StarGAN for Neighborhood Development

Zong Fan, Xiaobai Li, Zhongweiyang Xu, Dimitrios Gotsis

I. Introduction

Urban planning is an important task for socioeconomic development nowadays, especially for cities in developing countries and regions. Well-designed urban planning significantly changes the life quality of people living in the "slums" with poor infrastructures. Many analytical methods in neighborhood optimization were employed in previous studies, aiming to optimize the design of a particular city, which usually cannot be generalized to other cities directly. Nowadays, deep learning (DL)-based artificial intelligence (AI) techniques have revolutionized various aspects of industries and societies, showing the potential to solve the reblocking problem. Among these AI techniques, the generative methods are promising unsupervised methods in analyzing and understanding unlabeled data. They can capture the intrinsic probabilistic distribution of the desired target data to generate a class of data that is indistinguishable from the real examples, which have been widely employed in various applications ranging from the image domain to speech and text domain [1], [2]. Particularly, generative models have shown creativity in assisting explorations in various practical domains such as protein molecule design in drug synthesis, fragrance formulation design in culinary engineering, etc [2]. Inspired by their creativity in manipulating human knowledge, we try to develop a methodology by the use of generative models for neighborhood reblocking tasks. There might be two advantages of using generative methods: 1) they could provide diverse redesign possibilities to suit various scenarios; 2) the generation could be controlled by certain model designs to reach desired performances, performances, or characteristics, such as lift quality improvement degree, road construction budget, country or culture preference, historical and religious concern, etc. In this way, diverse generations can provide adequate options for the decision-maker to select the most suitable scheme in practice.

In the rest of the paper, we would discuss the background of the research in section II, including the topology of neighborhood and generative models. Then the proposed methodology is discussed in section III, including the data collection, selection of models, loss functions, and metrics.

II. Background

A. Topology of Neighborhoods

There are about 4 billion people living in urban areas, out of whom about 1 billion of them are living in slums. According to UN-Habitat, there will be 6.4 billion people living in cities and 3 billion living in slums by 2050, if no one address and try to solve this issue with a practical framework.

Belsford *et. al* [3] points out that people have been putting effort to solve this issue for years, and there's an increasing need for Urban planning. Urban planning is the forward-thinking process whereby people arrange the layout of towns, cities, and urban areas based on a set of objectives defined by the state or territory in conjunction with local councils. Urban planning is crucial since it can improve the life quality of peopling living in urban areas. If an urban area is badly planned or even neglected, then the city and its infrastructure become unsustainable and ultimately hinder the growth of the population and economy as discussed in [4], which is exactly what happens in the slums, and improvement is needed urgently. People start to approach cities as complex adaptive systems and treat this as the key to generating better knowledge and better solutions to get rid of all slums, building "city without slums".

In order to translate the problem into scientific terms, we need to place the character of slums in other urban neighborhoods besides slums. Such an approach is valid since although the nature of all cities varies quite a bit and the urban growth in recent years is unprecedented, the core of the problem itself is mostly the same.

People were trying to use data-driven statistical analysis to classify urban spatial patterns with an eye toward optimal design, but the quest for ideal forms has remained elusive. Space syntax methods within the urban morphology literature also recognize that the essential logic of many functional aspects of urban environments relies on connectivity rather than distance or geometry. Since topology was invented to describe problems of access and circulation in cities, this connectivity perspective naturally suggests a topological approach might be the optimal solution for our problem. The topology of any city block can be systematically measured and understood via the mathematical analysis of the relationship between the space of places and accesses, like block complexity, and people developed some algorithms to optimize the access networks in any neighborhood work which can be used in practice, according to Brelsford *et. al* [3].

Block complexity, a term that describes how difficult to access internal parcels in each block is used to classify all blocks in any particular city, and slums typically have much higher block complexity values. The higher the block complexity is, the harder it is for people in that block to access internal parcels. The algorithm minimizes the block complexity under constraints, leading to a sequence of constrained optimization problems whose solution is a completely accessible neighborhood with small travel costs between places, and the whole process is called "optimal reblocking".

The goal of the optimization algorithm proposed by Belsfold *et. al* [4] in 2019 is to reduce the k-complexity of all blocks to 2 with minimal connection added so that there are no longer

"slums" in the city. It does this by doing strict greedy optimization, which means it seeks to find the absolute smallest length of new accesses to be built in the city. Such an approach will give us the "optimal" solution to add the least number of connections into the city to make it "city without slums", thus reducing the cost of doing this process.

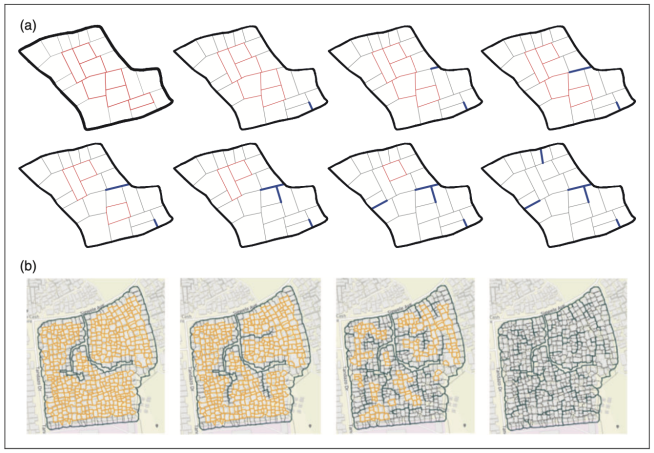

Fig. 1. (a) The incremental nature of the greedy search algorithm in which shorter paths are preferentially selected out of the set of many short paths that connect all parcels to the existing roads until a complete solution is achieved and (b) intermediate steps from the same process for the Khayelitsha neighborhood. The far-left map shows the original road layout, and the far-right map shows a final solution in which 1331 m of newly constructed streets are required. Source: Belsford *et. al* [5].

However, all these methods are using discriminative models and can only be used to optimize the existing cities' topology. What if we need to generate a plan for a new city or generate plans for current cities so that architects can learn from these generated topology methods and thus gain insight on how to improve the city structure? We need to think of using generative models instead.

B. Generative Models

A diverse array of DL-based techniques have been proposed to solve various generative problems. They could be roughly divided into four categories: 1) generative adversarial networks (GANs); 2) variational auto-encoders (VAEs); 3) auto-regressive (AR) models; and 4) normalizing flows (NFs) [1].

1) Generative Adversarial Network: Initially proposed by Goodfellow *et. al* [6] GANs and their variants are able to approximate target data distribution by using two adversarial networks playing a min-max game. The two networks in the GAN are a generator $G_\theta$and a discriminator D_ϕ. The generator works as a forger which generates data that captures the true sample distribution $p_d(x)$ from a low-dimension prior stochastic noise $p_z(z)$ (like Gaussian noise), while the discriminator works as a detective to tell whether the given data is from the training dataset or fabricated by the generator $G_\theta(z)$ precisely. The alternative training of the generator and discriminator is employed by optimizing the value function

$$\min_{G_\theta}\max_{D_\phi} \mathop{\mathbb{E}}_{x\sim p_d(x)}[\log D_\phi(x)] + \mathop{\mathbb{E}}_{z\sim p_z(z)}[\log(1-D(G(z)))] \tag{1}$$

2) Variational Auto-encoder: VAEs are a kind of latent variable-based models that assume the observed, high-dimensional data can be generated from an unobserved low-dimensional latent variable through a probabilistic process [7]. Considering a set of target samples x from a distribution parameterized by ground truth latent codes z, the encoder of a VAE is defined by a variational posterior $q_\phi(z|x)$, while the decoder is a generative distribution $p_\theta(x|z)$. VAE aims to learn a joint distribution between the latent space $p(z)$ and the input space $p(x)$, where the latent space is assumed to be a simple prior distribution as a standard Gaussian. However, optimizing the model through maximum likelihood is intractable since the marginal likelihood $p_\theta(x)$ is intractable. Instead, the tractable evidence lower bound (ELBO) is employed in practice to optimize the networks [8].

3) Auto-regressive Model: Auto-regressive models are based on chain rules of probability which are used for generating sequential data [1]. The model aims to predict the next element in a sequence given the previously generated elements. The recurrent neural networks (RNNs) can be employed as the AR modeling network architecture, since the natural architecture of RNNs could model sequential data by tracking information in the hidden state [1]. RNNs try to learn the joint probability distributions of sequences, so that they could generate a sequence by stochastic sampling.

4) Normalizing Flow: NFs are a class of generative models that define a sequence of parameterized invertible deterministic transformations that convert the simple distribution of latent space z to complex target distribution of the data space x [1]. The relationship between the density function of data x and the latent z can be determined through the change-of-variables rule, that is

$$p_x(x) = p_z\Big(f_\theta^{-1}(x)\Big)\left|\det\left(\frac{\partial f_\theta^{-1}(x)}{\partial x}\right)\right|, \tag{2}$$

where f_θ represents the invertible transformations. Therefore, each transformation must be sufficiently expressive while being easily invertible in order to compute the Jacobian determinant efficiently.

5) Graph Generations: In order to describe the city block structure easily and efficiently, graph data format was commonly-used in previous studies, where the parcels in the block were treated as the graph node and the roads were treated as the graph edges [3], [5], as shown in Fig.II-B5. The demand for generating reblocking designs is actually a graph generation task. Particularly, according to the generation process types, it has roughly two ways, including sequential generating and one-shot generating [8]. The former method generates the nodes and edges in a one-after-another sequential

way, while the latter one builds a probabilistic graph model that can generate all nodes and edges in one shot. Various graph generation technologies have been proposed which can refer to this review by Guo *et. al* [8].

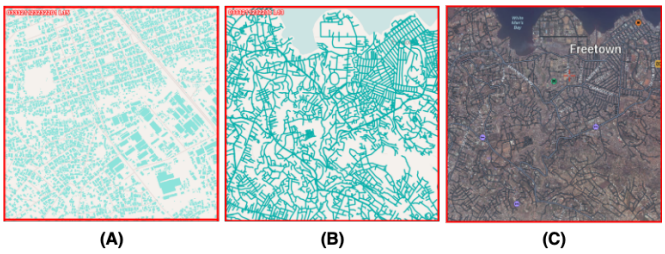

Fig. 2. Visualization of block geometric data. (A): block parcels; (B): block roads; (C) real remote-sensing image of (B).

III. Method

A. Annotation- vs Simulation-based datasets

To train the generative models, a block geometry dataset with both "good" blocks and "bad" blocks are required, where the "bad" block graph data is the input and the "good" block graph data is the output. According to the source of the "good" block graph, there are two types of training data that might be collected.

1) Simulation-based dataset: Since reblocking is an unrealized and irreversible process, it's usually difficult to get the real optimized "good" block graph data from a "bad" block. Therefore, one thinking is to simulate the "good" block graphs on the real "bad" block data through pre-defined analytical algorithms and metrics. The optimization method introduced in Brelsford *et. al* can be wrapped into simulation software, where its output graphs are treated as the ground-truth (GT) "good" block graphs. In this case, only "bad" block graphs are needed to be collected and annotated. Then simulation software is employed to produce the point-to-point mapped "good" block graphs with known performance improvements, forming the training graph pairs. This method could reduce the difficulties in obtaining large-scale dataset since geometric data collection and annotation is very complicated and time-consuming. **Pros**: the "good" and "bad" cases have point-to-point mapping, and the optimization graph changes and improvement degrees are prior known. This is an ideal mapping function to be learned by the generative model; **Cons**: the simulated "good" block may not be true "good".

2) Annotation-based dataset: Without the simulation, both "bad" and "good" block graphs are required to be collected and annotated manually. Generally, we suppose there are two types of real "good"/"bad" block graphs. 1) the block which was historically "bad" but is currently "good"; e.g. Tokyo (1950 vs 2010); 2) the good blocks and bad blocks in different regions at the same time. E.g. Cape Town (2010) vs New York (2010). For the former one, it's similar to the simulation method where the point-to-point mapping is available, except for the truth that amounts of "good" structures are limited. For the latter one, the "good" block is not directly optimized from the bad one, which means the overall structures before and after optimization may be seriously different. **Pros**: All the data is real. The optimization can be practical and meaningful in reality. **Cons**: the data size can be limited. The good/bad data may not be point-to-point mapping. The optimized block may change a lot in the structure.

B. Additional Useful Data

As mentioned above we plan on implementing a StarGAN. Besides using the topological neighborhood graph we wish to modify as the input, the dataset offers many more useful pieces of information we can take advantage of to improve our results. The country code will be used as a conditional variable for our generative model while the building count per block and the k complexity will be used in our loss function.

1) Country Code: One justification for using the country code is that each country's people may desire different types of layouts that could meet certain local cultural needs. Conditioning our generative models on the country code would thus force our model to correlate this culture specific layout to the country code further providing more appropriate generated results to local communities.

2) Construction and Material Cost: Construction and material costs change all the time and since the idea is to ultimately construct roads or pathways inside of these densely populated slums we need to take them into account. Assuming also budget which may change on a case by case basis local communities may want to see different layouts based on how much they can afford to spend in order to build more or fewer roads or pathways. Thus, we propose that the construction and material costs be used as a conditional variable for our generative model similar to the country code.

3) Building Count Per Block: The building count per block can be used to measure the density of people per block (assuming a relatively fixed number of people per building) and thus allows us to optimize our road selection in order to help the largest number of people get the most road access. In other words, some sort of dot product between the k-complexity of each both with the building count per block which we hope to minimize. We plan on implementing this in our loss function for the generator in order to constrain our generative models to produce more desirable results.

4) k-Complexity: We want our generated results to have a lower k-complexity in general than the original "bad" block neighborhoods. Ideally, we would like to simply add this to our loss function in order to minimize however it seems that both methods by which the k-complexity is found cannot be distilled to an analytic differential function for us to backpropagate through. In section III-D we propose a neural network that maps from our input block graph to its respective k-complexity graph. This solves the above issue of backpropagation. This network will be trained on existing data for which a k-complexity graph can be computed. The fully trained network will then be inserted into the generative model's loss function. We plan on minimizing the mean of the

k-complexity since we cannot backpropagate with respect to a maximum. See section III-D for more details.

C. CycleGAN and StarGAN

In the real world, it's impossible to get both good blockings and bad blockings of the same neighborhood, which means traditional supervised methods cannot be applied here. This general problem of unpaired data is actually a well developed area for style transferring, where different styles of the same content do not exist in the real world. Then with a dataset with blockings of different neighborhoods, we can manually separate them into different styles based on what metric we want to use. Then we can make the content information to be the general mapping of the neighborhood. The target is to achieve style transferring from a bad neighborhood blocking to a new blocking for the same neighborhood while not changing the neighborhood's rough map. With this setting, we can use the same techniques for general style transferring. More details of what possible styles we can set and how we apply style transferring will be covered in III-D2.

[9] defines a generative model for each one-to-one style transfer task, so m styles need $m(m-1)$ directed generative models to make sure the possibility of transferring from any style to another. By defining these models, it's possible to train the style transferring in a cyclic manner which can transfer back what's already transferred. The cycle consistency loss is exactly used to make sure the transferred sample still contains the content information while the style is changed.

However, if m is relatively large, $m(m-1)$ models are a lot and the training has be to done pairwise. Obviously, this would waste a lot of computation. Thus for the settings with multiple styles, StarGAN [10] is a more preferable training mechanism. Instead of having $m(m-1)$ models, StarGan simply uses one model to do all style transferring by taking a target style vector c into the network. c is just the one-hot vector to represent all kinds of styles. Here to make StarGan work, we have two discriminators D_{src} and D_{cls}. While D_{src} is just the common discriminator to output probability of a sample is real, $D_{cls}(c|x)$ outputs the probability of x is in style c. The objective function for training StarGAN is shown in III-D3.

D. Our Approach

1) k-Complexity Revisited: As discussed previously, the methods proposed in Belsford *et. al* [3] for calculating the k-complexity of a neighborhood block graph cannot be distilled into a differentiable function for backpropagation. In order to achieve this, we propose a neural network which maps from our neighborhood block graphs to a k-complexity labeling. This network will be pre-trained such that such that it can reliably produce this labeling.

2) Our Model: As we mentioned in III-C, we want to consider the reblocking problem as a style transferring problem because parallel data is impossible for this problem. Then the problem becomes what metrics we should consider in terms of style and what metric we should directly encode into our loss function. Let's first look at k-complexity. As mentioned in III-D1, k-complexity is actually differentiable, which means it's easy to just encode it as a metric in our loss function to optimize. We'll simply call the k-complexity loss be L_K, which is the summation of k-complexity of all generated blocks. Also, the population density of different blocks could be directly added in the loss to make sure that the population is small for regions with high complexity. This could be implemented by having a dot product of the k-complexity of all blocks with the population of all blocks, and we call this term $L_{density}$.

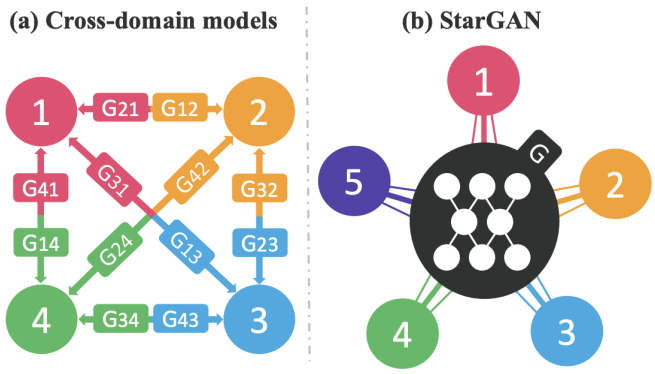

Fig. 3. CycleGAN (a) and StarGAN (b) Pipelines. In (a), CycleGAN needs two generators for each pair of styles, where StarGAN (b) only uses one single generator to transfer to all five possible styles. Source: Choi. *et. al* [10].

Besides these two metrics, other metrics are much harder to quantify. The country code is one example. Different regions have different neighborhood styles so we should encode the country as a style. In this case we can have a country style set $S_r = \{\text{all countries we want to consider as styles}\}$. Besides different neighborhood styles for different countries, we can define all kinds of styles like new roads preferred or canal preferred. In this case we can generate another style set $S_{rc} = \{road, canal\}$. If these are all styles we want to consider and generate new possible blockings to, then we can define all styles to be $S = S_r \times S_{rc}$ and let StarGAN make all kinds of transformations possible.

3) Loss Function: Then we consider the loss function. Below is the standard StarGAN objective [10].

$$L_{adv} = \mathbb{E}_x[\log(D_{src}(x))] + \mathbb{E}_{x,c}[\log(1 - D_{src}(G(x,c)))] \tag{3}$$

Which is the same as the traditional GAN loss 1 introduced by Goodfellow *et. al* [6] and discussed in II-B and is called the adversarial loss L_{adv}.

$$L^r_{cls} = \mathbb{E}_{x,c'}[-\log(D_{cls}(c'|x))] \tag{4}$$

$$L^f_{cls} = \mathbb{E}_{x,c}[-\log(D_{cls}(c|G(x,c)))] \tag{5}$$

Additionally, we need the loss functions L^r_{cls} and L^f_{cls} in order to train D_{cls} to classify real and fake examples, respectively. Finally, L_{rec} is the loss to make sure the generator still keeps the content information.

$$L_{rec} = \mathbb{E}_{x,c,c'}[||x - G(G(x,c),c')||_1] \tag{6}$$

where c and c' are conditional vectors. c' represents x's original style in equation 4 and 6. Putting the above together with the addition of hyperparameters λ_{cls} and λ_{rec} we get the following losses for the discriminator and the generator:

$$L_D = -L_{adv} + \lambda_{cls} L^r_{cls} \tag{7}$$

$$L_G = L_{adv} + \lambda_{cls} L^f_{cls} + \lambda_{rec} L_{rec} \tag{8}$$

However, in our case we want to optimize for k-complexity and population density directly, thus we add two extra terms about L_K and $L_{density}$ (mentioned in III-D2) in the generator loss so that we have which are defined as:

$$L_K = \mathbb{E}[\|K(G(x,c))\|_1] \tag{9}$$

$$L_d = \sum_{v \in V} K(G(x,c))_v \cdot B_v \tag{10}$$

where $K(\cdot)$ is the neural network that computes the k-complexity labeling for a given input neighborhood graph that the model has generated $G(x,c)$ and B is the building count per block labeling and V is the set vertices for our graph. This gives us the final equation for the generator's objective.

$$L_G = L_{adv} + \lambda_{cls} L^f_{cls} + \lambda_{rec} L_{rec} + \lambda_K L_K + \lambda_d L_{density} \tag{11}$$

where λ_K and λ_d are hyperparameters similar to λ_{cls} and λ_{rec}. Note that the hyperparameters λ_K and λ_d can be changed accordingly in order to put more emphasis on one type of criteria for a task or another or even removed altogether if not desired for a specific task.

E. Metrics for Evaluating our Generative Model

1) Travel Distance Matrix: Proposed in Belsford *et. al* [5] the travel distance matrix T_{ij} shows how long it takes to go from one block to another in our neighborhood graph. Ideally, a "good" neighborhood has an average travel distance value that is small, meaning that someone can get around the neighborhood relatively quickly. Thus, the average travel distance metric is

$$\bar{\mathcal{T}} = \frac{1}{n^2} \sum_{i,j}^{n} T_{ij} \tag{12}$$

2) Maximum K-Complexity: Also proposed in Belsford *et. al* [5], the maximum k-complexity of a neighborhood graph k_max is used as a metric to decide if a "good" neighborhood result is acceptable. We plan on using either of the two methods for calculating k-complexity on our result and afterward find the maximum value.

3) Building Count Per Block Weighted K-Complexity: In addition to the maximum k-complexity, one other possible metric we propose is the weighted k-complexity by building counts per block as discussed in previous sections, see III-B and III-D3. This metric is the same as the loss shown in Equation 10 except that the neural network $K(\cdot)$ is not used and instead one of the two methods for calculating k-complexity is used in its place. The idea behind this metric is that assuming each building has on average the same people, this will measure the access on a population density basis.

IV. Conclusion

Urban planning or neighborhood reblocking is extremely important in the modern world, especially considering there are still so many slum neighborhoods in different countries. So how can we utilize generative AI techniques to improve the neighborhood designs? We propose a novel pipeline with StarGAN to be able to create multiple new reblocking designs according to different metrics and requirements without using parallel data. We believe our method is able to generate possible good designs in diversity and thus provide new opportunities for reblocking the neighborhoods.

References

[1] S. Bond-Taylor, A. Leach, Y. Long, and C. G. Willcocks, "Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models," *arXiv preprint arXiv:2103.04922*, 2021.

[2] P. Das and L. R. Varshney, "Explaining artificial intelligence generation and creativity: Human interpretability for novel ideas and artifacts," *IEEE SIGNAL PROCESSING MAGAZINE, 1053-5888*, 2022.

[3] C. Brelsford, T. Martin, J. Hand, and L. M. Bettencourt, "Toward cities without slums: Topology and the spatial evolution of neighborhoods," *Science advances*, vol. 4, no. 8, p. eaar4644, 2018.

[4] K. Mouratidis, "Urban planning and quality of life: A review of pathways linking the built environment to subjective well-being," 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0264275121001293

[5] C. Brelsford, T. Martin, and L. M. Bettencourt, "Optimal reblocking as a practical tool for neighborhood development," *Environment and Planning B: Urban Analytics and City Science*, vol. 46, no. 2, pp. 303–321, 2019.

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[7] L. Girin, S. Leglaive, X. Bie, J. Diard, T. Hueber, and X. Alameda-Pineda, "Dynamical variational autoencoders: A comprehensive review," *arXiv preprint arXiv:2008.12595*, 2020.

[8] X. Guo and L. Zhao, "A systematic survey on deep generative models for graph generation," *arXiv preprint arXiv:2007.06686*, 2020.

[9] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2017. [Online]. Available: https://arxiv.org/abs/1703.10593

[10] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation," 2017. [Online]. Available: https://arxiv.org/abs/1711.09020