Computational Visual Media
https://doi.org/10.1007/s41095-018-0110-3 Vol. 4, No. 2, June 2018, 123–137
Research Article

Knowledge graph construction with structure and parameter learning for indoor scene design

Yuan Liang[1], Fei Xu[1], Song-Hai Zhang[1], Yu-Kun Lai[2], and Taijiang Mu[1](✉)

Abstract We consider the problem of learning a representation of both spatial relations and dependencies between objects for indoor scene design. We propose a novel knowledge graph framework based on the entity-relation model for representation of facts in indoor scene design, and further develop a weakly-supervised algorithm for extracting the knowledge graph representation from a small dataset using both structure and parameter learning. The proposed framework is flexible, transferable, and readable. We present a variety of computer-aided indoor scene design applications using this representation, to show the usefulness and robustness of the proposed framework.

Keywords knowledge graph; scene design; structure learning; parameter learning

1 Introduction

Indoor scene design is required for applications built on virtual environments (e.g., large-scale video games or virtual reality), and realistic interior design. Modern commercial CAD tools [1–3], as well as independently developed toolchains, have been utilized to help professional designers or architects to assemble indoor scenes with different levels of detail. However, generating a useful and aesthetically pleasing scene requires a relatively high level of expertise, in terms of both insight into interior design, and proficiency in use of design tools; this forms a high barrier for novice users. Even for expert users, a considerable amount of laborious and time-consuming interaction is needed.

To reduce the amount of interaction and expertise needed when using these tools, context-based models have been proposed, supporting semi-automatic [4] and fully-automatic [5] systems. Fundamental features for such systems include helping (i) to retrieve from a library the objects to be placed in the scene and (ii) to decide where in the scene to place various objects. These models usually embed a group of quantitative criteria learned from a training dataset for both features, e.g., co-occurrence frequencies as a prior for object retrieval, and for placement, e.g., Gaussian mixture models [4]. Such methods have achieved a certain level of success; however, they still have several shortcomings, including being inflexible given a limited training dataset, having limited transferability given a limited model library, and having poor readability of the underlying model for designers without a background in statistics or computer science.

On the other hand, knowledge graphs, a representation for facts in specific domains, have had great success in information retrieval and question–answer systems. Introducing knowledge graphs to such systems allows issues caused by long-tail queries or entities with few samples to be well addressed [6–8]. Moreover, knowledge graphs have good readability, which helps domain experts further explain and improve the model.

Two major challenges exist for utilizing knowledge graphs in indoor scene design: (i) designing a proper schema for modeling the functional and geometric relations between objects in the room and (ii) extracting the structure of the knowledge

1 TNList, Department of Computer Science, Tsinghua University, Beijing 100084, China. E-mail: Y. Liang, liangyua14@mails.tsinghua.edu.cn; F. Xu, xuf16@mails.tsinghua.edu.cn; S.-H. Zhang, shz@tsinghua.edu.cn; T. Mu, mmmutj@gmail.com (✉).
2 School of Computer Science and Informatics, Cardiff University, Cardiff, CF24 3AA, UK. E-mail: Yukun.Lai@cs.cardiff.ac.uk.
Manuscript received: 2018-01-08; accepted: 2018-01-13

graph from a few training samples of indoor layouts. To overcome these challenges, we design an entity-relation schema for modeling relations and properties of different objects. Based on this representation, we map the knowledge graph to a probabilistic graph model, which allows us to learn the structure of the knowledge graph from indoor scenes designed by professional designers, using structure and parameter learning. We show the usefulness of our knowledge graph framework by integrating it into indoor scene design algorithms and applying them to several typical application scenarios with good performance.

Our work has the following technical contributions:

- introduction of the knowledge graph representation to indoor scene design, with significant benefits of flexibility, transferability, and readability, compared to existing methods;
- an entity-relation knowledge graph schema that models functional, geometric, and hierarchical relations among objects in a library;
- a process that maps a knowledge graph to a probabilistic graph model, and an algorithm that learns both the structure and parameters of the knowledge graph.

The remainder of this paper is organized as follows. In Section 2 we discuss related work. We then introduce a general schema for knowledge graph representation in Section 3. Based on it, we present a corresponding underlying probabilistic graph model and an algorithm for parameter and structure inference in Section 4. We show several typical applications of our representation in Section 5, demonstrating the effectiveness of our approach. Finally, we conclude and discuss our work in Section 6.

2 Related work

2.1 Contextual modeling in indoor scene design

To reduce the number of interactions in indoor scene design, contextual modeling, which tries to evaluate if a model and its placement fit its context, is widely used [4] in practical applications. Different forms of interaction for indoor scene design have been proposed based on contextual modeling, including: proper placement for a user-selected model [4], retrieving a model and finding a proper orientation with a user selected placement [9], suggesting objects for adding greater levels of detail to scenes [10], co-retrieval and co-placement with user-selected samples [11], and scene synthesis from freehand sketch drawings [12] or natural language descriptions [13]. Fully-automatic methods have also been proposed to synthesize scenes in open world applications [5].

Frequently used methods for contextual modeling include rule-based criteria [5] and data-driven models [4]. Rule-based criteria apply a set of hand-crafted rules, e.g., "always place a plate on a table" or "the total area occupied by plates on a table should not exceed 70% of the area of the table". As the number of rules required to generate reasonable scenes increases rapidly with an increasing number of object categories, such an approach is generally restricted to synthesizing scenes of low complexity. Data-driven models usually model binary placement and co-occurrence relations between pairs of objects using quantitative models such as Gaussian mixtures [4], graphlet extraction [12], kernel density estimation [13], etc. The idea of contextual modeling has also been used for indoor scene modeling from scanned data [14].

Although such data-driven models have achieved success for certain categories of scenes, the usefulness of these methods is limited by their lack of flexibility given a limited set of training data, which usually causes the generated layout to overfit the training data. Also, these learned quantitative models are difficult to transfer to new scenarios. For example, they can perform poorly when placing a round table in a scene using a model trained with square tables. Furthermore, these data-driven models are hard for designers to interpret, lacking a background in statistics or computer science, making it difficult for them to improve the model when failures occur.

2.2 Knowledge graph in information retrieval

Outside the domain of geometric contextual modeling, knowledge graphs have been successfully used to support a wide variety of artificial intelligence applications: conventionally in information retrieval [15], and more generally in other applications such as question–answer systems [16], reasoning systems [17], and image classification [18]. They model a sophisticated network of real-world entities, representing their relations and supporting operations such as entity

identification, disambiguation, and completion.

Despite such wide practical success, building a structured knowledge graph from unstructured training data is conventionally considered a challenging problem for two reasons: (i) natural languages are not fully structured, and (ii) it is hard to fuse knowledge from different (or even conflicting) data sources. As a result, widely-used knowledge bases, such as DBpedia [19], use manually maintained ontologies as well as filtered training data as input.

In this paper, we build a knowledge graph for indoor scene design. As our training data is well-structured, it is much easier to generate structured knowledge. To solve the second challenge, we map our knowledge graph to a probabilistic model, which allows us to infer graph parameters as well as graph structure from our training data. The training process implicitly fuses different training data.

The spatial knowledge learning approach in Ref. [20] is most similar to our work, and uses prior probabilities directly as knowledge to support a text-to-scene application which allows missing information to be inferred. Our work differs in the following ways. (i) We give a detailed definition and discussion of relative position properties. (ii) Our representation is more general. In addition to spatial relations, our knowledge graph also addresses functional relations. (iii) Instead of using prior probabilities, we use a probabilistic graph model to learn a better joint distribution of different types of rooms for indoor design.

3 Knowledge graph schema

3.1 Preliminaries

In this section, we explain how we formulate facts in indoor scene design, including facts about different objects and guidelines for their typical placement.

In our knowledge graph, facts are formulated using an entity-relation (ER) model, which is conventionally applied in existing knowledge graphs such as DBpedia [19], Freebase [21], and YAGO [22]. An ER model consists of entities with different attributes and relations between pairs of entities. It can easily be represented as a graph with attributes associated with nodes and edges. Conventionally, the graph is stored as a node (entity) list with attributes of the entities, and an edge (relation) list, with triples describing the type of the relation and the related entities, such as $\langle \textbf{toad}, \textbf{is a kind of}, \textbf{amphibian} \rangle$.

The schema, a concept originating in relational databases, is fundamental to an ER model. The schema of our knowledge graph has the following parts:

- The types of entities and the attributes that describe each type of entity.
- The types of relations between entities, the attributes that describe each type of relation, and the types of entities that each type of relation can connect.

In the remainder of this section, we discuss how we designed our schema with the help from professional BIM (building information management) designers.

3.2 Entities

We interviewed several designers to ascertain how they select objects to be placed in a given room. Typical pipelines used by these designers included the following steps:

1. Consider the target functionality of the room, and select types of objects needed to support that functionality (e.g., a TV in a living room).
2. Add more types of objects which are used together with the selected objects and place them according to their usages.
3. Pick proper objects of the selected types to fit the general design style of the room.
4. Fine-tune the layout, including adding necessary lights, filling in empty space with decorative objects, etc.

Such a pipeline appeals to common sense. Using this generic pipeline, we designed a knowledge graph with the following entities:

- *Room type.* This encodes the general functionality of the room (e.g., living room, bedroom, gym, etc.) to be synthesized. This type of entity does not have attributes.
- *Object type.* As in ShapeNet [23], we use the WordNet [24] ontology as our basic hierarchical taxonomy. Initially we map each synset in ShapeNet to an object type in our entities, which allows us to organize facts at different levels of a hierarchy. E.g., we often put "cabinets" in a "living room", and a "TV" can only be put on a "base cabinet", which is a sub-synset of the synset "cabinet". The most important field of an object type's attributes

is its metadata, which indicates what fields are required for object instances of this object type. We extracted metadata from Autodesk Revit projects provided by these designers. We also add an attribute field "generative probability" to model the probability of adding an instance of this object type to the room even if it is unrelated to any room type or other instances in the room. This field is useful in handling decorative objects, e.g., paintings on a wall.

To make the taxonomy more specific to indoor design, e.g., to distinguish between a "tall cabinet" and a "base cabinet" in the synset "cabinet" in ShapeNet, we retrieved BIM families from several BIM websites and extracted their keywords. A designer then helped us filter these keywords and categorize them into existing synsets in ShapeNet.

- *Object instance.* Each object instance entity encodes an object in our library. Its attribute fields include its corresponding 3D mesh, and fields dependent on its object type (e.g., the material of the object, the width of a bed, the price of the object, etc.). All fields except the mesh are nullable, as some fields might be absent for models in the library.

These types of entities define the general skeleton of the knowledge graph, as they define the nodes in the graph. Each type of entity is encoded as a type of node in the knowledge graph.

3.3 Relations

By considering the design pipeline in Section 3.2, we identified the following explicit relations as being used:

- Functional requirements, e.g., a TV is required in a living room.
- Functional dependencies, e.g., a seat is required for watching TV.
- Placement dependencies, e.g., the seat should be placed facing the TV.

We can also identify some implicit relations in the pipeline. For example, when a seat is needed, either a sofa or a chair can satisfy the requirement, because both sofa and chair satisfy **a kind of** seat. Also, when we select a certain instance of a sofa, this implicitly encodes the relation "this object is **an instance of** a sofa".

A major challenge in modeling these relations is transferability. Due to lack of training scenes, we can only find a limited number of **object instance**s of a certain **object type** in the training dataset. As a result, we often need to transfer our model trained with our training dataset to new objects in practical applications, e.g., predicting the placement of an **object instance** that is absent from the training scenes.

A typical relation for which transferability is a problem is the relative placement relation of **object type**s. Other work, such as example-based scene synthesis [11], uses a single Gaussian mixture model (GMM) to model relative placements. However, relations modeled in this way are hard to transfer. For example, a model that characterizes the placement of chairs around a given table is hard to transfer to a table of different shape or size (see Fig. 1).

The analysis above leads to the relations we define, including:

- **Is needed in (R1).** This encodes that some **object type** is needed in some **room type**. We add an attribute "necessity" to measure the necessity (as a probability) of having this object type in the given room type. For example, a dining room needs a dining table, regardless of how small the room is. Given a larger room, we may place a TV in it. In this example, the object type "dining table" has a higher necessity than the type "TV". It can be written as $\langle$**Dining table**, **is needed in**, **dining room**$\rangle$.**necessity** $>$ $\langle$**TV**, **R1**, **dining room**$\rangle$.**necessity**. Relation names (e.g., **is needed in**) and relation IDs (e.g., **R1**) are used interchangeably in this paper. We quantify the necessity with a probabilistic model, which we introduce in Section 4.

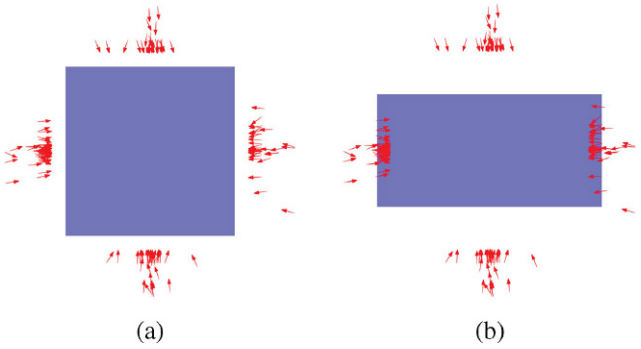


Fig. 1 Transferring a model for relative placement and orientation between **object types** "table" and "chair". (a) Training data with a square table. (b) For a table of a different shape, the model needs to be transferred.

- **Works with (R2).** This encodes the dependency between two **object type**s. Similarly to **R1**, we add an attribute field "dependency" to measure the level of dependency.
- **Relative placement (R3).** Existing work [20] also uses this type of relation, which maps the relative placements to keywords in a natural language. However, due to the ambiguity of natural languages, such a representation is inadequate for many applications. Although that paper disambiguates "on" (as in, e.g., on the wall versus on the desk), we can easily list several further cases (see Fig. 2) that also need disambiguation. We propose a multi-field representation, to more precisely represent the relative placements between objects, as follows:
  - *Primary direction.* Instead of using keywords [20], we use a quantitative representation to represent the most important placement constraints. For example, for plates placed **on** a table, the primary direction is $(0, 0, 1)$, where the z coordinate is assumed to be vertical. It is defined using the nearest pair of points $\boldsymbol{x}_{\text{plate}}$ and $\boldsymbol{x}_{\text{table}}$ belonging to the meshes of the plate and the table; the constraint here is $(\boldsymbol{x}_{\text{plate}} - \boldsymbol{x}_{\text{table}}) \cdot (0, 0, 1) \geqslant 0$. Specifically, when we calculate the primary direction between two objects touching each other, we use a primary direction that is perpendicular to their contact faces. Such a definition can help to overcome certain failures in specific cases, e.g., a desk with an integrated tray. With the help of this field, we can disambiguate concepts such as "in front of the desk" (with a primary direction of $(0, -1, 0)$) and "on the front of the desk" (with a primary direction of $(0, 0, 1)$).
  - *Primary distance.* With the defined primary direction, we can also define $(\boldsymbol{x}_{\text{plate}} - \boldsymbol{x}_{\text{table}}) \cdot (0, 0, 1)$ as the primary distance between objects "plate" and "table"; the primary distance is 0 for a pair of objects in contact. With the help of this field (along with the primary direction), we can easily disambiguate concepts such as "hanging on" (with a primary distance of 0) and "in front of" (with a primary distance larger than 0), which share the same primary direction.
  - *Projected placement.* With the defined primary direction, we can model more complex spatial relations. For example, to model the detailed placement of a plate on a table, we can further project the placement of the plate onto the table, $(\boldsymbol{x}_{\text{plate}} - \boldsymbol{x}_{\text{table}}) - (\boldsymbol{v} \cdot (\boldsymbol{x}_{\text{plate}} - \boldsymbol{x}_{\text{table}}))\, \boldsymbol{v}$, where $\boldsymbol{v}$ is the primary direction. We normalize this projected vector with the size of the object it relates to. With this field, we can disambiguate cases, e.g., as in Figs. 2(a) and 2(b), where the projections from the monitor and the laptop to the desk have different values, indicating where they are placed on the desk.
  - *Relative orientation.* Relative orientation is defined by the placement orientation of an object, relative to the primary direction.

  Note that for some pairs of objects, several alternative relative placements may exist between them. For example, a chair can be placed beside any of the four edges of a square table. Thus the relative placement relation is defined as a mixture of the above.
- **Is an instance of (R4).** This encodes the **object type** to which a certain **object instance** belongs.
- **Is a hypernym of (R5).** This type of relation is generated directly from the WordNet ontology. It encodes the conceptual relations between hypernyms and hyponyms, and also helps organize these concepts of object types in a hierarchical structure. It has an attribute "drilling probability", indicating the probability of picking an instance of that type, e.g., picking a "base cabinet" when a "cabinet" is needed.

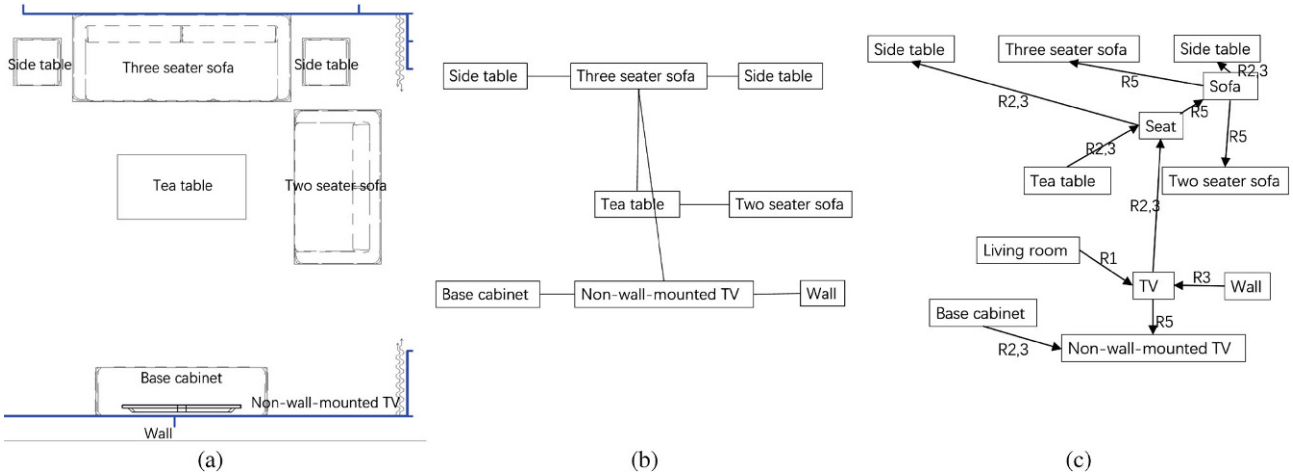
3.4 Indoor scene design analysis based on knowledge graphs

With the entities (nodes) and relations (edges) defined above, we can organize facts in indoor scene design in a knowledge graph.

Given an indoor scene, we can organize the hidden facts with a graph-based representation which is a subgraph of our knowledge graph. An example of such a subgraph is shown in Fig. 3, which gives a typical pattern for a living room. This graph gives facts about how this scene is constructed. A non-wall-mounted TV needs a base cabinet to support it (in contrast, a wall-mounted TV does not). To watch TV, we need a seat facing it, and a sofa is a kind of seat. We also add a tea table and two side tables to make better use of this seat. Although the representation in Fig. 3(c) is a more complex representation than

(a) (b) (c) (d)

Fig. 2 Ambiguity and imperfection of natural language in describing relative placement. (a, b) Difference between "a monitor on a desk" and "a laptop on a desk". Although the preposition "on" means "supported by" in both cases, monitors are usually placed far from the user, while laptops are usually placed near the user. (c, d) Difference between "a cabinet beside a bed" and "a window beside a bed". The preposition "beside" represents different spatial relations in these two scenarios.



(a) (b) (c)

Fig. 3 A typical indoor scene and its corresponding graph-based representation, which is a subgraph of our knowledge graph. (a) A corner of a room provided by an indoor designer. (b) A scene graph from the work [25]. (c) Our graph-based representation. To simplify this graph for drawing, we have removed nodes representing instances and corresponding **is an instance of** relations.

that in Fig. 3(b), it contains much more information and can easily be transferred. For a simple example, we can easily conclude from the representation in Fig. 3(c) that if we change the non-wall-mounted TV to a wall-mounted one, the base cabinet is no longer a necessary piece of furniture in the room. However, changing the three-seater sofa to a chair does not change any dependencies in the scene.

Such a representation can support many indoor scene design applications, with the help of subgraph searching or matching algorithms. Analyzers and designers without a background in statistics or computer science can also easily interpret this representation, to analyze the results of these algorithms, or manually correct mistakes in the graph.

4 Probabilistic mapping of the knowledge graph

To learn the knowledge graph from indoor training scenes, we propose use of a factorization model as an intermediate step. Based on our proposed factorization model, we can evaluate how the learned knowledge graph fits our training data by mapping entities and relations in our knowledge graph to the factorization model. Similar methodology has shown success in semantic processing, e.g., in topic models [26] in natural language processing. By learning the parameters and structure of the factorization model, we get an intermediate representation of hidden knowledge in indoor scene design. We may then map the factorization model back to a knowledge graph, and hence a knowledge graph representation is built.

Specifically, a factor graph [27] is utilized as our factorization model, which is a bipartite graph representing the factorization of a function. Handcrafted factor graphs have shown success in fully-automatic synthesis of indoor scenes with limited diversity of object types [5]. A factor graph has two types of nodes: variable nodes that encode random variables involved in the model, and factor

nodes that define the relations between variables. Given a factorization of function $g(X) = \prod_{j=1}^{m} f_j(S_j)$, where $S_j \in 2^X$, 2^X denotes the power set of X and $X = \{X_1, X_2, \ldots, X_n\}$ is the set of random variables, we can build a factor graph with $n + m$ nodes, including n variable nodes and m factor nodes. If $X_i \in S_j$, we connect the ith variable node and the jth factor node, which represents that the jth factor takes the ith variable as a dependent variable.

4.1 Mapping the knowledge base to a factor graph

To build a proper objective function for indoor scene design and derive a good mapping to our knowledge graph representation, the factor graph should satisfy the following design principles:

- **Factor nodes.** We should define factor nodes which correspond to the relations defined in Section 3.3, to map our defined factor graph to relations in the knowledge graph.
- **Variable nodes.** We should define variables that are closely related to objects placed in indoor scene designs, to model the indoor design results. The variables should also correspond to the entities defined in Section 3.2, to support the aforementioned factors.
- **Objective function.** The objective function to be factorized should be easily optimizable, to support structure and parameter learning.

We thus define the following variables in the factor graph:

- **Room type variables** $T_1, T_2, \ldots$. Each variable of this type encodes a room type entity. Given a room of a single type, we represent it with a one-hot representation, setting the variable corresponding to the room type to 1, and those for other room types to 0. Given a room of mixed type, e.g., a dining room with an integrated kitchen, we set $[T_1, T_2, \ldots]$ as a weighted vector of length 1.
- **Object type count variables** $C_1, C_2, \ldots$. Each variable of this type encodes the number of objects of this type we plan to place in the room (which is not the same as the actual number of objects placed). We split this variable into the sum of 3 components $\boldsymbol{C} = \boldsymbol{CI} + \boldsymbol{CD} + \boldsymbol{CT}$, representing the numbers of objects planned for supporting inheritance relations (R5), dependency relations (R2), and functionality relations (R1) respectively, as explained later in the factor definitions.
- **Instance placement variables** $I_1, I_2, \ldots$. Each variable of this type $\boldsymbol{I} = \{\text{instance}, \boldsymbol{x}, \boldsymbol{o}\}$ encodes the selection, placement $\boldsymbol{x}$, and orientation $\boldsymbol{o}$ of an object instance in the room.

In this model, only the room type and the placements of the instances $(\boldsymbol{T}, \boldsymbol{I})$ are observed. We define our objective function in a probabilistic form $p(\boldsymbol{T}, \boldsymbol{I} \,|\, \text{relations})$, which allows us to utilize probabilistic optimization methods in the estimation step.

The following factors are defined to represent the objective function in a probabilistic form:

- **Functionality factors.** A functionality factor corresponds to the relation **R1** and measures how the number $\boldsymbol{CT}$ of objects planned based on the room type fits the desired room type and the corresponding "necessity" property in the **R1** relation. We define a generative process for the $\boldsymbol{CT}$ variables using the following pipeline:
  1. For each room type i in the room type vector, sample the total number of all types of objects planned in the process $n_i = \text{Poisson}(ST_i)$ featuring the room's functionality, where S is proportional to the area of the room. Such sampling with a Poisson distribution is conventional in topic models [28].
  2. Plan the number of objects to be placed in the room corresponding to each function:
  $$\boldsymbol{CT} = \sum_i \boldsymbol{CT}^{(i)} + \boldsymbol{\epsilon}$$
  where
  $$\boldsymbol{CT}^{(i)} \sim \text{Multinomial}(n_i; pt_{i,1}, pt_{i,2}, \ldots)$$
  $$\epsilon_j \sim \text{Poisson}(g_j)$$
  Here $pt_{i,j}$ is the value of the field "necessity" of the relation $\langle j\text{th object type}, \textbf{R1}, i\text{th room type}\rangle$, or 0 if the relation does not exist in the knowledge graph. g_j is the field "generative probability" defined in the attributes of each object type. Hence we get a factor $p(\boldsymbol{CT}|\boldsymbol{T}, \text{room}) = \prod_i p(CT_i|\boldsymbol{n}) p(\boldsymbol{n}|\boldsymbol{T}, \text{room}) = f(\boldsymbol{CT}, \boldsymbol{n}) g(\boldsymbol{n}, \boldsymbol{T}, \text{room})$, as a factorized probability in the factor graph.
- **Selection factors.** A selection factor corresponds to relation **R5**, and models how we choose types of hyponym objects, e.g., whether to choose a chair or a sofa when a seat is needed. It measures how the number $\boldsymbol{CI}$ of objects planned from room types fits the knowledge graph and the

property "drilling probability" in **R5**. We define the generative process of variables $\boldsymbol{CI}$ as follows:

$$\boldsymbol{CI} = \sum_i \boldsymbol{CI}^{(i)}$$

where

$$\boldsymbol{CI}^{(i)} \sim \text{Multinomial}(C_i; pi_{i,1}, pi_{i,2}, \ldots)$$

Here $pi_{i,j}$ is defined as the value of the field "drilling probability" of the relation $\langle j$th object type, **R5**, ith object type$\rangle$, or 0 if the relation does not exist in the knowledge graph. We thus get a factor $p(\boldsymbol{CI}|\boldsymbol{C}) = \prod_i p(CI_i|\boldsymbol{C}) = f(\boldsymbol{CI}, \boldsymbol{C}) = f(\boldsymbol{C})$ as a factor node in the factor graph.

- **Dependency factors.** A dependency factor corresponds to the relation **R2** and models how we choose some types of objects to work with other objects, e.g., to choose a seat to work with a TV. It measures how the number $\boldsymbol{CD}$ of objects planned from the relation in the knowledge graph fits the property "dependency" in **R5**. We define the generative process of variables $\boldsymbol{CD}$ as follows:

$$\boldsymbol{CD} = \sum_i \boldsymbol{CD}^{(i)}$$

$$CD_j^{(i)} \sim \text{Poisson}(C_i \cdot dep_{i,j})$$

Here $dep_{i,j}$ is defined as the value of the field "dependency" of the relation $\langle j$th object type, **R2**, ith object type$\rangle$, or 0 if the relation does not exist in the knowledge graph. By modeling dependencies among object types in our knowledge base in this way, we get a factor $p(\boldsymbol{CD}|\boldsymbol{C}) = \prod_i p(CD_i|\boldsymbol{C}) = f(\boldsymbol{CD}, \boldsymbol{C}) = f(\boldsymbol{C})$ as a factor node in the factor graph.

- **Placement factors.** A placement factor corresponds to the relation **R3** and models how we place object instances, including determining the location $\boldsymbol{x}$ and orientation $\boldsymbol{o}$, e.g., placing a seat in front of a TV, so that it faces the screen of the TV. It measures how the layout in the room fits the facts and the parameters in the knowledge graph. From the defined properties of **R3** in Section 3.3, we can get a group of extracted properties:

$$(t_{i,j}, \boldsymbol{u}_{i,j}, \boldsymbol{v}_{i,j}, \boldsymbol{w}_{i,j}) = \text{extract}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{o}_i, \boldsymbol{o}_j)$$

Here $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$, and t denote the primary direction, projected placement, relative orientation, and primary distance of object instance i relative to object instance j.

Given an object instance pair (i, j), we only add a factor node when the hypernyms of their object types have relations of type **R3**. We denote this hypernym pair as (i', j'). We define the generative process for $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$, t as follows:

$$\begin{aligned}
z_{i,j} &\sim \text{Categorical}(\phi_{i',j'}) \\
t_{i,j} &\sim k_{i',j',z_{i,j}}\text{Beta}(\beta_{i',j',z_{i,j}}) + b_{i',j',z_{i,j}} \\
\boldsymbol{u}_{i,j} &\sim \text{Normal}(\mu_{\boldsymbol{u},i',j',z_{i,j}}, \sigma_{\boldsymbol{u},i',j',z_{i,j}}) \\
\boldsymbol{v}_{i,j} &\sim \text{Normal}(\mu_{\boldsymbol{v},i',j',z_{i,j}}, \sigma_{\boldsymbol{v},i',j',z_{i,j}}) \\
\boldsymbol{w}_{i,j} &\sim \text{Normal}(\mu_{\boldsymbol{w},i',j',z_{i,j}}, \sigma_{\boldsymbol{w},i',j',z_{i,j}})
\end{aligned}$$

Here $z_{i,j}$ is the identifier variable of this mixture model. All parameters are generated with a normal distribution, except for primary distance, as a beta distribution can better model the asymmetric impact of distance than a normal distribution. E.g., watching TV from 1.5 m away is the best distance, and while 2.5 m is also a good distance, 0.5 m is too close.

The values of k, b, μ, σ are learnable parameters in the attributes of **R3** relation. We write our factorized probability as $p(I_i|I_j) = f(I_i, I_j) = p(t_{i,j}|I_j)p(\boldsymbol{u}_{i,j}|I_j)p(\boldsymbol{v}_{i,j}|I_j)p(\boldsymbol{w}_{i,j}|I_j)/Z_{i,j}$, where $Z_{i,j}$ is a normalizing constant. Here, $k_{i',j'}$, $\beta_{i',j'}$, $b_{i',j'}$, $\mu_{\boldsymbol{u},i',j'}$, $\mu_{\boldsymbol{v},i',j'}$, $\mu_{\boldsymbol{w},i',j'}$, $\sigma_{\boldsymbol{u},i',j'}$, $\sigma_{\boldsymbol{v},i',j'}$, and $\sigma_{\boldsymbol{w},i',j'}$ by $\theta_{i,j}$ are the parameters in this factor.

- **Instance selection factors.** An instance selection factor corresponds to relation **R4**. We define it as 1 when for each leaf object type i in the hierarchy, the number of instances of that object type placed in the room is exactly C_i, and 0 otherwise. This gives the factorized probability $p(\boldsymbol{I}|\boldsymbol{C}) = f(\boldsymbol{I}, \boldsymbol{C})$.
- **Eligibility factor.** This corresponds to constraints in indoor scene layouts. We set it to 0 when there are collisions among the objects, or an object is floating in the air or placed outside the room, and 1 otherwise. The factorized probability is $p(\boldsymbol{T}, \mathbf{C}, \boldsymbol{I}) = f(\boldsymbol{T}, \boldsymbol{C}, \boldsymbol{I})$.

We thus get the final factorized objective function as follows:

$$\begin{aligned}
&f(\boldsymbol{T}, \boldsymbol{I}|\text{relations}) \\
&\propto \max_{\boldsymbol{C}} p(\boldsymbol{C}, \boldsymbol{I}|\boldsymbol{T}, \text{relations}) \\
&\approx \max_{\boldsymbol{C}} \big(p(\boldsymbol{CT}|\boldsymbol{n})p(\boldsymbol{n}|\boldsymbol{T}, \text{room})p(\boldsymbol{CI}|\boldsymbol{C})p(\boldsymbol{CD}|\boldsymbol{C}) \\
&\quad \cdot p(\boldsymbol{I}|\boldsymbol{C})\big) \prod_{\langle i, \mathbf{R3}, j\rangle} p(I_i|I_j) \\
&= \max_{\boldsymbol{C}} \big(p(\boldsymbol{n}|\boldsymbol{T}, \text{room})p(\boldsymbol{CT}|\boldsymbol{n}, \boldsymbol{pt})p(\mathbf{CI}|\boldsymbol{C}, \boldsymbol{pi}) \\
&\quad \cdot p(\boldsymbol{I}|\boldsymbol{C})\big) \prod_{\langle i, \mathbf{R3}, j\rangle} p(I_i|I_j, \theta_{i,j})
\end{aligned}$$

This gives a factor graph representation of our knowledge in indoor scene design, as shown in Fig. 4.

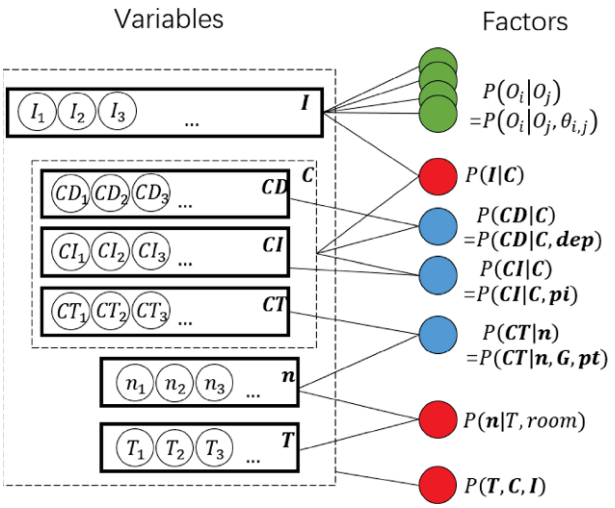


Fig. 4 Mapped factor graph of knowledge in indoor scene design.

4.2 Learning parameters and structure from training data

We learn the knowledge graph from a given training dataset with indoor scenes designed by professional designers. Such a learning problem includes 3 coupled tasks: learning the structure of the knowledge graph, estimating the parameters of relations, and inferring the hidden variables in the factor graph.

Existing work [29] on structure learning of factor graphs has shown that both structure learning and parameter inference can be completed in polynomial time. However the algorithm given there does not fit our work for the following reasons. (i) It assumes that all the variables in the factor graph are observable. However in our work, only the variables $\boldsymbol{I}$ and $\boldsymbol{T}$, which represent the objects in the room and the type of the room, are observed. (ii) It assumes discrete probability distributions over finite sets for each variable. In our case, although $\boldsymbol{n}$, $\boldsymbol{C}$ are discrete variables, each of their components ranges from 0 to $+\infty$, which is not a finite set. To make matters worse, the variables $\boldsymbol{I}$, are not discrete variables. Thus we propose our own solver for this problem.

As all parameters in the factors are independent of each other, optimization of the factorized objective function can also be performed separately:

$$\begin{aligned}&\max_{\text{relations}} f(\boldsymbol{T}, \boldsymbol{I}|\text{relations})\\ =&\max_{\boldsymbol{pt},\boldsymbol{pi},\boldsymbol{dep},\boldsymbol{C},\boldsymbol{n}} (p(\boldsymbol{CT}|\boldsymbol{n})p(\boldsymbol{n}|\boldsymbol{T}, \text{room})p(\boldsymbol{CI}|\boldsymbol{C})\\ &\cdot p(\boldsymbol{CD}|\boldsymbol{C})p(\boldsymbol{I}|\boldsymbol{C})) \times \prod_{\langle i,\mathbf{R3},j\rangle} \max_{\theta_{i,j}} p(I_i|I_j, \theta_{i,j})\end{aligned}$$

As shown in the equation above and Fig. 4, we split the multi-task optimization problem into the following parts:

- **Instance selection.** This corresponds to the factor nodes in red (factors without parameters) or blue (factors with parameters) in Fig. 4, and focuses on the first term in the equation above. The optimization problem includes discrete variables and is a parameter inference problem. The number of factors in this part is fixed. As the structure of the knowledge graph is mapped implicitly to non-zero elements in $\boldsymbol{pt}$, $\boldsymbol{pi}$, and $\boldsymbol{dep}$, this optimization part only needs to carry out (i) the inference of hidden variables $\boldsymbol{C}$ and $\boldsymbol{n}$, and (ii) the estimation of parameters $\boldsymbol{pt}$, $\boldsymbol{pi}$, and $\boldsymbol{dep}$. By solving for them jointly, we can recover both structure and parameters in our knowledge graph from the learned parameters.
- **Instance placement.** This corresponds to the green factor nodes in Fig. 4 and focuses on the second term in the equation above. The optimization problem involves continuous variables but no hidden variables. These factors are associated with **R3** relation in the knowledge graph explicitly, and the number of such factors needs to be learned. As a result, this part needs to (i) learn the structure related to this part and (ii) estimate parameters $\theta_{i,j}$.

Although the original learning problem has complex coupled tasks, the sub-problems we obtain above are much simpler. We can adapt existing solvers for both sub-problems independently:

- **Instance selection.** This includes coupled tasks of hidden variable inference and parameter estimation, which is a common problem in semantic modeling algorithms used with, e.g., topic models [28]. Given a group of parameters and hidden variables in this part, we can quickly evaluate the resulting objective function value for this part. However taking the derivative of the objective function with respect to each separate variable or parameter is difficult. Following work in semantic modeling, the Gibbs sampling method [30] is utilized to solve the coupled problem of hidden variable inference and parameter estimation. However, the total number of relations learned by the sampling algorithm, or the number of non-zero elements in the parameters, might be

huge. This could cause a severe overfitting issue given a limited number of training samples. Thus the Bayesian information criterion [31] (BIC) is employed in our objective function, to extract only the most salient [32] relations from our training samples. The BIC is defined as $k\log(n_{\text{sample}}) - 2\log\hat{L}$, where n_{sample}, k, and $\hat{L}$ denote the number of training samples, the number of free parameters in the model, and the maximized value of the likelihood function, respectively. Accordingly, the objective function for the Gibbs sampling solver is set to $-BIC$:

$$2\log(p(\boldsymbol{CT}|\boldsymbol{n})p(\boldsymbol{n}|\boldsymbol{T},\text{room})p(\boldsymbol{CI}|\boldsymbol{C})p(\boldsymbol{CD}|\boldsymbol{C}) \cdot p(I|C)) - k\log(n_{\text{sample}})$$

where k is the number of non-zero elements in all parameters. With Gibbs sampling, we can get sparse parameters, and map each non-zero element to a relation in our knowledge graph.

- **Instance placement.** This includes coupled tasks of structure and parameter estimation, which is a traditional problem in probabilistic graph models. Using our factorized objective function, each parameter $\theta_{i,j}$ can be estimated independently from other parameters and structure, as $\boldsymbol{I}$ is observable in the model, which is a simple parameter estimation problem with maximum a posteriori (MAP) estimation. The only difficulty in this problem is estimating the normalizing factor $Z_{i,j}$, as it requires integration over a joint distribution of all parameters in $\theta_{i,j}$. In our implementation, this is done with the VEGAS [33] Monte Carlo method.

  As each factor $p(I_i|I_j,\theta_{i,j})$ corresponds to a relation $\langle i,\mathbf{R3},j\rangle$ in our knowledge graph, we can also apply BIC to estimate the set of relations **R3**:

$$\arg\max_{\mathbf{R3}}\left(2\sum_{\langle i,\mathbf{R3},j\rangle}\log\left(p(I_i|I_j,\theta_{i,j})\right) - |\mathbf{R3}|\log(n_{\text{sample}})\right)$$

  which can be solved with greedy search.

5 Applications

5.1 Data sources

We have trained different versions of our knowledge base with a mixture of data sources, to support different indoor design tasks:

- **Dataset A**, built with 50 Autodesk Revit [1] projects collected from professional designers, including 460 rooms.
- **Dataset B**, built with 1194 rooms designed by players of a video game [34], with both usability and aesthetics considered.
- **Dataset C**, built with 233 rooms from the Stanford Scene Dataset [11] by non-professional users.

The library of objects was constructed from man-made objects from ShapeNet [23], and we also added objects from Datasets A, B, and C to it.

For object types and objects in the library, **R4** relation was obtained by classification, learned with MVCNN [35]. The resulting knowledge graph learned from each dataset consisted of more than 200,000 entities and 200,000 relations, most of which were object instance entities and **R4** relations.

We now show the effectiveness of our knowledge graph representation with several applications.

5.2 Alternative indoor scene design

One typical requirement for indoor scene design is to generate alternative designs. Given a design of a room, a user may have certain preferences for the room, and thus may like to generate alternative designs by changing the type of an object, changing a selected instance of a certain type, or moving some instances in the room. Thus it is desirable to generate alternative designs efficiently with these preferences addressed automatically. This application is similar to scene evolution [36], which aims to evolve layouts given a sequence of human activities. The major difference between alternative indoor scene design and scene evolution is with their desired outputs: alternative indoor scene design tries to generate a tidy layout to assist scene designers, whereas scene evolution aims to generate realistic and messy layouts that correspond to human activities.

One major challenge in generating alternative indoor scenes is to find the lead–lag relations in the scene. For example, when a user moves a table, it is highly possible that their intention is to move the chairs around it together with it. However when a user moves a chair, it is likely that they just want to move the chair elsewhere. Such challenges can be addressed with a simple graph-based algorithm by analyzing the lead–lag relations.

We address the problem with the following algorithm:

When the placement of an instance in the scene is changed, or when we replace an instance by

another instance of the same object type, we simply check for a connected component from the changed instance in our knowledge-graph-based representation with only **R3**, **R4**, and **R5** edges considered, and the placements of those instances in the connected components are to be influenced. Thus when the placement of a chair is changed, it does not influence the placement of the table, as there is no path representing dependency in the graph representation. In contrast, when a table is moved, we will move any chairs around it.

When an instance is replaced by another instance of a different object type, we check in our knowledge graph and compare the contexts of these object types in the knowledge graph. We then modify the scene accordingly. For example, a non-wall-mounted TV needs a base cabinet to support it, and a wall-mounted TV does not. When we replace a non-wall-mounted TV to a wall-mounted one, we should try to remove the base cabinet if the base cabinet is not a dependency of any other instance in the scene.

We show a typical example in Fig. 5. Figure 5(a) shows the original design of a living room, which can also be used for dining. As the room is nearly empty, the user first moves the table to the left half of the room (see Fig. 5(b)) and the chairs are also moved by our algorithm. The TV is not moved with our lead–lag analysis based on the knowledge graph, and our algorithm adds an armchair in front of the TV after re-analyzing the dependency relations of the TV. When the TV is changed to a wall-mounted TV (see Fig. 5(c)), our algorithm automatically removes the TV cabinet that originally supported the non-wall-mounted TV. In Fig. 5(d), the square table is replaced by a long table not included in the training dataset. Our algorithm automatically transfers the **R3** relation learned from other tables, and places some chairs around it. The right half of the room is still quite empty, so the user decides to add a coffee table (see Fig. 5(e)), and our algorithm changes the seat originally placed there to a sofa, places it next to the wall, and automatically adds another seat. It also adds two side tables beside the sofa. In Fig. 5(f), we move a chair around the dining table, to simulate a scenario of a family sitting around the coffee table, watching TV. In this case, the dining table in the chair's context is not moved, by the lead–lag analysis.

For the example in Fig. 5(e), we further compare

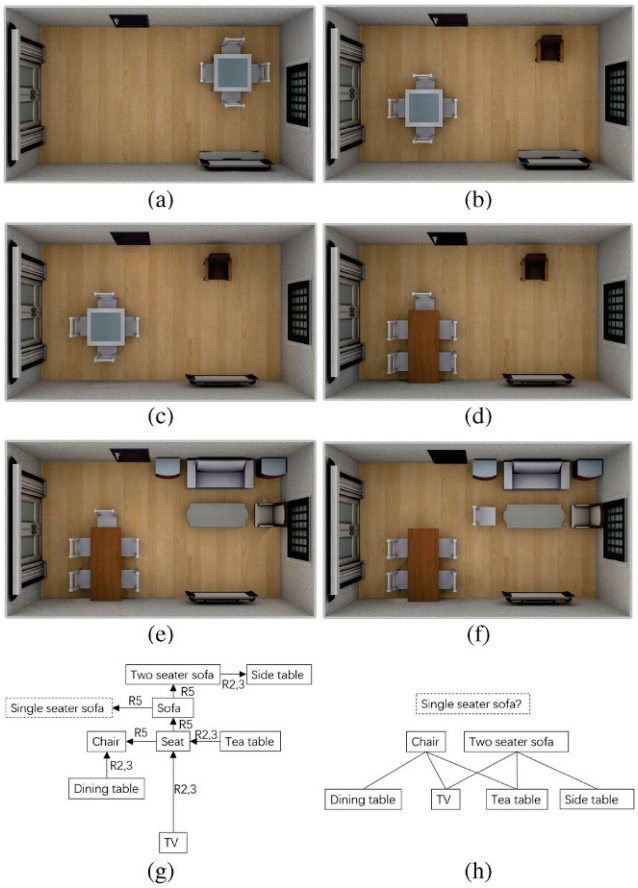


Fig. 5 An example of alternative room design showing several steps. (a) Original room design. (b) The user moves the table to the left half of the room. (c) The user changes the TV to a wall-mounted one. (d) The user changes the square table to a long table. (e) The user adds a coffee table in front of the TV. (f) We move a chair around the dining table. (g) Underlying knowledge graph for (e), with various entities and relations removed for clearer illustration. (h) Co-occurrence-based representation for (e) as commonly used in previous work.

our knowledge graph representation with the pairwise co-occurrence modeling used by Refs. [10, 25], as shown in Figs. 5(g) and 5(h), respectively. The major differences between these two representations are: (i) ours uses *directed* relations between entities, and (ii) ours introduces several levels of hypernyms for each object. In the alternative scene design application, such differences have significant benefits. By using directed relations, we can find lead–lag relations for those objects, and analyze dependencies between the objects. Figures 5(b), 5(d), and 5(e) show some successful modifications according to the dependency analysis, while Fig. 5(f) demonstrates the necessity of such analysis. By introducing different levels of hypernyms, the relations among the entities are easier to transfer, as shown in Fig. 5(d). A

graphical representation of such transfer is presented in Figs. 5(g) and 5(h), where we add a single-seater sofa not included in the training dataset. As shown in Fig. 5(g), the relations of its hypernym "sofa" and "seat" can be implicitly transferred to the hyponym "single seat sofa" with a single **R5** relation, whereas in co-occurrence models lacking a lexical hierarchy, the user has to go through all similar concepts and manually decide which co-occurrence relations to transfer. Such transferability can be very helpful in reducing the required scale and coverage of training data.

5.3 Inference tasks

Some interactions supported by existing works can be formulated as a simple MAP hidden variable inference problem based on our factor graph representation. The most typical interactions include finding a proper location for a user-selected instance [4], retrieving an instance for a user-selected placement [9], etc.

We present a typical example of inference tasks in Fig. 6. Figure 6(a) shows the result of finding a proper location for a chair in the room. The red chairs show six local maxima of posterior probability for placing the chair, indicating their **R2** and **R3** relations to the tea table and dining table, where the **R2** relation encodes the table that the chair depends on, and its position and rotation are further determined by the parameters of the **R3** relation. Figures 6(b) and 6(c) show the result of retrieving instances with user-selected locations with the relations of small objects learned from Dataset B. By retrieving **R2**, **R4**, **R5** relations in the knowledge graph for the hyponym concept and corresponding instances to be placed in the room filtered by **R3**, our algorithm identifies a plant, a tablet, a cabinet, and a plate for those locations. However, due to the limitation of our knowledge graph representation, without function labels for objects, our algorithm may select instances with duplicated function (see Fig. 6(d)).

5.4 Fully-automatic indoor design generation

Another challenging problem is the fully-automatic indoor design generation problem. Existing work [5] in open world synthesis can only synthesize a room with a certain room type with manually designed factor-based criteria for specific room types. Here we replace their objective function with our factorized formulation based on our knowledge graph, to adapt the model to different room types. Instead of designing criteria for each specific type, we include some general aesthetic measures [37] to avoid strange results. Figure 7 shows some results.

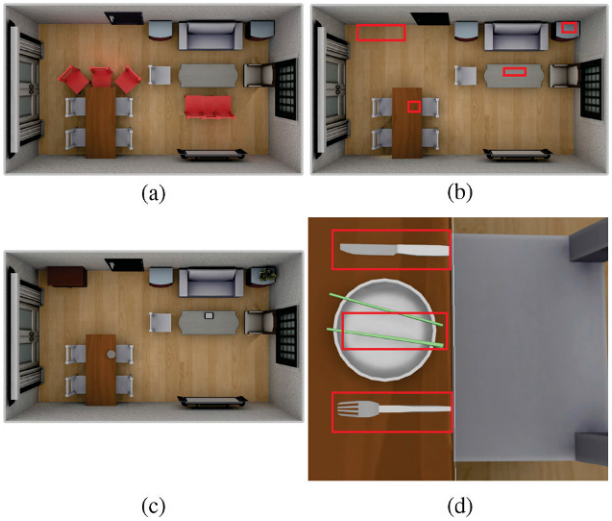


Fig. 6 Typical inference tasks. (a) Finding a proper placement for a user-selected instance. (b, c) Retrieving instances for selected placements. (d) A typical failure in retrieving instances for selected placements: both knife & fork and chopsticks are added.

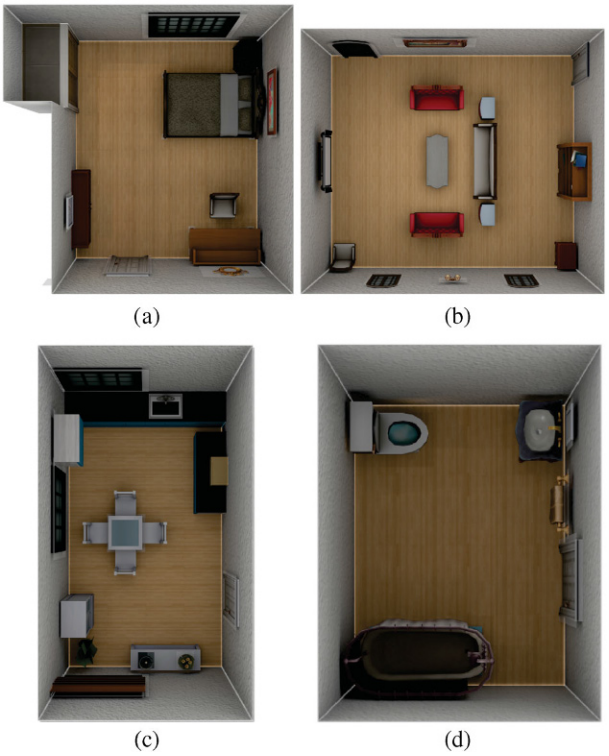


Fig. 7 Some rooms generated by automatic indoor design. Each case takes a different room type vector $\boldsymbol{T}$ as input, to generate (a) a bedroom, (b) a living room, (c) a dining room integrated with a kitchen, and (d) a toilet.

6 Discussion and conclusions

This work has proposed a knowledge-graph-based framework which addresses many practical problems in computer-aided indoor scene design. The relations can be learned with small-scale training datasets of indoor scenes, and can be easily transferred and adapted to suit practical needs. Various practical examples show that our framework is effective and can be easily embedded into existing applications. However, we can still point out several limitations and possible improvements of our work.

Firstly, to overcome sparsity and lack of training data, we employ an ontology based on ShapeNet [23] to assist our knowledge graph construction, which partially solves the issue. However, given a limited number of training scenes, we cannot claim that our knowledge graph trained with this dataset contains *every fact* about indoor scene design, since for some rare object types, we can find few if any instances in the training scenes, which makes it difficult to learn their relations. Possible improvements for such problems may include employing knowledge graph embedding techniques [38], which have shown success in relation prediction for long-tail entities.

Secondly, some placements of objects are not based on the contextual relations between them, but on their context in human interaction. For example, we may learn to "place a knife to the right side of a plate" and "place a fork to the left side of a plate". When we do not place a plate in the scene, the algorithms based on our framework do not know how to place knife and fork, even if a napkin has been placed. However with the underlying fact "it is conventional for a person to use a knife with their right hand and a fork with their left hand", a person can easily know how to place them. Modeling the human context makes it possible to handle cases like this. Therefore, a possible direction for further improvement is to include human activity information [36, 39] to our knowledge graph, an approach which has shown success in both realistic scene synthesis and in using evolutionary instructions.

Acknowledgements

This work was supported by the National Key R&D Program of China (No. 2017YFB1002604), the National Natural Science Foundation of China (No. 61772298), a Research Grant of Beijing Higher Institution Engineering Research Center, and the Tsinghua–Tencent Joint Laboratory for Internet Innovation Technology.

References

[1] Autodesk REVIT. Available at `https://www.autodesk.com/products/revit-family/overview`.

[2] SketchUp. Available at `https://www.sketchup.com/`.

[3] ARCHICAD. Available at `http://www.graphisoft.com/archicad/`.

[4] Fisher, M.; Hanrahan, P. Context-based search for 3D models. *ACM Transactions on Graphics* Vol. 29, No. 6, Article No. 182, 2010.

[5] Yeh, Y.-T.; Yang, L.; Watson, M.; Goodman, N. D.; Hanrahan, P. Synthesizing open worlds with constraints using locally annealed reversible jump MCMC. *ACM Transactions on Graphics* Vol. 31, No. 4, Article No. 56, 2012.

[6] Dalton, J.; Dietz, L.; Allan, J. Entity query feature expansion using knowledge base links. In: Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval, 365–374, 2014.

[7] Li, F.; Dong, X. L.; Langen, A.; Li, Y. Knowledge verification for long-tail verticals. *Proceedings of the VLDB Endowment* Vol. 10, No. 11, 1370–1381, 2017.

[8] Zhu, X.; Anguelov, D.; Ramanan, D. Capturing long-tail distributions of object subcategories. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 915–922, 2014.

[9] Savva, M.; Chang, A. X.; Agrawala, M. Scenesuggest: Context-driven 3D scene design. *arXiv preprint* arXiv:1703.00061, 2017.

[10] Yu, L. F.; Yeung, S.-K.; Terzopoulos, D. The clutterpalette: An interactive tool for detailing indoor scenes. *IEEE Transactions on Visualization and Computer Graphics* Vol. 22, No. 2, 1138–1148, 2016.

[11] Fisher, M.; Ritchie, D.; Savva, M.; Funkhouser, T.; Hanrahan, P. Example-based synthesis of 3D object arrangements. *ACM Transactions on Graphics* Vol. 31, No. 6, Article No. 135, 2012.

[12] Xu, K.; Chen, K.; Fu, H.; Sun, W.-L.; Hu, S.-M. Sketch2Scene: Sketch-based co-retrieval and co-placement of 3D models. *ACM Transactions on Graphics* Vol. 32, No. 4, Article No. 123, 2013.

[13] Chang, A. X.; Eric, M.; Savva, M.; Manning, C. D. SceneSeer: 3D scene design with natural language. *arXiv preprint* arXiv:1703.00050, 2017.

[14] Chen, K.; Lai, Y.-K.; Hu, S.-M. 3D indoor scene modeling from RGB-D data: a survey. *Computational Visual Media* Vol. 1, No. 4, 267–278, 2015.

[15] Mihalcea, R.; Radev, D. *Graph-based Natural Language Processing and Information Retrieval.* Cambridge University Press, 2011.

[16] Yao, X.; Van Durme, B. Information extraction over structured data: Question answering with freebase. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, 956–966, 2014.

[17] Socher, R.; Chen, D.; Manning, C. D.; Ng, A. Reasoning with neural tensor networks for knowledge base completion. In: Proceedings of the Advances in Neural Information Processing Systems 26, 926–934, 2013.

[18] Marino, K.; Salakhutdinov, R.; Gupta, A. The more you know: Using knowledge graphs for image classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2673–2681, 2017.

[19] Lehmann, J.; Isele, R.; Jakob, M.; Jentzsch, A.; Kontokostas, D.; Mendes, P. N.; Hellmann, S.; Morsey, M.; van Kleef, P.; Auer, S.; Bizer, C. DBpedia—A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* Vol. 6, No. 2, 167–195, 2015.

[20] Chang, A. X.; Savva, M.; Manning, C. D. Learning spatial knowledge for text to 3D scene generation. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2028–2038, 2014.

[21] Dong, X.; Gabrilovich, E.; Heitz, G.; Horn, W.; Lao, N.; Murphy, K.; Strohmann, T.; Sun, S.; Zhang, W. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 601–610, 2014.

[22] Hoffart, J.; Suchanek, F. M.; Berberich, K.; Lewis-Kelham, E.; de Melo, G.; Weikum, G. YAGO2: Exploring and querying world knowledge in time, space, context, and many languages. In: Proceedings of the 20th International Conference Companion on World Wide Web, 229–232, 2011.

[23] Chang, A. X.; Funkhouser, T.; Guibas, L.; Hanrahan, P.; Huang, Q.; Li, Z.; Savarese, S.; Savva, M.; Song, S.; Su, H.; Xiao, J.; Yi, L.; Yu, F. ShapeNet: An information-rich 3D model repository. *arXiv preprint* arXiv:1512.03012, 2015.

[24] Miller. G. A. WordNet: A lexical database for English. *Communications of the ACM* Vol. 38, No. 11, 39–41, 1995.

[25] Fisher, M.; Savva, M.; Hanrahan, P. Characterizing structural relationships in scenes using graph kernels. *ACM Transactions on Graphics* Vol. 30, No. 4, Article No. 34, 2011.

[26] Zeng, J.; Cheung, W. K.; Liu, J. Learning topic models by belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* Vol. 35, No. 5, 1121–1134, 2013.

[27] Kschischang, F. R.; Frey, B. J. Iterative decoding of compound codes by probability propagation in graphical models. *IEEE Journal on Selected Areas in Communications* Vol. 16, No. 2, 219–230, 1998.

[28] Blei, D. M.; Ng, A. Y.; Jordan, M. I. Latent dirichlet allocation. *Journal of Machine Learning Research* Vol. 3, 993–1022, 2003.

[29] Abbeel, P.; Koller, D.; Ng, A. Y. Learning factor graphs in polynomial time and sample complexity. *Journal of Machine Learning Research* Vol. 7, 1743–1788, 2006.

[30] Gelfand, A. E.; Hills, S. E.; Racine-Poon, A.; Smith, A. F. M. Illustration of Bayesian inference in normal data models using Gibbs sampling. *Journal of the American Statistical Association* Vol. 85, No. 412, 972–985, 1990.

[31] Schwarz, G. Estimating the dimension of a model. *The Annals of Statistics* Vol. 6, No. 2, 461–464, 1978.

[32] Soleimani, H.; Miller, D. J. Parsimonious topic models with salient word discovery. *IEEE Transactions on Knowledge and Data Engineering* Vol. 27, No. 3, 824–837, 2015.

[33] Lepage, G. P. A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics* Vol. 27, No. 2, 192–203, 1978.

[34] The SIMS 4. Available at `https://www.ea.com/games/the-sims?isLocalized=true`.

[35] Su, H.; Maji, S.; Kalogerakis, E.; Learned-Miller, E. Multi-view convolutional neural networks for 3D shape recognition. In: Proceedings of the IEEE International Conference on Computer Vision, 945–953, 2015.

[36] Ma, R.; Li, H.; Zou, C.; Liao, Z.; Tong, X.; Zhang, H. Action-driven 3D indoor scene evolution. *ACM Transactions on Graphics* Vol. 35, No. 6, Article No. 173, 2016.

[37] Merrell, P.; Schkufza, E.; Li, Z.; Agrawala, M.; Koltun, V. Interactive furniture layout using interior design guidelines. *ACM Transactions on Graphics* Vol. 30, No. 4, Article No. 87, 2011.

[38] Lin, Y.; Liu, Z.; Sun, M.; Liu, Y.; Zhu, X. Learning entity and relation embeddings for knowledge graph completion. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence, 2181–2187, 2015.

[39] Fisher, M.; Savva, M.; Li, Y.; Hanrahan, P.; Nießner, M. Activity-centric scene synthesis for functional 3D scene modeling. *ACM Transactions on Graphics* Vol. 34, No. 6, Article No. 179, 2015.

Yuan Liang is a Ph.D. candidate in the Department of Computer Science and Technology, Tsinghua University, China. His research interests include interactive multimedia analysis and geometry processing. He received his B.S. degree from the Department of Computer Science and Technology, Tsinghua University.

Fei Xu received his B.S. degree from Guangdong University of Technology. He is an interdisciplinary master student in the Department of Information Art and Design, Tsinghua University. His interests include interactive multimedia analysis, VR and AR, and human computer interaction.

Song-Hai Zhang received his Ph.D. degree from Tsinghua University, China, in 2007. He is currently an associate professor of computer science in Tsinghua University. His research interests include image and video processing as well as geometric computing.

Yu-Kun Lai received his bachelor and Ph.D. degrees in computer science from Tsinghua University, China, in 2003 and 2008, respectively. He is currently a senior lecturer at the School of Computer Science & Informatics, Cardiff University. His research interests include computer graphics, geometry processing, image processing, and computer vision. He is on the Editorial Board of *The Visual Computer*.

Taijiang Mu is currently a postdoctoral researcher in the Department of Computer Science and Technology, Tsinghua University, where he received his Ph.D. and B.S. degrees in 2016 and 2011, respectively. His research area is computer graphics, mainly focusing on stereoscopic image and video processing, and stereoscopic perception.