Composition Learning in “Modular” Robot Systems

Thesis by

Jiaheng Hu

In Partial Fulfillment of the Requirements for the
Degree of
Masters of Science in Robotics

CARNEGIE MELLON UNIVERSITY
Pittsburgh, Pennsylvania

2022
Defended 2nd August 2022

ACKNOWLEDGEMENTS

First of all, I want to thank my committee members, Dr. Howie Choset, Dr. Matthew Travers, Dr. Deepak Pathak and Julian Whitman. In particular, my advisor Howie Choset and my collaborator and mentor Julian Whitman helped me tremendously along the way, without whom this thesis would not have existed. I also want to give my huge appreciation to my family members, Su Zhang, Houzhi Hu and Tianzeng Chen, for their support which means everything to me. Lastly, I'd like to thank my lab-mates Howard Coffin, Richard Ren, Dr. Ian Abraham and Benjamin Freed for the fruitful discussions that consistently inspire me.

ABSTRACT

Modular robot and multi-robot systems share a concept in common: composition, i.e. the study of how parts can be combined so they can be used to achieve certain objectives. Our vision is to enable robotic systems to configure and reconfigure themselves during field deployment, either autonomously or with the help of users, to adapt to emerging tasks and conditions. This goal requires us to generate compositions in real-time, while maintaining the ability to handle emergent constraints and conflicting objectives. To address these challenges, we present evolution-guided generative adversarial networks (EG-GAN) that learns to map task to compositions. Our method trains a generative model to map a task to a distribution of compositions, with training signals guided by the output of evolutionary algorithm operations. Once trained, the EG-GAN can be used to produce compositions in a near real-time fashion. We demonstrate the effectiveness of our algorithm on two distinct composition problems: 1. designing modular robots and 2. forming teams for multi-robot systems, and show that our algorithm outperforms the previous state-of-the-art algorithms in solution quality, solution diversity, and the ability to handle multiple objectives.

A separate challenge in robot composition involves the complexity introduced by inter-component connectivity, which makes the composition space high-dimensional and topologically diverse, and therefore hard to search within. We introduce Grammar-guided Latent Space Optimization (GLSO), a framework that transforms the original composition space into a low-dimensional, continuous latent space via unsupervised learning. The transformation converts the composition problem into a continuous optimization problem, where we apply sample-efficient Bayesian Optimization to search in the latent space for high-performing compositions. Our method allows us to search in the high-dimensional robot composition space more efficiently than previous state-of-the-art.

PUBLISHED CONTENT AND CONTRIBUTIONS

[1] Jiaheng Hu, Julian Whitman, and Howie Choset. “GLSO: Grammar-guided Latent Space Optimization for Sample-efficient Robot Design Automation”. In: *under review*. 2022.

[2] Jiaheng Hu et al. “Large-scale Heterogeneous Multi-Robot Coverage via Domain Decomposition and Generative Allocation”. In: *International Workshop on the Algorithmic Foundations of Robotics*. 2022.

[3] Jiaheng Hu et al. “Modular Robot Design Optimization with Generative Adversarial Networks”. In: *Proceedings 2022 ICRA. IEEE Int. Conf. on Robotics and Automation*. 2022.

TABLE OF CONTENTS

Acknowledgements iii
Abstract iv
Published Content and Contributions v
Table of Contents v
List of Illustrations viii
List of Tables xi

I Introduction and Background 1

Chapter I: Introduction & Motivation 2
Chapter II: Background and Preliminaries 5
2.1 Evolutionary Algorithms 5
2.2 Learning-Based Composition Automation 5
2.3 Generative Adversarial Network 6
2.4 Graph Grammar 6
2.5 Latent Space Optimization 7

II Real-time Robot Composition via Evolution-guided Generative Model 9

Chapter III: Overview 10
Chapter IV: Evolution-guided Generative Model 12
4.1 Generator 13
4.2 Discriminator 14
4.3 Evolution-guided Data Creation 15
Chapter V: Results & Experiments 17
5.1 Modular Robot Design Automation 17
5.2 Multi-robot Team Formation 22

III Robot Composition with Arbitrary Connectivity via Grammar-guided Latent Space Optimization 30

Chapter VI: Overview 31
Chapter VII: Grammar-guided Latent Space Optimization 34
7.1 GLSO: Learning Latent Composition Representation 34
7.2 GLSO: Optimization in the Learned Latent Space 38
Chapter VIII: Results & Experiments 40
8.1 Comparisons and Baselines 40
8.2 Ablation Studies. 41

IV Conclusion and Future Work 42

Chapter IX: Conclusion 43

Chapter X: Future Works 44

10.1 Expeditionary structural composition 44

10.2 Real-time generation of composition with arbitrary connectivity . . . 44

10.3 Determine when to re-compose 44

Bibliography 45

LIST OF ILLUSTRATIONS

Number *Page*

1.1 We are interested in composition problem in robotic systems. Here we shown two examples of robotic systems where composition is crucial. On the left column we show a modular robot system, where modules need to be connected together to form designs. On the right column we show a multi-robot system, where heterogeneous robots need to be grouped into teams to maximize the overall system performance. Our goal is to determine the ideal composition for these systems based on given tasks during field deployment. 3

2.1 An example of graph grammar used for robot design 7

4.1 We present an abstracted version of our pipeline, where we assume that the components used for composition can be represented as shown in the blue box. At each training iteration, a task is generated at random from a predefined distribution of tasks. The generator implicitly maps the task into a population of compositions $\mathcal{X}_{gen}$, symbolized in the red box. The evolution-guided data creation step explores around the generated compositions by evolving them using a procedure inspired by Evolutionary Algorithms, and creates a population of evolved compositions $\mathcal{X}_{pop}$, symbolized in the green box. The discriminator takes as input the task and a robot composition that is either from the generated compositions or the evolved compositions, and tries to distinguish from which population the composition comes from. The output of the discriminator feeds into the loss function L_G, guiding the generator towards generating high-performing compositions. 13

5.1 From top to bottom, we show examples of queried input, top four designs outputted by the trained generator, randomly generated emergency constraints, and the designs filtered by the added constraints for a near-flat terrain (left) and a rough terrain (right). The emergency constraints restricts the maximum number of a certain type of modules, and are added to simulate unexpected module breaks or a shortage during deployment. The diverse design candidates produced by our method allows us to handle these constraints more robustly compared to prior ML approaches. 18

5.2 We consider the problem of multi-robot coverage, where our goal is to dispatch robot teams to each of the sub-regions in a given search domain to maximize the overall coverage. The sub-regions are precomputed from a terrain map of the target region using voronoi partition. In this work, we are interested in the transition from (c) to (d), which requires us to composition heterogeneous robots into teams based on the terrain features of each sub-region. 24

5.3 Our generator outputs a set of allocation plans that show the trade-off between coverage cost and deployment cost, i.e., a Pareto front of allocations. This plot shows the solution sets obtained by NSGA-ii [17] with population size 20, NSGA-ii with population size 50, and our generator, all evaluated on the same test terrain. Here, our generator performs similarly to NSGA-ii of population size 50 and outperforms NSGA-ii of population size 20, with an average runtime that is significantly shorter. On the right, we also include two different allocation plans from the generated solution set, which illustrate how the allocation plan varies with the deployment cost. 26

5.4 A visual comparison of coverage performance between decomposition with generative allocation (left column, ours), random allocation (middle column), or no decomposition (right column), on three different test terrains (top, middle, and bottom rows). On all three test terrains, our method achieves higher coverage over the target region than the other two methods. 28

6.1 Our framework begins by collecting a dataset of robot designs based on a set of graph grammar rules, as shown on the top left. This process of enumerating designs is computationally inexpensive, as no controller is needed. The collected data (example on bottom left) is subsequently used to train a graph variational autoencoder (VAE), which defines a mapping between a low dimensional continuous latent space and the combinatorial design space. A property predictor (shown as the green trapezoid) is simultaneously trained to predict the world space features grounding of the robots from the latent vector, in order to encourage physically similar robots to be grouped together in the latent space. After the VAE is trained, optimization can be performed in the latent space in search of high-performing designs. This VAE can further be used for multiple distinct tasks without the need for retraining. 32

7.1 Visualization of two different spaces (a) when the VAE is co-trained with the property predictor, and (b) without the property predictor. Here we project the latent points into a two-dimensional space, for visualization purposes, by taking the first two principal components. The color of the points correspond to its performance for a given task (here, traversing flat terrain), where a darker color denotes worse performance. We additionally show designs neighboring the same robot (with red border) in the two different latent spaces. We can see that the addition of the property predictor implicitly encourages designs with similar performance to be grouped together, which we find results in more efficient optimization. 36

7.2 We evaluate our framework on a set of locomotion tasks as shown in the figures. The above images showcase the best design identified by our method for four tasks. 39

8.1 Comparison against related methods. The solid line shows average over 3 different random seeds, and the error band represents the maximum and minimum. In each task, GLSO produces higher-reward designs within 500 steps (design samples). 41

8.2 Comparison of GLSO with ablated variants. The solid line shows average over 3 different random seeds, and the error band represents the maximum and minimum. These results show that both the graph grammar and the property predictor are important to GLSO. 41

LIST OF TABLES

Number *Page*

5.1 Performance comparison between real-time algorithms for modular robot design. ↑ means higher is better. Compare to other algorithms with constant test time, our approach produce designs that are superior both in performance and solution diversity, and as a result handles emergency constraints much better (as shown in the EC column). . . 19

5.2 Performance comparison between tested algorithms for modular robot design. ↓ means lower is better, ↑ means higher is better. Compare to evolutionary algorithms, our method can produce designs with on-par quality with significantly faster runtime. 20

5.3 In our experiment, each species of robot has different motion and sensing capabilities. The maximum velocity of a robot is terrain-dependent while the coverage weight is constant. This table shows the capabilities of each species of robot. 25

5.4 Performance comparison between tested algorithms for multi-robot team composition. ↓ indicates that lower measures are better, ↑ that higher are better. NSGA-ii:20 and NSGA-ii:50 [17] have a population size of 20 and 50 respectively. Compared to evolutionary algorithms, our generative allocation (EG-GAN) method produces allocations of similar quality with significantly faster runtime, making it suitable for time-critical deployment. 27

5.5 We studied the effect of decomposition and allocation on the coverage performance under different deployment cost budgets. ↓ indicates that a lower value is better, and ↑ that higher is better. RanA stands for Random Allocation, EG-GAN stands for Generative Allocation using Evolution-guided GAN. Each value is averaged over 100 test terrains. Our framework outperforms the other tested methods under all three deployment cost limits, demonstrating the effectiveness of both the decomposition and the generative allocation. 29

Part I

Introduction and Background

Chapter 1

INTRODUCTION & MOTIVATION

The study of composition, i.e. how parts can be put together to function as a system in the most efficient and effective manner, is an important problem that appears in various different disciplines, including chemistry (de-novo drug design [20]), machine learning (neural architecture search [19]), and robotics (robot composition automation [30]). This thesis focuses on composition in robotics - where a multi-part robotic system (e.g. modular robots, heterogeneous multi-robot teams) is configured and re-configured to address emerging tasks.

The field of robotics poses several unique challenges to the problem of composition [30, 15, 69, 64]. Firstly, the number of possible compositions in robotic systems typically grows exponentially with the number of components, which even for a small set of components can make search computationally intractable. In fact, since robot systems often involve field deployment, we want a solution that's not only tractable, but attainable in real time. Finally, such a system must be robust to emergent situations and unmodeled conditions, and may have to handle multiple conflicting objectives.

We claim that in order to tackle the challenges above, a composition generator must generate a distribution of solutions as oppose to a single one. This may be useful when our task is not properly modeled or there may be some slight mistakes with our generator in the first place. So, if we have a distribution of compositions and one cannot complete a task, we may be able to re-sample from that distribution to obtain more capable compositions. Our research group has already developed learning-based approaches [76, 75] that generate solutions in real-time. However, all previous learning-based approaches seek to learn a *one-to-one* mapping from task to design, which often result in ignoring a variety of the optimal or near-optimal compositions. We therefore present evolution-guided generative adversarial network that learns a *one-to-many* mapping from task to compositions based on supervision of evolutionary algorithms.

Unlike past works using GANs, where a dataset is present prior to training [23, 27, 56, 16, 24], we have no data *a priori* from which a mapping from task to robot compositions can be distilled. Worse yet, collecting such a dataset from scratch is

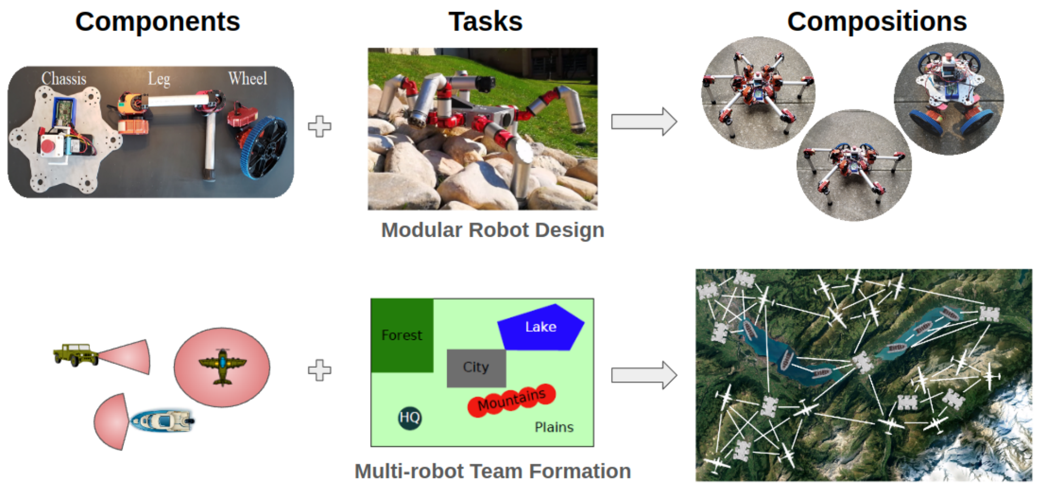


Figure 1.1: We are interested in composition problem in robotic systems. Here we shown two examples of robotic systems where composition is crucial. On the left column we show a modular robot system, where modules need to be connected together to form designs. On the right column we show a multi-robot system, where heterogeneous robots need to be grouped into teams to maximize the overall system performance. Our goal is to determine the ideal composition for these systems based on given tasks during field deployment.

enormously computationally burdensome, as obtaining each task-composition pair requires solving a (potentially multi-objective) combinatorial search problem. Instead, our approach actively collects data on-line during training through a novel evolution-guided data creation process inspired by EAs. This data creation process looks for new promising compositions around the current generated compositions, promoting the generator to improve the quality and diversity of its output. We apply Evolution-guided GAN to two distinct robot composition problems, as shown in Fig. 1.1. In the first case, we seeks to create mobile modular robots designs specialized for traversing varying terrains. In the second case, we compose heterogeneous robots into teams for coverage tasks. Results show that our algorithm outperforms the previous state-of-the-art algorithms for fast composition generation.

We then focus on tackling high-dimensional and unstructured composition space, such as modular robot with arbitrary connectivity and arbitrary number of modules. Previous works relied upon metaheuristic optimization algorithm such as Genetic Algorithms (GA)[29], Genetic Programming (GP) [48], and Particle Swarm Optimization (PSO)[80], where a diverse population of candidate compositions is maintained and updated in search of high-performing compositions. However, these methods requires repeated evaluation of the performance of the compositions, and is therefore sample inefficient during the optimization. We introduce Grammar-guided Latent Space Optimization (GLSO), a framework that transform the original composition space into a low-dimensional, continuous latent space via unsupervised learning. The transformation converts composition optimization into a continuous optimization problem, where we apply sample-efficient Bayesian Optimization (BO) [59] to search in the latent space for high-performing compositions.

This thesis is organized into four parts. In part one, we introduce the motivation of this thesis, and give an overview of the background and preliminaries. In part two, we focus on tackling real-time robot composition, and discuss in detail about evolution-guided generative adversarial network. In part three, we focus on robot composition problems where the composition space is high-dimensional and combinatorial, and discuss in detail Grammar-guided Latent Space Optimization. Finally in part four, we conclude the thesis and discuss about the future extension of this work.

Chapter 2

BACKGROUND AND PRELIMINARIES

In this section, we discuss two kinds of approaches that are popular for tackling composition optimization in robotics: Evolutionary Algorithm and Learning-Based Composition Automation. We then introduce three techniques that our approaches are based upon: generative adversarial network, graph grammar, and latent space optimization.

2.1 Evolutionary Algorithms

Population-based stochastic optimization algorithms, such as Evolutionary algorithms (EAs), have been the *de facto* choice for robot composition problems due to their ability to efficiently look for solutions in the composition search space [72, 28, 7, 8, 57, 51, 49, 66, 63, 73]. This class includes genetic algorithms [37], differential evolution [61], ant colony optimization [18], and particle swarm optimization [38]. EAs maintain and update a population of candidate solutions through a set of operations, and are capable of finding optimal or near-optimal solutions to NP-hard problems within tractable time [12]. They are also able to find multiple solutions due to their population-based nature, and are thus suitable for multi-modal and multi-objective optimizations [6, 71]. Specifically, [7] used a Genetic Algorithm with generative design encoding to evolve locomoting robots, and [72] used a Genetic Algorithm to discover locomoting and swimming robots. Similarly, [57] used a multi-objective particle swarm optimization (MOPSO) to composite multi-robot teams for coverage tasks. However, since EAs require repeated evaluation of the performance of different compositions, they can quickly become computationally expensive in the domain of composition automation, where evaluation of each composition requires running a simulation with a control policy. As a result, EAs are often too time-expensive to be deployed directly in the field, limiting their applicability to scenarios where time or computation is limited.

2.2 Learning-Based Composition Automation

An alternative to explicitly searching a composition space during deployment is to utilize machine learning techniques to map task to compositions. These methods utilize a long training phase to learn a mapping from task to composition, where the learned mapping can be directly queried during deployment to generate task-specific compositions. Specifically, [75, 76] cast modular robot design as a reinforcement learning (RL) problem, and use Q-learning to learn a policy that sequentially adds nodes and edges to the partial graph of designs conditioning on the task. Fit2form [25] learns an end-to-end mapping from task to 3D gripper. Importantly, these methods have the benefit of real-time execution. However, existing learning-based methods typically pose their objective as predicting a single composition that maximizes performance for the queried task. By doing so, the above mentioned approaches ignore the multimodal nature of the composition automation problem, and also cannot handle multiple objectives.

2.3 Generative Adversarial Network

Generative Adversarial Networks [23] have been widely adopted to learn generative models. A GAN is an implicit generative model that attempts to capture the patterns in a dataset, such that the model can be used to generate new samples as if they were drawn from the same underlying distribution as the data. A GAN consists of two components: a generator that learns a mapping from a noise vector to generated data sample, and a discriminator that learns to distinguish generated samples from real samples. These two components are typically implemented as deep neural networks and are optimized simultaneously through gradient descent. Since the two networks have competing objectives, training a GAN can be viewed as two players playing a minimax game [23]. The conditional GAN [56] was later introduced as a variant that output labeled samples by conditioning both the generator and the discriminator on a given label.

Typically, GANs work with continuous domain, such as images, due to the need for backpropagation through the generated data during training. In order to apply GANs to discrete data, Boundary Seeking GAN (BGAN) [27] trains the generator via policy gradient of the likelihood ratio estimated by the discriminator. Since a policy gradient loss doesn't need the data samples to be differentiable with respect to the generator parameters, this method makes it possible to train a GAN for discrete data such as graphs.

2.4 Graph Grammar

One popular composition space in robotics is the graph space, where components are represented as nodes and their inter-connectivity represented as edges. Graph grammar rules provide an effective means to confine a graph composition space,

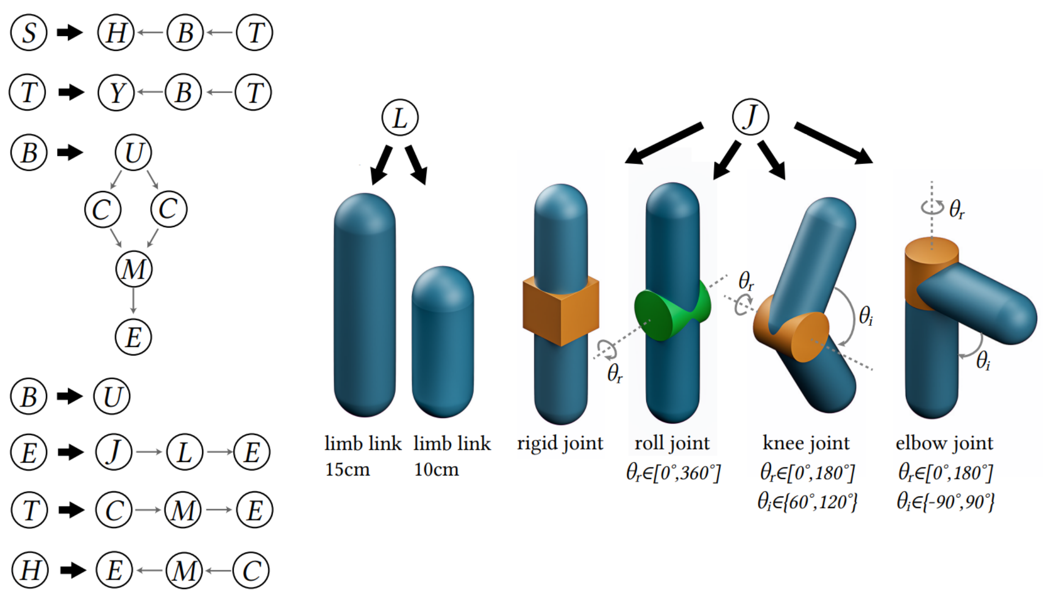

S → H←B←T
T → Y←B←T
B → U, C, C, M, E
B → U
E → J→L→E
T → C→M→E
H → E←M←C
L
J
limb link 15cm
limb link 10cm
rigid joint
roll joint $\theta_r \in [0°, 360°]$
knee joint $\theta_r \in [0°, 180°]$ $\theta_i \in \{60°, 120°\}$
elbow joint $\theta_r \in [0°, 180°]$ $\theta_i \in \{-90°, 90°\}$
θ_r
θ_i

Figure 2.1: An example of graph grammar used for robot design

which can improve optimization efficiency, especially when the composition space is high-dimensional [83, 29]. In robotics, graph grammars have been applied to model both physical structure as well as control laws [44, 67, 42]. More recently, RoboGrammar [83] introduced a set of recursive graph grammar rules for robot design automation. RoboGrammar operates on graphs composed of terminal and non-terminal symbols by recursively expanding non-terminal symbols using a set of predefined rules A final composition graph consist only of terminal symbols, and corresponds to a robot design where the hardware components are represented as nodes and the physical links are represented as edges. Fig. 2.1 demonstrates the set of rules proposed by RoboGrammar. To optimize the robot designs based on the grammar rules, RoboGrammar proposed Graph Heuristic Search (GHS), which operates on a search tree defined by the graph grammar rules. The search is conducted in a A* like manner, where a heuristic function in the form of a graph neural network [79] is trained during the search process. For more details about the RoboGrammar method, see [83].

2.5 Latent Space Optimization

Latent space optimization (LSO) is a framework designed to extend continuous optimization techniques to discrete combinatorial search space [70]. LSO first learns a generative model (typically a variational autoencoder) [41], where the encoder $g_{enc} : \mathbb{X} \to \mathbb{Z}$ maps from the discrete composition space $\mathbb{X}$ to a continuous latent space $\mathbb{Z}$ and the decoder $g_{dec} : \mathbb{Z} \rightarrow \mathbb{X}$ maps data points from the latent space to the combinatorial composition space. Continuous optimization can subsequently be performed in the latent space $\mathbb{Z}$ using standard continuous optimization algorithms such as BO. LSO has been shown to be an effective framework for domains involving discrete data, including natural language [4], arithmetic expressions [43], programs generation [14] and molecules synthesis [22, 36]. In the domain of robot composition automation, however, LSO has not seen much success. [39] made initial progress in learning a latent representation of robot morphology, but only with a small set of serial-chain topology designs, and no optimization was performed.

Part II

Real-time Robot Composition via Evolution-guided Generative Model

Chapter 3

OVERVIEW

In this chapter, we consider the problem of robot composition during field deployment, where compositions need to be generated in real-time. A *de facto* approach for solving composition problem in robotics is to utilize metaheuristic optimization algorithm such as Genetic Algorithms (GA)[29], Genetic Programming (GP) [48], and Particle Swarm Optimization (PSO)[80], where a diverse population of candidate compositions is maintained and updated in search of high-performing compositions. Unfortunately, this optimization process typically requires repeated simulation and evaluation of many different compositions, and can quickly become computationally expensive. Though they can produce high-quality compositions, these methods cannot be directly applied in scenarios like field deployment, where time or computation is limited.

Recent works [75, 76, 25] utilize machine learning (ML) to generate compositions with a low computational cost at run-time. Specifically, these works employ a training phase to learn a *one-to-one* mapping from task to composition for a range of different tasks. This mapping is then used during deployment to create new compositions, whether or not the mapping was trained on a given task. While such an approach may lead to non-optimal compositions, it does produce them in real-time.

The low run-time cost of existing ML approaches make them seemingly good candidates for use in the field. However, the problem of robot composition generation is often multimodal and multi-objective by nature, where for any given task, there are multiple competing objectives (e.g. maximizing speed and minimizing energy consumption for locomotion robot); and for each objective, there exist multiple distinct compositions that are similarly or equally capable. By modelling the task-to-composition mapping as a one-to-one mapping, previous learning-based methods tend to compute a most likely candidate for an optimal solution, as opposed to maintaining a distribution of candidate optimal solutions, thereby hinders a true search for an optimal solution. As a result, these methods fail to *fully describe* the relation between task and optimal compositions, which not only lead to inferior decisions, but also hamper exploration during training.

Furthermore, in a composition automation problem, keeping track of multiple distinct solutions can be valuable, as they provide alternatives in case the “first-choice” solution fails. For instance, if a module fails unexpectedly in the field, and there are not sufficient spare modules to replace it, the user will need to quickly identify and construct a “second-choice” design. Similarly, if the first solution deployed does not work as expected, perhaps due to a simulation to reality gap, then the user will need a different design. Finally, in case where there exist multiple competing composition objectives, a set of solutions can be used to demonstrate the optimal trade-off between these competing objectives (i.e. the pareto set). We therefore develop an algorithm that marries the solution quality and diversity of EAs with the low run-time cost of ML.

In this work, we introduce Evolution-guided GAN (EG-GAN), a framework for real-time generation of task-specific robot compositions. Our method learns a mapping from a task to a *distribution* of compositions through a modified generative adversarial network (GAN) [23, 27, 56], and can be used to generate diverse and high quality compositions in a computationally-efficient manner.

We apply Evolution-guided GAN to two distinct robot composition problems, as shown in Fig. 1.1. In the first setting, we seeks to create mobile modular robots designs specialized for traversing varying terrains. In the second setting, we compose heterogeneous robots into teams for coverage tasks. In both cases, we demonstrate that our method is capable of running in real-time during testing, and outperforms competing ML methods both in terms of solution quality and solution diversity, as well as providing alternative solutions when the first design fails. We further demonstrate the capability of Evolution-guided GAN to handle multi-objective composition on the problem of multi-robot team formation.

Chapter 4

EVOLUTION-GUIDED GENERATIVE MODEL

Our system is composed of three key components: a generator G_θ, a discriminator D_ϕ, and a novel evolution-guided data creator. An illustration of these components is shown in Fig. 4.1. The generator maps a task to a distribution of compositions. The discriminator tries to distinguish between compositions generated by the generator $\mathcal{X}_{gen}$, and compositions produced by the evolution-guided data creator $\mathcal{X}_{pop}$. Both the generator and the discriminator are implemented as neural networks, where θ and ϕ are the respective trainable weights. At every iteration, the evolution-guided data creator performs n steps of evolution on the robot compositions generated by the generator to obtain training data. These evolution steps iteratively improve the output of the generator, pushing the generator to approximate a distribution of increasingly higher-performing candidate solutions. The pseudocode of our training pipeline can be seen in Algorithm 1.

Notation: We present a general pipeline for generating compositions $\mathcal{X} \in \mathbb{X}$ conditioning on a task $\mathcal{T} \in \mathbb{T}$, where $\mathbb{X}$ is a pre-defined composition space and $\mathbb{T}$ is a pre-defined task space. The specific representation of $\mathcal{X}$ and $\mathcal{T}$, as well as the network architecture of the generator and the discriminator, depend on the target application. Notice that we slightly abuse the notation here, where $\mathcal{X}_{gen}$ and $\mathcal{X}_{pop}$ refers to a batch of compositions instead of a single composition.

Algorithm 1 Training of Evolution-guided GAN

```
1:  Initialize Generator $G_\theta$, Discriminator $D_\phi$
2:  for number of training iterations do
3:      Sample task $\mathcal{T}$ from task distribution $\mathbb{T}$
4:      Sample a batch of i.i.d noise vectors $z$ from noise prior $\mathcal{N}(0, I)$
5:      Generate a batch of compositions $\mathcal{X}_{gen}$ from $G_\theta(z, \mathcal{T})$
6:      Evolution-guided data creation (sec. 4.3):
7:          $\mathcal{X}_{pop} \leftarrow \text{EVO}(\mathcal{X}_{gen}, \mathcal{T})$
8:      Update the generator with (4.1):
9:          $\theta \leftarrow \theta - \nabla_\theta L_G(z, \mathcal{T}\}; \theta)$
10:     Update the discriminator with (4.3):
11:         $\phi \leftarrow \phi - \nabla_\phi L_D(\mathcal{X}_{pop}, \mathcal{X}_{gen}, \mathcal{T}; \phi)$
12: end for
```

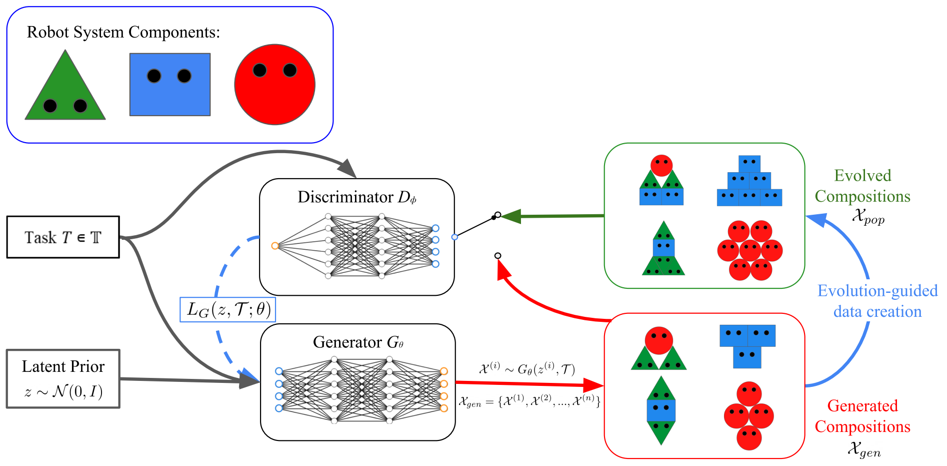

Robot System Components:
Task $T \in \mathbb{T}$
Latent Prior
$z \sim \mathcal{N}(0, I)$
Discriminator D_ϕ
Generator G_θ
$L_G(z, \mathcal{T}; \theta)$
$\mathcal{X}^{(i)} \sim G_\theta(z^{(i)}, \mathcal{T})$
$\mathcal{X}_{gen} = \{\mathcal{X}^{(1)}, \mathcal{X}^{(2)}, ..., \mathcal{X}^{(n)}\}$
Evolved Compositions $\mathcal{X}_{pop}$
Evolution-guided data creation
Generated Compositions $\mathcal{X}_{gen}$

Figure 4.1: We present an abstracted version of our pipeline, where we assume that the components used for composition can be represented as shown in the blue box. At each training iteration, a task is generated at random from a predefined distribution of tasks. The generator implicitly maps the task into a population of compositions $\mathcal{X}_{gen}$, symbolized in the red box. The evolution-guided data creation step explores around the generated compositions by evolving them using a procedure inspired by Evolutionary Algorithms, and creates a population of evolved compositions $\mathcal{X}_{pop}$, symbolized in the green box. The discriminator takes as input the task and a robot composition that is either from the generated compositions or the evolved compositions, and tries to distinguish from which population the composition comes from. The output of the discriminator feeds into the loss function L_G, guiding the generator towards generating high-performing compositions.

4.1 Generator

The generator $G_\theta : \mathbb{R}^P \times \mathbb{T} \to \mathbb{X}$ takes as input a numerical encoding of the queried task $\mathcal{T} \in \mathbb{T}$, along with a P-dimensional vector $z \in \mathbb{R}^P$ sampled from a standard multivariate Gaussian distribution, $z \sim \mathcal{N}(0, I)$, and output a numerical representation of a robot composition $\mathcal{X} \in \mathbb{X}$. I.e. $\mathcal{X} \sim G_\theta(z, \mathcal{T})$

For compositions with a continuous representation (e.g. large-scale multi-robot team formation), the generator directly outputs a composition and is trained using the standard conditional GAN loss [56]:

$$L_G(z, \mathcal{T}; \theta) = \log(1 - D_\phi(G_\theta(z, \mathcal{T}), \mathcal{T})) \tag{4.1}$$

where $D_\phi(G_\theta(z, \mathcal{T}), \mathcal{T})$ is the scalar output of the discriminator on the (composition, task) pair, explained in detail in section 4.2.

However, not all composition problem have a continuous representation. For compositions that are discrete (e.g. design graph), the output of the generator is still a continuous matrix, which now represents a distribution $\widetilde{\mathcal{X}}$ over the composition, from which we can sample discrete compositions $\mathcal{X}$. I.e. $\widetilde{\mathcal{X}} = G_\theta(z, \mathcal{T})$, $\mathcal{X} \sim \widetilde{\mathcal{X}}$. This sampling process is non-differentiable and prevents gradients from flowing through it. This prevents us from using the more common continuous GAN loss [23]. We therefore adopted Boundary Seeking GAN (BGAN) [27], by training the generator via policy gradient based on the likelihood ratio estimated by the discriminator. Since our generator is conditioned on the task, we developed a conditional variant of BGAN, where the generator loss is

$$\begin{aligned} &L_G(z, \mathcal{T}, \{\mathcal{X}^{(1)}, ..., \mathcal{X}^{(m)}\}; \theta) \\ &= -\sum_m \frac{D_\phi(\mathcal{X}^{(m)}, \mathcal{T})}{\sum_{m'} D_\phi(\mathcal{X}^{(m')}, \mathcal{T})} P_\theta(\mathcal{X}^{(m)} | z, \mathcal{T}) \end{aligned} \tag{4.2}$$

where $\mathcal{T}$ is the task; $P_\theta(\mathcal{X}|z, \mathcal{T})$ is the posterior probability for composition $\mathcal{X}$ specified by $\widetilde{\mathcal{X}} = G_\theta(z, \mathcal{T})$; $\{\mathcal{X}^{(1)}, ..., \mathcal{X}^{(m)}\}$ are m compositions sampled from $P_\theta(\mathcal{X}|z, \mathcal{T})$, and $D_\phi(\mathcal{X}^{(m)}, \mathcal{T})$ is the scalar output of the discriminator.

During training, $\mathcal{T}$ is uniformly sampled from the task space. Both $\mathcal{T}$ and z are re-sampled at the start of each training iteration.

4.2 Discriminator

The discriminator $D_\phi : \mathbb{X} \times \mathbb{T} \rightarrow (0, 1)$ takes in a robot composition $\mathcal{X}$ and the task $\mathcal{T}$, and outputs a scalar d between 0 and 1. I.e. $d = D_\phi(\mathcal{X}, \mathcal{T}) \in (0, 1)$. Similar to the canonical GAN, this scalar can be viewed as a prediction of whether the composition is generated by the generator. The discriminator has the same network structure as the generator except for the input and output layer, which are adjusted to correspond to the input and output dimensions. The training data for the discriminator comes from two sources. Half of the compositions are synthesised by the generator, and are labeled 0 (“fake”). The other half are obtained from the evolution-guided data creation step (explain in detail in section 4.3) and are labeled 1 (“real”).

The discriminator is trained to minimize the cross entropy loss between predicted and actual composition labels:

$$\begin{aligned} L_D(\mathcal{X}_{pop}, \mathcal{X}_{gen}, \mathcal{T}; \phi) = \quad & -[\log D_\phi(\mathcal{X}_{pop}, \mathcal{T}) + \\ & \log(1 - D_\phi(\mathcal{X}_{gen}, \mathcal{T}))] \end{aligned} \tag{4.3}$$

where $\mathcal{X}_{pop}$ is a batch of compositions obtained from the evolution-guided data creation step, and $\mathcal{X}_{gen}$ is a batch of compositions generated from the generator.

Algorithm 2 Evolution-guided Data Creation (single-objective)

```
1: function EVO($\mathcal{X}_{gen}, \mathcal{T}$)
2:     $\mathcal{X}_{cur} \leftarrow \mathcal{X}_{gen}$
3:     for $n$ iterations do
4:         $\mathcal{X}_{new} \leftarrow \emptyset$
5:         for each (parent1, parent2) $\in \mathcal{X}_{cur}$ do
6:             child1, child2 ← Crossover(parent1, parent2)
7:             child1, child2 ← Mutation(child1, child2)
8:             $d_{ij} \leftarrow$ Dis(parent i, child j), $i, j \in \{1, 2\}$
9:             if $d_{11} + d_{22} < d_{12} + d_{21}$ then
10:                candidate1 = $\arg\max_{\text{(parent1, child1)}} \text{Fitness}(x)$
11:                candidate2 = $\arg\max_{\text{(parent2, child2)}} \text{Fitness}(x)$
12:            else
13:                candidate1 = $\arg\max_{\text{(parent1, child2)}} \text{Fitness}(x)$
14:                candidate2 = $\arg\max_{\text{(parent2, child1)}} \text{Fitness}(x)$
15:            end if
16:            $\mathcal{X}_{new}$.append(candidate1)
17:            $\mathcal{X}_{new}$.append(candidate2)
18:        end for
19:        $\mathcal{X}_{cur} \leftarrow \mathcal{X}_{new}$
20:    end for
21:    return $\mathcal{X}_{new}$
22: end function
```

4.3 Evolution-guided Data Creation

Our method is fundamentally different from a canonical GAN since we do not have a dataset collected *a priori*. Instead, we generate training data online through a novel evolution-guided process, inspired by evolutionary algorithms, which iteratively pushes the generator towards generating designs with desirable properties.

During each training iteration, we pass a task $\mathcal{T}$ and a batch of randomly sampled latent vectors z through the generator to obtain a batch of compositions $\mathcal{X}_{gen}$. These compositions are then treated as if they are the population of an EA, and iterate through n EA steps (e.g. mutations, cross-over, evaluation, and elite selection) to create an evolved population $\mathcal{X}_{pop}$. The choice of the the specific EA algorithm determines the property of the evolved population: for a single-objective EA, the evolved population $\mathcal{X}_{pop}$ should have higher diversity and performance; for a multi-objective EA, $\mathcal{X}_{pop}$ should have better Pareto optimality. Finally, $\mathcal{X}_{pop}$ are passed into the discriminator in place of what would be considered the “real” data in a conventional GAN. By training the GAN with the evolved samples, we effectively guide the generator to model a task-conditioned distribution that is iteratively shifted towards higher-quality regions in the solution space.

Algorithm 3 Evolution-guided Data Creation (multi-objective)

```
1:  function EVO($\mathcal{X}_{gen}, \mathcal{T}$)
2:      $\mathcal{X}_{cur} \leftarrow \mathcal{X}_{gen}$
3:      for $n$ iterations do
4:          Evaluate Objectives Values for $\mathcal{X}_{cur}$ on $\mathcal{T}$
5:          for $x$ in $\mathcal{X}_{cur}$ do
6:              Assign Rank (level) based on Pareto Optimality
7:              Generate sets of Nondominated Solutions
8:              Determine Crowding Distance
9:          end for
10:         Generate New Population $\mathcal{X}_{new}$:
11:             Roulette Wheel Selection
12:             Crossover and Mutation
13:         $\mathcal{X}_{cur} \leftarrow \mathcal{X}_{new}$
14:     end for
15:     return $\mathcal{X}_{new}$
16: end function
```

The generator and discriminator are both conditioned on the task description. Without this task-conditioning, the training procedure would be similar to a standard EA, wherein a population of allocation plans are evolved for a single target region. By conditioning the generator and the discriminator on the tasks, and randomly sampling tasks at each iteration, the GAN learns to interpolate between different tasks, which is the key to how our approach is able to generalize to unseen tasks during deployment.

Any variety of population-based optimization algorithm could be used inside the data creation process. However, the specific algorithm chosen will affect the convergence behavior of the generator. For single objective composition, we used a deterministic crowding genetic algorithm [54], which explicitly encourages diversity of the population by enforcing a pair-wise replacement between parents and children, with detail shown in algorithm 2. For multi-objective composition, we used NSGA-ii [17], a well-known fast sorting and elite multi-objective evolutionary algorithm [82], with detail shown in algorithm 3.

Chapter 5

RESULTS & EXPERIMENTS

We examine EG-GAN on two distinct robot composition problems: 1. designing modular robot and 2. forming teams for multi-robot systems. We conducted all training and testing on a desktop computer with Ubuntu 18.04, Intel i7 eight-core processor at 1.9240GHz, and an NVIDIA GTX 1070 graphics card.

5.1 Modular Robot Design Automation

We first evaluated our method by composing modular robots for traversing different terrains, as shown in the left column of Figure 1.1. We evaluate our method by querying them with randomly sampled terrains and simulating the output designs. Some of the designs generated by EG-GAN can be seen in Figure 5.1.

Composition Space

The composition space of this experiment consists of robots composed from four different types of modules: leg, wheel, leg-wheel, and chassis, where each of these modules corresponds to physical hardware produced by Hebi Robotics [26]. A “none” module type is also allowed, so that designs may have fewer than the maximum allowable number of modules. Each robot contains one chassis, and each chassis has six ports where modules can be attached. Therefore, there is a total of $4^6 = 4096$ different designs in our design space.

Each composition can be think of as a fixed topology graph, which is numerically represented as a $m \times n$ one-hot matrix $\mathcal{X}$, where n is the number of different node types and m is the total number of nodes. Each node type corresponds to a module type, and therefore $\mathcal{X}$ specifies the type of module for each node. Notice that this composition space is discrete, which as mentioned in sec 4.1 requires the generator to output a distribution over compositions $\widetilde{\mathcal{X}}$. We set $\widetilde{\mathcal{X}}$ to be continuous $m \times n$ matrix. Each row of $\widetilde{\mathcal{X}}$ sums to 1 and can be viewed as a categorical distribution over all possible module selections for the given node. We can then obtain a discrete composition $\mathcal{X}$ through a categorical sampling of $\widetilde{\mathcal{X}}$.

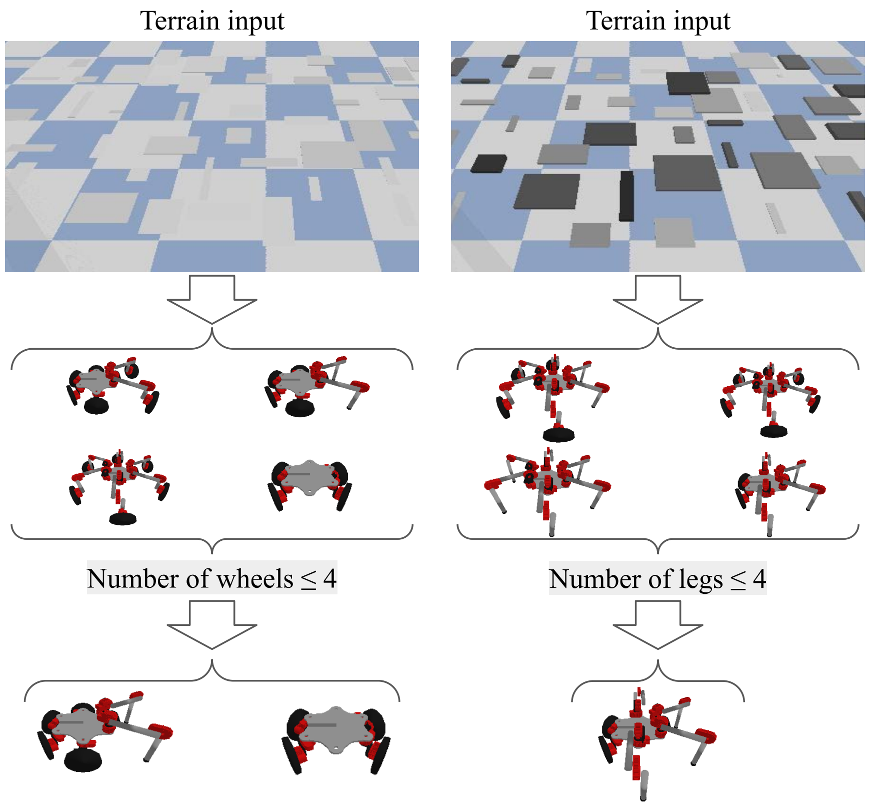


Figure 5.1: From top to bottom, we show examples of queried input, top four designs outputted by the trained generator, randomly generated emergency constraints, and the designs filtered by the added constraints for a near-flat terrain (left) and a rough terrain (right). The emergency constraints restricts the maximum number of a certain type of modules, and are added to simulate unexpected module breaks or a shortage during deployment. The diverse design candidates produced by our method allows us to handle these constraints more robustly compared to prior ML approaches.

Task Space

We define the task space as the space of different terrains. Each task $\mathcal{T}$ is therefore represented as a 2D elevation map of the given terrain. We generate random terrains by adding blocks of randomly selected height, width, and spacing to flat ground. Examples of such terrain can be seen in the first row of Figure. 5.1. We defined the performance of each design on a given terrain as the distance travelled within a fixed amount of time, and obtain the performance by running simulations in Pybullet [11].

Network Architecture

To generate compositions conditioning on the task $\mathcal{T}$, $\mathcal{T}$ is first passed through two convolution layers, each with a 3×3 kernel of filter size 32. The output of the convolution layer is then flattened, passed through a full connected layer of output

	Comparison Metrics		
Methods	Max distance travelled (m)↑	# of distinct solutions ↑	Max dist. (EC) (m)↑
Fit2form [25]	4.03 ± 0.91	1.2 ± 0.2	2.55 ± 1.31
DQN based [76]	4.31 ± 0.26	2.7 ± 0.9	3.10 ± 0.92
MolGAN [16]	3.82 ± 0.55	1.8 ± 0.3	2.93 ± 1.00
Random Sampling	0.72 ± 0.64	–	0.63 ± 0.69
EG-GAN (ours)	5.10 ± 0.42	2.2 ± 0.7	3.77 ± 1.18
EG-GAN-crowding (ours)	**5.73 ± 0.22**	**5.8 ± 1.5**	**5.36 ± 0.52**

Table 5.1: Performance comparison between real-time algorithms for modular robot design. ↑ means higher is better. Compare to other algorithms with constant test time, our approach produce designs that are superior both in performance and solution diversity, and as a result handles emergency constraints much better (as shown in the EC column).

dimension P, concatenated with z and passed into a multi-layer perceptron (MLP) network with 3 hidden layers of size 64 to generate the output. Batch normalization [34] and ReLU activation [2] are applied at each hidden layer.

Control Policy & Objective Evaluation

In order to evaluate the performance of each design, we need a policy for each design in the design space. In other words, the performance $\mathcal{J}$ is obtained by simulating a design at a task given a policy. We adopted a learned modular policy trained through deep reinforcement learning to control the generated designs. The implementation and training of the controller follows [74], and is completed before EG-GAN starts training. Note that the methods of this chapter could be applied regardless of the type of controller used, since the performance evaluation of a design includes both physical and control parameters. We investigate the effect of the controller in Sec. 5.1.

Comparison Metrics

We compare our algorithm to a variety of related methods using a set of metrics for the quality and diversity of the output solutions. To measure the quality of the generated designs, we record the performance of the best robot design found by each algorithm, and report it as *max distance travelled*. We then quantify the diversity of the generated solutions following the benchmark proposed in [46] to obtain the *number of distinct solutions*. Specifically, given a set of candidate solutions produced by an algorithm, a candidate solution counts as a distinct solution if it is at least δ distance away from any existing distinct solution, and its performance is within ϵ distance from the population's highest performance. We used $\delta = 1$ and $\epsilon = 0.5$ in our experiment, where the distance between two compositions is

	Comparison Metrics		
Methods	Max distance travelled (m) ↑	# of distinct solutions ↑	Avg. runtime (min.) ↓
GA-20 [37]	4.21 ± 1.43	1.8 ± 0.8	25.4 ± 0.7
GA-50 [37]	5.63 ± 0.34	2.0 ± 1.4	71.9 ± 1.3
GA-crowding [54]	5.33 ± 0.25	**6.4 ± 1.3**	97.6 ± 2.1
EG-GAN-crowding (ours)	**5.73 ± 0.22**	5.8 ± 1.5	**0.003**

Table 5.2: Performance comparison between tested algorithms for modular robot design. ↓ means lower is better, ↑ means higher is better. Compare to evolutionary algorithms, our method can produce designs with on-par quality with significantly faster runtime.

measured by their hamming distance.

To quantitatively examine the benefit of having solution diversity, we introduce the concept of "emergency constraints" into our experiment. Emergency constraints take the form of limitations on the maximum number of each type of modules a design can contain. These constraints model a scenario where a module unexpectedly broke, or a module shortage is discovered during the prototyping process, and these constraints are generated at random during testing. If an algorithm fails to generate any design that satisfies the constraints, a random design that satisfies the constraints is used. We record this quantity as *max distance travelled with emergency constraints*. For evolutionary algorithms, these emergency constraints can be bypassed by rerunning the search process from scratch while taking these constraints into account. Therefore, we only report this quantity for real-time algorithms.

Lastly, when comparing against evolutionary algorithms, we included *average runtime*, which measures the wall time of each algorithm during execution in minutes, to demonstrate the computational efficiency of our algorithm.

Comparison with other real-time algorithms

We first compare against learning-based and rule-based approaches. As with our algorithm, these methods can run in near real-time after a longer training procedure. While the related ML algorithms are all trained to generate a single design, their trained model can often be queried stochastically to obtain multiple designs, albeit without any guarantee that designs obtained in this manner are near-optimal. All learning-based algorithms are trained for 12 hours on i.i.d. sampled tasks. Specifically, we compare against:

Fit2form: Our implementation of Fit2form follows [25], where a generator is trained to map task to a single design by maximizing the expected performance of the design. Unlike in the original work, the modular robot we are trying to design in our experiment is not differentiable. Therefore, instead of back-propagating directly with respect to the expected performance, we used a policy gradient loss.

DQN based: Our implementation of DQN follows [76], where a policy sequentially adds nodes and edges to the partial graph of designs. A value function, implemented as neural network, is learned to serve as a search heuristic.

MolGAN: MolGAN [16] was originally introduced to optimize molecule structure given a labelled dataset. This approach involves using a GAN to keep diversity while using reinforcement learning to promote performance. We implement and evaluate a conditional variant of MolGAN for modular robot design. Since we do not have a dataset required for the MolGAN training, we randomly sample valid robot designs from the design space to train the GAN.

Random Sampling: As a simple baseline, we also examined the result of randomly sampling in the design space. Specifically, a random design is generated by sampling module types for each node of the design graph. For each test terrain, we sample 10 i.i.d. designs, and record the one with the highest performance.

Results are presented in Table 5.1. Each entry is averaged over 10 randomly sampled terrains, each with 20 independent runs. We found that both versions of our algorithm outperform the competing algorithms both in terms of design quality and design diversity. Further, when the emergency constraints are introduced, the crowding version of our algorithm outperforms all other algorithms by a significant margin, demonstrating the advantage gained from outputting multiple designs when the first-choice design fails.

Comparison with Evolutionary Algorithms

We also compare our algorithm against evolutionary algorithms. These algorithms are computationally expensive, but can produce high-quality designs. We compare against:

Classical genetic algorithm: Our implementation of a classical genetic algorithm follows [37], with uniform crossover and uniform mutation. We run two versions of the classical genetic algorithm with population size of 20 and 50 respectively, with a max iteration of 50, and report their results as GA-20 and GA-50 in Table 5.2.

Crowding genetic algorithm: Crowding approaches are introduced to EAs as an effective way to generate multi-modal solutions [6]. We therefore use a deterministic crowding genetic algorithm as a benchmark to examine the capability of our algorithm to generate diverse solutions. Our implementation follows [54], using a population size of 50 and a max iteration of 50.

Results are shown in Table 5.2. Compared to EAs, both versions of our algorithm are able to generate comparable designs more efficiently. It is important to note that our execution efficiency comes at the cost of the training time, which EAs do not require. However, the training takes place before the generator is deployed, and need only be trained once before being used for many tasks. Also note that EAs tend to find better solutions given larger population sizes and more of iterations, but their time cost scales proportionally.

Effect of the controller applied

We also examined how the performance of our method is affected by the controller applied to the robots. For this experiment, we replaced the neural controller in section 5.1 by a hand-crafted controller, where the legs follow a cyclic alternating-tripod gait, and the wheels spin forward. We retrained EG-GAN with this new controller, and compare the max distance travelled by generated designs with Fit2form and Random Sampling:

Fit2form	Random	**EG-GAN-crowding**
3.81 ± 1.22	0.86 ± 0.81	**4.96 ± 0.45**

The hand-crafted controller is rather simple, resulting in a lower average distance travelled for each design than did the neural control. Even so, our algorithm is still able to find designs that are well-suited with this controller, demonstrating that our algorithm is agnostic to the controller used, as long as it is consistent at training and testing times.

Notice that here we focus on relatively restrictive design space (fixed topology), where composition space with arbitrary connectivity is addressed in Part. III.

5.2 Multi-robot Team Formation

In our second experiment, we consider composition in multi-robot systems. Specifically, we consider the problem of multi-robot coverage, where our goal is to dispatch robot teams to each of the sub-regions in a given search domain to maximize the overall coverage (Fig. 5.2). We assume that the sub-regions are obtained through voronoi partitioning of target regions based on terrain features before the start of the training, where the detail of partition follows [32]. We focus on using EG-GAN to allocate robots to each of the sub-terrains (sub-tasks) in this work, which can be think of as a multi-objective composition problem with two conflicting objectives: the coverage cost and the deployment cost. Our generative model for team composition is trained for 20 hours on i.i.d. satellite images sampled from the training dataset before testing. Both the generator and the discriminator are optimized using Adam optimizer [40] with a learning rate of $1e^{-5}$.

Composition Space

The composition problem in this experiment is spanned by a set of heterogeneous robots $\mathcal{R} = \{r_1, ..., r_N\}$ to allocate to each sub-region. Each robot r_i belongs to a species [62], $o_i \in O$ which determines its motion and sensing capability. The motion capability of a robot is quantified by its maximum velocity in each terrain. Each robot takes 20 observations during an episode. These observations are given a "coverage weight" based on the species, with a higher coverage weight indicating better sensing capabilities. We set our robot bank to contain three different species of robots: aerial robots, ground robots and aquatic robots. We hand-selected the max velocities and coverage weights of each species, as summarized in Table 5.3.

A robot team a_i is defined as a subset of available robots $\mathcal{R}$ which act as a team to cover a sub-region. A composition corresponds to an allocation plan $\mathcal{X} = \{a_1, \dots a_K\}$, which consists of K teams, one for each task, where $\bigcup_{i=1}^{K} a_i \subseteq \mathcal{R}$, and $\forall i \neq j : a_i \cap a_j = \emptyset$. We numerically represent an allocation plan as a continuous non-negative $K \times O$ matrix $\mathcal{X}_{gen}$, where O is the total number of robot species. Each entry of $\mathcal{X}_{gen}(i, j)$ specifies the fraction of robots of species j to deploy to task i. The integer number of robots of species j to deploy to sub-region i can then be calculated through a continuous relaxation $N_j^{(i)} = \text{round}(N_j \times \mathcal{X}_{gen}(i, j))$, where N_j is the total available number of robots of species j.

Task Space

Our task space consists of a set of sub-terrains to be covered. The set of sub-terrains is determined through applying domain decomposition [50] on the DroneDeploy real-world satellite image dataset [60], which contains 6888 satelite image chips of 300x300 pixels, where each pixel has a corresponding terrain label. In our experiment, each terrain label belongs to one of the following four categories: [Water, Mountain, Plain, City]. The task $\mathcal{T}$ is numerically represented as a $K \times M$

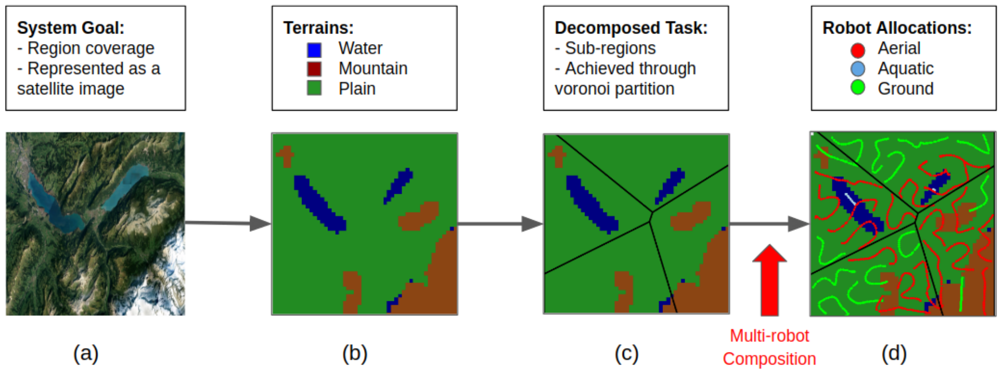


Figure 5.2: We consider the problem of multi-robot coverage, where our goal is to dispatch robot teams to each of the sub-regions in a given search domain to maximize the overall coverage. The sub-regions are pre-computed from a terrain map of the target region using voronoi partition. In this work, we are interested in the transition from (c) to (d), which requires us to composition heterogeneous robots into teams based on the terrain features of each sub-region.

terrain features matrix, where K is the number of sub-regions, and M is the number of distinct terrain types. Each row of the terrain features matrix correspond to the terrain features distribution of a specific sub-region, and sum to 1.

Network Architecture

The generator $G_\theta(z, \mathcal{T})$ is implemented as a multi-layer perceptron (MLP) network with 3 hidden layers of size 64, with batch normalization [34] and ReLU activation [2] at each hidden layer. $G_\theta(z, \mathcal{T})$ takes in a P-dimensional vector $z \in \mathbb{R}^P$ sampled from a standard multivariate Gaussian distribution, $z \sim \mathcal{N}(0, I)$, along with a $K \times M$ terrain features matrix, and output a $K \times O$ allocation plan $\mathcal{X}_{gen}$.

Control Policy & Objectives Evaluation

We denote the *coverage cost* of a robot team a on task t as $f(a, t) \in \mathbb{R}$, and use the sum of the coverage cost on all tasks as the total coverage cost for a given team composition. The coverage cost for each robot team a_i is measured via the path ergodicity [52], a popular metric for information-gathering and search tasks [55, 53, 3, 1]. Intuitively, and disregarding the effect of the species weights w_j, the ergodic metric is minimized when the number of observations taken in any area of the full region is proportional to the amount of information in that area. In this work, we assume that the information is uniformly distributed over the entire region, so, ideally, observations should be equally spaced. We used a model predictive control policy similar to [53] to optimize the trajectories of the robots relative to the ergodic metric.

In addition, each robot r_i deployed incurs a deployment cost c_i, where the total deployment cost is the sum of the cost of all deployed robots. Both the deployment cost and the coverage cost are objectives to be minimized. We thus frame multi-robot team formation as an multi-objective composition problem, where the goal is to find a set of allocations with optimal trade-offs between the two conflicting objectives, i.e. the Pareto-optimal allocations.

	Max velocity for different terrains (m/s)				
Species	Water	Mountain	Plain	City	Coverage Weight
Ground robot	0	20	40	50	20
Aerial robot	50	50	50	30	10
Aquatic robot	50	0	0	30	20

Table 5.3: In our experiment, each species of robot has different motion and sensing capabilities. The maximum velocity of a robot is terrain-dependent while the coverage weight is constant. This table shows the capabilities of each species of robot.

Pareto front Comparison

To the best of our knowledge, no other real-time algorithm can generate / approximate a set of Pareto front solutions for the problem of composition. We therefore compare EG-GAN against traditional multi-objective optimization approaches such as multi-objective genetic algorithms (MOGA). Although MOGAs do not fit our problem statement needs due to their high computational cost, they can nevertheless be used as a benchmark for evaluating the quality of the Pareto front solutions generated by our generative model. The goal of this experiment is to determine whether generative allocation can produce allocations that match or surpass traditional centralized task allocation approaches, while reducing computational expense at run-time.

We compare between the generative allocation and NSGA-ii [17], a well-known fast sorting and elite multi-objective evolutionary algorithm [82]. Our implementation of NSGA-ii used single-point arithmetic crossover with a crossover probability of 0.5 and uniform mutation with a mutation rate of 0.1. The maximum number of iterations is set to be 50, where NSGA-ii:20 has a population size of 20, and NSGA-ii:50 has a population size of 50.

We numerically evaluate the Pareto fronts discovered by the tested algorithms on *hypervolume* [84], a common set-quality indicators for stochastic multi-objective optimizers [45]. The hypervolume set-quality indicator maps a point set in $\mathbb{R}^d$ to the

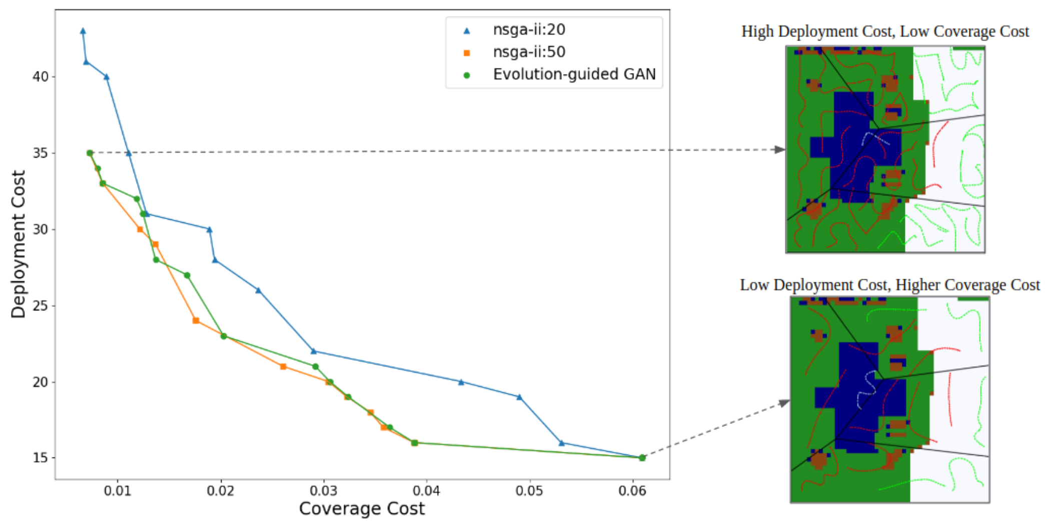


Figure 5.3: Our generator outputs a set of allocation plans that show the trade-off between coverage cost and deployment cost, i.e., a Pareto front of allocations. This plot shows the solution sets obtained by NSGA-ii [17] with population size 20, NSGA-ii with population size 50, and our generator, all evaluated on the same test terrain. Here, our generator performs similarly to NSGA-ii of population size 50 and outperforms NSGA-ii of population size 20, with an average runtime that is significantly shorter. On the right, we also include two different allocation plans from the generated solution set, which illustrate how the allocation plan varies with the deployment cost.

measure of the region dominated by that set and bounded above by a given reference point, also in $\mathbb{R}^d$, where d is the number of objectives. As is visually evident, a larger hypervolume is better. In our experiment, we used $(0.25, 60)$ as the reference point. we additionally report *average runtime*, which measures the wall time of each algorithm during execution in minutes, to demonstrate the computational efficiency of our algorithm.

Quantitative results of the experiments are presented in Table 5.4, where each entry is averaged over five independent runs. A visualization of the Pareto fronts discovered by the tested algorithms for a specific terrain and the corresponding solutions can be seen in Fig. 5.3. Compared to NSGA-ii, our trained generator is able to generate comparably effective allocations more efficiently, since its execution only consist of a neural network forward pass. It is important to note that our execution efficiency comes at the cost of the training time, which NSGA-ii do not require. However, the training takes place before the generator is deployed, and need only be trained once before being used for many tasks. Also note that NSGA-ii solution quality tends to improve solutions given larger population sizes and more iterations, but its time cost scales proportionally.

Methods	Hypervolume ↑	Avg. runtime (min.) ↓
NSGA-ii:20	9.4 ± 1.0	43.1 ± 0.6
NSGA-ii:50	**11.9** ± 1.5	107.6 ± 0.9
EG-GAN (ours)	11.8 ± 0.8	**0.02**

Table 5.4: Performance comparison between tested algorithms for multi-robot team composition. ↓ indicates that lower measures are better, ↑ that higher are better. NSGA-ii:20 and NSGA-ii:50 [17] have a population size of 20 and 50 respectively. Compared to evolutionary algorithms, our generative allocation (EG-GAN) method produces allocations of similar quality with significantly faster runtime, making it suitable for time-critical deployment.

Coverage Comparison

The primary goal of this set of experiments is to justify the effectiveness of the overall decomposition and allocation pipeline. Specifically, we provide comparison results on the following three variants:

- *No Decomposition*. Robots are randomly distributed across the entire search and directly planned their trajectories using distributed MPC.
- *Decomposition + Random Allocation*. A decomposition step is first carried out and then the robots are randomly grouped into teams and allocated to each sub-region, as a baseline.
- *Decomposition + Generative Allocation (ours)*. The robots are grouped into teams based on the output of our trained generative model.

We present coverage performance under different deployment cost limits. In this work, we assume for simplicity that each robot incurs a deployment cost of 1. Therefore, the total deployment cost corresponds to the number of robots deployed. A higher deployment cost corresponds to more deployable robots, which naturally results in higher coverage performance. Notice that with generative allocation, the allocation plans under different deployment cost limits are generated simultaneously as solutions on the same Pareto front, i.e., different parts of the latent space input to the generator map to different Pareto-optimal solutions for the same task. Visualizations of selected results can be seen in Fig. 5.4. Quantitative results of the experiments are collected through evaluation on 100 test terrains, and are presented in Table 5.5.

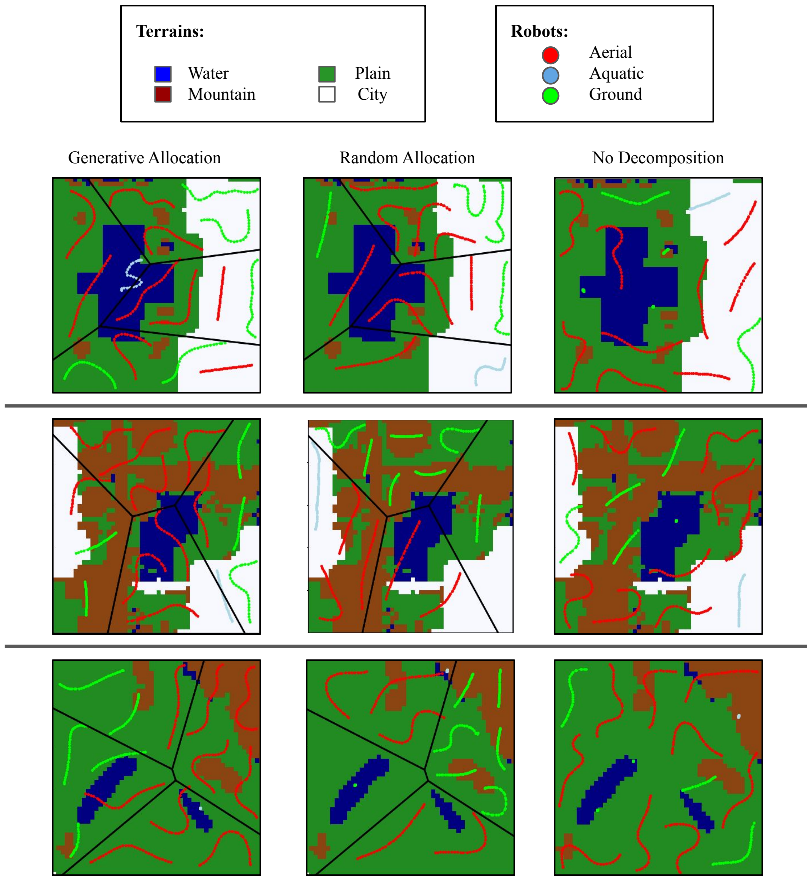


Figure 5.4: A visual comparison of coverage performance between decomposition with generative allocation (left column, ours), random allocation (middle column), or no decomposition (right column), on three different test terrains (top, middle, and bottom rows). On all three test terrains, our method achieves higher coverage over the target region than the other two methods.

We observe from the visualization that with *No Decomposition*, some of the robots would get trapped in a undesirable local optima due to unfavorable initialization; while with *Decomposition* + *Random Allocation*, there tends to be an uneven distribution of robots that leaves certain sub-regions relatively unexplored. As a result, both of them leave a considerable amount of area uncovered. Our framework outperforms these other approaches in this setting, demonstrating the effectiveness of both the decomposition and the generative allocation.

	Coverage Cost ($\times 10^{-2}$) $\downarrow$		
Methods	Deploy Cost ≤ 15	Deploy Cost ≤ 30	Deploy Cost ≤ 45
No Decomposition	0.365 ± 0.076	0.153 ± 0.035	0.056 ± 0.037
Dec + RanA	0.732 ± 0.231	0.401 ± 0.144	0.062 ± 0.015
Dec + EG-GAN (ours)	**0.049** ± 0.006	**0.018** ± 0.004	**0.007** ± 0.001

Table 5.5: We studied the effect of decomposition and allocation on the coverage performance under different deployment cost budgets. $\downarrow$ indicates that a lower value is better, and $\uparrow$ that higher is better. RanA stands for Random Allocation, EG-GAN stands for Generative Allocation using Evolution-guided GAN. Each value is averaged over 100 test terrains. Our framework outperforms the other tested methods under all three deployment cost limits, demonstrating the effectiveness of both the decomposition and the generative allocation.

Part III

Robot Composition with Arbitrary Connectivity via Grammar-guided Latent Space Optimization

Chapter 6

OVERVIEW

In the previous chapter, we primarily focus on relatively simple composition space (e.g. modular robot with fixed topology). However, real world robot composition problem can involve high-dimensional and complex composition space (e.g. modular robot with arbitrary topology), which makes search and optimization in the composition hard. Classic design automation approaches resort to discrete black-box optimization techniques such as Genetic Algorithms (GA)[29], Genetic Programming (GP) [48], and Random Graph Search (RGS)[72]. While these methods typically work well when the objective function is inexpensive to evaluate (e.g. obtaining the locomotion speed of a simple robot in simulation with a pre-defined controller), they require a large number of objective function evaluations and are not suitable for situations where evaluation is expensive (e.g. through real-world evaluation, or creating customized dynamic motion plans for each new design). Recent works utilize graph grammar rules to confine the search space [83, 29], such that the number of evaluations can be reduced. However, they still operate in a high-dimensional combinatorial search space and require a considerable number of sample evaluations.

In this work, we introduce Grammar-guided Latent Space Optimization (GLSO), a framework for sample-efficient robot composition automation. Given a robot composition space defined by the possible combinations of a set of discrete primitive components, GLSO first learns a low-dimensional, continuous representation of the robot composition space through unsupervised learning. The learned representation allows us to then convert the combinatorial composition automation search into a continuous optimization problem, where we apply sample-efficient Bayesian Optimization (BO) [59] to search in the latent space for high-performing composition. GLSO uses a graph variational auto-encoder (VAE) to learn mappings between the composition space and latent space. Importantly, the learned latent space can be used to optimize compositions for multiple different tasks without the need for retraining.

GLSO is inspired by recent advancements in molecule synthesis [43, 36]. However, unlike molecule optimization, where existing large-scale datasets such as ZINC [35]

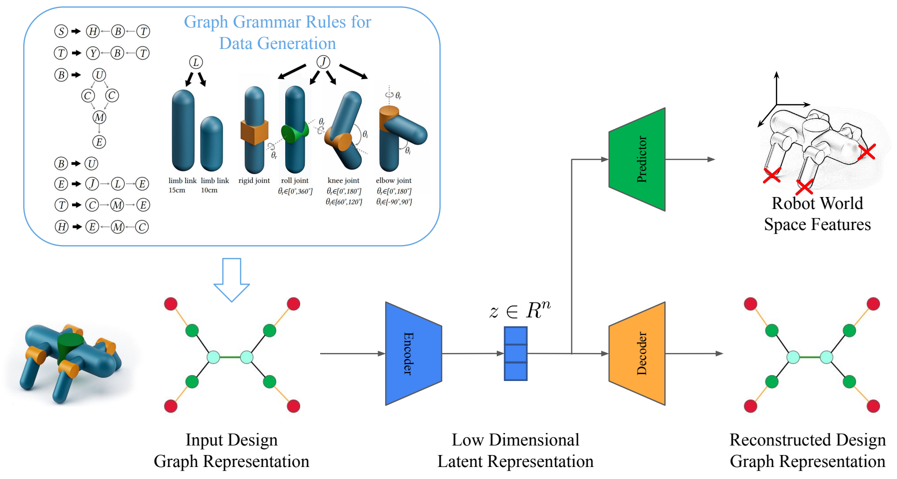

Graph Grammar Rules for Data Generation
limb link 15cm
limb link 10cm
rigid joint
roll joint
knee joint
elbow joint
Input Design Graph Representation
Encoder
$z \in R^n$
Low Dimensional Latent Representation
Predictor
Decoder
Robot World Space Features
Reconstructed Design Graph Representation

Figure 6.1: Our framework begins by collecting a dataset of robot designs based on a set of graph grammar rules, as shown on the top left. This process of enumerating designs is computationally inexpensive, as no controller is needed. The collected data (example on bottom left) is subsequently used to train a graph variational autoencoder (VAE), which defines a mapping between a low dimensional continuous latent space and the combinatorial design space. A property predictor (shown as the green trapezoid) is simultaneously trained to predict the world space features grounding of the robots from the latent vector, in order to encourage physically similar robots to be grouped together in the latent space. After the VAE is trained, optimization can be performed in the latent space in search of high-performing designs. This VAE can further be used for multiple distinct tasks without the need for retraining.

are readily available to supervise training, the domain of robot composition has no such dataset available. Instead, GLSO generates training data by leveraging graph grammar rules [83], which confine the search space and implicitly inject a prior into our learned latent representation.

An additional challenge associated with robot composition automation is the potentially “chaotic” objective function, i.e. that two structurally similar robot compositions may have very different performance for a given task. For example, consider the designs shown in Fig. 7.1b: they have nearly the same design graph, but they all have drastically different locomotive performance. We address this problem by including an additional objective in our VAE training, which ensures that neighboring compositions in the latent space not only have similar graphs, but also share similar world space features, such as the bounding box of a robot or the end-effector workspace. That is, in addition to the conventional VAE encoder/decoder reconstruction loss, GLSO simultaneously trains a neural network to learn world space feature grounding the robot compositions, which we called the property prediction network. The complete pipeline can be seen in Fig. 6.1 [1].

We evaluate GLSO by designing robots for a set of locomotion tasks, with a component library that includes different joints and links with various rotational angles and axes, sizes, and weights. Our method outperforms state-of-the-art robot design automation methods, consistently identifying higher-performing designs when given the same number of sample evaluations.

[1]Upper left portion of the figure adopted from [81] with author's approval

Chapter 7

GRAMMAR-GUIDED LATENT SPACE OPTIMIZATION

7.1 GLSO: Learning Latent Composition Representation

GSLO trains a generative model that defines a mapping between a continuous latent space and the discrete composition space. Similar to previous LSO works, our method uses a VAE as a generative model. The VAE consists of an encoder (section 7.1) and a decoder (section 7.1). In addition to the encoder and decoder, we co-train a property prediction network (section 7.1) to predict world space features of the compositions. The goal of co-training the property predictor is to encourage compositions with similar physical properties to be close together in the latent space; we find that in turn, this makes optimization in the latent space more efficient. The training data for the VAE is generated through recursively applying a set of graph grammar rules (section 7.1). The grammar ensures that the training compositions are valid, which implicitly bias the mapping from latent space to composition space towards promising compositions. Importantly, the trained VAE can be reused to search for compositions specialized to multiple different tasks without retraining.

Notation. In this work, we define a composition as a modular robot design, and use design and composition interchangeably. Each design is represented as an acyclic graph $\mathcal{G} = (V, E)$ where V corresponds to the hardware components (nodes) and E the connectivity between them (edges). Note that an acyclic design graph can also be viewed as a tree, where the root node corresponds to the one component on the body, and the each of the leaf nodes correspond to the robot's end-effectors. We use i, j, k to refer to nodes indices, and define $N(i)$ as the neighbor of a node i. We denote the sigmoid function as $\sigma(\cdot)$ and the ReLU function as $\tau(\cdot)$, and trainable weights in the models as W and U.

Data Collection via Graph Grammar

For a given set of robot hardware components, GSLO begins by collecting a dataset of robots composed of these hardware components. This dataset will then be used to train the graph VAE. We desire a dataset that covers much of the design space, while containing few invalid designs.

One key idea behind GSLO is that graph grammar rules can serve as a guiding tool to generate a large dataset of designs, due to their ability to filter out invalid designs and to produce a tractable and meaningful subspace of designs. Specifically, we adopt the set of grammar rules proposed in RoboGrammar [83]. To generate a random design using RoboGrammar, we start from the start symbol "S" and randomly apply valid grammar rules until the design graph consists of only terminal nodes. Each terminal node corresponds to a specific hardware component, and therefore each design graph has a one-to-one correspondence with a physical robot. We additionally collect each design's contact locations when that design is placed at rest on flat ground in simulation. These contact locations will be used to train the property predictor in section 7.1. Since the data collection process does not involve creating controllers, it can be readily applied to a new set of grammar rules / hardware components. In our experiments, the data collection process produces around $5e^5$ designs on a 8-core desktop computer in approximately four hours.

Robot Encoder

Since the robots are represented as graphs, we use a graph autoencoder to encode the robots via a graph message passing neural network (MPNN)[13, 36]. MPNNs operate on graph-structured inputs, where hidden representations of the graph nodes are updated iteratively via messages sent along graph edges [79]. Each node v_i contains a one-hot feature vector x_i indicating its type. We denote a message from a node v_i to v_j as $m_{i,j}$, and $m_{j,i}$ vice versa.

At each message passing iteration, messages are updated as:

$$m_{i,j} = \mathrm{GRU}(x_i, \{m_{ki}\}_{k \in N(i)/j}) \tag{7.1}$$

where GRU refers to Gated Recurrent Unit [9] adapted for graph message passing. The GRU formula is shown below, where W and U refer to trainable weights, s, z and r are internal variables, and m refers to the messages.

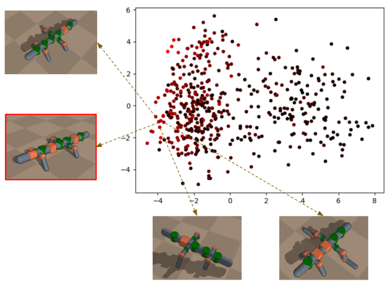
$$s_{ij} = \sum_{k \in N(i)/j} m_{ki} \tag{7.2}$$

$$z_{ij} = \sigma(W^z x_i + U^z s_{ij} + b^z) \tag{7.3}$$

$$r_{ki} = \sigma(W^r x_i + U^r m_{ki} + b^r) \tag{7.4}$$

$$\tilde{m}_{ij} = tanh(W x_i + U \sum_{k \in N(i)/j} r_{ki} \bigodot m_{ki}) \tag{7.5}$$

$$m_{ij} = (1 - z_{ij}) \bigodot s_{ij} + z_{ij} \bigodot \tilde{m}_{ij} \tag{7.6}$$


(a) Latent Space with PPN

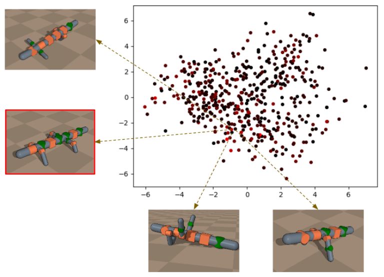

(b) Latent Space without PPN

Figure 7.1: Visualization of two different spaces (a) when the VAE is co-trained with the property predictor, and (b) without the property predictor. Here we project the latent points into a two-dimensional space, for visualization purposes, by taking the first two principal components. The color of the points correspond to its performance for a given task (here, traversing flat terrain), where a darker color denotes worse performance. We additionally show designs neighboring the same robot (with red border) in the two different latent spaces. We can see that the addition of the property predictor implicitly encourages designs with similar performance to be grouped together, which we find results in more efficient optimization.

After t iterations of message passing, we obtain the latent representation of each node by aggregating its inward messages,

$$h_i = \tau(W^e x_i + \sum_{k \in N(i)} U^e m_{ki}). \tag{7.7}$$

The robot graph representation is calculated as the sum of the latent representation of the leaf nodes,

$$h_{\mathcal{G}} = \sum h_{\text{leaf}}. \tag{7.8}$$

Finally, the mean $\mu_{\mathcal{G}}$ and the log variance $\sigma_{\mathcal{G}}$ of the variational posterior approximation are computed from $h_{\mathcal{G}}$ by applying two separate affine layers, where the latent vector $z_{\mathcal{G}}$ is sampled from a Gaussian distribution $z_{\mathcal{G}} \sim \mathcal{N}(\mu_{\mathcal{G}}, \sigma_{\mathcal{G}})$

Robot Decoder

The robot decoder maps the continuous latent vector $z_{\mathcal{G}}$ back to a robot design tree $\mathcal{G}$ by sequentially adding nodes to a partial design tree in depth-first order, starting from the root. For every visited node, the decoder first predicts whether it has children to be generated. If so, a new node is created and attached to the current node, with its node type predicted by the decoder. This procedure is recursively applied to the newly created nodes until the current node has no more children to generate, where the decoder backtracks to its parent node and repeat this process.

The decoder makes predictions based on message propagation in the current partial tree at each time step. The messages are propagated using the same GRU structure as in the encoder. The decision of whether to create new node at each time step is predicted as a probability p_t based on the inward messages $h_{k,i}$, latent vector $z_{\mathcal{G}}$ and the node features x_i,

$$p_t = \sigma(u^c \cdot \tau(W_1^c x_{i_t} + W_2^c z_{\mathcal{G}} + W_3^c \sum_k h_{k,i_t})). \tag{7.9}$$

For a new node j created from parent i, the label q_j is predicted as,

$$q_j = \text{softmax}(U^l \tau(W_1^l z_{\mathcal{G}} + W_2^l h_{i,j})). \tag{7.10}$$

The decoder is trained to maximize the reconstruction likelihood $p(\mathcal{G}|z_{\mathcal{G}})$. Let $\hat{p}_t$ and $\hat{q}_j$ be the ground truth values of p_t and q_j respectively, the decoder loss can be written as:

$$\mathcal{L}_d(\mathcal{G}) = \sum_t \mathcal{L}_c(p_t, \hat{p}_t) + \sum_j \mathcal{L}_l(q_j, \hat{q}_j) \tag{7.11}$$

Where $\mathcal{L}_c$ and $\mathcal{L}_l$ are cross-entropy losses. Similar to language generation, we apply *teacher forcing* [78] during training, where ground truth topology and labels are used at each step for prediction.

Property Predictor

We hypothesize that a learned latent space in which designs with similar capabilities are close together, and which contains few invalid designs, will allow for more efficient optimization. To obtain such a latent space, we propose to co-train a property prediction network (PPN) with the VAE. The PPN maps the mean of the variational encoding distribution $\mu_{\mathcal{G}}$ to the corresponding robot's world space features, which implicitly encourages robots with similar world space features to have similar latent representation. In this work, since we are primarily experimenting with locomotion tasks, we define the world space features as a vector v_{contact} consisting of the robot's 2D contact locations on flat ground. The contact locations are sorted based on their x values and zero padded to a fixed length to form v_{contact}. v_{contact} is collected together with the robot data as described in section 7.1. The PPN is trained to minimize the mean square error between the predicted contact vector $\hat{v}_{\text{contact}}$ and the ground truth contact vector v_{contact}, i.e. $\mathcal{L}_{PPN} = ||\hat{v}_{\text{contact}} - v_{\text{contact}}||^2$.

Training:

The final loss of the graph VAE is a weighted sum of the decoder loss $\mathcal{L}_d$, the PPN loss $\mathcal{L}_{PPN}$, and the kl-divergence $\mathcal{L}_{kl}$ between the variational posterior and the prior latent distribution $\mathcal{N}(0, I)$:

$$\mathcal{L}_{vae} = \mathcal{L}_d + \mathcal{L}_{kl} + \lambda \mathcal{L}_{PPN} \tag{7.12}$$

We minimize $\mathcal{L}_{vae}$ using gradient descent. We use a learning rate of 1e-3, which is decayed every 40000 training steps at a rate of 0.9. We optimize the trainable weights using Adam [40], while applying a gradient clipping of magnitude 50. We trained the model for 400000 steps with a batch size of 32 to obtain the reported results. Notice that we anneal the weight of KL loss from zero to one during training, which helps ensure that the encoder does not start by pushing the KL loss to zero, as noted in [4].

Training took around five hours on a 8-core desktop computer with a NVIDIA GTX 1070 graphics card.

7.2 GLSO: Optimization in the Learned Latent Space

The trained VAE associates each robot design with a latent vector, given by the mean of the variational encoding distribution $\mu_{\mathcal{G}}$. We can use the VAE to transform robot design automation from a combinatorial optimization problem into a continuous one. We then apply Bayesian Optimization to search for the latent vector where the associated robot design has the highest performance.

Controller & Evaluation

To evaluate the performance of a given robot structure, we need to derive its controls. This reveals one challenge in robot composition optimization– hidden inside each evaluation of the composition optimization objective function is a trajectory or policy optimization problem, which itself is often computationally expensive. In this work, we utilized model predictive control (MPC) [21] to create controllers for different robot structures across different terrains. MPC predicts future behaviour using a model of the system, optimizes controls for a finite horizon, executes a small number of the optimized control, and then replans. Specifically, we used model predictive path integral control (MPPI) [77], a sampling based MPC method for controlling the generated robot designs. Our implementation follows [83]. Each robot is controlled in a simulation implemented using the Bullet Physics library [10]. We adopt the same objective function as in [83], where robots are rewarded for forward progression while maintaining its initial orientation. Computing the trajectory of each robot takes 30 - 60s on a 8-core computer, and is the primary computational bottleneck of the design optimization.

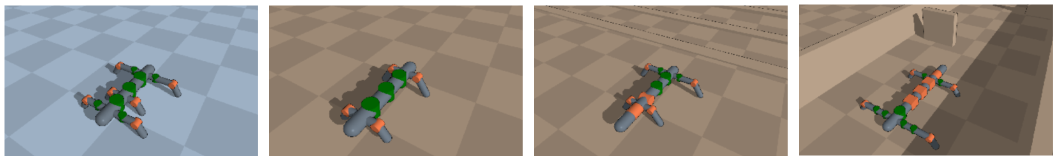

(a) Frozen Lake (b) Flat Terrain (c) Ridged Terrain (d) Wall Terrain

Figure 7.2: We evaluate our framework on a set of locomotion tasks as shown in the figures. The above images showcase the best design identified by our method for four tasks.

Bayesian Optimization in Latent Space

We use Bayesian Optimization (BO) [59], a sample-efficient black-box optimizer, to perform optimization in the latent space. Specifically, our BO implementation uses Gaussian Process (GP) [65] to build a model of the objective function, which includes both the current estimation of the function and the uncertainty around the estimation. We then use expected improvement (EI) as the acquisition function to determine the optimal point to sample next. The sampled point is subsequently evaluated and used to update the GP model. This procedure is repeated until we reach the computation limit, defined as the maximum number of objective function evaluations. We additionally apply domain reduction [68], where the bounds of the latent space are contracted during the optimization to reduce oscillation. Our Bayesian Optimization (BO) implementation begins with 50 steps of random exploration, which helps diversify the exploration space. Subsequent sampling points are determined by the Expected Improvement (EI) acquisition function. Every 10 steps, a random point is sampled to explore. We used an optimization bound of $[-3, 3]$ across all latent dimensions. Our EI acquisition function has $\xi = 0.01$, which controls the exploration rate. Our Gaussian Process uses a Matern kernel with $\nu = 2.5$. We additionally apply domain reduction, with shrinkage parameter $\gamma_{\text{osc}} = 0.7$, panning parameter $\gamma_{\text{pan}} = 1.0$, zoom parameter $\eta = 0.9$. We used [58] for our BO implementation.

Chapter 8

RESULTS & EXPERIMENTS

We evaluate our method by generating designs for the following four locomotion tasks, each defined by its corresponding terrain:

- **Flat Terrain**: A flat plane with a friction coefficient of 0.9.
- **Frozen Lake Terrain**: A flat plane with a low friction coefficient of 0.05.
- **Ridged Terrain**: Includes hurdles that the robots must jump or crawl over.
- **Wall Terrain**: Includes high walls placed in a slalom-like manner.

The reward of each robot is measured as described in section 7.2. We report comparisons to previous approaches as well as ablated versions of our approach. For each each task, we allow each method a maximum of 500 objective function evaluations, i.e., control and trajectory optimization for up to 500 designs. Images of the tasks and optimized designs are shown in Fig. 7.2.

8.1 Comparisons and Baselines

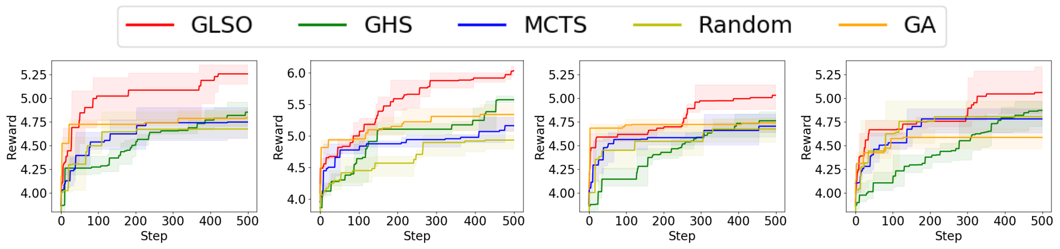
To demonstrate the effectiveness of GLSO, we benchmark our method against the following:

Random Search: Random designs are generated using the graph grammar rules.

Monte Carlo Tree Search (MCTS): A baseline adopted from [83], performing Monte Carlo Tree Search (MCTS) [5] on a search tree defined by the graph grammar rules.

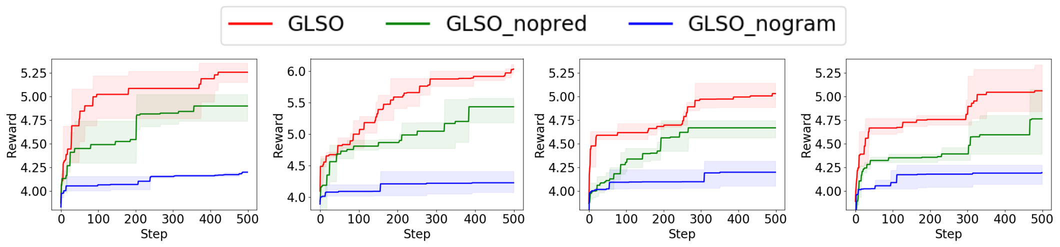
Graph Heuristic Search (GHS): A design automation method proposed in [83]. GHS performs search on the same graph grammar search tree as does MCTS. Our implementation follows [83].

Genetic Algorithm (GA): Our implementation of Genetic Algorithm (GA) follows [72], where graph mutation with uncertainty is used to mutate the population at each iteration. We used a population size of 50 and evolved them until the number of evaluation limit was reached. Note that unlike the other comparison methods, GA does not operate in the grammar space, as crossover and mutation operations are not clearly defined for the graph grammar proposed by [83].



(a) Frozen Lake (b) Flat Terrain (c) Ridged Terrain (d) Wall Terrain

Figure 8.1: Comparison against related methods. The solid line shows average over 3 different random seeds, and the error band represents the maximum and minimum. In each task, GLSO produces higher-reward designs within 500 steps (design samples).



(a) Frozen Lake (b) Flat Terrain (c) Ridged Terrain (d) Wall Terrain

Figure 8.2: Comparison of GLSO with ablated variants. The solid line shows average over 3 different random seeds, and the error band represents the maximum and minimum. These results show that both the graph grammar and the property predictor are important to GLSO.

The optimization curves for each of the tested algorithms is shown in Fig. 8.1. We found our method outperform all comparison methods and baselines.

8.2 Ablation Studies.

We performed ablation studies to investigate the effects of the graph grammar rules and the property prediction network. Results are shown in Fig. 8.2. In the "GLSO_nogram" test, we removed the graph grammar rules during data collection, and the VAE training set is created through random generation of topology and node labels. In the "GLSO_nopred" test, we removed the property prediction network during VAE training. We found that both the graph grammar and property predictor are crucial to the success of GLSO. Additionally, we created a visualization of how the property prediction network influences the latent space. Fig. 7.1 presents points in the latent space created by training the VAE with and without the property prediction.

Part IV

Conclusion and Future Work

Chapter 9

CONCLUSION

In this thesis, we presented our works that improve the robustness, sample-efficiency and quality of existing robot composition automation algorithms. To robustly generate robot composition in real-time, we proposed an evolution-guided generative adversarial network which learns a one-to-many mapping from task to designs. By combining ideas from both ML and EA, our method can quickly provide multiple suitable composition candidates for a given task, and could be extended to multi-objective settings. One limitation of our method, shared by other ML methods, is the assumption that new tasks will be from the same distribution as those seen during training, such that the outputs from out-of-distribution tasks may be poor. Secondly, we only experiment with two objectives in our work. It would be interesting to see if EG-GAN will scale to higher dimensional objectives spaces.

To efficiently search in complex robot composition space, we proposed GLSO which transforms the composition search space into low dimensional latent space. We believe that our work is general to the domain of robot composition, and represents a step towards achieving efficient robot composition automation, in particular when evaluation of each composition is computationally expensive. An important limitation of our approach is the requirement for a pre-defined set of graph grammar rules as well as world space features. Extending this work to different sets of hardware components and tasks will likely take some expert knowledge or domain intuition. A second limitation of this work is that we restricted our design space to only acyclic graphs. Allowing cycles in the design graphs is theoretically viable as shown in the recent progress on graph generation [47], but would require a new set of grammar rules and likely additional computation.

Chapter 10

FUTURE WORKS

While our works can potentially be extended in lots of different directions, we present three future works that we believe are most crucial and fundamentally impactful. We refer the reader to the conclusion sections of our publications [33, 32, 31] for more domain-specific future works.

10.1 Expeditionary structural composition

An exciting extension of our work is to go beyond robot composition and see if they can also be applied to other composition problems. We are particularly interested in applying GLSO to expeditionary structural composition, where the goal is to quickly set up field hospital using modular components. We believe that GLSO can be useful in determining the suitable field hospital structure, for more efficient deployment.

10.2 Real-time generation of composition with arbitrary connectivity

An interesting future direction is to use the latent representation of the compositions learned in GLSO to help with fast composition generation. For example, evolution-guided generative adversarial network may benefit from learning to output a continuous latent vector instead of a complete composition. This would potentially provide a way to output high-dimensional composition in real time.

10.3 Determine when to re-compose

In this thesis, we primarily focus on the problem of “how to compose”. However, a parallel problem that is also extremely interesting is “when to compose / re-compose”. This would likely require us to reason about the re-composition cost versus the potential gain we can obtain, and would be an important step towards the full automation of modular / multi-agent robotic systems.

BIBLIOGRAPHY

[1] Ian Abraham, Anastasia Mavrommati, and Todd Murphey. “Data-Driven Measurement Models for Active Localization in Sparse Environments”. In: *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania, June 2018. DOI: `10.15607/RSS.2018.XIV.045`.

[2] Abien Fred Agarap. “Deep learning using rectified linear units (relu)”. In: *arXiv preprint arXiv:1803.08375* (2018).

[3] Elif Ayvali et al. “Utility-guided palpation for locating tissue abnormalities”. In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 864–871.

[4] Samuel R Bowman et al. “Generating sentences from a continuous space”. In: *arXiv preprint arXiv:1511.06349* (2015).

[5] Cameron B Browne et al. “A survey of monte carlo tree search methods”. In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012), pp. 1–43.

[6] Noe Casas. “Genetic algorithms for multimodal optimization: a review”. In: *arXiv preprint arXiv:1508.05342* (2015).

[7] Nick Cheney et al. “Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding”. In: *ACM SIGEVOlution* 7.1 (2014), pp. 11–23.

[8] O. Chocron and P. Bidaud. “Evolutionary algorithms in kinematic design of robotic systems”. In: *Proc. of the 1997 IEEE/RSJ Int’l Conf.on Intelligent Robots and Systems*. Vol. 2. 1997, pp. 1111–1117. DOI: `10.1109/IROS.1997.655148`.

[9] Junyoung Chung et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (2014).

[10] Erwin Coumans. “Bullet physics simulation”. In: *ACM SIGGRAPH 2015 Courses*. 2015, p. 1.

[11] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. `http://pybullet.org`. 2016–2019.

[12] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. “Exploration and exploitation in evolutionary algorithms: A survey”. In: *ACM computing surveys (CSUR)* 45.3 (2013), pp. 1–33.

[13] Hanjun Dai, Bo Dai, and Le Song. “Discriminative embeddings of latent variable models for structured data”. In: *Int. Conf. on machine learning*. PMLR. 2016, pp. 2702–2711.

[14] Hanjun Dai et al. “Syntax-directed variational autoencoder for structured data”. In: *arXiv preprint arXiv:1802.08786* (2018).

[15] Mehdi Dastani, Farhad Arbab, and Frank De Boer. “Coordination and composition in multi-agent systems”. In: *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. 2005, pp. 439–446.

[16] Nicola De Cao and Thomas Kipf. “MolGAN: An implicit generative model for small molecular graphs”. In: *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models* (2018).

[17] Kalyanmoy Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197.

[18] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. “Ant colony optimization”. In: *IEEE computational intelligence magazine* 1.4 (2006), pp. 28–39.

[19] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. “Neural architecture search: A survey”. In: *The Journal of Machine Learning Research* 20.1 (2019), pp. 1997–2017.

[20] Daniel C Elton et al. “Deep learning for molecular design—a review of the state of the art”. In: *Molecular Systems Design & Engineering* 4.4 (2019), pp. 828–849.

[21] Carlos E Garcia, David M Prett, and Manfred Morari. “Model predictive control: Theory and practice—A survey”. In: *Automatica* 25.3 (1989), pp. 335–348.

[22] Rafael Gómez-Bombarelli et al. “Automatic chemical design using a data-driven continuous representation of molecules”. In: *ACS central science* 4.2 (2018), pp. 268–276.

[23] Ian J Goodfellow et al. “Generative Adversarial Nets”. In: *Neural Information Processing Systems*. 2014.

[24] Gabriel Lima Guimaraes et al. “Objective-reinforced generative adversarial networks (ORGAN) for sequence generation models”. In: *arXiv preprint arXiv:1705.10843* (2017).

[25] Huy Ha, Shubham Agrawal, and Shuran Song. “Fit2Form: 3D Generative Model for Robot Gripper Form Design”. In: *Conference on Robotic Learning (CoRL)*. 2020.

[26] *Hebi Robotics*. 2020. URL: `www.hebirobotics.com`.

[27] R Devon Hjelm et al. “Boundary-seeking generative adversarial networks”. In: *International Conference on Learning Representations*. 2018.

[28] G. S. Hornby, H. Lipson, and J. B. Pollack. "Evolution of generative design systems for modular physical robots". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 4. 2001, 4146–4151 vol.4. DOI: `10.1109/ROBOT.2001.933266`.

[29] Gregory S Hornby, Hod Lipson, and Jordan B Pollack. "Evolution of generative design systems for modular physical robots". In: *Proceedings 2001 ICRA. IEEE Int. Conf. on Robotics and Automation (Cat. No. 01CH37164)*. Vol. 4. IEEE. 2001, pp. 4146–4151.

[30] Gregory S Hornby, Hod Lipson, and Jordan B Pollack. "Generative representations for the automated design of modular physical robots". In: *IEEE transactions on Robotics and Automation* 19.4 (2003), pp. 703–719.

[31] Jiaheng Hu, Julian Whitman, and Howie Choset. "GLSO: Grammar-guided Latent Space Optimization for Sample-efficient Robot Design Automation". In: *under review*. 2022.

[32] Jiaheng Hu et al. "Large-scale Heterogeneous Multi-Robot Coverage via Domain Decomposition and Generative Allocation". In: *International Workshop on the Algorithmic Foundations of Robotics*. 2022.

[33] Jiaheng Hu et al. "Modular Robot Design Optimization with Generative Adversarial Networks". In: *Proceedings 2022 ICRA. IEEE Int. Conf. on Robotics and Automation*. 2022.

[34] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[35] John J Irwin et al. "ZINC: a free tool to discover chemistry for biology". In: *Journal of chemical information and modeling* 52.7 (2012), pp. 1757–1768.

[36] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. "Junction tree variational autoencoder for molecular graph generation". In: *Int. Conf. on machine learning*. PMLR. 2018, pp. 2323–2332.

[37] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. "A review on genetic algorithm: past, present, and future". In: *Multimedia Tools and Applications* (2020), pp. 1–36.

[38] James Kennedy and Russell Eberhart. "Particle swarm optimization". In: *Proceedings of ICNN'95-international conference on neural networks*. Vol. 4. IEEE. 1995, pp. 1942–1948.

[39] J Taery Kim et al. "Learning Robot Structure and Motion Embeddings using Graph Neural Networks". In: *arXiv preprint arXiv:2109.07543* (2021).

[40] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[41] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[42] Eric Klavins, Robert Ghrist, and David Lipsky. "Graph grammars for self assembling robotic systems". In: *IEEE Int. Conf. on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*. Vol. 5. IEEE. 2004, pp. 5293–5300.

[43] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. "Grammar variational autoencoder". In: *Int. Conf. on machine learning*. PMLR. 2017, pp. 1945–1954.

[44] Manfred Lau et al. "Converting 3D furniture models to fabricatable parts and connectors". In: *ACM Transactions on Graphics (TOG)* 30.4 (2011), pp. 1–6.

[45] Miqing Li, Tao Chen, and Xin Yao. "How to Evaluate Solutions in Pareto-based Search-Based Software Engineering? A Critical Review and Methodological Guidance". In: *IEEE Transactions on Software Engineering* (2020), pp. 1–1. ISSN: 2326-3881. DOI: `10.1109/tse.2020.3036108`. URL: `http://dx.doi.org/10.1109/TSE.2020.3036108`.

[46] Xiaodong Li, Andries Engelbrecht, and Michael G Epitropakis. "Benchmark functions for CEC'2013 special session and competition on niching methods for multimodal function optimization". In: *RMIT University, Evolutionary Computation and Machine Learning Group, Australia, Tech. Rep* (2013).

[47] Yujia Li et al. "Learning deep generative models of graphs". In: *Proceedings of the 35th International Conference on Machine Learning (PMLR)*. 2018.

[48] Hod Lipson. "How to draw a straight line using a GP: Benchmarking evolutionary design against 19th century kinematic synthesis". In: *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference, Seattle, Washington, USA*. Vol. 26. 2004.

[49] Chun Liu and Andreas Kroll. "A centralized multi-robot task allocation for industrial plant inspection by using a* and genetic algorithms". In: *Int. Conference on Artificial Intelligence and Soft Computing*. Springer. 2012, pp. 466–474.

[50] Yang Liu et al. "On centroidal Voronoi tessellation—energy smoothness and fast computation". In: *ACM Transactions on Graphics (ToG)* 28.4 (2009), pp. 1–17.

[51] Alexandro López-González et al. "Multi robot distance based formation using Parallel Genetic Algorithm". In: *Applied Soft Computing* 86 (2020), p. 105929.

[52] George Mathew and Igor Mezić. "Metrics for ergodicity and design of ergodic dynamics for multi-agent systems". In: *Physica D: Nonlinear Phenomena* 240.4-5 (2011), pp. 432–442.

[53] Anastasia Mavrommati et al. “Real-time area coverage and target localization using receding-horizon ergodic exploration”. In: *IEEE Transactions on Robotics* 34.1 (2017), pp. 62–80.

[54] Ole J Mengshoel and David E Goldberg. “The crowding approach to niching in genetic algorithms”. In: *Evolutionary computation* 16.3 (2008), pp. 315–354.

[55] Lauren M Miller et al. “Ergodic exploration of distributed information”. In: *IEEE Transactions on Robotics* 32.1 (2015), pp. 36–52.

[56] Mehdi Mirza and Simon Osindero. “Conditional generative adversarial nets”. In: *arXiv preprint arXiv:1411.1784* (2014).

[57] Carla Mouradian et al. “A coalition formation algorithm for multi-robot task allocation in large-scale natural disasters”. In: *2017 13th Int. Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE. 2017, pp. 1909–1914.

[58] Fernando Nogueira. *Bayesian Optimization: Open source constrained global optimization tool for Python*. 2014. URL: `https://github.com/fmfn/BayesianOptimization`.

[59] Martin Pelikan, David E Goldberg, Erick Cantú-Paz, et al. “BOA: The Bayesian optimization algorithm”. In: *Proceedings of the genetic and evolutionary computation conference GECCO-99*. Vol. 1. Citeseer. 1999, pp. 525–532.

[60] Nicholas Pilkington, Stacey Svetlichnaya, and Tom Holmes. “GitHub - dronedeploy/ddml-segmentation-benchmark: DroneDeploy Machine Learning Segmentation Benchmark”. In: (2019).

[61] Kenneth V Price. “Differential evolution”. In: *Handbook of optimization*. Springer, 2013, pp. 187–214.

[62] Amanda Prorok, M. Ani Hsieh, and Vijay Kumar. “Fast Redistribution of a Swarm of Heterogeneous Robots”. In: *Proceedings of the 9th EAI Int. Conference on Bio-Inspired Information and Communications Technologies (Formerly BIONETICS)*. BICT’15. New York City, United States, 2016, pp. 249–255.

[63] Amit Rauniyar and Pranab K Muhuri. “Multi-Robot Coalition Formation and Task Allocation Using Immigrant Based Adaptive Genetic Algorithms”. In: *Computational Intelligence in Emerging Technologies for Engineering Applications*. Springer, 2020, pp. 205–225.

[64] Amit Rauniyar and Pranab K Muhuri. “Multi-robot coalition formation problem: Task allocation with adaptive immigrants based genetic algorithms”. In: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2016, pp. 000137–000142.

[65] Eric Schulz, Maarten Speekenbrink, and Andreas Krause. “A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions”. In: *Journal of Mathematical Psychology* (2018).

[66] Onn Shehory and Sarit Kraus. “Methods for task allocation via agent coalition formation”. In: *Artificial intelligence* 101.1-2 (1998), pp. 165–200.

[67] Brian Smith et al. “Multi-robot deployment and coordination with embedded graph grammars”. In: *Autonomous Robots* 26.1 (2009), pp. 79–98.

[68] Nielen Stander and KJ Craig. “On the robustness of a simple domain reduction scheme for simulation-based optimization”. In: *Engineering Computations* (2002).

[69] Tarik Tosun et al. “Computer-aided compositional design and verification for modular robots”. In: *Robotics Research*. Springer, 2018, pp. 237–252.

[70] Austin Tripp, Erik Daxberger, and José Miguel Hernández-Lobato. “Sample-efficient optimization in the latent space of deep generative models via weighted retraining”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 11259–11272.

[71] David A Van Veldhuizen and Gary B Lamont. “Multiobjective evolutionary algorithms: Analyzing the state-of-the-art”. In: *Evolutionary computation* 8.2 (2000), pp. 125–147.

[72] Tingwu Wang et al. “Neural Graph Evolution: Towards Efficient Automatic Robot Design”. In: *International Conference on Learning Representations*. 2018.

[73] Changyun Wei, Ze Ji, and Boliang Cai. “Particle swarm optimization for cooperative multi-robot task allocation: a multi-objective approach”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2530–2537.

[74] Julian Whitman, Matthew Travers, and Howie Choset. *Learning Modular Robot Control Policies*. 2021. arXiv: `2105.10049 [cs.RO]`.

[75] Julian Whitman, Matthew Travers, and Howie Choset. “Modular mobile robot design selection with deep reinforcement learning”. In: *Conference on Neural Information Processing Systems. NeurIPS Workshop on ML for engineering modeling, simulation and design*. 2020.

[76] Julian Whitman et al. “Modular robot design synthesis with deep reinforcement learning”. In: *Proc. of the AAAI Conf. on Artificial Intelligence*. Vol. 34. 2020, pp. 10418–10425.

[77] Grady Williams et al. “Aggressive driving with model predictive path integral control”. In: *2016 Int. Conf. on Robotics and Automation*. 2016.

[78] Ronald J Williams and David Zipser. “A learning algorithm for continually running fully recurrent neural networks”. In: *Neural computation* 1.2 (1989), pp. 270–280.

[79] Zonghan Wu et al. "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[80] Bo Xu et al. "Coalition formation in multi-agent systems based on improved particle swarm optimization algorithm". In: *International Journal of Hybrid Information Technology* 8.3 (2015), pp. 1–8.

[81] Jie Xu et al. "Multi-Objective Graph Heuristic Search for Terrestrial Robot Design". In: *2021 IEEE Int. Conf. on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 9863–9869.

[82] Yusliza Yusoff, Mohd Salihin Ngadiman, and Azlan Mohd Zain. "Overview of NSGA-II for optimizing machining process parameters". In: *Procedia Engineering* 15 (2011), pp. 3978–3983.

[83] Allan Zhao et al. "Robogrammar: graph grammar for terrain-optimized robot design". In: *ACM Transactions on Graphics (TOG)* 39.6 (2020), pp. 1–16.

[84] Eckart Zitzler and Lothar Thiele. "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach". In: *IEEE transactions on Evolutionary Computation* 3.4 (1999), pp. 257–271.