Creating Foldable Polyhedral Nets Using Evolution Control

Yue Hao
George Mason University
Fairfax, VA
yhao3@gmu.edu

Yun-hyeong Kim
Seoul National University
Seoul, South Korea
yunhyk1202@gmail.com

Zhonghua Xi and Jyh-Ming Lien
George Mason University
Fairfax, VA
{zxi, jmlien}@gmu.edu

***Abstract*—Recent innovations enable robots to be manufactured using low-cost planar active material and self-folded into 3D structures. The current practice for designing such structures often uses two decoupled steps: generating an unfolding (called net) from a 3D shape, and then finding collision-free folding motion that brings the net back to the 3D shape. This raises a foldability problem, namely that there is no guarantee that continuous motion can be found in the latter step, given a net generated in the former step. Direct evaluation on the foldability of a net is nontrivial and can be computationally expensive. This paper presents a novel learning strategy that generates foldable nets using an optimized genetic-based unfolder. The proposed strategy yields a fitness function that combines the geometric and topological properties of a net to approximate the foldability. The fitness function is then optimized in an evolution control framework to generate foldable nets. The experimental results show that our new unfolder generates valid unfoldings that are easy to fold. Consequently, our approach opens a door to automate the design of more complex self-folding machines.**

I. Introduction

The concept of self-folding machines has recently inspired many innovations [1], [2], [3]. Especially for robots made from low costs 2D active material that can be folded into 3D shape using light or heat [4], [5]. However, most of these robots cannot perform complex folding motions, e.g., can only uniformly rotate their hinges [6]. Even for self-folding robots equipped with mechanical hinges [1], [7], it is highly desirable to reduce the complexity of folding motion. These limitations and constraints raise a problem: given a target 3D shape, how should one design a 2D pattern that can be rigidly folded into the desired 3D shape without stretching or self-collision by following a folding motion that is as simple as possible?

Creating unfolding of a 3D polyhedron by cutting along its edges, called edge unfolding or *polyhedral net* [8] (or simply net), is known to be computationally intractable [9]. Finding folding motion that brings the unfolded 2D state to the target 3D state rigidly (i.e., without deforming the polyhedral facets) along the uncut edges (i.e., the crease lines) is also computationally intractable, if the shape is non-convex [10]. Because of these tremendous challenges in designing foldable nets, finding a net and its folding motion are almost always addressed as two independent steps, with the exception of convex shapes. However, there are abundant examples showing that this decoupled approach fails to create foldable nets even for simple non-convex shapes. As illustrated in Fig. 1, the foldability of the nets created from a single polyhedron can vary significantly. Fig. 1 shows three nets generated from an L-shaped object. From their shapes, It is not obvious which a net is foldable, at least not until that we see the self-collisions (shown in red in Fig. 1(a)&(b)). In fact, these three nets have different levels of foldability. For example, the net shown in Fig. 1(c) is foldable by linearly interpolating the initial and target configurations, while the others are not. We call such a net: *linearly foldable*. There are also significant differences in computation cost for planning folding paths of these nets. Finding a collision-free folding motion for an arbitrary net in Fig. 1(b) takes about 30 times longer than that of the optimized nets in Fig. 1(c).

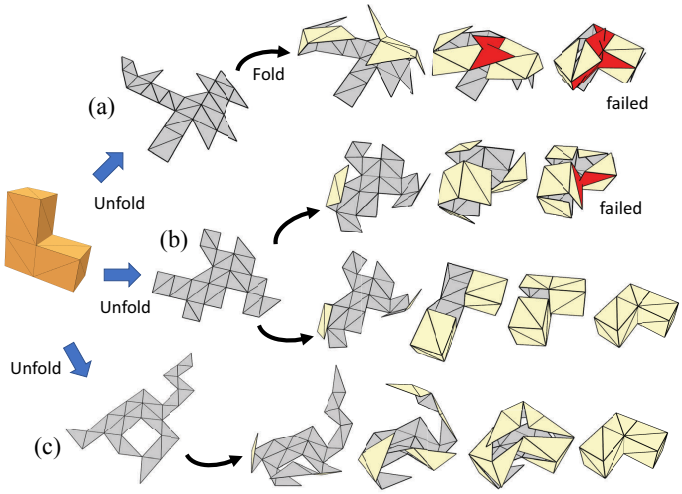


Fig. 1: Foldability of nets of an L mesh with 28 triangles. (a) A net with deep self-intersections (shown in red) throughout the path linearly interpolated between start and goal configurations (linear path). (b) Another net with shallower self-collisions along the linear path (top). This net can be folded by a motion planner that uncovers the *implicit ordering* (bottom); (c) A net generated using the proposed method is *linearly foldable*.

Challenges and Contributions. Given a polyhedron mesh with $|F|$ faces, there are $O(2^{\sqrt{|F|}})$ possible unfoldings [11]. It is simply not feasible to enumerate all of them and evaluate their foldability. In this paper, we seek to unfold 3D shapes

into polyhedral nets that can be easily folded, such as the net shown in Fig. 1(c). As to be formally defined in Section III-A, our approach can create nets that are linearly foldable and *uniformly foldable*, if such nets exist.

To our best knowledge, this is the first work that designs self-folding machines by optimizing the foldability as well as the *quality of foldability* of nets. Existing methods either generate nets without considering foldability [12], [11], [13], [14] or plan motion on given nets ignoring the fact that net structures have significant impact on foldability [15], [16], [17], [18]. The challenge of considering both net design and foldability is significant; if we are to draw some analogy from robot design, both the form and motion of the robot are unknown to us, only the target structure and constraints are given.

Inspired by the ideas of *fitness approximation* and *evolution control* [19], we propose a novel learning strategy (shown in Fig. 2) to overcome two major obstacles in both net design and robot design: large number of design parameters and expensive evaluation function. We will show that foldable net can be generated without sophisticated hypothesis models for the fitness function (in Sections IV). Our approach is computationally efficient and requires significantly fewer evaluations in each iteration. To handle complex 3D structures, we further reduce the training time via biased sampling and transfer learning (in Section V). Our experiments demonstrate that the polyhedral nets generated using our method are up to 87 times faster for the planner to fold (in Section VI). We would also like to refer the readers to [20] that describes the fabrication aspects and user studies of the proposed method.

II. Background

In this section, we briefly reviews recent developements in polyhedral unfoldings and folding motion planning. For the convenience of narration in this paper, we use *unfolding* to denote the process of creating a net without considering motion and use *folding* exclusively to address the motion aspect of the problem. When combining both unfolding and folding, it raises a designing problem: for a given shape, how one can design a foldable unfolding.

In the existing literature, this designing problem has only been addressed for convex shapes. Demaine et al. [10] presented a polynomial time algorithm that can unfold and fold all convex polyhedra without self-intersection using source unfolding [21].

A. Unfolding a Polyhedron

Given a polyhedron P, edge unfolding is a function $u(P)$ that rigidly transforms the faces of P to produce a net N by cutting along the edges of P. Finding an edge unfolding of P is equivalent to finding a spanning tree in the dual graph of P and ensuring that the flattened faces in a spanning tree do not overlap. An edge unfolding free of overlapping faces is a *net*. Finding nets of convex P is much easier than non-convex polyhedra. Effective heuristic methods [12], [11] have been developed to unfold convex polyhedra by designing edge weights so that the resulting minimum spanning tree is likely to be a net.

Finding nets of non-convex shapes is significantly more challenging. Often segmentation is needed to avoid overlapping [11]. To avoid over segmentation, genetic-based optimization methods [13], [14] evolve the unfoldings by mutating the cut edges until a net with zero overlaps is found. The evolution is controlled by a fitness function $f : N \to \mathbb{R}$ that evaluates an unfolding N. An examples of $f(N)$ [14] is defined as:

$$f(N) = -(\lambda_0 \delta_0 + \lambda_1 \delta_1) \ , \tag{1}$$

where δ_0 is the number of overlaps in the unfolding N, δ_1 is the number of hyperbolic vertices that cause local overlaps in N, and λ_0 and λ_1 are user parameters. Obviously, the value $f(N)$ of a valid net N must be 0, however, it is not guaranteed that N can be folded back to P without self-intersection [22].

B. Folding a Net

Planning folding motion of a given net is nontrivial. Early research focused mostly on folding origami [15], [16], [23], [24] that usually have many faces but low Degrees of Freedom (DoF) due to the closure constraints. On the other hand, polyhedral nets have high DoF. The motion of these nets is typically found by a motion planner that models the net as a tree-like articulated robot. For example, Song and Amato [17] proposed a PRM-based planner for nets generated from non-convex polyhedra. Recently, another motion planner has been developed by Xi and Lien [18] that focuses on finding implicit folding order of a net. This approach is proved to be more effective, especially for non-convex models. However, depending on the polyhedra nets, the planning may require long computation time even for simple shapes. For large nets, it can take hours find a path and ends up with complex motions that are not desirable for physical realization. Moreover, these approaches ignore the fact that foldability of an unfolding not only depends on its motion but also on its structure.

C. Optimal Robot Design and Evolving Robot Morphology

The problem of finding the foldable nets is closely related to the process of designing a robot. The research of optimal robot design focuses on automating the design of robot's physical morphology to improve certain general performances, e.g. dexterity, stiffness, accuracy etc., or performance on a specific application. An early research traces back to Kirdanski [25], who optimized the link lengths to boost the robot's operations near isotropic configurations. The optimization is limited and the topology is fixed as a two or three DoF serial manipulator. Later, Stocco et. al. [26] and van Henten et. al. [27] considered the robot's structure, demonstrating the decision of structures may having a larger impact on the performance than other parameters. These methods rely on quantifying certain features of robot's performance, which, in our case, is extremely difficult to obtain efficiently. Estimating the overlapping workspace of the manipulators of the folding nets is not feasible nor necessary. Lund et. al. [28] takes a different approach towards this problem. In their research,

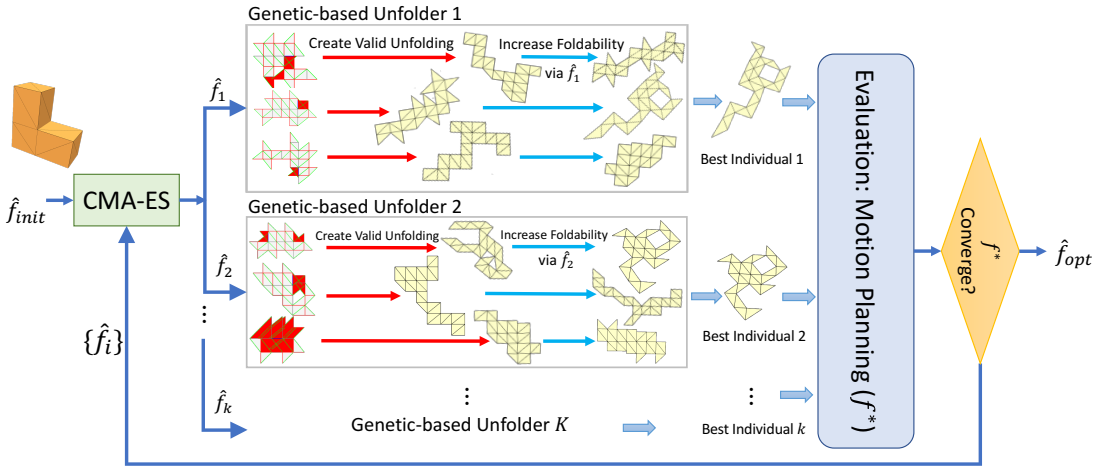


Fig. 2: Overview of the proposed learning strategy. In each iteration, multiple genetic-based unfolders running in parallel (shown in the boxes) generate polyhedral nets using the assigned fitness functions $\{\hat{f}\}$. The best individuals are then evaluated by the motion planner f^*. An external CMA-ES optimizer then updates the fitness functions $\{\hat{f}\}$ according to their performance from f^*. This process is repeated until the evaluated folding time converges.

they use genetic algorithms to encode the body parameters of robot hardware into a morphological space. These parameters, e.g. robot wheel size, are then evolved through evaluating of the robot's performance on a specific task in the simulation. However, similar to previous work, the parallel/serial structure or prismatic and rotational joints are still manually chosen, rather than free parameters.

The idea of evolving virtual creatures, originally attempted by Sims [29], where the robot morphology is represented in directed graphs. This method evolves new topologies but lacks improvements in complexity and does not scale to large parameter spaces. Some recent researches were able to evolve more complex robots using compositional pattern-producing networks [30], [31], [32]. Even with the advanced encoding methods, evolving folding robots is still computationally infeasible with constraint kinematics of folding robots. Our work seeks to combine the domain knowledge of polyhedral nets to facilitate the encoding.

III. Preliminaries

This section defines two important building blocks of our method: (1) foldability and (2) the concept of fitness approximation and evolution control.

A. Foldability of Polyhedral Nets

If a polyhedral net N is modeled as a tree-like articulated robot, the configuration space (C-space) $\mathbb{C}$ of N is the set of valid configurations $c = \{\theta_i\}$, where θ_i is the folding angle of each hinge. The free configuration space (C-free) is a subset of $\mathbb{C}$ where N has no self-intersection. For self-folding machines made of active materials, the folding motion is more constrained, as most hinges can only rotate monotonically or uniformly at the same speed. Due to a lack of coverage in the existing literature, we provide the following definitions to better describe the foldability of nets under these constraints. Note that dynamics are often not considered in the context of computational folding, and in our discussion, the folding motion is considered quasi-static.

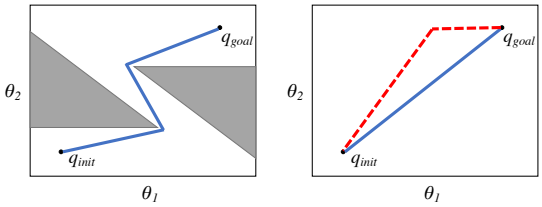


Fig. 3: Illustration of 2D C-space for different foldabilities. (left) the blue path does not satisfy monotonic foldability, as folding angle θ_1 does not monotonically increase. (right) the solid blue line shows the linear folding path and dashed red line shows the uniform folding path.

a) ***Continuously Foldable****:* Continuous foldable means a net N can be folded back to P without self-collision regardless the complexity of the folding motion.

b) ***Monotonically Foldable****:* The monotonic foldability is a special case in continuous foldability, where the angular velocity must remain positive $\dot{\theta}_i \geqslant 0$ or negative $\dot{\theta}_i \leqslant 0$ throughout the folding, as many active material driven hinges cannot reverse the direction. Shown in Fig. 3 (left), some hinges need to fold forward and backward to avoid self-collision for non-monotonically foldable nets.

c) ***Linearly Foldable****:* A net is linearly foldable if there exists a straight line connecting initial and target states in C-free. Shown in Fig. 3 (right), linearly foldable net has the shortest folding path in C-free, and the length of the path is

$$L_l = \sqrt{\sum_{i=1}^{n} \theta_i^2} \tag{2}$$

Physically, a linearly foldable net can be folded under the constraint that all hinges rotate at a speed proportional to the folding angle.

*d) **Uniformly Foldable**:* Distinct from linear folding, uniformly foldable nets can fold under the constraint that their hinges must rotate at the same speed. Shown in Fig. 3 (right), θ_1 is a larger folding angle than θ_2, so θ_2 is fully folded first (assume that the robot has a mechanism to stop folding each hinge at goal angle), then θ_1 continues to fold.

The linear and uniform foldability are both a special case of monotonic foldability, as it requires a certain unique path to be collision-free. The length of uniformly folding path is

$$L_u = \sqrt{n\theta_1^2} + \sum_{j=1}^{n-1} \sqrt{(n-j)(\theta_{j+1} - \theta_j)^2} \qquad (3)$$

where $\theta_j \leqslant \theta_{j+1}$ for $j = 1, 2..n$. If all θ_j are equal, then $L_l = L_u$ and the linear folding path and uniform path is the same for a given net.

Note that L_u is upper bounded by $\sum_{i=1}^{n} \theta_i$, so $\sqrt{2} > \frac{L_u}{L_l} \geqslant 1$. This suggests that the uniformly folding path is longer than linear one by a constant, which suggests that the uniform foldability may be slightly harder to achieve.

To evaluate the foldability of a net, motion planning time can be a useful metric. Shorter planning time usually indicates an easier foldable net. Thus, we let $f^*(N) = T_{N,\Pi}/T_{N,A}$ be the foldability metric of a net N, where $T_{N,\Pi}$ is the time needed to check the validity of the path Π, and $T_{N,A}$ is the time needed for a planner A to generate a collision-free path Π. If a net N is continuously foldable, $0 < f^*(N) \leq 1$. And, if N is linear or uniform foldable, $f^*(N)$ is 1. Our objective is to design an unfolder $u(P, f^*)$ that maps a polyhedron P to a net N with $f^*(N)$ as close to 1 as possible.

B. Fitness Approximation and Evolution Control

A naive unfolder $u(P, f^*)$ uses f^* as a fitness function in the genetic-based algorithm. However, evaluating the foldability using motion planning is extremely time consuming; a single call of motion planner involves a large amount of sampling and collision checking and can take minutes to hours to complete. Additionally, as many planners are sampling-based algorithm, the estimation of motion planning time is stochastic. In order to obtain a stable estimation of foldability landscape, one can either sample the fitness function multiple times, or average over more individuals. Both methods will, again, dramatically increase the computational cost.

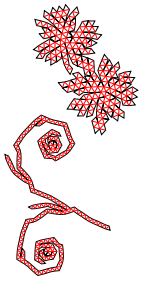
The fitness approximation methods [19] have been widely applied to similar problems. Specifically, the functional approximation methods can be used in our problem domain. In functional approximation, an alternate and explicit expression is constructed for the fitness function. Take the most popular polynomial approximation for example, the constructed fitness function has the following form: $f' = w_0 + \sum_i w_i x_i + \sum_{i<j} w_{i+j} x_i x_j$, where $\{x_i\}$ is a set of criteria used to approximate f^* and they are combined by coefficients $\boldsymbol{W} = \{w_i\}$. The coefficients $\boldsymbol{W}$ are then optimized using *evolution control* [19] so that $|f' - f^*|$ is minimized.

The success of fitness approximation methods stems from the sparse evaluation of the computationally expensive objective f^*; only once in each iteration of evolution control. The functional approximation method is essentially a supervised regression learner embedded in the evolution control framework. However, in polyhedron unfolding, it is difficult to find a good hypothesis fitness function f', even using sophisticated non-linear models, to substitute the folding path planning time f^*. Minimizing $|f' - f^*|$ relies on a large number of samples through evaluation of f^*, which remains extremely time-consuming to obtain.

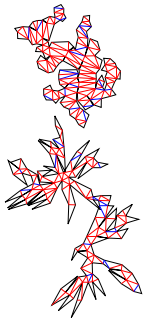
Fortunately, because our goal is finding the foldable nets, obtaining a perfect approximation that minimizes the error of $|f^* - f'|$ is unnecessary. That means, our fitness functions f' does not approximate the foldability metric f^* in a way like traditional functional approximation does. Instead, f' is a *policy* that controls the evolution by ensuring that f' and f^* are positively correlated. In other words, we would like to find f' such that $\delta f^* \delta f'$ is maximized locally. This idea is essential to the success of our method.

IV. Geometric and Topological Features of Nets

Our first goal is to map a polyhedral net N into a *feature space* $\mathbb{X}$, in which the foldability can be efficiently approximated. To construct a good hypothesis fitness function, we observe that different unfoldings for a polyhedron may have distinct geometric and topological features. Examples of these features include the cut length, hull area, and the number of leaf nodes. A complete list of 12 features studied can be found in supplementary materials. Due to a trade-off between the model complexity and training time and the underlying correlations between these 12 features, only four key features are chosen for constructing our fitness function and are normalized within the range $[0, 1]$.



*a) **Node Degrees**:* The first property is a topological feature defined as the ratio between the number of leaf (degree on) node and the total number of nodes in a net. Recall that nets are spanning trees, and each node has 1, 2 or 3 neighbors in triangular meshes. As we can observe in a net, the number of leaf nodes equals to the number of branches and approximates the branch lengths of the net. The figure on the right shows the nets with large (top) and small (bottom) number of leaf nodes obtained from the same polyhedron.



*b) **Cut Length**:* Edges of a net are either cut or crease edges. Given a polyhedron with $|F|$ faces and $|E|$ edges, all net of this polyhedron must have $\sigma = 2(|E| - |F| + 1)$ cut edges and $|F| - 1$ crease edges. The length of these σ cut edges affects foldability and is usually minimized [11], [13]. For example, cutting two longest edges of a narrow triangle creates a spike and these spikes are likely to collide. The cut length is upper bounded and lower

bounded by the total length of σ longest edges and σ shortest edges, respectively, thus is normalized by these bounds. The figure above shows the nets with short (top) and long (bottom) cut length obtained from the same polyhedron.

*c) **Border Cuts Length**:* Cut length does not vary a lot if the polyhedron is tessellated with (nearly) equilateral triangles (e.g. a geodesic dome in Fig. 4). Thus, we introduce the concept of border cuts to capture additional features. Definition of border cuts can be found in Fig. 4. The intuition behind this is that some cut edges can be easily "zipped" without collision. On the contrary, pairs of matching border cuts are often separated far away and harder to be joined back. The border cut length is normalized by the total cut length.

*d) **Hull Area**:* The convex hull bounding the flattened net provides information about the clearance between branches. If the hull area is close to the total area of the net surface, this would indicate that the net is tightly bounded. We use the ratio of the surface area with respect to the hull area as a normalized factor in our fitness function.

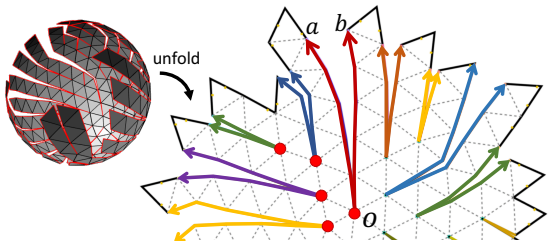


Fig. 4: **Border cut edges.** In this net, cut edges along the thick arrows can be "zipped" without collision. For example, the edges from o to a can be zipped with those from o to b. Edges that cannot be zipped are border cuts (in solid black).

V. Learning the Fitness Function

In the previous section, we mapped nets into a low dimensional feature space $\mathbb{X}$. Although these features may be insufficient to construct a hypothesis fitness function as a regression model to approximate the foldability metric f^*, we only need to find a policy $\hat{f}$ ensuring that $\hat{f}$ and f^* are positively correlated. Therefore, despite resemblance with fitness approximation and evolution control, our method is fundamentally different and is much more efficient.

Based on this observation, we propose two novel hypothesis functions (Section V-A) that are iteratively updated by a learning strategy (Section V-B).

A. The Fitness Functions

Let us denote our hypothesis fitness as $\hat{f} : \mathbb{X} \to \mathbb{R}$. Given a net N (a polyhedral unfolding without overlapping), our objective is to optimize $\hat{f}(g(N))$ so that the foldability metric $f^*(N)$ and $\hat{f}(g(N))$ are positively correlated. Our model does not rely on complex hypothesis minimizing $|\hat{f} - f^*|$. We propose two hypothesis fitness $\hat{f}$ called the planar fitness and the paraboloid fitness.

*1) **The Planar Fitness Function**:* The planar fitness function is linear combination of the m normalized features x_i,

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{m} w_i x_i = \boldsymbol{W}^\intercal \boldsymbol{x} \ , \tag{4}$$

where w_i is the unknown weight of i-th feature. The function also forms a m dimensional hyperplane, where $\boldsymbol{W}$ is the normal direction. Shown in Fig. 5a, if $f^{'} = \beta + \sum_{i=1}^{m} w_i x_i$ is used as the fitness function in the genetic-based unfolder. The unfolder evolves the nets towards the direction that achieves a higher combined value of the characteristics. Visually, a point x, which corresponds to certain nets in the feature space, will slide up against the tilt of the hyperplane.

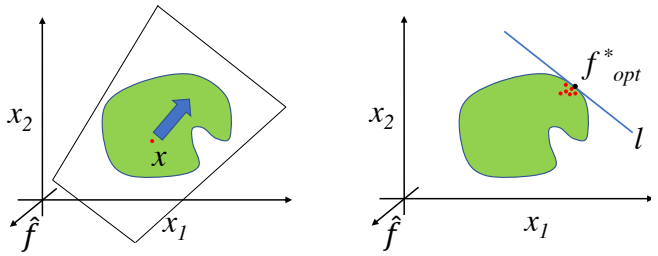


Fig. 5: An illustration of the planar fitness function with two features x_1, x_2. The green area illustrates all valid unfoldings in the feature space (defined in Section IV). (a) The trapezoid is a perspective view of the fitness plane. (b) l is the projected tangent line and red dots are the optimized nets.

Our learning objective is to find such hyperplane that helps the genetic-based unfolder push the individuals towards the point of the optimal foldability. We can think of this as an *inverse linear programming* problem, which is given a set of constraints on x_i, find an optimal linear function $\hat{f} = \sum_{i=1}^{m} w_i x_i$ that has the maximum value only on a known optimal point. Shown in Fig. 5b, the line l is a vertical projection of the hyperplane tangent to the boundary of the feature space. The target point f^*_{opt} corresponds to the nets with the optimal foldability in the feature space. The optimal hyperplane cannot be solved by inverse linear programming algorithms as our constraints for x_i is the anomalous boundary of the feature space, rather than a set of linear inequalities. Also, the foldability metric landscape f^* can be estimated, but the optimal point f^*_{opt} is unknown. We designed a learning strategy that learns the optimal solution through a sampling approach which will be discussed in detail in Section V-B.

The planar fitness function works for many simple shapes, however, it requires that (1) the nets with optimal foldability f^*_{opt} lie on the boundary of the feature space, and (2) the boundary of the feature space is convex. In our experiment, we found the first assumption is often satisfied, as our features incorporate the characteristics related to the foldability. This means the optimal foldability often lies on the *pareto front* of the multiple characteristics. The second assumption, however,

is often invalid. In many cases, the feature space is not even a connected component. For complex non-convex polyhedron, valid unfoldings are already difficult to find and the feature space is likely to contain many small islands. In these situations, $\hat{f}$ becomes harder to converge. The paraboloid fitness function is introduced next to address these issues.

*2) **The Paraboloid Fitness Function**:* The paraboloid fitness function overcomes the drawbacks of the planar fitness function by adding a higher-degree curve on the fitness landscape and has the following formula:

$$\hat{f}(\boldsymbol{x}) = -\sum_{i=1}^{m} (x_i - w_i)^2 \tag{5}$$

The right side of the function is an m dimensional inverted paraboloid centered at $\boldsymbol{W}$; illustrated in Fig. 6, the genetic-based unfolder trim the individual nets in the feature space x_i, and optimize them towards the point $\boldsymbol{W}$, with respect to our fitness function $\hat{f}$. The learning is a process of finding the $\boldsymbol{W}_{opt}$ that as close as possible to the optimal point on the actual foldability landscape. Because the paraboloid fitness function has only one unique optimal point, the individuals will not converge to some sub-optimal points, even if the optimal point is not on the convex part of the boundary or even within the boundary.

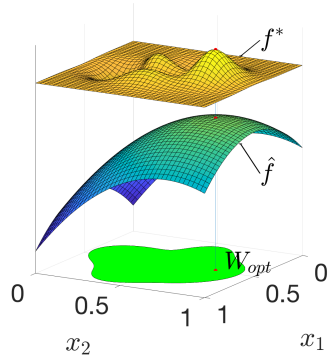


Fig. 6: An illustration of a paraboloid fitness function $\hat{f}$. f^* is the landscape of the foldability metric.

A limitation of the paraboloid fitness function is that the initial weights $\boldsymbol{W}_{init}$ may cause the optimization to stuck in a local optimal, or even worse, will not generate valid unfoldings because $\boldsymbol{W}_{init}$ is far out of the boundary of the feature space. In practice, we can apply planar fitness function first to probe a possible f^*_{opt}, then obtain an initial guess of paraboloid fitness function through induction.

B. Updating Fitness Function

Recall that the fitness function $\hat{f}$ needs to be updated after each iteration. In this section, we describe a learning strategy that, in a sense, mimics the reinforcement learning process. In this learning strategy, we model the genetic-based unfolder $u(P, \hat{f})$ as an agent that produces polyhedral nets of a given polyhedron P. The behavior of our agent is controlled by a *policy* $\hat{f}$, and the goal of the agent is to produce nets N with the highest *reward*, i.e. the optimal foldability $f^*(N)$.

As shown in Fig. 2, the policy $\hat{f}$ is updated by, evaluating f^* on the single best individual produced by the unfolder $u(P, \hat{f})$. Due to the non-convexity of f^*, we use the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm [33] as an external optimizer to update the policy. CMA-ES is a stochastic algorithm that uses a population-based approach similar to the genetic algorithm. For each iteration, CMA-ES tracks the mean and covariance of the samples (i.e. $\boldsymbol{W}$s), and the mean is shifted from its previous position towards the samples with better performance. The new covariance matrix is also adapted to the trajectory of the mean over the past few iterations. The new distribution is then re-sampled for the next iteration. The intuition is that, in every iteration, the CMA-ES will generate a set of fitness functions $\{\hat{f}_i\}$ to be employed by the genetic-based unfolders, then each unfolder will produce the best individual net N_i using $\hat{f}_i$. The evaluation $f^*(N_i)$ of the best net N_i serves as an estimation of the performance of $\hat{f}_i$ to be considered by CMA-ES for the next iteration.

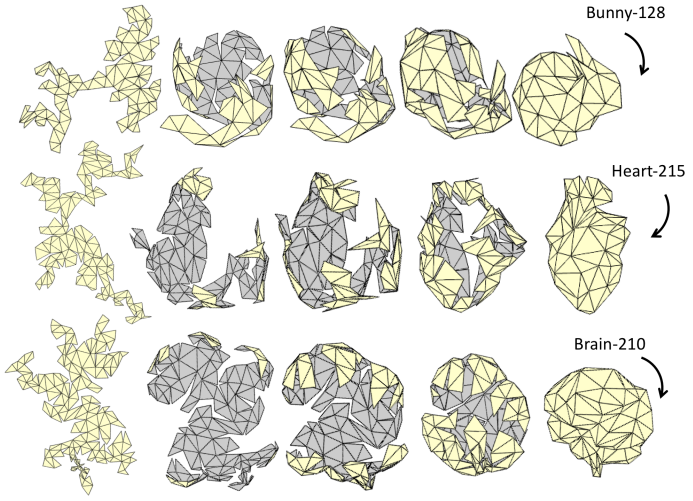


Fig. 7: Linearly foldable nets created by the proposed method.

VI. Experiments and Results

In this section, we evaluate the foldability of our optimized polyhedral nets in terms of path planning time (Sections VI-A). We will discuss a biased sampling technique that improves the efficiency of our training in Sections VI-B and a transfer learning strategy to handle complex models in Section VI-C. In Section VI-D, we demonstrate that we can use our framework to find nets with user-defined motion constraints.

All experimental results reported are set up as follows. To learn the fitness function, we run 40 genetic-based unfolders and the maximum training iteration is 50. The population size within the genetic-based unfolder is 400, and the maximum number of generations is 200 for polyhedra with less than 50 faces and 300 for over 50 faces. In each iteration, 40 best individuals are produced, and we use motion planner to evaluate each net 10 times and report the average. Parallelization is used in most stages. All data are collected from a C++ implementation on a workstation with two 2.3GHz Intel CPUs.

A. Optimized Polyhedral Nets

We first show that the optimized unfolder generates more foldable polyhedral nets than arbitrary nets produced by an existing unfolder discussed in Section II-B. We evaluate the

foldability using the Average Path Planning Time **APPT**. The metric is defined as follow:

- For a polyhedron, we unfold it into 40 nets using the unfolder to be tested;
- For each net, we plan the folding path 10 times using a given motion planner;
- APPT is the average planning time for the 400 paths.

The 3D models in Figs. 1, 8 and 10 are used in our experiment. Among these models, only the star is not linearly foldable. Our algorithm uncovers the linearly foldable nets if such nets exist. Table I clearly shows that the optimized nets have much smaller APPTs, thus are easier to fold.

Fig. 8: The models used in our experiments: L-28, Star-24, Moneybox-48, Bunny-64 and Fish-96.

TABLE I: Comparison of APPT of Arbitrarily Generated Nets and Nets Generated Using Optimized Unfolder

Model	DoF	APPT (s)		Speedup
		Arbitrary	Optimized	
L-28	27	0.892	0.021	42.5x
Star-24	23	17.587	1.195	14.7x
Moneybox-48	47	59.546	1.211	48.2x
Bunny-64	63	100.223	1.152	87.0x
Fish-96	95	82.418	2.923	28.1x

B. Biased Sampling Technique for Motion Planning

From our experiments, the training time is bottlenecked by the motion planner. To reduce the computation cost for motion planning, we employed a biased sampling technique that increases the sampling rate near the target region. Various biased-sampling techniques [34], [35], [36] have been proved to be effective in many applications. The key idea is to manipulate the probabilistic distribution of random sampling to increase sample rates near the obstacle dense regions.

Our biased sampling technique is based on an observation which we call the *Target Region Prior*: self-collisions are more likely to happen when the configuration is near the target region than that near the initial state. As around the initial state, the net is almost flat and has large clearance between branches; While near the target, the configuration space is crowded with obstacles. More importantly, even some self-collisions occur near the initial state, it is easy to find an alternative path that circumvents the obstacles due to the large majority of the region is collision-free. However, the configuration space near the target is almost full of obstacles and narrow passages, and it often requires certain critical samples to be connected.

In our experiment, we found that the *Target Region Prior* applies to all polyhedra tested. Our biased sampling algorithm outperforms the planner without bias [18] for arbitrarily generated nets is shown in Table II. Table II also shows the benefit of biased sampling in reducing the training time. If the motion planner failed to fold a net within the timeout limit, the folding is aborted and a large penalty is given to the evaluation. As the learning progresses, the median planning time decreases (shown in Fig. 9) and the training speeds up in each iteration. For polyhedral nets with large DoF, the median planning time is still high in the first few iterations, which raises the need for the transfer learning strategy.

TABLE II: Comparison of APPT for Arbitrarily Generated Nets and Training Time (TT) for the first 20 Iterations.

Model	APPT (s)		TT (min.)	
	w/o Biased	w/ Biased	w/o Biased	w/ Biased
L-28	0.892	0.424	19	19
Star-24	17.587	1.145	44	19
Moneybox-48	59.546	6.036	84	46
Bunny-64	100.223	31.125	106	47
Fish-96	82.418	29.826	138	95

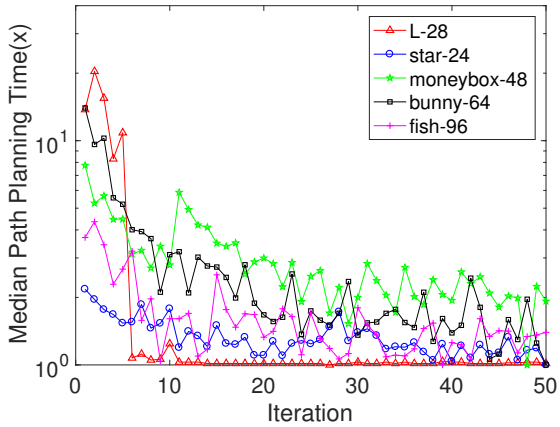


Fig. 9: Learning Curve: relative path planning time (actual time normalized by dividing the shortest time) is used here. Each data point is the *median* planning time of the 40 best individuals in the iteration, not using *APPT* here to avoid impact by large penalties for failed foldings.

C. Transfer Learning

As shown in the previous experiments, the APPT increases significantly for models with many faces and highly non-convex geometries. We use a transfer learning [37] technique, illustrated in Fig. 10, to handle complex models. We reduce the complexity of the model by remeshing it with fewer faces. The simplified mesh, though may lose details, retain most geometric characteristics. We can easily obtain a fitness function $\hat{f}_{simp}$ from the simplified model using the proposed learning strategy. Then $\hat{f}_{simp}$ serves as a shortcut for the training on the complex model. Table III compares the APPT of folding the nets of various complex models using transfer learning.

Fig. 7 shows the linearly foldable nets for select complex models obtained using transferred learning strategy. Finding a linear folding path for these nets is almost instantaneous, while an arbitrarily generated net may need half an hour of path planning time. Video showing folding motion

TABLE III: Comparison of APPT of Complex Nets Generated Using Fitness $\hat{f}_{init}$: Initial Fitness (for Arbitrary Nets); $\hat{f}_{simp}$: the Fitness of Simplified Models; $\hat{f}_{opt}$: a Further Optimized Version of $\hat{f}_{simp}$

Model	DoF of $\hat{f}_{simp}$	APPT (s)		
		$\hat{f}_{init}$	$\hat{f}_{simp}$	$\hat{f}_{opt}$
Moneybox-128	47	350.1	52.5	15.1
Bunny-128	63	344.5	46.6	12.2
Fish-150	95	397.9	60.6	18.4

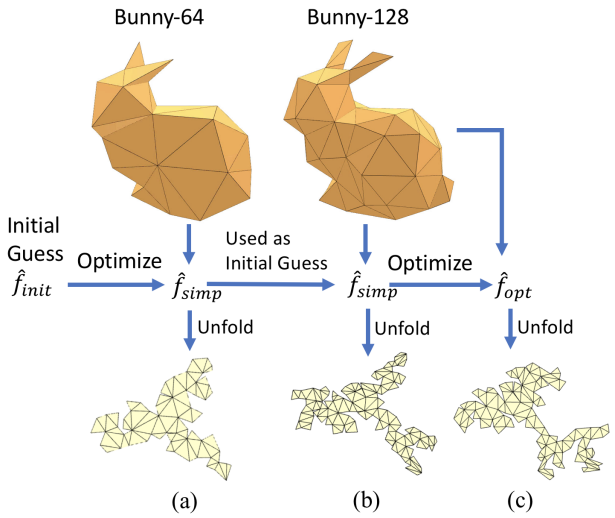


Fig. 10: An optimal fitness $\hat{f}_{opt}$ for Bunny-128 is obtained from $\hat{f}_{simp}$ trained on Bunny-64. If we unfold Bunny-128 via $\hat{f}_{simp}$ we can obtain net (b) which shares common topological and geometrical features with net (a) and has foldability already far better than arbitrary nets. The fitness $\hat{f}_{simp}$ is subsequently used as initial fitness for the learning of $\hat{f}_{opt}$.

for various nets is available on http://masc.cs.gmu.edu/wiki/LinearlyFoldableNets.

D. Foldable Nets with User-Defined Motion Constraints

In this experiment, we show that the proposed learning strategy is flexible and can be easily adapted to generate nets with additional user-defined motion constraints. In Fig. 11, the user specifies a motion constraint that requires an ancestor crease (i.e., creases closer to the root face) start to fold only after all its descendant nodes being fully folded. We simply replace the evaluator f^* in our framework with a new motion planner that checks if the nets satisfy the constraint.

VII. Conclusion, Limitations and Future Work

In this paper, we addressed the problem of creating foldable polyhedral nets. We combine the topological and geometric features into a fitness function and use it to approximate the computationally expensive foldability metric. A learning strategy is proposed to optimize the fitness function in an efficient manner. This synthesis of the domain knowledge with learning technique relieves us from the problem of the complex hypothesis model design and acquisition of large

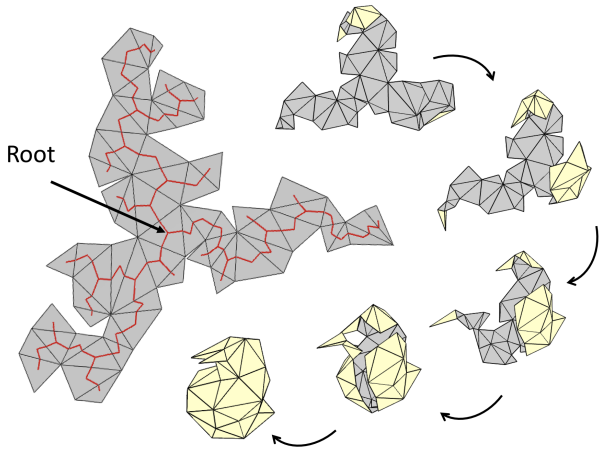


Fig. 11: A foldable net of Bunny-64 satisfying the motion constraints that all the descendant creases must be folded prior to the ancestor creases. Left: the net and its dual graph (shown in red) created by our method with this new constraint. The root face is designated as the topological center of the dual graph. Right: the folding sequence of the net.

sample data that the traditional evolution control method faces. The proposed learning strategy can be easily adapted to find polyhedral nets with other features or constraints.

Another noticeable achievement is that we are able to find the linearly and uniformly foldable nets for highly non-convex polyhedra, making it possible to design and manufacture high DoF self-folding robots constrained actuators. As our framework is based on an evolutional algorithm, despite the ability to distribute the computation using parallelization, the amount of total computation is still too large for the personal computing environment. For nets with hundreds and thousands of faces, the foldability without segmentation is not yet discussed. One of the reasons is that we only use four characteristics to define the feature space, but, for high DoF nets, nets with distinct foldability may be mapped to similar spots in the feature space, which makes optimization difficult. For future work, new features will be investigated, and new learning techniques can be employed to replace our current mapping mechanism.

Acknowledgement

This work was supported in part by NSF EFRI-1240459 and CNS-1205260, AFOSR FA9550-17-1-0075 and Nvidia GPU Grant Program. Some early ideas were inspired by the coversation with In-Suk Choi of Seoul National University, Takahashi Shigeo of University of Aizu, Japan, Hsu-Chun Yen of National Taiwan University.

References

[1] S. M. Felton, M. Tolley, E. D. Demaine, D. Rus, and R. Wood, "A method for building self-folding machines," *Science*, vol. 345, no. 6197, pp. 644–646, 2014.

[2] D. Davis, B. Chen, M. D. Dickey, and J. Genzer, "Self-folding of thick polymer sheets using gradients of heat," *Journal of Mechanisms and Robotics*, 2015.

[3] Y. Liu, J. K. Boyles, J. Genzer, and M. D. Dickey, "Self-folding of polymer sheets using local light absorption," *Soft Matter*, vol. 8, no. 6, pp. 1764–1769, 2012.

[4] B. An, S. Miyashita, M. T. Tolley, D. M. Aukes, L. Meeker, E. D. Demaine, M. L. Demaine, R. J. Wood, and D. Rus, "An end-to-end approach to making self-folded 3d surface shapes by uniform heating," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1466–1473.

[5] H. Shigemune, S. Maeda, Y. Hara, U. Koike, and S. Hashimoto, "Kirigami robot: Making paper robot using desktop cutting plotter and inkjet printer," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 1091–1096.

[6] S. Sundaram, D. S. Kim, M. A. Baldo, R. C. Hayward, and W. Matusik, "3d-printed self-folding electronics," *ACS Applied Materials & Interfaces*, vol. 9, no. 37, pp. 32 290–32 298, 2017, pMID: 28825288. [Online]. Available: http://dx.doi.org/10.1021/acsami.7b10443

[7] B. An, N. Benbernou, E. D. Demaine, and D. Rus, "Planning to fold multiple objects from a single self-folding sheet," *Robotica*, vol. 29, no. 1, pp. 87–102, 2011.

[8] A. Dürer, "Underweyssung der messung mit dem zyrkel und rychtscheyd," *Nürnberg (1525). English translation with commentary by Walter L. Strauss The Painter's Manual, New York*, 1977.

[9] J. O'Rourke, "Unfolding polyhedra," 2008.

[10] E. D. Demaine, M. L. Demaine, V. Hart, J. Iacono, S. Langerman, and J. O'Rourke, "Continuous blooming of convex polyhedra," *Graphs and Combinatorics*, vol. 27, no. 3, pp. 363–376, 2011.

[11] H. Prautzsch and R. Straub, "Creating optimized cut-out sheets for paper models from meshes," in *Proceedings of the 9th SIAM Conference on Geometric Design and Computing*, 2005.

[12] W. Schlickenrieder, "Nets of polyhedra," Master's thesis, Technische Universität Berlin, 1997.

[13] S. Takahashi, H.-Y. Wu, S. H. Saw, C.-C. Lin, and H.-C. Yen, "Optimized topological surgery for unfolding 3d meshes," in *Computer Graphics Forum*, vol. 30, no. 7. Wiley Online Library, 2011, pp. 2077–2086.

[14] Z. Xi, Y.-H. Kim, Y. J. Kim, and J.-M. Lien, "Learning to segment and unfold polyhedral mesh from failures," in *Shape Modeling International (SMI); also appears in Journal of Computers & Graphics*, Berlin, Germany, Jun. 2016.

[15] E. D. Demaine, S. L. Devadoss, J. S. B. Mitchell, and J. O'Rourke, "Continuous foldability of polygonal paper," in *CCCG*, 2004.

[16] T. Tachi, "Simulation of rigid origami," *Origami*, vol. 4, pp. 175–187, 2009.

[17] G. Song and N. M. Amato, "A motion-planning approach to folding: From paper craft to protein folding," *Robotics and Automation, IEEE Transactions on*, vol. 20, no. 1, pp. 60–71, 2004.

[18] Z. Xi and J.-M. Lien, "Continuous unfolding of polyhedra - a motion planning approach," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, Sep. 2015, pp. 3249 – 3254.

[19] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft computing*, vol. 9, no. 1, pp. 3–12, 2005.

[20] Y. Hao, Y.-H. Kim, and J.-M. Lien, "Synthesis of fast and collision-free folding of polyhedral nets," in *Proceedings of the ACM Symposium on Computational Fabrication (SCF)*, Boston, MA, Jun 2018.

[21] E. Miller and I. Pak, "Metric combinatorics of convex polyhedra: cut loci and nonoverlapping unfoldings," in *Twentieth Anniversary Volume:*. Springer, 2009, pp. 1–50.

[22] T. Biedl, A. Lubiw, and J. Sun, "When can a net fold to a polyhedron?" *Computational Geometry*, vol. 31, no. 3, pp. 207 – 218, 2005, 11th Canadian Conference on Computational Geometry. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925772104001361

[23] Z. Xi and J.-M. Lien, "Folding rigid origami with closure constraints," in *International Design and Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE)*. Buffalo, NY: ASME, Aug. 2014.

[24] Z. Xi and J.-M. Lien, "Plan folding motion for rigid origami via discrete domain sampling," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, May. 2015.

[25] M. V. Kircanski, "Robotic isotropy and optimal robot design of planar manipulators," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, May 1994, pp. 1100–1105 vol.2.

[26] L. Stocco, S. E. Salcudean, and F. Sassani, "Matrix normalization for optimal robot design," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 2, May 1998, pp. 1346–1351 vol.2.

[27] E. J. Van Henten, D. A. Van't Slot, C. W. J. Hol, and L. G. Van Willigenburg, "Optimal manipulator design for a cucumber harvesting robot," *Comput. Electron. Agric.*, vol. 65, no. 2, pp. 247–257, Mar. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.compag.2008.11.004

[28] H. H. Lund, J. Hallam, and W.-P. Lee, "Evolving robot morphology," in *Evolutionary Computation, 1997., IEEE International Conference on*, 1997, pp. 197–202.

[29] K. Sims, "Evolving virtual creatures," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '94. New York, NY, USA: ACM, 1994, pp. 15–22. [Online]. Available: http://doi.acm.org/10.1145/192161.192167

[30] J. E. Auerbach and J. C. Bongard, "Dynamic resolution in the co-evolution of morphology and control," in *Artificial Life XII: Proceedings of the Twelfth International Conference on the Synthesis and Simulation of Living Systems*, no. EPFL-CONF-191277. MIT Press, 2010, pp. 451–458.

[31] J. E. Auerbach and J. C. Bongard, "On the relationship between environmental and morphological complexity in evolved robots," in *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '12. New York, NY, USA: ACM, 2012, pp. 521–528. [Online]. Available: http://doi.acm.org/10.1145/2330163.2330238

[32] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson, "Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '13. New York, NY, USA: ACM, 2013, pp. 167–174. [Online]. Available: http://doi.acm.org/10.1145/2463372.2463404

[33] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.

[34] P. Leven and S. Hutchinson, "Using manipulability to bias sampling during the construction of probabilistic roadmaps," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 6, pp. 1020–1026, 2003.

[35] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle, "Dynamic-domain rrts: Efficient exploration by controlling the sampling domain," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005, pp. 3856–3861.

[36] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How, "Motion planning for urban driving using rrt," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, 2008, pp. 1681–1686.

[37] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.