A FastMap-Based Encoder-Decoder Architecture and Its Applications

No Author Given

No Institute Given

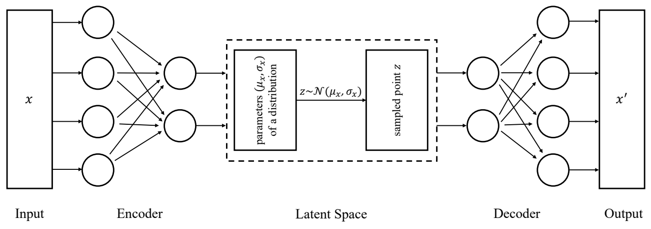
Abstract. FastMap is a linear-time algorithm that embeds a given set of complex objects into a Euclidean space while approximately preserving domain-specific distances between them. It has numerous applications and can be viewed as an efficient encoder from a Machine Learning (ML) perspective. In this paper, we propose a "decoder" counterpart to FastMap and present a FastMap-based encoder-decoder (FMED) architecture. While FMED is inspired by generative ML architectures such as the variational autoencoder (VAE), it is based on comparing pairs of objects via a distance function instead of learning the characteristics of individual objects. Hence, FMED is advantageous when a good domain-specific distance function on pairs of objects is readily available: In such cases, it not only requires a significantly smaller amount of training time and data but also supports more expressive queries compared to VAEs. We discuss some potential applications of our FMED architecture and show a concrete application, in which we efficiently generate new kinds of environment maps as testbeds for robot path-finding algorithms.

Keywords: FastMap · Encoders and Decoders · Generative Models.

1 Introduction

FastMap [3] is a Data Mining algorithm that embeds complex objects—like audio signals, seismograms, DNA sequences, electrocardiograms, or magnetic-resonance images—into a K-dimensional Euclidean space, for a user-specified value of K and a user-supplied distance function $D(\cdot, \cdot)$ on pairs of objects. The Euclidean distance between any two objects in the embedding approximates the domain-specific distance between them. Therefore, similar objects, as quantified by $D(\cdot, \cdot)$, map to nearby points in Euclidean space while dissimilar objects map to distant points. Although FastMap preserves $O(N^2)$ pairwise distances between N objects, it generates the embedding in only $O(KN)$ time.

The embedding generated by FastMap enables geometric interpretations, algebraic manipulations, and downstream Machine Learning (ML) algorithms. Moreover, because of its efficiency, FastMap has already found numerous real-world applications, including in Data Mining [3], shortest-path computations [1], community detection and block modeling [6], centrality computations on the vertices of large networks [7], and in solving many other combinatorial optimization problems on graphs [8].



x
parameters (μ_x, σ_x) of a distribution
$z \sim \mathcal{N}(\mu_x, \sigma_x)$
sampled point z
x'
Input
Encoder
Latent Space
Decoder
Output

Fig. 1. Shows the schematic representation of a VAE. The encoder transforms an input object x to the parameters (μ_x, σ_x) of a normal distribution in the latent space. The decoder transforms a point z sampled from this distribution to an object x' that is intended to match x.

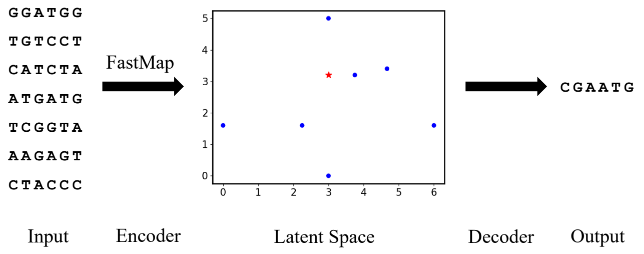


Fig. 2. Shows the schematic representation of our FMED architecture. The input space contains complex objects such as DNA strings. FastMap acts as an encoder and represents these objects as (blue) points in Euclidean latent space. The decoder proposed in this paper transforms a (red) point in the latent space to an object in the input space.

From an ML perspective, the various applications of FastMap stem from its ability to serve as an efficient "encoder". In general, encoder-decoder architectures are used extensively in generative AI. An encoder transforms an input object to a point in latent space; and a decoder transforms a point in latent space to an object. An autoencoder is a special case of an encoder-decoder, in which the output is intended to match the input. Hence, a point in the latent space of an autoencoder that corresponds to an object serves as a compact representation of that object and can be used in lieu of it for a variety of tasks.

In a variational autoencoder (VAE) [5], the encoder transforms an input object to the parameters of a distribution in latent space; and the decoder transforms a sample point from the latent space to an object. Figure 1 shows a VAE that uses a neural network as an encoder and another neural network as a decoder. A distribution in the latent space is usually a normal distribution with a centroid and a covariance matrix. VAEs are very useful for data generation: They have been used in a wide range of applications such as image generation [5, 11] and molecular structure reconstruction [9].

In this paper, we present a new encoder-decoder architecture for generative ML, which is inspired by VAEs but based on FastMap. In doing so, we first note that FastMap can already be viewed as a special kind of encoder that transforms an input object to a point in Euclidean latent space. Hence, we propose only the necessary "decoder" counterpart to FastMap. Figure 2 shows the resulting FastMap-based encoder-decoder (FMED) architecture.

While the FMED architecture of Figure 2 resembles the VAE architecture of Figure 1, the former bears some critical differences and important advantages over the latter. One critical difference is that FMED is based on comparing *pairs* of objects via a distance function instead of learning the characteristics of *individual* objects. Another critical difference is that FMED preserves the domain-specific distances between pairs of objects as straight-line distances between them in the Euclidean latent space, a property that is not featured in VAEs.

These critical differences impart certain advantages to FMED over VAEs. First, since FMED is based on comparing pairs of objects, it is more advantageous when a good domain-specific distance function is readily available. In fact, this difference is similar to the difference between FastMapSVM and other ML frameworks: FastMapSVM [12, 13] is a combination of FastMap and Support Vector Machines (SVMs), which also works by comparing pairs of objects via a domain-specific distance function instead of learning the characteristics of individual objects. Hence, FMED exhibits the same advantage as that of FastMapSVM over other ML frameworks, namely, that it requires a significantly smaller amount of training time and data across a wide range of applications.

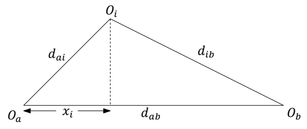
Second, since FMED preserves the domain-specific distances between pairs of objects in the Euclidean latent space, it is able to support more expressive queries compared to VAEs. For example, FMED can generate a complex object that resembles the target objects O_1, O_2, and O_3 in the proportions 30%, 45%, and 25%, respectively. This capability of FMED is particularly useful for "mixing" objects from different categories. VAEs are unable to answer such queries since their latent space is highly nonlinear: They are unable to retrieve the weighted mean of O_1, O_2, and O_3 in the latent space.

In the experimental section of this paper, we showcase some of the above advantages of the FMED architecture over VAEs. We show that FMED can be used to efficiently generate new kinds of environment maps as testbeds for robot path-finding algorithms.

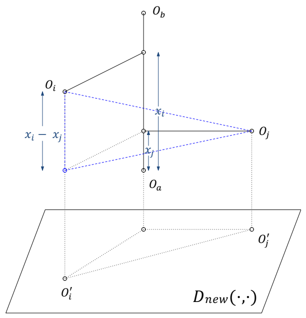
2 Background

FastMap [3] embeds a collection of complex objects in an artificially created Euclidean space. It gets as input a collection of complex objects $\mathcal{O}$, where $D(O_i, O_j)$ represents the domain-specific distance between objects $O_i, O_j \in \mathcal{O}$.[1] It generates a Euclidean embedding that assigns a K-dimensional point $\mathbf{p}_i \in \mathbb{R}^K$ to each object O_i. A good Euclidean embedding is one in which the Euclidean distance $\|\mathbf{p}_j - \mathbf{p}_i\|_2$ between any two points $\mathbf{p}_i$ and $\mathbf{p}_j$ closely approximates $D(O_i, O_j)$.

[1] $D(\cdot, \cdot)$ is assumed to be a non-negative symmetric function that yields 0 for identical objects.



(a) the "cosine law" projection in a triangle



(b) projection onto a hyperplane that is perpendicular to $\overline{O_aO_b}$

Fig. 3. Illustrates how the Euclidean coordinates are computed and recursion is carried out in FastMap; borrowed from [1].

In the first iteration, FastMap heuristically identifies the farthest pair of objects O_a and O_b in linear time. Once O_a and O_b are determined, every other object O_i defines a triangle with sides of lengths $d_{ai} = D(O_a, O_i)$, $d_{ab} = D(O_a, O_b)$, and $d_{ib} = D(O_i, O_b)$, as illustrated in Figure 3a. The lengths of the sides of the triangle define its entire geometry, and the projection of O_i onto the line $\overline{O_aO_b}$ is given by

$$x_i = (d_{ai}^2 + d_{ab}^2 - d_{ib}^2)/(2d_{ab}). \tag{1}$$

FastMap sets the first coordinate of $\mathbf{p}_i$, the embedding of O_i, equal to x_i. In the subsequent $K-1$ iterations, the same procedure is followed for computing the remaining $K-1$ coordinates of each object; however, the distance function is adapted for each iteration. For example, for the first iteration, the coordinates of O_a and O_b are 0 and d_{ab}, respectively. Because these coordinates fully explain the true distance between these two objects, from the second iteration onward, the rest of $\mathbf{p}_a$ and $\mathbf{p}_b$'s coordinates should be identical. Intuitively, this means that the second iteration should mimic the first one on a hyperplane that is perpendicular to the line $\overline{O_aO_b}$. Figure 3b illustrates this. Although the hyperplane is never explicitly constructed, it conceptually implies that the distance function for the second iteration should be changed for all i and j in the following way:

$$D_{new}(O'_i, O'_j)^2 = D(O_i, O_j)^2 - (x_i - x_j)^2, \tag{2}$$

in which O'_i and O'_j are the projections of O_i and O_j, respectively, onto this hyperplane, and $D_{new}(\cdot,\cdot)$ is the new distance function. The distance function is recursively updated according to Equation 2 at the beginning of each of the $K-1$ iterations that follow the first.

In each of the K iterations, FastMap heuristically finds the farthest pair of objects according to the distance function defined for that iteration. These objects are called pivots and are stored as reference objects. There are very few, that is, $\leq 2K$, reference objects. Technically, finding the farthest pair of objects in any iteration takes $O(N^2)$ time. However, FastMap uses a linear-time "pivot changing" heuristic [3] to efficiently and effectively identify a pair of objects O_a and O_b that is very often the farthest pair. It does this by initially choosing a random object O_b and then choosing O_a to be the farthest object away from O_b. It then reassigns O_b to be the farthest object away from O_a, reassigns O_a to be the farthest object away from O_b, and so on, until convergence or a maximum of C iterations, for a small constant $C \leq 10$.

3 A FastMap-Based Encoder-Decoder

In this section, we propose a decoder counterpart to FastMap to complete the FMED architecture. Similar to the decoder of a VAE, our decoder transforms a query point in the latent space to an object in the input space. First, we formalize the task of the decoder as an optimization problem. Second, we study this optimization problem under efficiency considerations and propose various improvements. Third, we discuss the classification capabilities of FMED as an upshot of enabling SVMs in the latent space. Fourth, we discuss the data generation capabilities of FMED with an emphasis on the more expressive queries that it can support compared to VAEs.

3.1 Formalization of the Decoder

Our decoder can be understood by revisiting the functionality of FastMap. Given a collection of objects $\mathcal{O}$, we note that FastMap attempts to preserve the domain-specific distances between pairs of objects as straight-line distances between their Euclidean embeddings by heuristically minimizing the total distortion given by

$$\sum_{O_i, O_j \in \mathcal{O}:\ O_i \neq O_j} \left(D(O_i, O_j) - \|\mathbf{p}_i - \mathbf{p}_j\|_2\right)^2. \tag{3}$$

Now, suppose that one more object $\tilde{\mathbf{O}}$ is added to $\mathcal{O}$; and suppose that the FastMap embedding of $\tilde{\mathbf{O}}$ is $\tilde{\mathbf{p}}$. The distortion that FastMap attempts to minimize for the collection of objects $\mathcal{O} \cup \{\tilde{\mathbf{O}}\}$ is of the same form as Equation 3 but has the additional terms

$$\sum_{O_i \in \mathcal{O}} \left(D(O_i, \tilde{\mathbf{O}}) - \|\mathbf{p}_i - \tilde{\mathbf{p}}\|_2\right)^2. \tag{4}$$

Hence, if $\tilde{\mathbf{p}}$ is a query point for our decoder in the Euclidean latent space created by FastMap, the appropriate choice for $\tilde{\mathbf{O}}$ is given by

$$\arg\min_{\mathbf{O} \in \mathbb{O}} \sum_{O_i \in \mathcal{O}} \left(D(O_i, \mathbf{O}) - \|\mathbf{p}_i - \tilde{\mathbf{p}}\|_2\right)^2. \tag{5}$$

Here, $\mathbb{O}$ is the input *space*, which we note is different from the *collection* of input objects $\mathcal{O}$.

From the above arguments, we can conclude that our decoder is essentially an *optimization algorithm* that solves the optimization problem in Equation 5.

3.2 Efficiency Considerations for the Decoder

Although the objective function is as specified in Equation 5, the optimization algorithm encapsulated in the decoder should be practically viable and efficient. Two strategies can be used towards this end.

Simplifying the Objective Function. We note that, if N is too large, then the number of terms in the objective function may be unwieldy. Hence, we may instead choose to minimize a truncated version of the objective function in Equation 5. One such truncated version can be formulated as follows:

$$\arg\min_{\mathbf{O}\in\mathbb{O}} \sum_{\substack{O_i\in\mathcal{S}\\ \mathcal{S}\subseteq\mathcal{O}:\ |\mathcal{S}|=k}} \left(D(O_i,\mathbf{O}) - \|\mathbf{p}_i - \tilde{\mathbf{p}}\|_2\right)^2 . \tag{6}$$

Here, $\mathcal{S}$ is a subset of the collection of input objects $\mathcal{O}$; and the cardinality of $\mathcal{S}$ is $k \leq N$, for a user-specified value of k. Hence, Equation 6 considers only k input objects instead of all the N input objects.

While choosing k to be significantly smaller than N increases the efficiency of the decoder, doing so also incurs the cost of decreasing its effectiveness. In particular, it is undesirable to fix the subset $\mathcal{S}$ a priori, since doing so would be equivalent to conspicuously ignoring the remaining objects in the input. Such a strategy would not harness the resourcefulness of all the available data. Hence, for a user-specified value of k, $\mathcal{S}$ must be chosen judiciously to achieve both efficiency and effectiveness of the decoder. We propose that a good choice for $\mathcal{S}$ is the set of k nearest neighbors of the query point $\tilde{\mathbf{p}}$. There are two reasons for this choice, as discussed below.

First, this choice for $\mathcal{S}$ does not ignore any objects in the input: Instead, it simply makes $\mathcal{S}$ pliable and dependent on $\tilde{\mathbf{p}}$, while all objects and their point representations in the Euclidean latent space are still retained to be able to process any other query. Moreover, making $\mathcal{S}$ pliable and dependent on $\tilde{\mathbf{p}}$ readily harnesses the resourcefulness of all the available data by commensurately refining the neighborhoods of the objects in the latent space.

Second, it is easy to retrieve the k nearest neighbors of a query point $\tilde{\mathbf{p}}$. This is because the latent space of FMED is Euclidean and preserves the domain-specific distances between pairs of objects as straight-line distances between them. In Euclidean space, the problem of retrieving the k nearest neighbors of a query point is known to admit a very efficient algorithm that is based on Locality Sensitive Hashing (LSH) [2].

Approximation and Heuristic Algorithms. Generally speaking, the optimization problem in Equation 5—or, for that matter, in Equation 6—can be either polynomial-time solvable or NP-hard. When the problem is NP-hard, it may still be possible to invoke practically viable approximation and/or heuristic algorithms for solving it. However, the scope of such possibilities depends on the exact nature of the optimization problem that becomes evident only after it is grounded in a given specific domain.

The best choice of the optimization algorithm, whether it solves the problem exactly, approximately, or heuristically, can also be of many different kinds: an Integer Linear Programming (ILP) solver, a local search algorithm, or a genetic algorithm, to name a few. In each case, some domain-specific knowledge is required: This is similar to the domain-specific function that is required by the encoder, that is, the function $D(\cdot,\cdot)$ required by FastMap. If the decoder optimization algorithm employs an ILP solver, we need domain-specific knowledge on how to represent an object in the ILP formulation. If it uses local search, we need domain-specific knowledge on what constitutes the local search "move" operators in the input space. If the decoder uses a genetic algorithm, we need domain-specific knowledge on how to "mutate" an object and how to "cross" two objects; and so on.

3.3 Classification Capabilities

Since the latent space of FMED is Euclidean—in which the domain-specific distances are preserved as straight-line distances—it enables the applicability of SVMs and kernel methods for classification tasks on complex objects. In fact, FastMapSVM [12, 13] elegantly combines the strengths of FastMap and SVMs. In the first phase, it invokes FastMap to generate the point representations of the input objects in a Euclidean latent space. In the second phase, it invokes SVMs and kernel methods for learning to classify the points in this Euclidean latent space. FastMapSVM has several advantages over other methods.

First, FastMapSVM leverages domain-specific knowledge via a distance function instead of relying on complex ML models to infer the underlying structure in the data entirely. Second, it facilitates interpretability and explainability. In fact, it even provides a perspicuous visualization of the objects and the classification boundaries between them [12]. Third, since FastMapSVM obviates the need to learn a complex transformation of the input objects, it uses significantly smaller amounts of time and data for model training compared to other ML algorithms. Fourth, since SVMs and kernel methods require the objects to be represented as points in a Euclidean space for classification tasks, FastMapSVM extends their applicability to complex objects by representing these objects as points in a low-dimensional Euclidean latent space.

3.4 Data Generation Capabilities

With the decoder in place, FMED has all the standard benefits of VAEs for data generation. In fact, in Section 5, we not only showcase the data generation capabilities of FMED but also demonstrate its superior efficiency and quality of results compared to VAEs. Similar to VAEs, FMED can transform any chosen point in the latent space to an object in the input space. Such a point or a collection of such points can be chosen either randomly to generate diverse objects or in a certain neighborhood to generate similar objects but with some variance.

In the context of data generation, when $k = 1$, FMED has a zero reconstruction loss for the objects in $\mathcal{O}$ but a potentially non-zero reconstruction loss for the objects not in $\mathcal{O}$. In any case, $k = 1$ is not a particularly good choice since it significantly under-approximates the objective function in Equation 5. Hence, $k \approx 10$ is a much more practically viable alternative.

In comparison to VAEs, FMED can support more expressive queries. These capabilities of FMED stem from the fact that its Euclidean latent space preserves the domain-specific distances between pairs of objects as straight-line distances between them. In other words, FMED's Euclidean latent space is "linear". Hence, FMED can support queries for mixed objects that resemble target objects, say, O_1, O_2, and O_3, in various proportions, say, 30%, 45%, and 25%, respectively. Suppose $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$ are the point representations of the objects O_1, O_2, and O_3, respectively, in the latent space. In FMED, the query point $\tilde{\mathbf{p}} = 0.30\mathbf{p}_1 + 0.45\mathbf{p}_2 + 0.25\mathbf{p}_3$ is expected to retrieve the same weighted mean of O_1, O_2, and O_3, since the Euclidean latent space is linear. However, in VAEs, the latent space is highly nonlinear, because of which the same query point does not necessarily retrieve the desired weighted mean of O_1, O_2, and O_3.

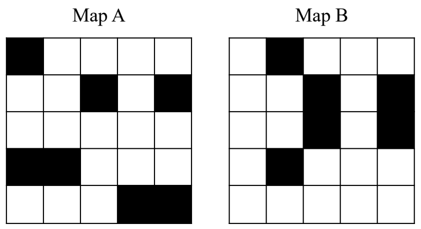
4 An Example Application Domain

In this section, we discuss an example application domain that consists of environment maps used as testbeds for single-robot and multi-robot path-finding algorithms. In this domain, there are various categories of maps that have been traditionally used for benchmarking: Not surprisingly, different single-robot and multi-robot path-finding algorithms have demonstrated different strengths and weaknesses in the various categories. Hence, from an evaluation perspective, it is very useful to generate new maps on demand that mix the characteristics of the various categories in specified proportions.

Below, we formally identify the objects in the application domain, design an appropriate distance function on pairs of objects, discuss how to efficiently solve the decoder's optimization problem, and present an evaluation metric for mixing specified objects. Since environment maps used to test path-finding algorithms are typically 2-dimensional grid-world maps with free cells and obstacle cells, they can also be interpreted as 2-dimensional images with black and white pixels. Hence, some of the discussion that follows can also be made relevant to image processing after sufficient generalization.

4.1 Objects

We identify the objects in our domain as 64×64 2-dimensional grid-world maps. Each cell in such a map is either a free cell, sometimes indicated by a '0', or an obstacle cell, sometimes indicated by a '1'. We derive the objects from the Moving AI dataset [10]. Since the Moving AI dataset contains maps of different sizes, we standardize them as follows. We first ignore maps with less than 64 cells height-wise or width-wise. We then randomly crop a contiguous area of 64×64 cells from the remaining maps, sometimes generating multiple objects from one map. There are five categories of maps: 'Game', 'Street', 'Warehouse', 'Maze', and 'Room'.



Matrix A

1	0	0	0	0
0	0	1	0	1
0	0	0	0	0
1	1	0	0	0
0	0	0	1	1

Matrix B

0	1	0	0	0
0	0	1	0	1
0	0	1	0	1
0	1	0	0	0
0	0	0	0	0

Vector A: [1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1]

Vector B: [0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

$\mu_A = 0.28$; $\mu_B = 0.24$; $\sigma_A = 0.45$; $\sigma_B = 0.43$; $r(A,B) = \frac{1}{25} \times \frac{1.32}{0.19} = 0.275$; $D(A,B) = 1 - r(A,B) = 0.725$

Fig. 4. Illustrates how the distance between two objects is computed in our example application domain. The 64×64 grid-world maps are reduced to 5×5 grid-world maps for ease of exposition. The matrices A and B correspond to the maps A and B, respectively. In turn, these are vectorized to vectors A and B, respectively. The distance between the two objects is then derived from the Pearson similarity of the two vectors.

4.2 Distance Function

Because the objects in our domain can be interpreted as vectors, matrices, or images, there are several kinds of distance functions that we can design. However, for simplicity of exposition, we define the distance between two objects to be 1 minus the Pearson similarity between them.

The Pearson similarity, or the Pearson correlation, measures the linear correlation between two equal-sized arrays of real numbers. If $\mathbf{u} = \langle u_1, u_2 \dots u_L \rangle$ and $\mathbf{v} = \langle v_1, v_2 \dots v_L \rangle$ are two arrays of size L, the Pearson correlation r is calculated as follows:

$$r(\mathbf{u}, \mathbf{v}) = \frac{1}{L} \sum_{i=1}^{L} \frac{(u_i - \mu_{\mathbf{u}})(v_i - \mu_{\mathbf{v}})}{\sigma_{\mathbf{u}} \sigma_{\mathbf{v}}}, \tag{7}$$

where $\mu_{\mathbf{u}}$ and $\mu_{\mathbf{v}}$ are the means of $\mathbf{u}$ and $\mathbf{v}$, respectively, and $\sigma_{\mathbf{u}}$ and $\sigma_{\mathbf{v}}$ are their respective standard deviations. The Pearson similarity is guaranteed to be in the interval $[-1, 1]$.

An object in our domain can be interpreted as a 64×64 matrix of 0s and 1s. It can also be vectorized and interpreted as an array of size $4,096$. Hence, given two objects, the Pearson similarity between them is well defined. Consequently, the distance function, equated to 1 minus the Pearson similarity function, is also well defined and is guaranteed to be in the interval $[0, 2]$. Figure 4 illustrates how the distance between two objects is computed.

4.3 The Decoder's Optimization Problem

To suit our application domain, we first formulate the optimization problem of Equation 6 as a well-studied constrained optimization problem. We then invoke a powerful off-the-shelf solver for efficiently solving it.

Since the decoder outputs a 64×64 matrix of 0s and 1s, we first create $4,096$ Boolean variables X_i, for $1 \leq i \leq 4096$, to represent a candidate target object $\mathbf{O}$. In doing so, we assume that the matrix representation of an object is vectorized row-wise to $4,096$ bits. We then create three continuous variables μ, V, and σ to represent the mean, variance, and the standard deviation, respectively, of these Boolean variables. Moreover, we create continuous variables $d_1, d_2 \ldots d_k$, where each d_i, for $1 \leq i \leq k$, represents $D(O_i, \mathbf{O})$. Here, $\mathcal{S} = \{O_1, O_2 \ldots O_k\}$ is the set of objects that correspond to the k nearest neighbors of the query point $\tilde{\mathbf{p}}$.

While the optimization space involves all the above variables, the objective function can be stated using the variables $d_1, d_2 \ldots d_k$ as follows:

$$\underset{\substack{X_1, X_2 \ldots X_{4096} \\ \mu, V, \sigma, d_1, d_2 \ldots d_k}}{\arg\min} \sum_{i=1}^{k} \left(d_i - \|\mathbf{p}_i - \tilde{\mathbf{p}}\|_2\right)^2 . \tag{8}$$

Here, we note that $\tilde{\mathbf{p}}$ and $\mathbf{p}_1, \mathbf{p}_2 \ldots \mathbf{p}_k$ are known quantities: $\tilde{\mathbf{p}}$ is the given query point; and $\mathbf{p}_1, \mathbf{p}_2 \ldots \mathbf{p}_k$ are its k nearest neighbors returned by LSH.

The constraints of the problem can be stated as follows:

$$\mu = \frac{1}{4096} \sum_{i=1}^{4096} X_i, \tag{9}$$

$$V = \frac{1}{4096} \sum_{i=1}^{4096} (X_i - \mu)^2, \tag{10}$$

$$V = \sigma^2, \tag{11}$$

$$\sigma \geq 0, \tag{12}$$

$$\sigma d_i = \sigma - \frac{1}{4096} \sum_{j=1}^{4096} \frac{(X_j - \mu)(v_{i_j} - \mu_{\mathbf{v}_i})}{\sigma_{\mathbf{v}_i}} \qquad \forall 1 \leq i \leq k. \tag{13}$$

Here, $\mathbf{v}_i = \langle v_{i_1}, v_{i_2} \ldots v_{i_{4096}} \rangle$ is the vectorized form of object O_i. $\mu_{\mathbf{v}_i}$ and $\sigma_{\mathbf{v}_i}$ are the mean and standard deviation of $\mathbf{v}_i$, respectively. $\mathbf{v}_i$, $\mu_{\mathbf{v}_i}$, and $\sigma_{\mathbf{v}_i}$ are known quantities.

Since both the objective function and the constraints are quadratic in nature, the overall constrained optimization problem is a Quadratically Constrained Quadratic Program (QCQP). While QCQPs are generally NP-hard to solve optimally, the size of our problem is quite manageable. In fact, it can be solved fairly efficiently using Gurobi [4], a state-of-the-art solver for ILPs, QCQPs, and many other optimization programs.

4.4 Evaluation Metric

Suppose that we are required to generate an object $\tilde{\mathbf{O}}$ as the weighted combination of the specified objects $O_1, O_2 \ldots O_q$. That is, suppose that we are required to generate the object $\sum_{i=1}^{q} w_i O_i$, for $\sum_{i=1}^{q} w_i = 1$ and $\forall i \; w_i \geq 0$. Both VAEs and FMED may be able to generate $\tilde{\mathbf{O}}$ by first substituting the specified objects in the weighted sum with their point representations in latent space and then decoding the result. In fact, $\tilde{\mathbf{O}}$ may conceivably be generated in any other way, perhaps even by working directly on the specified objects. In any case, an evaluation metric is required to measure how good the generated $\tilde{\mathbf{O}}$ is.

Towards this end, we propose the following evaluation score:

$$\sum_{i=1}^{q} w_i D(O_i, \tilde{\mathbf{O}})^2. \tag{14}$$

The value of this score decreases with increasingly better results.

Moreover, there is an intuitive reason for the proposed form of the evaluation score: This is apparent when the objects are construed as points and the distances between them are the natural straight-line distances. On the one hand, the query is a weighted mean of the specified points. On the other hand, the evaluation score is the weighted sum of the squared distances of a candidate point to the specified points: The optimal value of the candidate point for minimizing this score is also the weighted mean of the specified points. This equivalence between the query and the optimal value of the candidate point in the evaluation score justifies the form of Equation 14.

5 Experimental Results

In this section, we present experimental results drawn from the application domain discussed in Section 4. We present two sets of experimental results. First, we demonstrate the impressive classification capabilities of FMED using FastMapSVM. Second, we demonstrate the superior data generation capabilities of FMED in comparison to a standard VAE. For a fair comparison of FMED and the VAE, we compared their performances using their best setting of the dimensionality of their latent space: We used 32 dimensions for the Euclidean latent space of FMED and 128 dimensions for the latent space of the VAE. In the decoder part of FMED, we used $k = 10$.

In both sets of experiments, we used 200 64×64 maps from each of the five categories as training instances. Hence, there were a total of $1,000$ training instances. In the experiments with FastMapSVM, we used an additional 40 maps from each of the five categories as test instances. Hence, there were a total of 200 test instances in the first set of experiments.

In the second set of experiments, we designed our VAE architecture as follows. The encoder part of the VAE consists of six ReLU convolution layers followed by two ReLU linear layers. Each convolution layer has 32 3×3 filters. Their strides are 1, 1, 2, 1, 2, and 1, respectively. The two linear layers have 128 units each, for the mean and covariance, respectively, of each latent variable's normal distribution. The decoder part of the VAE consists of one ReLU linear layer followed by six ReLU deconvolution layers. The linear layer has 128 units because the latent space has 128 dimensions. Each deconvolution layer has 32

Table 1. Shows the performance of FastMapSVM as a classifier on the five categories of maps. The first five rows are for binary one-vs.-rest classification tasks. The last row is for the overall five-category multi-class classification task. The mean and variance are reported on five trials for the accuracy, recall, precision, and the F1 score.

Category	Accuracy		Recall		Precision		F1	
	Mean	Var	Mean	Var	Mean	Var	Mean	Var
Game	84.5	2.03	77.0	4.38	90.6	5.16	83.5	1.29
Street	87.3	0.31	74.5	1.25	100.0	0.00	85.4	0.54
Warehouse	98.5	1.09	99.0	1.88	98.0	1.22	98.5	1.10
Maze	97.3	0.31	94.5	1.25	100.0	0.00	97.2	0.36
Room	99.5	0.47	99.5	1.25	99.5	1.19	99.5	0.47
All	88.3	7.95	-	-	-	-	-	-

3×3 filters. Their strides are 1, 2, 1, 2, 1, and 1, respectively. We trained our VAE using $16,000$ epochs on the full batch of training instances.

We implemented all algorithms in Python3. We ran FMED and FastMapSVM on an Apple M2 chip laptop with an 8-core 3.5 GHz CPU and 16 GB RAM. The training time for FMED was only about 44 seconds. We trained the VAE on an x86 64 GNU/Linux machine with an i7-8700 8-core 3.2 GHz CPU and 64 GB RAM.[2] The training time for the VAE was about 36 hours.

5.1 Classification Capabilities

In our first set of experiments, we test the classification capabilities of FMED. We do so by invoking FastMapSVM on the point representations of the objects in FMED's Euclidean latent space. If indeed the pairwise distances between objects are well preserved in this space, similar objects are expected to map to nearby points and dissimilar objects are expected to map to distant points, thereby enabling FastMapSVM to readily identify the classification boundaries between the five categories of training instances.

While we used the same distance function on pairs of objects, as described in Section 4.2, we rotated one object with respect to the other by $0°$, $90°$, $180°$, and $270°$ for the best possible alignment: We choose the rotation angle that yields the minimum distance between the two objects.

Table 1 shows the performance results of FastMapSVM on maps of all the five categories. We invoked bagging on FastMapSVM with five estimators. For the multi-class classification task, we used 200 instances of each category for training and an additional 40 instances of each category for testing. For the individual binary one-vs.-rest classification tasks, we used 200 instances of one identified category and 50 instances of each of the remaining four categories for training. We used an additional 40 instances of the identified category and an additional 10 instances of each of the remaining four categories for testing.

The high-quality performance of FastMapSVM demonstrates the high fidelity of FMED's Euclidean latent space to the input space of objects. As a consequence, it also showcases the high-quality classification capabilities of FMED.

[2] We used a slightly more powerful machine for training the VAE since its training time is about $3,000$ times more than that of FMED.

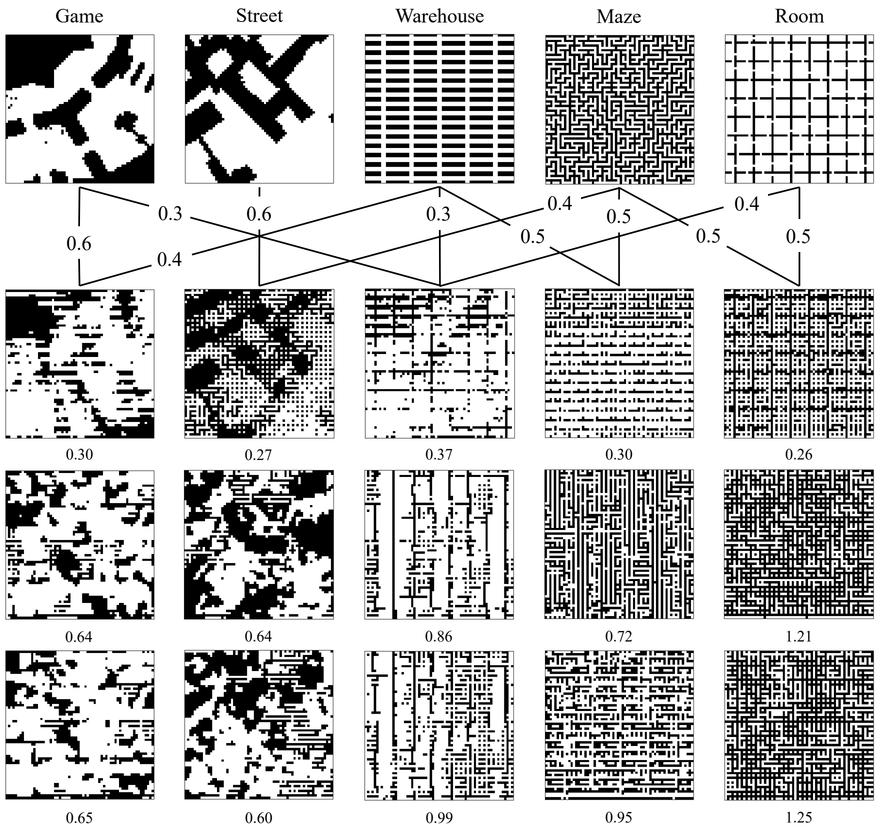


Fig. 5. Compares the data generation capabilities of FMED and the VAE. The top row shows five different input maps, one from each of the five categories. The second row shows five maps generated by FMED. Each map in this row is generated as a mixture of the maps that it is connected to in the top row in the proportions annotating the connections. The third row is similar to the second row but shows the five maps generated by the VAE for the same mixtures of the input maps. The last row is similar to the third row but shows a different sample of the five maps generated by the VAE. Each map generated by FMED or the VAE is annotated with its evaluation score.

5.2 Data Generation Capabilities

In our second set of experiments, we compare the data generation capabilities of FMED against those of the VAE. While the decoders of both FMED and the VAE can transform any query point in the latent space to an object, their overall data generation capabilities can be compared by how well they support more expressive queries. Hence, we focus on queries that generate an object as a weighted combination of other specified input objects. The generated object is then evaluated as discussed in Section 4.4.

Table 2. Shows the performance of FMED and the VAE on different mixture models of queries. Each mixture model first identifies three of the five categories. One map is then chosen from each of these three categories; and a query is formulated to mix the three chosen maps in equal proportions.

Mixture Model of Query	FMED		VAE	
	Mean	Var	Mean	Var
Game-Street-Warehouse	0.60	0.025	0.93	0.002
Game-Street-Maze	0.49	0.013	0.92	0.001
Game-Street-Room	0.60	0.010	0.94	0.003
Game-Warehouse-Maze	0.60	0.010	0.92	0.001
Game-Warehouse-Room	0.53	0.010	0.90	0.002
Game-Maze-Room	0.59	0.012	0.92	0.003
Street-Warehouse-Maze	0.58	0.011	0.98	0.002
Street-Warehouse-Room	0.57	0.017	1.01	0.002
Street-Maze-Room	0.48	0.005	1.00	0.001
Warehouse-Maze-Room	0.54	0.027	1.07	0.002

We used 200 maps from each category as training instances. The training times for FMED and the VAE are as low as 44 seconds and as high as 36 hours, respectively. After training, data generation queries of the form $\sum_{i=1}^{q} w_i O_i$, for $\sum_{i=1}^{q} w_i = 1$ and $\forall i\ w_i \geq 0$, are presented. Given such a query, FMED first computes $\tilde{\mathbf{p}} = \sum_{i=1}^{q} w_i \mathbf{p}_i$, where $\mathbf{p}_i$ is the point representation of object O_i in its Euclidean latent space. It then uses the decoder algorithm to transform $\tilde{\mathbf{p}}$ to the output object. In contrast, the VAE first computes $\tilde{\mathbf{p}} = \sum_{i=1}^{q} w_i \mathbf{p}_i$, where $\mathbf{p}_i$ is the distribution that represents object O_i in its latent space. It then samples points from $\tilde{\mathbf{p}}$ and uses the decoder to transform them to the output objects.

Figure 5 compares FMED and the VAE on queries of the above kind that mix two or more maps. For the chosen input maps, we observe that FMED produces higher-quality results, both in terms of the visual appeal and the evaluation score. In fact, for the queries in the five columns, FMED produces maps with evaluation scores of 0.30, 0.27, 0.37, 0.30, and 0.26, respectively. In contrast, measured over 10 samples, the VAE produces maps with evaluation scores of low variance and mean 0.68, 0.60, 0.95, 0.80, and 1.26, respectively.

Table 2 compares FMED and the VAE more extensively on the same kind of expressive queries. It shows ten mixture models of queries, derived from all possible ways of choosing three of the five categories. Within each mixture model, 10 queries are formulated. Each query identifies one map from each of the three chosen categories and mixes them in equal proportions. For each mixture model, the table reports the mean and variance for FMED, measured over the 10 queries, and the same quantities for the VAE, measured over the 10 queries and 10 sample maps it outputs for each query. We observe that FMED consistently produces higher-quality results compared to the VAE, proving that the linear Euclidean latent space of FMED is advantageous for expressive data generation.

6 Conclusions

In this paper, we proposed a decoder counterpart to FastMap and presented the FMED architecture. FMED is based on comparing pairs of objects via a distance function instead of characterizing individual objects. It not only has data generation capabilities similar to that of VAEs but also supports more expressive queries since it preserves domain-specific distances between objects in its Euclidean latent space. We demonstrated two advantages of FMED for data generation in the path-finding domain. First, FMED requires a significantly smaller training time compared to VAEs. Second, it supports more expressive queries for mixing maps. The maps generated by FMED can serve as new testbeds for path-finding algorithms. In general, FMED is advantageous over VAEs when a good domain-specific distance function on pairs of objects is readily available.

References

1. Cohen, L., Uras, T., Jahangiri, S., Arunasalam, A., Koenig, S., Kumar, T.K.S.: The FastMap Algorithm for Shortest Path Computations. In: Proceedings of the International Joint Conference on Artificial Intelligence (2018)
2. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.: Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In: Proceedings of the Twentieth Annual Symposium on Computational Geometry (2004)
3. Faloutsos, C., Lin, K.I.: FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (1995)
4. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2024), https://www.gurobi.com
5. Kingma, D.P., Welling, M.: Auto-Encoding Variational Bayes. In: Proceedings of the International Conference on Learning Representations (2014)
6. Li, A., Stuckey, P., Koenig, S., Kumar, T.K.S.: A FastMap-Based Algorithm for Block Modeling. In: Proceedings of the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (2022)
7. Li, A., Stuckey, P., Koenig, S., Kumar, T.K.S.: A FastMap-Based Framework for Efficiently Computing Top-K Projected Centrality. In: Proceedings of the Ninth International Conference on Machine Learning, Optimization, and Data Science (2023)
8. Li, J., Felner, A., Koenig, S., Kumar, T.K.S.: Using FastMap to Solve Graph Problems in a Euclidean Space. In: Proceedings of the International Conference on Automated Planning and Scheduling (2019)
9. Liu, Q., Allamanis, M., Brockschmidt, M., Gaunt, A.: Constrained Graph Variational Autoencoders for Molecule Design. In: Proceedings of the Conference on Neural Information Processing Systems (2018)
10. Sturtevant, N.: Benchmarks for Grid-Based Pathfinding. Transactions on Computational Intelligence and AI in Games (2012)
11. Vahdat, A., Kautz, J.: NVAE: A Deep Hierarchical Variational Autoencoder. In: Proceedings of the Conference on Neural Information Processing Systems (2020)
12. White, M., Sharma, K., Li, A., Kumar, T.K.S., Nakata, N.: Classifying Seismograms Using the FastMap Algorithm and Support-Vector Machines. Communications Engineering (2023)
13. Zheng, K., Li, A., Kumar, T.K.S.: FastMapSVM/FastMapSVR for Predictive Tasks on CSPs, SAT, and Weighted CSPs. In: Proceedings of the International Conference on Machine Learning and Applications (2024)