Graph Similarity Learning of Floor Plans

Casper van Engelenburg
Jan van Gemert
Seyran Khademi

Delft University of Technology
Delft, NL

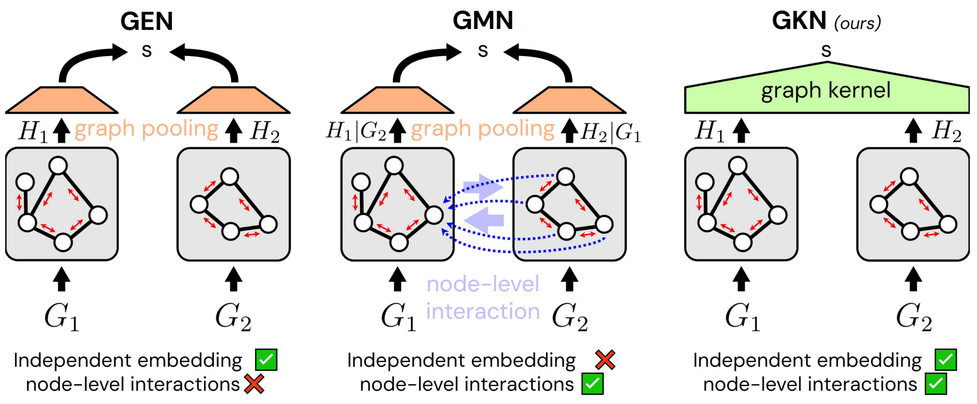

Figure 1: **Joint embeddings architectures for graph similarity learning.** (***Left***) Graph embeddings networks (GENs) independently embed graphs G_1 and G_2 in a vector space using parameter-shared graph networks (gray box) followed by pooling (orange box). The similarity s is computed based on the vector embeddings. (***Middle***) In addition to GENs, graph matching networks (GMNs) model cross-graph node-level interactions (blue arrows) between the two sets of node features in each layer. Embeddings (H_i) depend on the other. (***Right***) Our graph kernel network (GKN) shifts node-level interactions to the similarity function via a graph kernel (green box), enabling independent embeddings while preserving node-level interactions.

Abstract

Floor plans depict building layouts and are often represented as graphs to capture the underlying spatial relationships. Comparison of these graphs is critical for applications like search, clustering, and data visualization. The most successful methods to compare graphs *i.e.*, graph matching networks, rely on costly intermediate cross-graph node-level interactions, therefore being slow in inference time. We introduce **LayoutGKN**, a more efficient approach that postpones the cross-graph node-level interactions to the end of the joint embedding architecture. We do so by using a differentiable graph kernel as a distance function on the final learned node-level embeddings. We show that LayoutGKN computes similarity comparably or better than graph matching networks while significantly increasing the speed. Code and data are open.

1 Introduction

In contrast to the natural environment, the built environment follows a remarkably simple spatial logic – which, more remarkably so, seems to be consistent anywhere on earth. When stripped of its details, our built environment can be adequately described as a hierarchy of networks: the main infrastructure as a network of roads connecting cities and towns; cities as a network of roads that divide and bring about the composition of neighborhoods and buildings; buildings as stacks of floor plans; floor plans as networks of rooms and corridors; etc. Often modeled as graphs of nodes and edges, these networks are embedded in physical space, with nodes and edges carrying information about location, shape, and spatial relations. Encoding such relations in machine-readable form has enabled computer-aided methods to understand [21], analyze [9, 22], and synthesize [8, 19] our built environment. In this work, we focus on floor plans and floor plan representation learning. Nonetheless, most ideas could prove worthy in the other 'levels' of our built environment.

As visual depictions of building layouts, floor plans offer an elementary yet powerful abstraction of spatial structure—an aspect central to the quality of space [1, 10, 28]. Of key importance to floor plan machine understanding is the ability to effectively compare floor plans based on their spatial structure, as well as the ability to do so *quickly* – especially for search-related tasks, data visualization, and clustering or classification. The question of spatial similarity is inherently subjective, multi-faceted, and task-dependent. However, we believe, as do others [12, 20, 21, 24], that floor plan similarity can be adequately modeled as a graph comparison problem, given the availability of the graph through image vectorization.

Most notably, graph matching networks (GMNs) [15] have been successful at 'solving' graph comparison problems, and have been adopted for use in floor plan similarity [12, 21]. A GMN is a form of a joint embedding architecture (JEAs), in which two parameter-sharing neural networks (*i.e.*, encoders) process two distinct data samples simultaneously to arrive at two corresponding embeddings [4, 5]. The similarity is measured based on a distance (*e.g.*, Euclidean) between the embeddings. For GMNs, the encoders are graph neural networks (GNNs). The key distinction between classical JEAs is that, in GMNs, information is exchanged between nodes across both graphs during both training and inference. These cross-graph node-level interactions allow for the explicit modeling of node-level correspondences, which are otherwise thrown away when the graph-level embeddings are computed independently. Although effective, cross-graph node-level interactions do not allow for the offline computation of embeddings for a given database of floorplans, due to the pairwise dependence of the embeddings. Cross-interaction modules hinder the usability of GMNs in real-time retrieval systems where similarity, i.e., the distance between the embeddings, needs to be computed fast. Our contributions enable an effective yet efficient solution for graph-based floor plan retrieval, and are summarized as follows:

- We propose LayoutGKN: a GNN model built on a differentiable graph similarity metric, which effectively captures the spatial similarity in floor plan representation learning.
- We model the node-level interactions at the end of the JEA by using a differentiable path-based graph kernel.
- Experiments on RPLAN and zero-shot generalization to MSD demonstrate that LayoutGKN achieves comparable or better ranking performance than LayoutGMN while being significantly faster.

2 Related works

Floor plan similarity Floor plan retrieval, cast as a problem of similarity learning, in which the goal is to rank a gallery database of floor plans to a query, has been the subject to previous works [12, 13, 20, 21, 24, 26, 29, 30, 33]. Relevant other applications include the retrieval of interface layouts and documents [2, 18, 21, 35] and room layouts [7]. Determining the similarity between two floor plans, as is framed in [21], is a multi-faceted task, regarding semantic (*e.g.* room functions), geometric (*e.g.*, room shapes), and relational (*e.g.*, permeability or adjacency) information layers. The *intersection-over-union* (IoU) is often considered as an instrumental measure for spatial similarity [12, 18, 21]. While relatively quick to compute, IoU is sensitive to translations, rotations, and scale changes – and, most importantly, does not explicitly measure the relational similarity between the components that compose the layout [30]. Similar to most other works [12, 13, 18, 20, 21, 24, 29, 30], we frame floor plan similarity as a graph comparison problem, in which we model floor plans as spatially-attributed graphs of nodes that represent rooms and edges that define their spatial relations. Unlike previous learning-based methods [12, 18, 21] that train and evaluate under the IoU metric, we do so using a graph-based similarity metric, which we show better aligns with human judgment of floorplan similarity.

Graph similarity To measure similarity between graphs, many works resort to the use of a *graph edit distance* (GED) [25], which for example is used to assess compatibility in the generation of floor plans [19], or the *maximum common subgraph* (MCS) used directly for floor plan retrieval [29]. GED is a known NP-complete problem (details in [38]), exponential in the number of nodes, thus hindering the use where similarity needs to be computed fast (*e.g.*, in search engines). To address limitations in efficiency, graph kernels (GKs), such as those defined in [6, 27], have been proposed to solve graph comparison or related problems. We encourage the reader to read a recent review on GKs [14]. While graph kernels are effective for comparing relational structures, a key limitation is their reliance on handcrafted input features, such as node attributes. We use graph kernels to compute similarity between floor plans; however, unlike traditional methods, our approach learns the node features directly from data.

Graph similarity learning In turn, learning-based solutions using *graph neural networks* (GNNs) have been proposed for the problem of comparing graphs [3, 15, 17, 23]. Most notable is the line on *graph matching networks* (GMNs)[15] that explicitly encodes cross-graph node-level interactions in the joint embedding architecture, which captures fine-grained structural correspondences between the nodes. Benchmarks on floor plan similarity, such as *LayoutGMN* [21] extensively use GMNs. Problematic to the use of GMNs is the fact that the node/graph embeddings cannot be computed in isolation [39], drastically decreasing efficiency in real-time retrieval. Instead of modeling expensive cross-graph node-level interactions *across the GNNs*, we only model such interactions at the end of the pipeline using a differentiable path-based graph kernel, *GraphHopper* [6], on top of the final node embeddings. Because the node embeddings can be precomputed independently, our method is much faster than is the case for GMNs (Fig. 1).

3 Problem formulation

3.1 Floor plans as attributed graphs

To explicitly model spatial relations, a floor plan is represented as an undirected graph $G = (\mathcal{V}, \mathcal{E})$ of nodes $u \in \mathcal{V}$ that describe rooms connected by edges $(u,v) \in \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ that describe the permeability between the rooms. We should not forget that the use of the graph as representational device of a floor plan is by no means a new idea, and stems, most notably, from a vast line of works in *Space Syntax* (*e.g.*, as in [10] in the mid 70s) and other works around the same time (*e.g.*, in Alexander's [1] and Steadman's [28] seminal works). In these works, only the graph's connectivity (*i.e.*, its topology) was usually explored. Similar to other works of late [19, 21], we expand the graphs by populating the nodes and edges with a rich set of attributes, which we do as follows. Specifically, each node u is endowed with a package of node features. 1) A one-hot encoding of the category of the room's function $\mathbf{c}^{(u)} \in \mathbb{R}^8$. Categories are: "living room", "bedroom", "kitchen", "bathroom", "dining", "store room", "balcony", "corridor". For example, $[1;0;0;0;0;0;0]$ describes the living room. 2) A vector that characterizes the room's shape $\mathbf{s}^{(u)} \in \mathbb{R}^6$, defined as: $\mathbf{s} = [c_x; c_y; w; h; \sqrt{a}; p/4]$ where (c_x, c_y) denotes the center of the room, l_x the maximum size in x and l_y in y, a the area, and p the perimeter. 3) To make use of the graph kernel, we need for each node the shortest-path histogram matrix $\mathbf{M}^{(u)}$ [6]. A path is a sequence of non-repeated nodes connected through edges present in a graph, which for a floor plan captures how one could walk from one space to another. The shortest path between two nodes is the one which traverses the least edges. The entry $[\mathbf{M}^{(u)}]_{ij}$ counts how often u occurs at the i-th position on shortest paths of length j. We set the size of $\mathbf{M}^{(u)}$ to $\mathbb{R}^{\delta \times \delta}$, in which δ is the maximum path length, usually taken as the longest shortest path i.e., the graph diameter. Since graph diameters of floor plan graphs are relatively small, we set $\delta = 4$. Each edge (u,v) carries a vector $\mathbf{e}^{(u,v)}$ indicating the permeability of two adjacent spaces: $[1;0]$ for access connectivity and $[0;1]$ for adjacent-only.

3.2 Floor plan similarity as graph comparison problem

We formulate the challenge of floor plan similarity as a graph comparison problem: We seek a function s, operating on two floor plans represented as attributed graphs G_i and G_j, $s(G_i, G_j) : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}^+$, such that, when G_i is similar to G_j, s is large; and when dissimilar, s is small. We narrow down the set of possible solutions by framing the problem in terms of a distance metric learning formulation [37]: We seek a learnable function, e.g., an encoder, f_θ, parameterized by θ, that embeds G in a representation space $\mathcal{H}$ (*i.e.*, $H = f_\theta(G)$). A similarity measure $s_\mathcal{H}$ is required that computes the similarity between two such representations:

$$s(G_i, G_j) = s_\mathcal{H}\left(H_i = f_\theta(G_i), H_j = f_\theta(G_j)\right). \tag{1}$$

Given the similarity function $s_\mathcal{H}$, the objective is to learn $f_\theta(G)$ through a differentiable loss. To decide upon an proper metric for training and evaluation, a user study is conducted, in which we asked architects to rank floor plans. Albeit some level of disagreement, we found a positive and significant correlation between human judgment and a similarity based on a graph edit distance (GED) [25], while almost none for IoU ($\rightarrow$ suppl. mat. for details). Node and edge categories are considered when computing GED. Node categories correspond to the room types and edge categories correspond to the type of room-to-room permeability, which can either be access (*i.e.*, door) or adjacent-only (*i.e.*, wall but no door). We consider the weight of each edit operation (*e.g.*, changing node category, deleting / adding edge) the same. Unlike other learning-based frameworks that train and measure the goodness of retrieval based on IoU [12, 18, 21], we train and evaluate based on a normalized variant of GED to express the similarity: $\text{sGED}(G_i, G_j) = \exp(-\text{GED}(G_i, G_j)/(|G_i| + |G_j|))$, where $|G|$ is the size of the graph (*i.e.*, the number of nodes).

4 Method

4.1 Graph kernel network

The joint embedding architecture of our network contains two elements: a graph neural network (GNN) to learn node embeddings, and a graph kernel to compute the similarity between two sets of node embeddings. If f_θ denotes the GNN and $s_\mathcal{H}$ the similarity function, our framework, coined as graph kernel network (GKN), computes the similarity between graphs as given in Eq 1. How are methods fits the others is highlighted in Fig. 1. An overview of our method is given in Fig. 2.

Graph neural network for feature learning Node feature vectors are encoded using separate MLPs: $\mathbf{f}_\text{cat} : \mathbb{R}^8 \to \mathbb{R}^d$ on top of the room function vector; $\mathbf{f}_\text{shape} : \mathbb{R}^6 \to \mathbb{R}^d$ on the room shape vector; and $\mathbf{f}_\text{edge} : \mathbb{R}^2 \to \mathbb{R}^d$ on the edge vector. The outputs of $\mathbf{f}_\text{cat}$ and $\mathbf{f}_\text{shape}$ are concatenated and subsequently fed into another MLP, $\mathbf{f}_\text{node} : \mathbb{R}^{2d} \to \mathbb{R}^d$, to form the first hidden state:

$$\mathbf{h}_0^{(u)} = \mathbf{f}_\text{node}\left(\begin{bmatrix}\mathbf{f}_\text{cat}(\mathbf{c}^{(u)}) \\ \mathbf{f}_\text{shape}(\mathbf{s}^{(u)})\end{bmatrix}\right). \tag{2}$$

To learn expressive node embeddings, we use a message passing network (MPN) similar to the one in [15]. The MPN is a product of L consecutive graph convolutional layers. Each layer contains a message and an update function. The message to node u in layer l, $\mathbf{m}_l^{(\to u)}$, aggregates node information from its direct neighbors $v \in \text{ne}(u)$ (and itself) and takes into account the type of edges between them. Specifically, for each neighbor v, the hidden node features of the node itself $\mathbf{h}_l^{(u)}$, the one of the neighbor $\mathbf{h}_l^{(v)}$, and the edge encoding $\mathbf{r}^{(u,v)}$ are concatenated and fed into a shared MLP $\mathbf{f}_\text{intra} : \mathbb{R}^{3d} \to \mathbb{R}^d$. We use average aggregation over all the individual messages and arrive at the following definition for $\mathbf{m}_l^{(\to u)}$:

$$\mathbf{m}_l^{(\to u)} = \sum_{v \in \text{ne}(u)} \mathbf{f}_\text{intra}\left(\begin{bmatrix}\mathbf{h}_l^{(u)} \\ \mathbf{h}_l^{(v)} \\ \mathbf{r}^{(u,v)}\end{bmatrix}\right). \tag{3}$$

To arrive at the node features at layer $l+1$, the aggregated message and the node feature itself are stacked and fed into a GRU, $\mathbf{f}_\text{update} : \mathbb{R}^{2d} \to \mathbb{R}^d$:

$$\mathbf{h}_{l+1}^{(u)} = \mathbf{f}_\text{update}\left(\begin{bmatrix}\mathbf{h}_l^{(u)} \\ \mathbf{m}_l^{(\to u)}\end{bmatrix}\right). \tag{4}$$

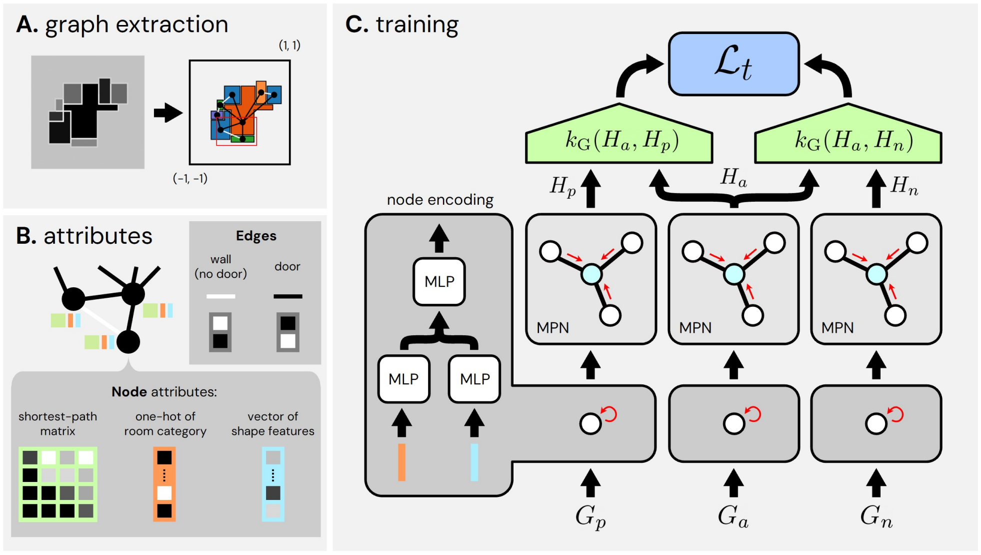

Figure 2: **Overview of LayoutGKN.** (***A. graph extraction***) Semantic images from RPLAN are converted into richly-attributed access graphs. The floor plans' geometries (*i.e.*, the rooms represented as polygons) are centered at $(0,0)$ and scaled to fit within the unit square box. The unit box amounts to 20 x 20 meters in reality (*i.e.*, 0.1 equals 1 meter). The color indicates the room's semantic category (*e.g.*, dark orange for living room, green for balcony). Edges are modeled when two rooms share a door (black) or a wall (white edge). (***B. attributes***) Each node represents a room and is endowed with 3 attributes: the shortest-path matrix, a one-hot encoding of the room's category and a vector of shape features. (***C. training***) LayoutGKN is trained using triplets of graphs, each containing an anchor (G_a), positive (G_a), and negative (G_a) graph. The goal is to penalize the relative distance between anchor-positive and anchor-negative. Anchor, positive, and negative are simultaneously fed into parameter-shared graph neural networks, which consist of a node encoder (dark gray box), followed by a series of L graph message passing network layers (light gray box). This results in embedded graphs (H_i). The anchor-positive and anchor-negative similarities are computed as $s_{\text{ap}} = k_G(H_a, H_p)$ and $s_{\text{an}} = k_G(H_a, H_n)$, respectively. The triplet loss $\mathcal{L}_t$ penalizes the relative distance $\log(s_{\text{an}}/s_{\text{ap}})$, given a margin m.

Graph kernel as similarity function We use the GraphHopper path-based graph kernel [6] to compute the similarity between two graphs. Specifically, given two graphs and their corresponding sets of learned node embeddings by the GNN, the graph kernel similarity k_G, as shown by the authors, can be computed in a remarkably simple form: as a weighted sum over node kernels across all pairs of nodes and their corresponding vector embeddings:

$$k_G(G_1, G_2) = \sum_{u \in \mathcal{V}_1} \sum_{v \in \mathcal{V}_2} \left\langle \mathbf{M}^{(u)}, \mathbf{M}^{(v)} \right\rangle \cdot k_{\text{node}} \left(\mathbf{h}_L^{(u)}, \mathbf{h}_L^{(v)} \right), \tag{5}$$

where the node kernel $k_{\text{node}}(\bullet, \bullet)$ computes the similarity between two nodes, and is typically Gaussian ($\mu \propto d^{-1}$ [14]):

$$k_{\text{node}} \left(\mathbf{h}_L^{(u)}, \mathbf{h}_L^{(v)} \right) = \exp \left(-\mu \left\| \mathbf{h}_L^{(u)} - \mathbf{h}_L^{(v)} \right\|^2 \right). \tag{6}$$

4.2 Training and inference

We train LayoutGKN under a triplet network setting [11]; thus, penalizing *relative* distances. Given a triplet of graphs (G_a, G_p, G_n), in which a, p, and n denote anchor, positive, and negative. We feedforward each graph through f_θ resulting in embeddings H_a, H_p, and H_n. We use a normalized variant of the graph kernel (Eq. 5) to compute the similarity between two such embeddings: $s_{\mathcal{H}}(H_i, H_j) = k_G(H_i, H_j) / \sqrt{k_G(H_i, H_i) \cdot k_G(H_j, H_j)}$. Given that we compute the relative distance from anchor-positive to anchor-negative as $d_{\text{apn}} = s_{\mathcal{H}}(H_a, H_n)/s_{\mathcal{H}}(H_a, H_p)$, we formulate the triplet loss as

$$\begin{aligned}\mathcal{L}_t &= \left[m + \log\left(d_{\text{apn}}\right)\right]_+ \\ &= \left[m + \log\left(\frac{k_G(H_a, H_n)}{k_G(H_a, H_p)}\sqrt{\frac{k_G(H_p, H_p)}{k_G(H_n, H_n}}\right)\right]_+,\end{aligned} \tag{7}$$

where $[\bullet]_+ = \max(\bullet, 0)$. We use the log to effectively penalize small relative distances more. Because the node (Eq. 6) and graph kernel (Eq. 5) are differentiable w.r.t. to the node embeddings, the loss (Eq. 7) is as well. The second line in 7 shows an efficient implementation of our loss in which $k_G(H_a, H_a)$ is canceled out. Similar to [12, 18, 21], we mine triplets based on the computable metrics we aim to mimic. Our objective, as mentioned before in Sec. 3, is different though: we mimic sGED not MIoU. To find informative triplets, we first rank each floor plan in the training set on MIoU, filter the best 50, and re-rank them on sGED. Each triplet is formed by the query and some combination of positive and negative found in the re-ranked list. Given some anchor, we first pick a positive s.t. $0.9 > \text{sGED}(G_a, G_p) > 0.6$. To ensure hard negatives, we pick a negative s.t. $0.7 < \text{sGED}G_a, G_n)/\text{sGED}(G_a, G_p) < 0.9$. In the case of GKN, during training, we pre-compute the shortest-path histogram matrices. During inference, we pre-compute the final node embeddings and denominator of $s_{\mathcal{H}}$ as well.

5 Results and discussion

5.1 Experiment setup

(***Datasets***) We train and test on RPLAN [34] and evaluate generalization to MSD [31]. After careful cleaning of both datasets, RPLAN and MSD contain 46K+ and 16K+ apartment-level floor plans, respectively. Statistics, descriptions, and pre-processing steps can be found in the suppl. mat. We train our models on RPLAN, and split training and test data with a ratio of 8:2. (***Baselines***) We compare LayoutGKN, which we will refer to as GKN in the remainder of the text, with *LayoutGMN* [21], in short GMN, and basic baseline methods including GEN and GK. GK is the *GraphHopper* graph kernel as-is, for which we model the node features by concatenating $\mathbf{s}^{(u)}$ and $\mathbf{g}^{(u)}$. For all methods, we use the same graph representation, as given in Sec. 3. In the case of GEN, GMN, and GKN, the same node encoder and intra-graph message passing mechanisms are used. In the case of GMN, an inter-cross message $\mathbf{mc}^{(\to u)} \in \mathbb{R}^d$ is concatenated to the input of $\mathbf{f}_{\text{update}}$ (Eq. 4), equivalent to [21] (Eq. 4, pp. 4). For GMN and GEN, we use the same graph pooling mechanism as in [15] and use the conventional triplet margin loss on the relative distances between the final graph-level vector embeddings: $\mathcal{L}_t = [d_{\text{an}} - d_{\text{an}} + m]_+$. (***Evaluation***) For comparing the methods, we report the triplet accuracy, Precision at 5 and 10 (P@5 and 20), and inference times (t). We use

Table 1: **Performance comparisons on RPLAN and MSD.** We report: the triplet accuracy; precision (P) scores at 5 and 10; and inference time t per 10K pairs. Best in **bold**. Note that the graph kernel (GK) does not involve any learning: it is GraphHopper kernel [6] as-is, for which we model the node features by concatenating the categorical and shape-based node features (*i.e.*, for each node u: $[\mathbf{c}^{(u)};\mathbf{s}^{(u)}]$). GK is a strong baseline for retrieval. During inference, embeddings (H_i) can only be precomputed for GEN and LayoutGKN, which creates the order of magnitude difference in inference time between these methods and LayoutGMN. * We train and represent the data in LayoutGMN [21] in exactly the same way as is the case for the other methods, which is slightly different from the original implementation in which the triplets are mined using MIoU and floor plans are represented as fully connected graphs.

	RPLAN				**Zero-shot to MSD**	
	Accuracy (↑)	P@5 (↑)	P@10 (↑)	t [s] (↓)	P@5 (↑)	P@10 (↑)
LayoutGK (*baseline*)	65.63±0.00	0.389±0.000	0.439±0.000	1.2±0.4	na	na
LayoutGEN (*baseline*)	96.24±0.07	0.603±0.007	0.665±0.004	**0.7±0.1**	0.595±0.015	0.605±0.018
LayoutGMN* CVPR'21	**97.74±0.05**	0.616±0.004	0.675±0.002	35.6±10.5	0.585±0.026	0.596±0.020
LayoutGKN (*ours*)	**97.78±0.10**	**0.623±0.004**	**0.683±0.002**	1.8±0.5	**0.674±0.024**	**0.697±0.017**

4-fold cross-validation, and report the average scores across the folds on the test set. To evaluate the generalization of the methods, we report the zero-shot precision scores on MSD only. (***Implementation***) Each model is trained for at most 200 epochs, with early stopping (patience is 10), on an NVIDIA GeForce RTX 4090 Ti GPU using AdamW [16] with a learning rate of 10^{-4} and batch size of 64. We use layer and batch normalization for the node encoder parts and message passing updates, respectively.

5.2 Effectiveness

Table 1 reports, on the left side of the vertical line, the overall effectiveness of all methods. Although similar to GMN in accuracy, GKN outperforms all other methods on ranking. The difference between GKN (or GMN) and GEN scores shows that explicit inclusion of cross-graph node-level interactions leads to a significant increase in accuracy and precision. However, we show that modeling expensive cross-graph node-level interactions *across the GNNs* might not be necessary for effective graph similarity computation. Instead, which is the case in GKN, the node-level interactions can be "postponed" to the end, when the final node embeddings are already computed, without losing quality.

Effect of size Fig. 3 shows the effect of network size on triplet accuracy, varying the hidden node dimension (d) and number of graph convolutional layers (L). When changing d, we fix $L = 5$; when changing L, we fix $d = 64$. We observe that performance drops much faster for GEN and GMN than for GKN as d decreases. We attribute this to the different roles of the embeddings. In GEN and GMN, node embeddings must be sufficiently large to capture both (i) their topological role in the graph and (ii) their semantic attributes (e.g., geometry and room function in floor plans). Although GMNs explicitly model cross-graph node interactions, they still depend on expressive node embeddings. In contrast, GKN leverages a topology-aware similarity function (*i.e.*, the graph kernel), which naturally accounts for the topology of the graphs when computing the similarity between them. As a result, node embeddings in GKN can be smaller without too much of a performance drop, since they need not encode topology as strongly themselves. We have not yet tested with more

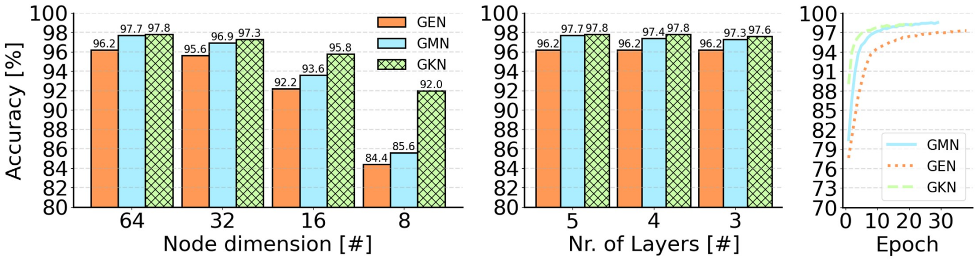

Figure 3: (***Left and middle***) The effect of the number of learnable parameters on the triplet accuracy. We vary the number of parameters by changing the hidden node dimension and number of graph convolutional layers. The hidden node dimension has a profound effect on the accuracy: the smaller the node dimension, the more a decrease in accuracy. The rate at which the accuracy drops for decreasing node dimension is lowest in the case of GKN. In the case of the number of layers there are negligible differences in accuracy (across all methods). (***Right***) Typical training curves (on the validation split) showing that: 1) GKN converges the fastest and 2) has a head-start because of the kernel.

'expressive' GNN encoders, such as GAT [32] or GIN [36], but the same capacity limitation seems likely to remain.

Qualitative studies Fig. 4 (**A**) shows a case of ranking under the sGED metric on RPLAN. A useful property of training floor plan similarity under a sGED metric, is that retrievals are (practically) invariant to grid transformations. In Fig. 4 (**B**), the emergence of rotational and flip invariance is depicted. Because of the two first entrees ($c_x^{(u)}$ and $c_x^{(u)}$) of the room shape vector $\mathbf{s}^{(u)}$, it is important to note that the model is not strictly invariant to such grid transformations. Nonetheless, if you want the model to be *provably* invariant to such grid transformations, simply remove the first two entrees of $\mathbf{s}^{(u)}$.

5.3 Efficiency

In the case of GMN, we cannot pre-compute the node embeddings, because of the cross-graph node-level interactions. Depending on d, L, and $N_i = |G_i|$ for $i \in (1,2)$, the number of FLOPs is order of magnitudes larger for GMNs: $\approx$ 5-10k FLOPs in the case of GKN and GEN; $\approx$ 10-20M FLOPs in the case of GMN. Therefore, the differences in inference time t, shown on the direct left of the vertical line in Tab. 1, are not surprising: GMN takes $\approx$ 20X more time than the rest. In real-time search engines, speed is of great importance: results should pop-up in seconds, not minutes. GMNs compute the similarity between 10K pairs in about 30-40 seconds. GKNs, on the other hand, compute the similarity between 10K pairs in about 1 to 2 seconds: a drastic speedup compared to GMNs – a speedup that, most importantly, aligns with what we are after in real-time search.

5.4 Generalization

We further assess the models in a zero-shot setting (*i.e.*, no additional training or fine-tuning) by evaluating directly on the more complex MSD dataset [31]. As shown on the right side of Tab. 1, all methods retain relatively high precision despite the distribution shift from RPLAN to MSD. For GKN, precision is even higher on MSD than on RPLAN, and it achieves the

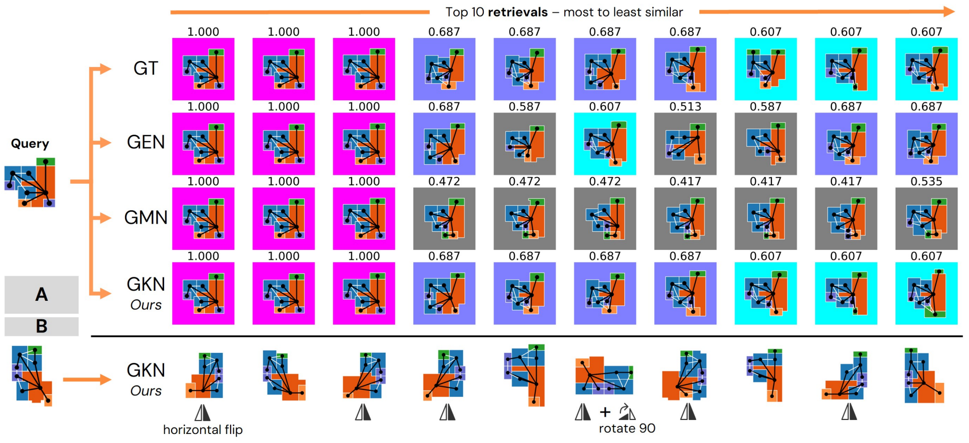

Figure 4: **Examples of ranking on RPLAN.** (**A**) Given a query floor plan on the left, the top row shows the ground truth (GT) ranking. The value on top of the floor plan indicates the similarity in terms of sGED. Since the GT scores are quite often equivalent for different retrievals, we color-coded them: the same color means the same GT score. Rows 2 to 4 depict the rankings based on GEN, GMN, and GKN. On top of the floor plans, we provide the GT score and indicate using the same color scheme as is the case in the top row where a floor plan would have landed in the GT ranking. A gray background means that it would fall outside the top-10 on GT. (**B**) We show an example of ranking on GKN in which the top-10 retrievals show signs that the retrieval results are invariant to the elementary grid transformation (flips and 90 degree rotations).

strongest performance overall. A plausible explanation is that MSD, which contains entire building complexes, exhibits a higher proportion of near-duplicate or structurally similar floor plans, thereby boosting retrieval scores. The advantage of GKN compared to the other methods may further stem from the use of the graph kernel, which provides a strong structural prior aligned with the GED-based evaluation and is thus less dependent on dataset-specific differences. These results indicate stronger zero-shot transfer for GKN, though disentangling intrinsic model robustness from dataset effects remains an open question.

6 Conclusion

We introduced LayoutGKN, a graph-based joint embedding architecture for measuring the spatial similarity between floor plan graphs. Unlike graph matching networks (GMNs), LayoutGKN decouples representation learning from similarity computation: node embeddings are learned independently, while important cross-graph node-level interactions are only imposed in the distance function via a differentiable path-based kernel. Our results on floor plan similarity learning indicate that cross-graph node-level interactions are not necessarily required within the graph encoders themselves, but can be effectively postponed to the similarity-based loss: compared to LayoutGMN, our method achieves comparable or better performance on floor plan retrieval on RPLAN, while drastically increasing the speed. We are curious whether our method generalizes to other graph domains (*e.g.*, molecules) as well.

Acknowledgments We thank all participants of our user study for their valuable time and feedback.

References

[1] Christopher Alexander, Ishikawa Sara, and Silverstein Murray. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[2] Yue Bai, Dipu Manandhar, Zhaowen Wang, John Collomosse, and Yun Fu. Layout Representation Learning with Spatial and Structural Hierarchies. In *AAAI*, 2023.

[3] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. SimGNN: A Neural Network Approach to Fast Graph Similarity Computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019.

[4] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature Verification using a "Siamese" Time Delay Neural Network. In *NeurIPS*, 1993.

[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. *ICML*, 2020.

[6] Aasa Feragen, Niklas Kasenburg, Jens Petersen, Marleen de Bruijne, and Karsten Borgwardt. Scalable Kernels for Graphs with Continuous Attributes. In *NeurIPS*, 2013.

[7] Matthew Fisher, Manolis Savva, and Pat Hanrahan. Characterizing Structural Relationships in Scenes Using Graph Kernels. In *ACM SIGGRAPH*, 2011.

[8] Liu He and Daniel Aliaga. COHO: Context-Sensitive City-Scale Hierarchical Urban Layout Generation. In *ECCV*, 2024.

[9] Congrui Hetang, Haoru Xue, Cindy Le, Tianwei Yue, Wenping Wang, and Yihui He. Segment Anything Model for Road Network Graph Extraction. In *CVPRw*, 2024.

[10] B Hillier, A Leaman, P Stansall, and M Bedford. Space Syntax. *Environment and Planning B: Planning and Design*, 1976.

[11] Elad Hoffer and Nir Ailon. Deep Metric Learning Using Triplet Network. In *Similarity-Based Pattern Recognition*, 2015.

[12] Jiongchao Jin, Zhou Xue, and Biao Leng. SHRAG: Semantic Hierarchical Graph for Floorplan Representation. In *International Conference on 3D Vision*, 2022.

[13] Rasika Khade, Krupa Jariwala, and Chiranjoy Chattopadhyay. An Interactive Floor Plan Image Retrieval Framework Based on Structural Features. *Arabian Journal for Science and Engineering*, 2023.

[14] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A Survey on Graph Kernels. *Applied Network Science*, 2020.

[15] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph Matching Networks for Learning the Similarity of Graph Structured Objects. In *ICML*, 2019. arXiv:1904.12787.

[16] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *ICLR*, 2019.

[17] Guixiang Ma, Nesreen K. Ahmed, Theodore L. Willke, and Philip S. Yu. Deep Graph Similarity Learning: A Survey, 2020. arXiv:1912.11615.

[18] Dipu Manandhar, Dan Ruta, and John Collomosse. Learning Structural Similarity of User Interface Layouts Using Graph Networks. In *ECCV*, 2020.

[19] Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. House-GAN: Relational Generative Adversarial Networks for Graph-Constrained House Layout Generation. In *ECCV*, 2020.

[20] Hyejin Park, Hyegyo Suh, Jaeil Kim, and Seungyeon Choo. Floor Plan Recommendation System using Graph Neural Network with Spatial Relationship Dataset. *Journal of Building Engineering*, 2023.

[21] Akshay Gadi Patil, Manyi Li, Matthew Fisher, Manolis Savva, and Hao Zhang. LayoutGMN: Neural Graph Matching for Structural Layout Similarity. In *CVPR*, 2021.

[22] Pablo N. Pizarro, Nancy Hitschfeld, Ivan Sipiran, and Jose M. Saavedra. Automatic Floor Plan Analysis and Recognition. *Automation in Construction*, 140, 2022.

[23] Pau Riba, Andreas Fischer, Josep Lladós, and Alicia Fornés. Learning Graph Edit Distance by Graph Neural Networks. *Pattern Recognition*, 2020.

[24] Qamer Uddin Sabri, Johannes Bayer, Viktor Ayzenshtadt, Syed Saqib Bukhari, Klaus-Dieter Althoff, and Andreas Dengel. Semantic Pattern-based Retrieval of Architectural Floor Plans with Case-based and Graph-based Searching Techniques and their Evaluation and Visualization. In *Proceedings of the 6th International Conference on Pattern Recognition Applications and Methods*, 2017.

[25] Alberto Sanfeliu and King-Sun Fu. A Distance Measure Between Attributed Relational Graphs for Pattern Recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13, 1983.

[26] Divya Sharma and Chiranjoy Chattopadhyay. High-level Feature Aggregation for Fine-Grained Architectural Floor Plan Retrieval. *IET Computer Vision*, 2018.

[27] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 2011.

[28] Philip Steadman. *Architectural Morphology: An Introduction to the Geometry of Building Plans*. Pion, 1983.

[29] Yuki Takada, Naoto Inoue, Toshihiko Yamasaki, and Kiyoharu Aizawa. Similar Floor Plan Retrieval featuring Multi-Task Learning of Layout Type Classification and Room Presence Prediction. In *ICCE*, 2018.

[30] Casper van Engelenburg, Seyran Khademi, and Jan van Gemert. SSIG: A Visually-Guided Graph Edit Distance for Floor Plan Similarity. In *ICCVw*, 2023.

[31] Casper van Engelenburg, Fatemeh Mostafavi, Jan van Gemert, and Seyran Khademi. MSD: A Benchmark Dataset for Floor Plan Generation of Building Complexes. In *ECCV*, 2024.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *ICLR*, 2018.

[33] Raoul Wessel, Ina Blümel, and Reinhard Klein. The Room Connectivity Graph: Shape Retrieval in the Architectural Domain. In *The 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2008.

[34] Wenming Wu, Xiao-Ming Fu, Rui Tang, Yuhan Wang, Yu-Hao Qi, and Ligang Liu. Data-driven Interior Plan Generation for Residential Buildings. *ACM ToG*, 38, 2019.

[35] Xingjiao Wu, Luwei Xiao, Xiangcheng Du, Yingbin Zheng, Xin Li, Tianlong Ma, and Liang He. Cross-Domain Document Layout Analysis via Unsupervised Document Style Guide. *Expert Syst. Appl.*, 2022.

[36] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *ICLR*, 2019.

[37] Liu Yang and Rong Jin. Distance Metric Learning: A Comprehensive Survey. `https://www.cs.cmu.edu/~liuy/frame_survey_v2.pdf`, 2006.

[38] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing Stars: On approximating Graph Edit Distance. *Proc. VLDB Endow.*, 2009.

[39] Haoran Zheng, Jieming Shi, and Renchi Yang. GraSP: Simple yet Effective Graph Similarity Predictions. In *AAAI*, 2025.