的设计仍然在一分钟内返回。相比之下，在大多数测试集中，遗传算法无法在五分钟内找到可行的构型。

在最直接相关的研究中（Ha 等人，2018），搜索在运行时遇到了维度诅咒。当分支因子（来自可用模块类型的数量）增加时，启发式函数估值的数量呈指数增长。相反，当添加更多模块时，我们必须对 DQN 进行额外时间的训练，但仍然使用 DQN 前向传播，将启发式估值一次分配给扩展节点的所有子节点。与相关方法相比，我们的方法最强的地方是在找到可行的安排之前所需的低计算量，这既源于计算搜索启发式的计算效率（来自 DQN 的前向传播），也源于所评估的完整机器人数量较低。随着任务变得更加复杂，我们预计完整的机器人运动规划评估次数将主导搜索时间，导致相关方法的性能下降，但只会增加我们方法的训练时间。

我们在补充材料中包括了训练网络的代码、预训练的网络权重以及我们实验的代码和结果。

5 存在缺点和未来研究方向

我们研究的一个不足之处是，如果模块类型的集合发生变化，需要重新训练神经网络；未来的研究将考虑使用一个经过训练的网络，从小差异的模块集中开始训练。另一个不足是我们的公式不考虑速度/运动平滑度。在未来的研究中，我们将转向动力学运动规划，而不是准静态 IK。此外，未来的研究将是将基于机器人设计、任务和环境的学习控制策略（Chen、Murali 和 Gupta 2018）与模块选择策略端到端的结合起来。

本研究中的模块构型输入用独热编码的向量组成的列表表示，每个向量表示序列中的模块，并用零填充，直到构型中允许的最大模块数。这种编码方式将构型限制在串联拓扑上，我们将在未来的研究中加以解决这一限制。类似地，我们将设计限制为由离散的组件构成。机器人设计由连续设计参数和离散组件组成，这是一个更一般但更复杂的案例，也是一个正在进行的研究领域（Whitman 和 Choset 2018）。

6 结论

在本文中，我们提出了一种算法，该算法使用数据驱动的启发式图搜索算法来综合出特定任务的模块化机器人设计。我们表明，我们的方法比类似的最先进的方法更有效地返回成本更低的解决方案。在构型搜索中，“维数诅咒”来自于一系列离散模块选择中的多分支节点。需要搜索效率来减轻为每个候选构型创建运动规划的计算量。我们的方法通过使用深度神经网络前向传播一次逼近所有候选模块的值来解决这些挑战，将绝大多数计算转移到离线训练中。尽管我们将重点限制在串联机械臂，但类似的方法也可以应用于更复杂的构型设计。设想我们

的方法可以用于零件质量或预算受限的应用，并且需要将相同的模块应用于频繁变化的应用，但模块类型集合保持固定的应用，例如，在空间、小批量制造或军事应用中。随着模块化机器人的生产成本越来越低，我们希望像我们这样的自动化设计工具能够让非专家也能轻松创建和定制机器人。

7 致谢

这项研究得到了美国宇航局空间技术研究奖学金 NNX16AM81H 的支持。

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

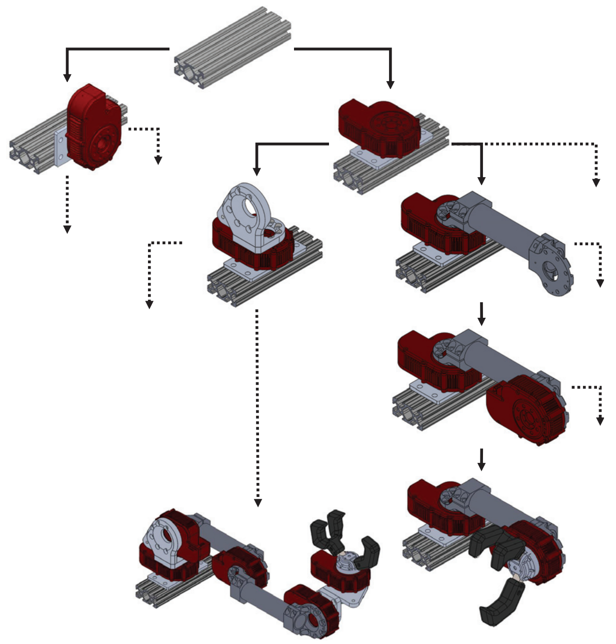


Figure 1: Our approach searches for modular manipulator designs by viewing the space of arrangements as a tree, where modules are sequentially added to the end of the robot. The arrangement at the root of the tree is a base mounting location. Solid arrows represent module additions, and dashed arrows 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).

(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,

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

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,

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

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.

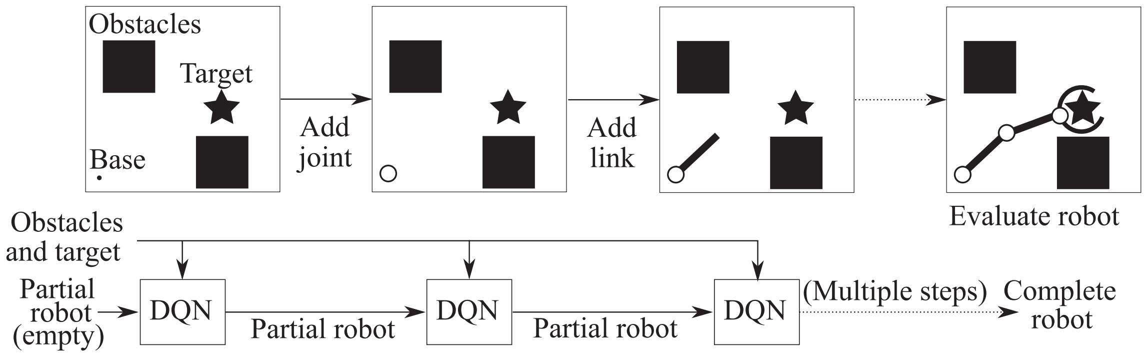


Figure 2: During training, the DQN is used repeatedly to evaluate the contribution each module type would have toward reaching a target. The arrangement is assembled sequentially (top) with modules selections made by the DQN (bottom)

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,

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

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,

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

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.

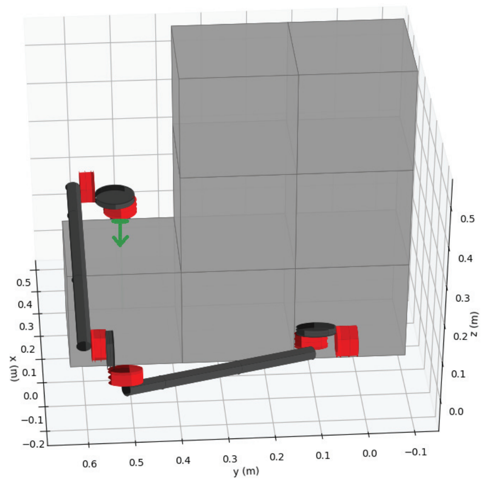
During training we build up an arrangement by sequentially selecting modules. At each step in the episode, the network outputs Q-values for each module type. In our module set, each type of module can connect to only a subset of the other module types. We mask out invalid module connection actions, and 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.



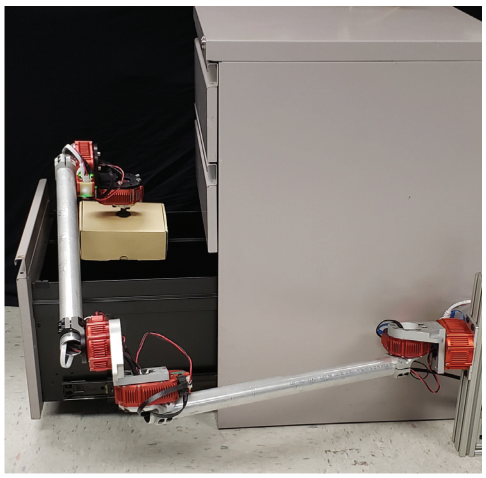


Figure 3: Top: An arrangement of modules (dark grey and red) with base located at the origin reaches a single workspace target position and tip axis (green point with arrow) without colliding with voxelized obstacles (grey cubes). Bottom: The physical modular robot matches the arrangement and environment.

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),

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

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,

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

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

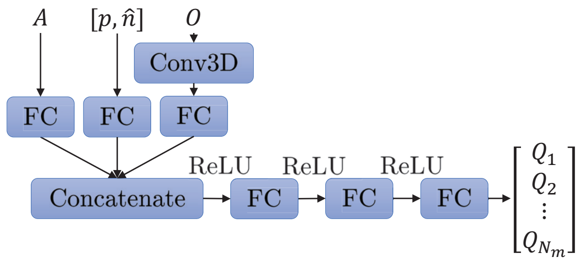


Figure 4: The neural network architecture we used for our DQN consists of fully connected (FC) layers with rectified linear unit (ReLU) activation, and a 3D convolution (Conv3D) over the grid of obstacles. The inputs to the DQN are the current arrangement A, target $T = [p, \hat{n}]$, and obstacle grid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

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.