Synthesizing Open Worlds with Constraints using Locally Annealed Reversible Jump MCMC

Yi-Ting Yeh* Stanford University
Lingfeng Yang* Stanford University
Matthew Watson* Stanford University
Noah D. Goodman* Stanford University
Pat Hanrahan* Stanford University

Figure 1: *The table-chair sets, arm chairs, plants, shelves, and floor lamps in this coffee shop were arranged using our locally annealed reversible jump MCMC sampling method. The users don't need to specify the number of objects beforehand.*

Abstract

We present a novel Markov chain Monte Carlo (MCMC) algorithm that generates samples from transdimensional distributions encoding complex constraints. We use factor graphs, a type of graphical model, to encode constraints as factors. Our proposed MCMC method, called locally annealed reversible jump MCMC, exploits knowledge of how dimension changes affect the structure of the factor graph. We employ a sequence of annealed distributions during the sampling process, allowing us to explore the state space across different dimensionalities more freely. This approach is motivated by the application of layout synthesis where relationships between objects are characterized as constraints. In particular, our method addresses the challenge of synthesizing *open world* layouts where the number of objects are not fixed and optimal configurations for different numbers of objects may be drastically different. We demonstrate the applicability of our approach on two open world layout synthesis problems: coffee shops and golf courses.

CR Categories: G.3 [Probability and Statistics]: Probabilistic algorithms—Markov chain Monte Carlo;

Keywords: factor graphs, open worlds, constrained synthesis

Links: DL PDF

1 Introduction

Constraints are powerful ways to describe the layout of man-made and natural objects. For example, the arrangement of furniture in a room can be described by a set of constraints that specify that furniture does not overlap, that chairs face each other in seating arrangements, and that sofas are placed with their backs against the wall, etc. Books have been written describing the constraints that characterize pleasing design patterns [Alexander et al. 1977].

Recently in graphics, researchers have applied optimization algorithms and probabilistic inference to the problem of constrained layout synthesis. For example, a parallel tempered Metropolis-Hastings algorithm has been used to synthesize furniture layout [Merrell et al. 2011]. A similar layout problem has been solved using simulated annealing [Yu et al. 2011]. In these two systems, the designers need to specify the number and types of objects in advance. However, a general layout algorithm needs to choose the number of tables and the types of furniture as well as their positions and orientations. For example, consider the placement of tables and chairs in the design of a coffee shop. Good layouts satisfy the constraints of physical plausibility and functionality, but also crowdedness and appropriateness of the furniture. An open layout will have fewer tables than a crowded layout.

We use probability distributions that have a variable set of random variables to characterize these kinds of layout problems. In the field of AI, if the world has a fixed set of objects, it is called *closed universe*. Whereas, if it has a variable number of objects, it is known as *open universe* [Milch et al. 2005].

In graphics, open universe distributions can be created using procedural grammars. Talton et al. [2011] have employed reversible jump MCMC, an inference algorithm for open universe distributions, to constrain the output of grammars describing models such as trees, cities, and Mondrian paintings. In particular, they use parallel tempered RJMCMC with delayed rejection in order to find high probability models more efficiently.

In this paper, we focus on open universe layout problems for which there do not exist good descriptions as grammars, that also have complex constraints. We describe the layout problem in two parts: (i) a probabilistic generative model that creates random variables, and (ii) a system of constraints encoded as a factor graph. We call such layouts *open world* layouts.

Our goal is to synthesize diverse layouts that satisfy constraints. In order to do this, we formulate the space of layouts as a probability distribution and use a sampling algorithm to synthesize layouts. Sampling, where the chance of a layout being produced is proportional to its probability, is useful if more than one layout needs to be

*e-mail:{yitingy, lfyg, mdwatson, ngoodman, hanrahan}@stanford.edu

generated, and not just the best ones. This is in contrast to pure optimization methods, which are used to produce the highest scoring members of a space.

However, we find that state of the art MCMC techniques often cannot efficiently generate open world layouts. In this paper, we propose a new probabilistic inference algorithm called locally annealed RJMCMC (LARJ-MCMC) to address this challenge. The idea behind LARJ-MCMC is to introduce a sequence of intermediate distributions between jump moves. The sequence interpolates between the old factors and the new factors. We prove that it is a valid MCMC method and show how, through exploiting knowledge of how the structure of the constraints change when adding and removing variables, LARJ-MCMC can more efficiently generate good configurations with different sets of variables. To our knowledge, we are the first to address the combination of complex constraints and varying number of objects which are prevalent in many instances of layouts. Our contributions are:

1. We formalize open world layout synthesis as sampling from an open universe probability distribution with constraints encoded as factor graphs.
2. We propose a novel MCMC method, called locally annealed reversible jump MCMC, which exploits the structure of the repeated constraints to synthesize layouts more efficiently.

We formulate open world distributions in Section 3. In Section 4, we first describe the challenges of synthesizing open world layouts using previous methods and then present the core of our algorithm, the locally annealed jump proposal. In Section 5, we show how our algorithm performs on complex examples.

2 Related Work

Probabilistic graphical models. Probabilistic graphical models have been used for various prediction and learning tasks. Two common classes of graphical models are directed (e.g. Bayesian networks (BN)) and undirected (e.g. Markov random fields (MRF)). Koller et al. [2009] summarize these models. Factor graphs [Kschischang et al. 2001], a comparatively modern invention in graphical models, subsume both BN and MRF [Frey 2003]. However, it is not necessary for a graphical model to be purely directed or undirected. Hybrid graphical models including both directed and undirected relations, such as chain graphs [Frydenberg 1990], have also proven to be useful.

Graphical models typically specify probability distributions over a fixed set of random variables. It is also possible to define distributions where the number of random variables can change. Recently, a number of representations have appeared that explicitly handle such probability distributions. One can specify the number and dependence of random variables explicitly in BLOG [Milch et al. 2005], quantify over random variables in Markov logic networks [Richardson and Domingos 2006], or construct the variables during the execution of a probabilistic program [Goodman et al. 2008]. In this paper, we use factor graphs to encode constraints in open world layouts and describe a specification language to construct factor graphs dynamically.

Markov chain Monte Carlo. Markov chain Monte Carlo (MCMC) methods have been widely used in probabilistic inference. Two of the most common MCMC methods used to sample from distributions are Metropolis-Hastings (MH) [Hastings 1970] and Gibbs sampling [Geman and Geman 1984]. Green [1995] included jump moves (birth and death) in his reversible jump MCMC method to sample from distributions that can vary in their number of dimensions. In MH settings, proposal functions are critical to the performance of the sampler, especially when there are isolated modes in the target distribution. One class of algorithms tries to generate better proposals by constructing a sequence of intermediate auxiliary states, leading to candidate states with higher acceptance probabilities. Members of this class include tempered transitions [Neal 1994], delayed rejection [Green and Mira 1999] [Mira 2001], mode jumping incorporating deterministic optimization [Tjelmeland and Hegstad 1999], and multiple-try methods [Liu et al. 2000]. Our method belongs to this category. In particular, we exploit knowledge of the graphical model structure to improve the efficiency of exploring states across different dimensionalities. We anneal in new factors and anneal out old factors during jump moves that change the number of dimensions.

Content generation via probabilistic inference. The optimal spatial arrangement of a pre-specified set of elements according to a cost function is a well-studied problem that occurs in many disciplines. Recently, techniques for the optimization of layouts based on MCMC have been applied to specific domains in graphics, such as the arrangement of furniture in a room [Yu et al. 2011] [Merrell et al. 2011]. While these methods handle the problem for a fixed set of elements, it is still the user's job to pick the set of elements beforehand. In this work we consider the problem of *open world* layouts, where the existence and number of elements is not considered as given, but rather as another design variable that is influenced by the cost function. This is similar to the work by Talton et al. [2011] where parallel tempered RJMCMC is used to sample models from a grammar that satisfy additional constraints. In this paper, we focus on models that are better described declaratively as systems of constraints rather than as being generated by a grammar-like procedural process.

3 Representation

We define open world layouts as states $\mathbf{x}$ in a state space $\mathbf{X}$ of varying dimensionality. We synthesize layouts by sampling from a target distribution π over $\mathbf{X}$. Informally, we can consider open world layouts as being generated in two phases. The first phase starts by randomly generating objects. In this phase, both the types and the numbers of objects can vary. Constraints can then be defined on these configurations of random objects using *factors*. Factors are functions that map the values of a random variable to real-valued scores. Configurations that satisfy the constraints have higher scores. These functions change the landscape of the probability distribution from a relatively flat one to one where there are peaks corresponding to satisfying configurations.

We then decompose π into two components corresponding to these two phases. One extends the state space by creating new random variables, and the other shapes the energy landscape of the existing state space. Our synthesis algorithm is designed to work with any such π. Thus, we formulate π for open world layouts by using a programming language extended to include probabilistic primitives. The use of programming languages to describe probability distributions is a recent development in AI and machine learning [Lunn et al. 2000] [Milch et al. 2005] [McCallum et al. 2009] [Goodman et al. 2008]. We will first give a formal specification of the semantics of the language, and walk through a simple example to provide intuition for the semantics.

3.1 Specification of Open World Distributions

Our specification language consists of a standard imperative programming language augmented with two probabilistic primitives, random variables and factors, corresponding to the creation and constraint components of π.

(a)

```
1  def sample_plate():
2    return Plate(
3           position ~ (Uniform(-10, 10), Uniform(-10, 10))
4           size ~ Uniform(0.1, 2.0))
5
6  def sample_table():
7    return Table(
8       num_plates ~ randint(1, 10)
9       size ~ Uniform(15.0, 25.0))
10
11 Table = sample_table()
12
13 for i in 1:Table.num_plates:
14   Plates[i] = sample_plate()
15
16 #Occupied area factor
17 Factor(SoftEq(0.7, total_area(Plates)/area(Table)))
18
19 #Inside factor
20 for Plate_i in Plates:
21   Factor(SoftEq(0.0, area_outside_table(Plate_i)))
22
23 #Non-overlap factor
24 for (Plate_i, Plate_j) in pairs(Plates):
25   Factor(SoftEq(0.0, overlap_area(Plate_i, Plate_j))
```

(b)

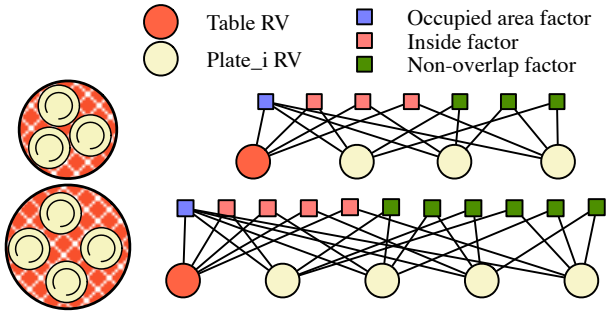

Figure 2: *Distributions of open world layouts can be described by imperative programs. Here we show an example of placing plates on a table. The program (a) dynamically instantiates a set of plates and constraints (probabilistic factors) over them, pictured in (b). Here, the constraints are that plates do not overlap, they are inside the table boundaries, and occupy 70% of the table area.*

In our language, random variables are declared by calling sampling functions that draw values from a standard probability distribution or other random variables. For example, the `Uniform(a, b)` sampling primitive declares a real-valued random variable, and the expression `Uniform(0, 1) < 0.5` declares a boolean random variable whose distribution is uniform over $\{\texttt{true}, \texttt{false}\}$. In general, to create random variables of a given type, the programmer provides a sampling function that produces values of that type. Defining random variables by providing functions to sample them is very similar in spirit to the probabilistic language Church [Goodman et al. 2008].

In addition to constructing and computing random variables, we can associate scoring functions, or factors, to configurations of random variables. In our setting, factors represent constraints, and the execution of each `Factor` statement corresponds to the instantiation of a constraint on the open world layout. The subset of the language dealing with factors is very similar in spirit to FACTORIE [McCallum et al. 2009].

To give intuition for the semantics of our language, we present a simple layout problem: placing plates on a table (see Figure 2(a)).

$$P(\mathbf{X}) \propto \prod_{i=1}^{4} f_i$$

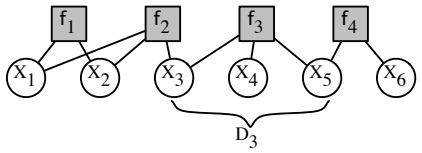

Figure 3: *A factor graph that describes the joint distribution $P(X_1, \ldots, X_6)$ with the factorization $f_1(X_1, X_2)$, $f_2(X_1, X_2, X_3)$, $f_3(X_3, X_4, X_5)$, and $f_4(X_5, X_6)$. $D_3 = (X_3, X_4, X_5)$ is the scope of f_3.*

The `sample_table` function at line 11 in the program creates a random variable `Table` with a corresponding prior distribution $P_{\texttt{Table}}$, defined in terms of how its fields are sampled. In particular, `Table.num_plates` is another random variable representing the number N of place settings `num_plates`, sampled uniformly between 1 and 10. Next, on lines 13 to 14, we use the sampled value of `Table.num_plates` to create a set of Plate-type random variables, `Plates[i]`, $i \in \{1 \ldots N\}$, sampling their locations uniformly from the region $[-10, 10] \times [-10, 10]$.

The rest of the probability distribution is defined by the next part of the program where we impose constraints on the random variables. On line 17, we create a factor that specifies that the total area occupied by the plates is 70%. One line 20, we create a factor that ensures each plate is on the table. Finally, in lines 24-25, we create factors that enforce non-overlapping of plates.

Formally, the overall probability distribution resulting from the program is defined as follows:

$$\pi \propto \pi_1 \pi_2 \pi_3 \pi_4,$$

where π_1 corresponds to the prior distribution induced by the sampling of the table and plates, and π_2, π_3, π_4 correspond to the factors instantiated:

$$\pi_1 = P_{\texttt{Table}}(x_0) \prod_{i=1}^{N} P_{\texttt{Plate}}(x_i).$$

$$\pi_2 = \text{Eq}(0.7, \text{total\_area}(x_1 \ldots x_N)/\text{area}(x_0))$$

$$\pi_3 = \prod_{i=1}^{N} \text{Eq}(0.0, \text{area\_outside\_table}(x_i)),$$

$$\pi_4 = \prod_{i=1}^{N} \prod_{j=1, j \neq i}^{N} \text{Eq}(0.0, \text{overlap\_area}(x_i, x_j)).$$

and Eq represents soft equality of real numbers. The precise formulation of Eq is given in the supplementary materials.

3.2 Relationship to Factor Graphs

Another view of this program is that it creates a mathematical model of the probability distribution as a factor graph. A factor graph (Figure 3) is composed of random variable nodes, factor nodes, and edges between them. Circles represent random variable nodes and squares represent factor nodes. Each factor node is connected to a set of random variable nodes, known as the *scope* of the factor, D_i. For each factor node, there is a *factor function*, f_i, that takes assignments to the random variables in its scope to real-valued numbers. The unnormalized joint probability encoded by a factor graph is the product of all its factor functions.

Figure 2(b) depicts two factor graphs specified by the plate layout program, corresponding to two layouts with different numbers of

objects. Our algorithm, described in Section 4, exploits knowledge of how the structure of the factor graph changes when the number of dimensions changes. This allows us to interpolate between energy landscapes of different dimensionalities, resulting in more efficient sampling in the presence of complex constraints.

3.3 Staged Synthesis

Many layout synthesis problems may be broken up into different stages. For example, a layout synthesis involving tables and chairs in addition to place settings can be broken up into two stages. First, we synthesize the layout of tables and chairs only. Then, the placement of individual place settings containing plates, tableware, and glasses is synthesized while holding the tables and chairs fixed. Although this is an approximation to the full synthesis problem, stages capture the natural hierarchy found in many layouts.

A stage is defined to be an independent constrained synthesis problem conditioned on the results from previous stages; that is, there can be no dependency from the current stage back to the previous one. We can exploit this property to speed up the synthesis process. Each stage runs a sampler conditioned on the values sampled in previous stages. If there is a branching hierarchy of stages, then each subsequent stage may contain multiple independent sub-problems; each of these can then be run in parallel. The technique of breaking down a sampling problem into separate stages has been used to speed up probabilistic inference in general [Pfeffer 2007].

4 Locally Annealed Reversible Jump MCMC

The main difficulty our algorithm addresses is layout problems where the set of good layouts changes drastically when the number of dimensions changes.

4.1 Metropolis-Hastings Basics

We formulate the problem of generating layouts as sampling from the target distribution π. Our algorithm belongs to a class of Markov chain Monte Carlo (MCMC) methods known as Metropolis-Hastings (MH).

The MH procedure generates a sequence of samples from a target distribution by iteratively applying a proposal step and an acceptance step. Let $q(x^*|x)$ be the proposal function generating a candidate state x^* given the current state x. The next state is set to x^* with the probability equal to the MH acceptance ratio:

$$\alpha = \min\left\{1, \frac{\pi(x^*)q(x|x^*)}{\pi(x)q(x^*|x)}\right\}. \quad (1)$$

Otherwise, x remains as the next state. After an initial burn-in period, states traversed using this procedure can be considered as samples from the target probability distribution π.

In the most general setting, the proposal function can be any transformation of the current state, including moves that change the number of dimensions. In the case where dimension changing moves are used as proposals, it is called Reversible Jump Markov chain Monte Carlo (RJMCMC). The following acceptance probability is used for a change from m to n dimensions:

$$\alpha = \min\left\{1, \frac{\pi(x_n^*)q(x_m|x_n^*)}{\pi(x_m)q(x_n^*|x_m)} \times \mathcal{J}_{f_{m\to n}}\right\}. \quad (2)$$

$f_{m\to n}$, the dimension matching function [Green 1995], is used to map the variables at dimensionalities m and n into a space of common dimensionality where $\pi(x_m)q(x_n^*|x_m)$ and $\pi(x_n^*)q(x_m|x_n^*)$ can be meaningfully compared. This is usually done by introducing additional $n-m$ parameters $u_{m,n}$ that compensate for an increase in dimensionality, or projecting out the corresponding $m-n$ parameters in the case of a decrease. $\mathcal{J}_{f_{m\to n}}$ is the Jacobian of the dimension matching function:

$$\mathcal{J}_{f_{m\to n}} = \left|\det \frac{\partial f_{m\to n}(x_m, u_{m,n})}{\partial(x_m, u_{m,n})}\right|. \quad (3)$$

In the examples used in this paper, the Jacobian is 1 since the new variables are always independently sampled from their domains.

The key to the design of MH-based algorithms is the proposal functions because it would affect the rate of mixing of the chain. That is, it affects how quickly the high probability states are explored. Our locally annealed RJMCMC employs a jump proposal specially designed to more efficiently traverse high probability states of different dimensionality.

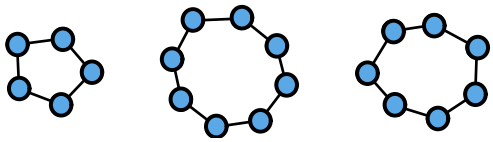

Figure 4: *A simple open world layout example that nevertheless exhibits isolated modes across dimensions. There are two constraints: (1) A fixed distance between consecutive pairs of circles. (2) The circles are arranged as straight as possible.*

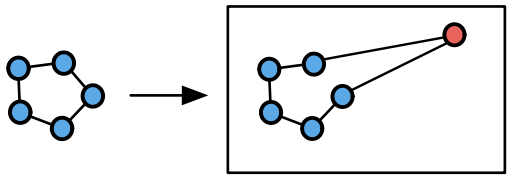

Figure 5: *The situation after a typical reversible jump proposal. In order for the jump itself to be immediately accepted, the new circle needs to be in the correct place in addition to all other circles fitting around it. Our algorithm addresses this challenge by the letting the new object adapt to the newly instantiated constraints.*

4.2 Reversible Jump MCMC

In principle, we can apply the original RJMCMC algorithm to synthesize open world layouts. However, it is often the case that adding or removing a random variable to/from the current configuration produces a low probability state. This results in a very low acceptance probability for jump moves.

We illustrate this effect for a simple layout problem (Figure 4). There are two soft constraints: (1) Every consecutive pair of circles is a fixed distance d apart. (2) The curvature among any three consecutive circles is 0, i.e., constrained to be as straight as possible. Figure 4 shows three high probability states with different numbers of objects.

Whenever a new random variable is added, there are new factors introduced as well as factors removed, resulting in change in the energy landscape. Consider the jump proposal function where the position of this new variable is uniformly sampled from the domain.

Figure 5 shows two states before and after a jump move is made that adds a new circle to the layout. We can see that the likelihood that this new variable is placed at a good location is very low. In

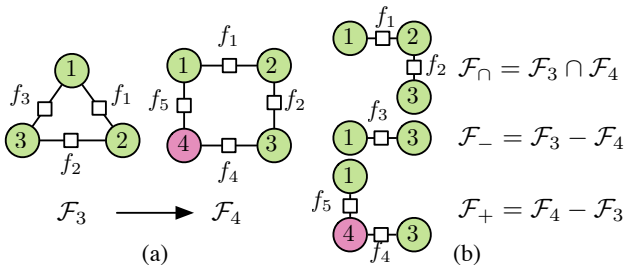

Figure 6: *(a) The set of factors in dimension 3 and 4 are $\mathcal{F}_3 = \{f_1, f_2, f_3\}$ and $\mathcal{F}_4 = \{f_1, f_2, f_4, f_5\}$, respectively. (b) Factors may be added, removed or kept when changing the number of dimensions. LARJ-MCMC introduces a sequence of annealed distributions based on this structural change in the factor graph.*

addition, without adjusting the rest of the random variables, it is unlikely to obtain a good proposal state because the chain has already spent time mixing in 5 dimensions. The result is low acceptance rate for jump proposals.

4.3 Locally Annealed Jump

In this section, we describe the proposed locally annealed RJMCMC. The central idea of our method is to use a sequence of intermediate annealed distributions when jumping between dimensionalities, instead of immediately accepting or rejecting the candidate state generated by a single big jump.

Suppose the Markov chain is currently in state $y \in \mathcal{R}^m$, we are considering a jump move to $\mathcal{R}^n$, and the functions for instantiating factors and creating random variables are defined.

Let $\mathcal{F}_m$ denote the set of factors instantiated in $\mathcal{R}^m$ and $\mathcal{F}_n$ be the set of factors instantiated in $\mathcal{R}^n$. The following changes in the factor graph occur:

1. add new factors $\mathcal{F}_+ = \mathcal{F}_n - \mathcal{F}_m$
2. remove factors $\mathcal{F}_- = \mathcal{F}_m - \mathcal{F}_n$
3. keep the common factors $\mathcal{F}_\cap = \mathcal{F}_n \cap \mathcal{F}_m$

Figure 6 shows an example of these three sets of factors as we jump from $\mathcal{F}_3$ to $\mathcal{F}_4$.

Our locally annealed jump move generates a sequence of intermediate states while annealing in $\mathcal{F}_+$ and annealing out $\mathcal{F}_-$. The sequence of annealing distributions is

$$p_{\beta_t}(\mathbf{x}) \propto \prod_{f \in \mathcal{F}_\cap} f(x_f) \prod_{f \in \mathcal{F}_+} f^{\beta_t}(x_f) \prod_{f \in \mathcal{F}_-} f^{(1-\beta_t)}(x_f). \quad (4)$$

$\beta_0 = 0.0, \ldots, \beta_t, \ldots \beta_{T-1} = 1.0$ is the annealing schedule. At β_0, $\mathcal{F}_+$ has no effect. As β_t increases, the effect of $\mathcal{F}_+$ increases while the effect of $\mathcal{F}_-$ decreases. At β_{T-1}, factors in $\mathcal{F}_-$ have no effect. We use the name "locally annealed" RJMCMC to indicate that we are not using globally annealed distributions as in simulated annealing or parallel tempering.

The process of our locally annealed jump move from $\mathcal{R}^m$ to $\mathcal{R}^n$ is described below and pictured in Figure 7. We use y to denote the actual samples and x to denote the intermediate auxiliary states generated in the annealing stage. Let $q_{mn}(\cdot|\cdot)$ denote a jump proposal from a state of dimension m to n.

1. **Jump.** Make a jump proposal $x_0^* \in \mathcal{R}^n$ from density $q_{mn}(x_0^*|y)$. If $n < m$, set x_0^* to have $m-n$ random variables marked for removal at the end of the annealing stage.
2. **Annealing.** In the annealing stage, we generate states $(x_1^*, \cdots, x_T^*)$ in sequence. Let t be the annealing index. x_{t+1}^* is generated from x_t^* using the transition kernel Q_{β_t} that has p_{β_t} as an invariant distribution (4). The transition kernel Q_{β_t} can be constructed by MH steps. In our implementation, we generate x_{t+1}^* from x_t^* by using one MH update.
3. **Final acceptance.** The last annealed state $x_T^* \in \mathcal{R}^n$ is either accepted as the next state y^* or rejected. If rejected, the chain remains in y. We use the following acceptance probability $\alpha_{mn}(y \to y^*)$:

$$\min\left\{1, \frac{\pi(y^*)}{\pi(y)} \frac{q_{nm}(x_T^*|y^*)}{q_{mn}(x_0^*|y)} \prod_{t=0}^{T-1} \frac{p_{\beta_t}(x_t^*)}{p_{\beta_t}(x_{t+1}^*)} \mathcal{J}_{f_{m\to n}}\right\} \quad (5)$$

The proof of detailed balance can be found in Appendix A.

To provide further intuition, Figure 8 depicts the intermediate states generated during an annealing run for a simple point layout problem. The constraints are the same as those in Figure 4.

5 Results

We implemented the LARJ-MCMC algorithm in Python. The library consists of a sampler that takes a specification consisting of (1) how to create random variables, (2) how to attach factors to given a set of random variables, and (3) proposal functions. All probabilities were computed in log space to avoid underflow.

For all experiments in this paper, we use a linearly and evenly spaced annealing schedule. We employ two types of proposal functions: 1) shift moves that change an attribute of an object, and 2) jump moves that add or remove one object. When adding an object, the attributes of the new object are randomly selected from their domains. For a LARJ-MCMC run with n annealing steps, each locally annealed jump consists of one jump move and n shift moves. In our performance comparisons where the number of proposals is used as a metric, each locally annealed jump thus counts as $1 + n$ proposal moves.

In this section, we show that LARJ-MCMC is more efficient than simple RJMCMC, delayed rejection (DR), and parallel tempering. We also show two complex examples: synthesizing the layouts of coffee shops and golf courses. For all the examples shown in the paper, we include the precise mathematical formulation of all distributions in the supplemental materials.

5.1 Comparison with delayed-rejection and parallel tempering

We compare our localized annealing algorithm (using 50 annealing steps) with traditional RJMCMC, delayed rejection, and parallel tempering. We test the relative performance using two probability distributions. Each distribution is over strings of different lengths consisting of random characters. Jump moves change the number of characters by one, selecting the position of the character to add or remove along with its value uniformly at random. Shift moves select a character uniformly at random in the string and replace it with a different character, also selected uniformly at random.

In the first experiment, the strings are constrained in different ways depending on the length (dimensionality) of the string: 1) when the number of dimensions is odd, the characters must all be `a`'s, and 2) when the number of dimensions is even, the characters must all be `b`'s. The dimensionality is allowed to vary between 5 and 10. Each character can be `a` or `b`. Strings that satisfy the above constraints are assigned an unnormalized log probability 0.0, while strings that

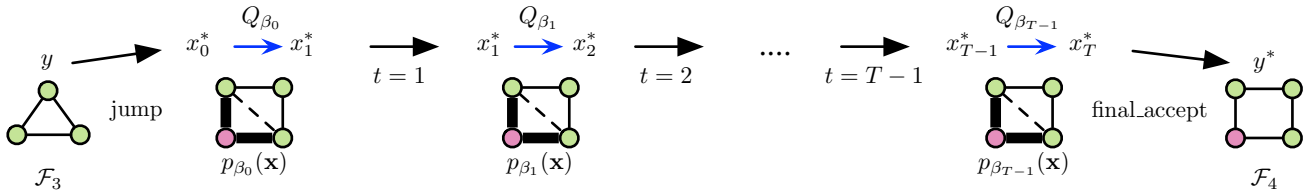

Figure 7: *The process of executing a single jump move in LARJ-MCMC. The sequence of annealing distributions allows us to gradually adapt to different energy landscapes caused by the addition or removal of a single random variable.*

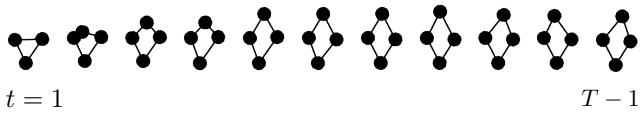

Figure 8: *One full sequence of intermediate states generated during a locally annealed jump move.*

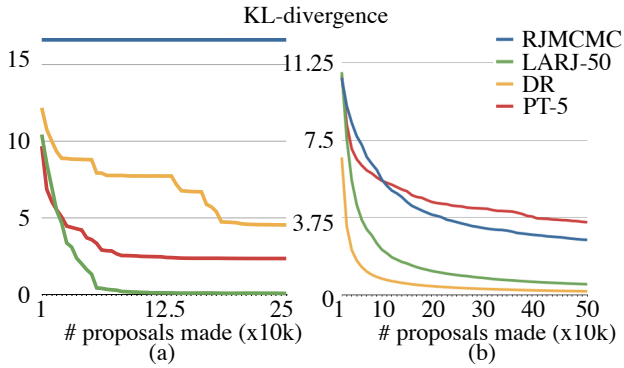

Figure 9: *The change in KL-divergence over the number of proposals made for distributions with (a) a large change in the energy landscape between different dimensions (b) a partially changed energy landscape. We compare 4 different algorithms: locally annealed RJMCMC algorithm with 50 annealing steps (LARJ-50), RJMCMC, delayed rejection (DR), parallel tempering with delayed rejection with five chains (PT-5). 9 trials were used for each combination of algorithm and distribution.*

do not are penalized proportional to the number of wrong letters. All satisfying strings are assigned the same probability.

Figure 9(a) shows how KL-divergence changes as a function of the number of proposal moves. The KL-divergence is computed between the samples from the distribution and the exact distribution. We see that RJMCMC and delayed rejection converge slowly, being slowed down at particular values of KL-divergence. These results are expected since it is difficult for these algorithms to reach certain parts of the space that involve changes in the set of constraints and variables. In contrast, parallel tempering and LARJ-MCMC converge within the first 10k proposal moves. Parallel tempering and localized annealing both allow the distribution to gradually adapt to the new constraints.

Although parallel tempering works well in this experiment, global annealing schemes can be inefficient when changing the number of dimensions only changes a small percentage of the constraints. To illustrate this problem, we created a new test distribution with different constraints: 1) pairs of opposing letters of the string should be the same, and 2) every consecutive pair of letters should be different. Each character can be `a`, `b`, or `c`. The length of the string ranges from 6 to 9. These constraints are local, so adding or deleting a single character keeps most of the factors the same.

Figure 9(b) shows the performance of all the algorithms when run on this distribution. We see that DR performs better because the constraints are simpler; it is possible to jump between high probability states in just a few proposal moves. There is a tradeoff between the number of annealing steps used and efficiency. We see that LARJ-MCMC still converges significantly more quickly than parallel tempering and RJMCMC.

5.2 Synthesizing diverse layouts

Several recent papers have addressed furniture layout in closed worlds. In this section, we show how to lay out furniture in an open world. We use coffee shops as instances of open world layouts. We demonstrate not only the standard functional constraints on individual pieces of furniture, but ones that influence the number of furniture objects, such as constraints on the density of tables and chairs that create open or crowded environments.

The input to our algorithm is a room shape and potential furniture objects. We also define zones for placing different kinds of furniture. There are zones corresponding to two-seated tables, four-seated tables, and armchairs. Zones may overlap, allowing for mixtures of different types of furniture.

We synthesize coffee shop layouts in two stages. Each of these two stages specifies an open world layout problem. In the first stage we place table and chair arrangements, armchairs, and shelves. Once this furniture has been placed, the second stage places indoor plants and floor lamps. We run LARJ-MCMC for each of these stages. Finally, for additional detail, we sampled tableware and food on each table from a set of predetermined configurations.

Furniture in the first stage were constrained to not overlap, to be inside their designated zones, and to align to a wall. Shelves were additionally constrained to be flush against a wall or a user specified segment (for example, the edge of a pathway). Armchairs were also constrained to face outwards if they were near a window, or to face nearby armchairs. In the second stage we place plants and lamps. We imposed the constraints that they do not overlap other objects, plants are close to furniture, and lamps are close to just armchairs. Finally, there is a factor that controls the density of tables and chairs.

Proposal functions Each piece of furniture or a group of tables is parameterized by 1) position $p = (p_x, p_y) \in \mathbb{R}^2$, 2) orientation $\phi \in \{0, \frac{1}{12}\pi, \ldots, 2\pi\}$ for sofas and shelves; $\phi \in \{0, \frac{1}{8}\pi, \ldots, 2\pi\}$ for table groups and 3) type $t \in \{0, \ldots, 4\}$ for table groups, $\{0, 1\}$ for sofas. Groups of tables also include extra parameters 1) the number of table/chair sets $n \in \{2, \ldots, 10\}$, and 2) the offset vector between each one $v = (r, \theta)$ where $r \in \mathbb{R}^+$ and $\theta \in \{0, \frac{1}{4}\pi, \ldots, 2\pi\}$.

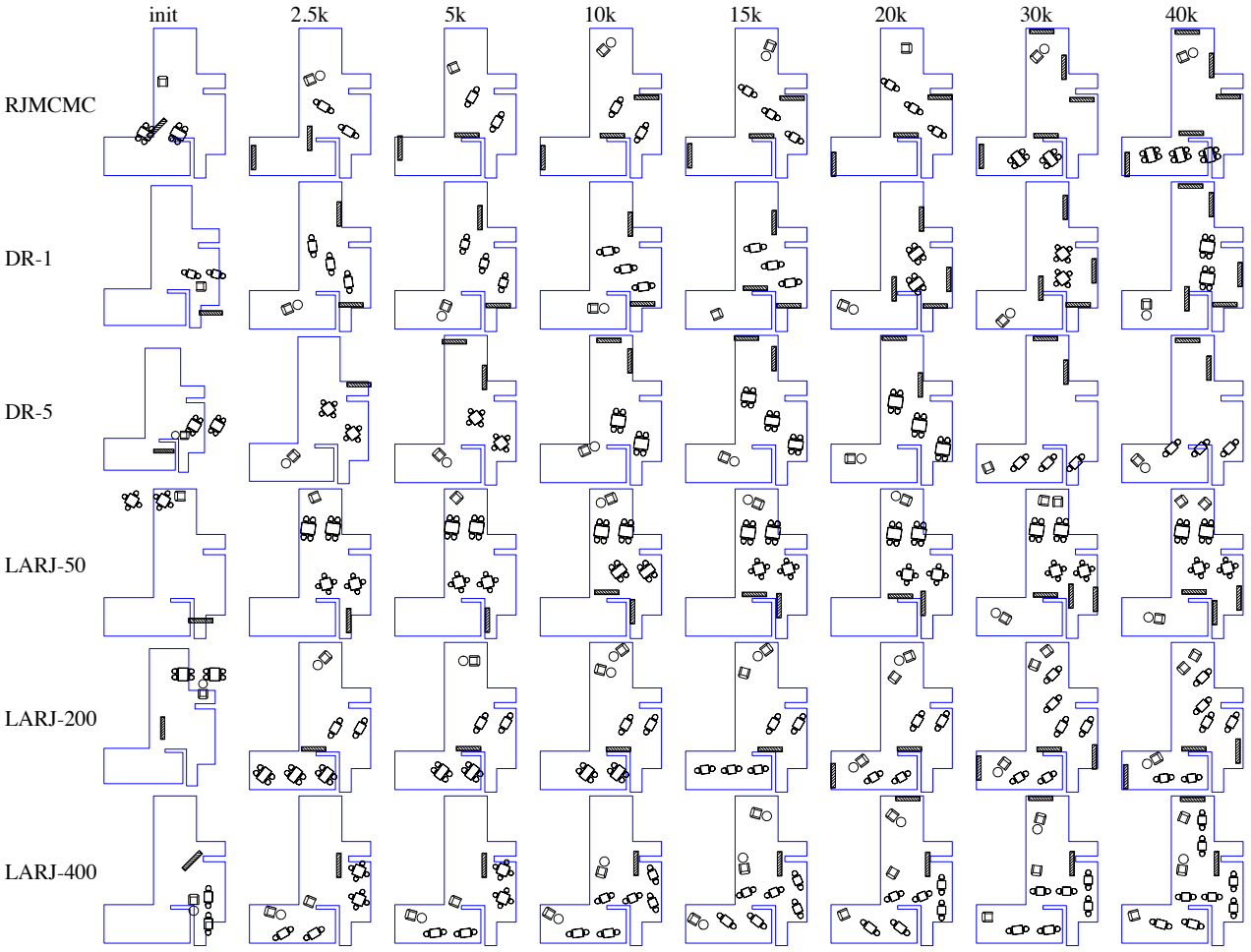

Figure 10: *Sequences of cafe layouts generated using different algorithms, indexed by the number of proposals used. Regular RJMCMC and delayed rejection schemes, DR-1 and DR-5 (with 1 and 5 steps of delayed rejection), synthesize layouts with fewer objects than LARJ-MCMC does (showing results with 50, 200, 400 annealing steps).*

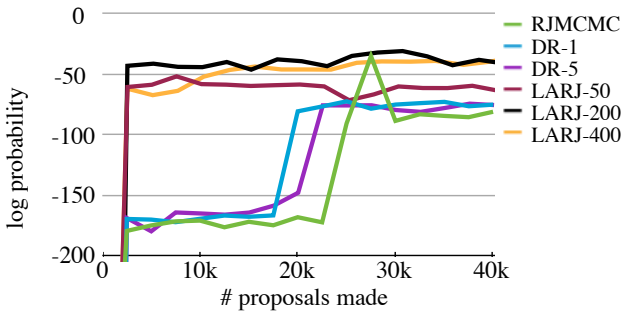

Figure 11: *Plot of unnormalized log probability versus number of proposals used for the runs in Figure 10. LARJ-MCMC converges faster than regular RJMCMC and delayed rejection.*

Each jump move adds or removes one furniture or table group from the scene. The attributes of the new object are uniformly randomly selected from their domains. Because they are uniformly selected, when jumping up, the reversible jump correction $q(x|x^*)/q(x^*|x)$ is set to d, the product of domain sizes for each attribute of the new object. When jumping down, it is set to $1/d$.

Each shift move consists of selecting one of the above random attributes to perturb according a probability distribution that is determined empirically, then applying the corresponding operations:

- Perturb p_x, p_y, and r by offsetting it with a value sampled from a Gaussian $\mathcal{N}(0, \sigma)$.
- Perturb ϕ and θ by the corresponding fractional unit of π.
- Perturb t by uniform random selection from its domain.

Figure 10 compares LARJ-MCMC with delayed rejection using different parameters for each algorithm by arranging table groups, sofas and shelves in a given room shape. Each sequence of samples is indexed by the number of proposals made. One annealing step or delayed rejection counts as one proposal. In addition to single-step delayed rejection, we also include the "DR-5" algorithm which allows 5 delayed rejection steps before final acceptance. We found that more delayed rejection steps do not help. We ran LARJ-MCMC using different annealing steps: 50, 200, and 400. We see that LARJ-MCMC manages to introduce more furniture objects and lay them out properly. It is able to make accepted jump proposals even in this highly constrained space, placing new table groups in satisfying locations. In contrast, DR runs are stuck at suboptimal arrangements in lower dimensional states. Although more annealing steps result in earlier jumps, the effective number of

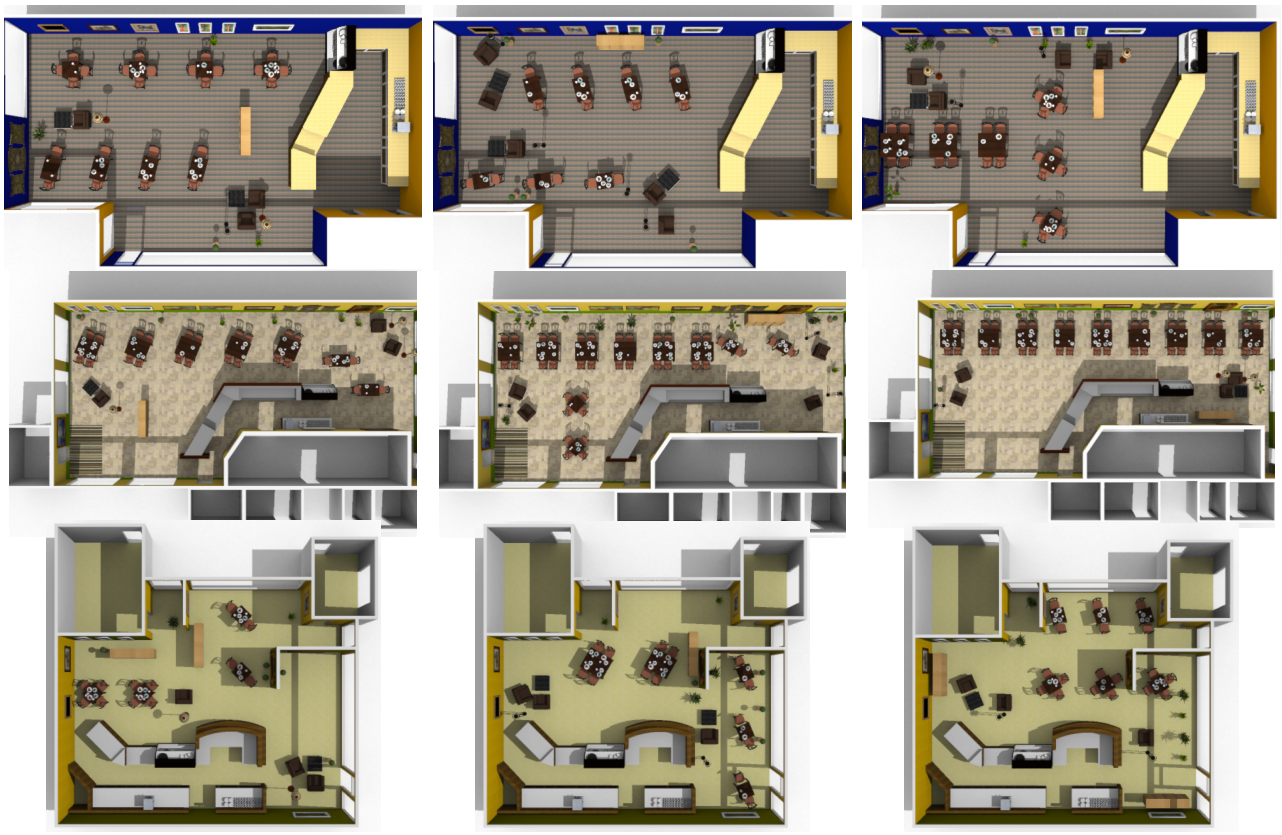

Figure 12: *Coffee shops with various room shapes synthesized by our algorithm. The 3-D models were obtained from Google 3D Warehouse.*

samples is reduced. We use 200 annealing steps for the other cafe layouts shown in the paper. Figure 11 shows the evolution of log-probability over the number of the proposals for each of the runs in Figure 10, demonstrating LARJ-MCMC's ability to reach higher probability regions of the space more quickly.

Figure 12 shows several coffee shop layouts synthesized using LARJ-MCMC under these constraints. In particular, the coffee shops have different numbers of tables, yet still satisfy the constraints in different ways. The figure shows examples where the density constraint acting in concert with alignment constraints can lead to a few long rows of tables or T-shaped arrangements of smaller groups of tables. We also see that the total number of furniture adapts to the available area. We used 200 annealing steps. The total time taken to generate 100k samples was 36 minutes. All timings were obtained using an Intel Xeon desktop clocked at 2.66 GHz with 8GB RAM.

5.3 Automatic game level design

We also apply our technique to the problem of generating golf course layouts for the golf game Tiger Woods PGA Tour 2008. We chose golf courses because the design of a golf course is complex [Graves and Cornish 1998]. At one level, the course has to adapt to the landscape, at another level, it has to follow the rules of the game, and finally, it must be challenging, yet fun to play.

The input to the system is the boundary shape, the lakes, and desired level of difficulty. We generate 9-hole golf courses given three different boundary shapes and three target difficulty levels in Figure 13(a). We synthesize a course in two stages. First, we lay out the hole start, middle and end positions within the boundary along with the positions of the greens and flags for a fixed number of holes (closed universe). Second, we lay out the fairways and bunkers for each hole, sampling over varying sets of control points that determine fairway and bunker shapes (open universe).

The holes are represented as a path consisting of 9 segments, where the desired length of each segment depends on the par chosen for that hole. The total par for the 9-hole course is softly constrained to be 36, and there is a target distribution for the number of par 3s, par 4s, and par 5s. The course is also constrained to stay within the course boundaries and form a walkable chain from hole to hole.

Each hole is then broken into start, middle and end points. The end point determines the green placement; the flag is a random point on the green. The middle point allows us to generate doglegs. We then layout the fairway, which we represent as a group of blobs. The blobs are constrained to lie near the segments that define the hole, and to be a certain distance apart. Finally, we determine the expected position of each shot, and add the bunkers. The bunkers are constrained to avoid each other and cluster around the probable shot locations.

Proposal functions. Each control point is parameterized by its position $(x, y) \in \mathbb{R}^2$ and radius $r \in \mathbb{R}$. In the first stage, only shift moves are used. Similar to the ones used in the cafe layouts, each shift move perturbs the position or radius with a Gaussian sample.

In the second stage, each jump move perturbs the number of control points by one. The position of the new control point is uniformly sampled from its domain. For fairway points, the domain is the bounding box of the hole start, middle, and end points padded by 50 units. For bunker points, the domain consists of 80×80-unit

rectangles around the green and the expected position of each shot.

We generated a total of 6.3k samples for the pathing stage, and 72k samples for the hole layout stage, 8k samples per hole. Each hole in the second stage is independent of the other holes, and can be laid out separately. The total time to generate these samples was 43 minutes. We used 150 annealing steps for the fairway points and 50 annealing steps for the bunker points for all the courses.

Each target difficulty level in Figure 13 maps to a different set of parameters for the factor constraints. At the easy level, holes are constrained to be shorter in length, with wide straight fairways that are hard to miss. At the hard level, holes are constrained to be longer and will often feature sharp doglegs and shots over water hazards. Fairways are allowed to vary more in shape, and sand traps cluster more aggressively around predicted shot locations.

We evaluate the playability and difficulty of the synthesized layouts by importing our courses into the actual Tiger Woods PGA Tour 2008 game. The screen shots in Figure 13(b) are taken from the game. We then run the game in automatic mode and record the stroke totals for AI players of varying skill levels. We play 15 rounds of golf per course on the three different course locations, testing just the easy and hard layouts. For the long golf courses in the topmost row of Figure 13(a), AI players score an average of 1.2 strokes over par on the hard course and 1.53 strokes under par on the easy course. For the pentagon- and square-shaped courses, the average score on the hard course is 2.67 and 0.466 over par, respectively, and the average score on the easy course is 2.53 and 0.27 under par. This shows that our system can synthesize playable golf courses at different levels of difficulty.

6 Discussion

In this paper, we have developed a novel MCMC algorithm for sampling generative models involving complex constraints. Our approach uses a sequence of annealed distributions during jump moves. This allows the sampler to propose states that better adapt to the change in energy landscape when the dimensionality changes. We applied this method to the application of open world layout synthesis, demonstrating that this increases the efficiency with which layouts with different number of objects are sampled under complex constraints.

Optimization and heuristic search methods are also commonly used in generating layouts. However, we believe the advantage of casting the layout synthesis problem as sampling is the ability to characterize design spaces. For example, an artist may want to select from a set of possibilities, not just the best one. In the case of procedural modeling, it is often necessary to generate a collection of models, not a single model. Sampling also enables natural variation in the layouts. The golf course example encodes constraints that give the highest score to perfectly straight holes, but in our application we would like to generate curved holes centered around the mode. Finally, the presented techniques do not have to be used for only sampling. In order to be a valid MCMC sampling method, LARJ-MCMC requires extra terms in the acceptance ratio to ensure detailed balance. For applications where it is not important to sample from the distribution, we believe the localized annealing scheme without the correction terms can also be useful as a stochastic optimization/search technique.

In general, sampling from a factor graph requires greater computational resources than producing a derivation from a generative model such as a stochastic grammar. However, by declaratively specifying constraints on the layout of patterns, we gain greater control of the results. As computers become faster and probabilistic inference algorithms become more efficient, the cost of sampling becomes less significant compared to the increase in controllability.

In this paper, we have concentrated on efficiently sampling from constrained probabilistic models. We have not addressed the issue of creating more intuitive higher-level representations, possibly in the form of editing tools or modeling languages, for which our formalism can serve as a backend. However, we believe our approach will lead to more powerful modeling interfaces.

Finally, this paper has only considered the forward direction: generating patterns from a predefined probability distribution. A natural next step is to learn the constraints from a set of examples. We believe our representation can potentially serve as a new approach for inverse procedural modeling.

A Proof of Detailed Balance

To show that we have a valid MCMC sampler, it is sufficient to show that the detailed balance condition is satisfied. We follow the proof of Tempered Transitions [Neal 1994].

To satisfy detailed balance, we would like the probability of being in state $y \in \mathcal{R}^m$, jump to the launch state $x_0^* \in \mathcal{R}^n$, and generate $(x_1^*, \cdots, x_T^*) = x_{1:T}^*$ in sequence and accepting this proposal $x_T^* \in \mathcal{R}^n$ as the new state y^* to be the same as the probability of being in state $y^* \in \mathcal{R}^n$, jump to the launch state $x_0' \in \mathcal{R}^m$, and generate $(x_1', \cdots, x_T') = x_{1:T}' = (x_{T-1}^*, \cdots, x_0^*)$ and accepting this proposal $x_T' \in \mathcal{R}^m$. Let $\alpha_{mn}(y \to y^*)$ and $\alpha_{nm}(y^* \to y)$ denote these two final acceptance probabilities. We want that

$$\pi(y) q_{mn}(x_0^*|y) \mathcal{Q}(x_0^* \cdots x_T^*) \alpha_{mn}(y \to y^*) = \pi(y^*) q_{nm}(x_T^*|y^*) \mathcal{Q}(x_T^* \cdots x_0^*) \alpha_{nm}(y^* \to y) \mathcal{J}_{f_{m \to n}} \quad (6)$$

where $y^* = x_T^*$. Our goal is to find an appropriate choice of $\alpha_{mn}(y \to y^*)$ such that (6) is satisfied.

Acceptance Probability. Our acceptance probability is

$$\alpha_{mn}(y \to y^*) = \min \left\{ 1, \frac{\pi(y^*)}{\pi(y)} \frac{q_{nm}(x_T^*|y^*)}{q_{mn}(x_0^*|y)} \prod_{t=0}^{T-1} \frac{p_{\beta_t}(x_t^*)}{p_{\beta_t}(x_{t+1}^*)} \mathcal{J}_{f_{m \to n}} \right\} \quad (7)$$

Proof. Consider the sequence of intermediate states generated in the annealing stage $(x_1^*, \cdots, x_T^*)$. Each state x_{t+1}^* is generated from state x_t^* using the transition kernel Q_{β_t} that has p_{β_t} as invariant distribution. Let $\mathcal{Q}(x_0^* \cdots x_T^*)$ denote the transition probability for the sequence of moves from x_0^* to x_T^* through kernels $Q_{\beta_0} \cdots Q_{\beta_{T-1}}$, each of which satisfy detailed balance with respect to their corresponding p_{β_t}. We can write down $\mathcal{Q}(x_0^* \cdots x_T^*)$ as follows:

$$\mathcal{Q}(x_0^* \cdots x_T^*) = Q_{\beta_0}(x_1^*|x_0^*) Q_{\beta_1}(x_2^*|x_1^*) \cdots Q_{\beta_{T-1}}(x_T^*|x_{T-1}^*). \quad (8)$$

Because our design of the intermediate distributions is symmetric for the forward and backward directions, the probability of a transition through the reverse sequence of states $(x_T^*, \cdots, x_0^*)$ can be written as

$$\mathcal{Q}(x_T^* \cdots x_0^*) = Q_{\beta_{T-1}}(x_{T-1}^*|x_T^*) Q_{\beta_{T-2}}(x_{T-2}^*|x_{T-1}^*) \cdots Q_{\beta_0}(x_0^*|x_1^*). \quad (9)$$

Since each intermediate update satisfies detailed balance with respect to its annealed distribution p_{β_t}, i.e.,

$$p_{\beta_t}(x_i^*) Q_{\beta_t}(x_j^*|x_i^*) = p_{\beta_t}(x_j^*) Q_{\beta_t}(x_i^*|x_j^*), \quad (10)$$

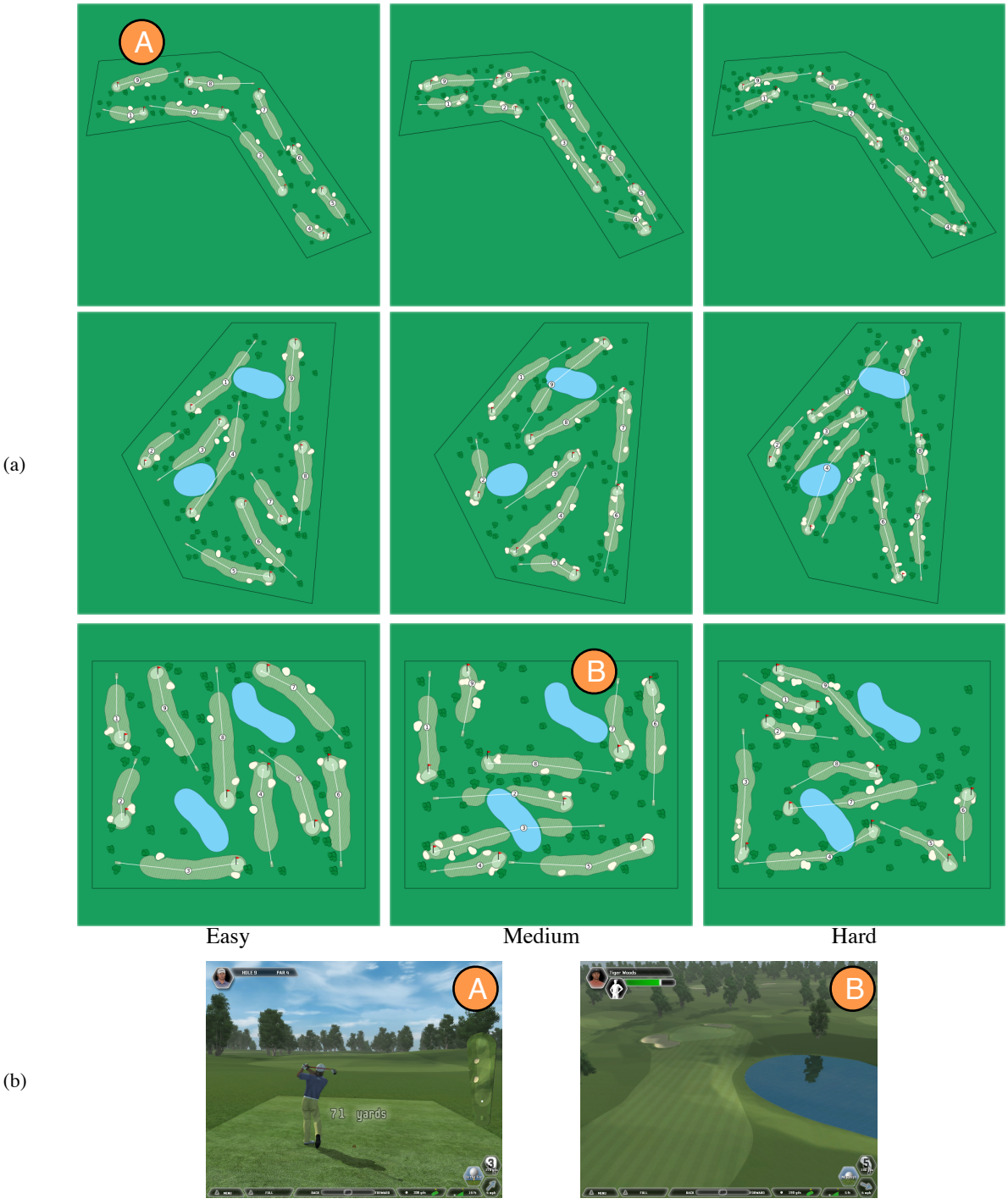

Figure 13: *(a) Golf courses of varying difficulty levels and course boundaries synthesized by our algorithm. (b) The in-game screenshots of the synthesized golf courses, corresponding to the marked locations A and B in the overhead view.*

we can rewrite the forward transition probability (8) as

$$
\begin{aligned}
&\mathcal{Q}(x_0^* \cdots x_T^*) \\
&= Q_{\beta_0}(x_1^*|x_0^*) Q_{\beta_1}(x_2^*|x_1^*) \cdots Q_{\beta_{T-1}}(x_T^*|x_{T-1}^*) \\
&= \frac{p_{\beta_0}(x_1^*)}{p_{\beta_0}(x_0^*)} Q_{\beta_0}(x_0^*|x_1^*)) \cdots \times \\
&\quad \frac{p_{\beta_{T-1}}(x_T^*)}{p_{\beta_{T-1}}(x_{T-1}^*)} Q_{\beta_{T-1}}(x_{T-1}^*|x_T^*) \\
&= Q_{\beta_0}(x_0^*|x_1^*) \cdots Q_{\beta_{T-1}}(x_{T-1}^*|x_T^*) \times \\
&\quad \frac{p_{\beta_0}(x_1^*)}{p_{\beta_0}(x_0^*)} \cdots \frac{p_{\beta_{T-1}}(x_T^*)}{p_{\beta_{T-1}}(x_{T-1}^*)} \\
&= \mathcal{Q}(x_T^* \cdots x_0^*) \frac{p_{\beta_0}(x_1^*)}{p_{\beta_0}(x_0^*)} \cdots \frac{p_{\beta_{T-1}}(x_T^*)}{p_{\beta_{T-1}}(x_{T-1}^*)} \\
&= \mathcal{Q}(x_T^* \cdots x_0^*) \prod_{t=0}^{T-1} \frac{p_{\beta_t}(x_{t+1}^*)}{p_{\beta_t}(x_t^*)}
\end{aligned} \tag{11}
$$

where

$$p_{\beta_t}(\mathbf{x}) = \prod_{f \in \mathcal{F}_\cap} f(x_f) \prod_{f \in \mathcal{F}_+} f^{\beta_t}(x_f) \prod_{f \in \mathcal{F}_-} f^{(1-\beta_t)}(x_f). \quad (12)$$

With our acceptance probability $\alpha_{mn}(y \to y^*)$ in (7), the LHS of (6) becomes

$$\begin{aligned}
&LHS \\
&= \pi(y) q_{mn}(x_0^*|y) \mathcal{Q}(x_T^* \cdots x_0^*) \left[\prod_{t=0}^{T-1} \frac{p_{\beta_t}(x_{t+1}^*)}{p_{\beta_t}(x_t^*)} \right] \\
&\quad \times \alpha_{mn}(y \to y^*) \\
&= \pi(y) q_{mn}(x_0^*|y) \mathcal{Q}(x_T^* \cdots x_0^*) \left[\prod_{t=0}^{T-1} \frac{p_{\beta_t}(x_{t+1}^*)}{p_{\beta_t}(x_t^*)} \right] \\
&\quad \times \min \left\{ 1, \frac{\pi(y^*)}{\pi(y)} \frac{q_{nm}(x_T^*|y^*)}{q_{mn}(x_0^*|y)} \prod_{t=0}^{T-1} \frac{p_{\beta_t}(x_t^*)}{p_{\beta_t}(x_{t+1}^*)} \mathcal{J}_{f_{m \to n}} \right\} \\
&= \mathcal{Q}(x_T^* \cdots x_0^*) \\
&\quad \times \min \left\{ \pi(y) q_{mn}(x_0^*|y) \prod_{t=0}^{T-1} \frac{p_{\beta_t}(x_{t+1}^*)}{p_{\beta_t}(x_t^*)}, \right. \\
&\qquad \left. \pi(y^*) q_{nm}(x_T^*|y^*) \mathcal{J}_{f_{m \to n}} \right\} \\
&= \mathcal{Q}(x_T^* \cdots x_0^*) \pi(y^*) q_{nm}(x_T^*|y^*) \mathcal{J}_{f_{m \to n}} \\
&\quad \times \min \left\{ \frac{\pi(y) q_{mn}(x_0^*|y)}{\pi(y^*) q_{nm}(x_T^*|y^*)} \prod_{t=0}^{T-1} \frac{p_{\beta_t}(x_{t+1}^*)}{p_{\beta_t}(x_t^*)} \mathcal{J}_{f_{n \to m}}, 1 \right\} \\
&= RHS \quad (13)
\end{aligned}$$

where $\mathcal{J}_{f_{n \to m}} = 1/\mathcal{J}_{f_{m \to n}}$, since $f_{n \to m}$ is the inverse of $f_{m \to n}$. ■

Acknowledgements

We are grateful to the reviewers for their constructive comments. We would also like to thank the Google 3D warehouse for providing us the models used in the cafe layout example. Support for this research was provided by the ISTC-VC Intel Science and Technology Center for Visual Computing fund, DARPA SEEC grant HR0011-11-C-0007- P00003, and ONR grant N00014-09-1-0124.

References

ALEXANDER, C., ISHIKAWA, S., AND SILVERSTEIN, M. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York.

FREY, B. 2003. Extending factor graphs so as to unify directed and undirected graphical models. In *Proceedings of the Proceedings of the Nineteenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, Morgan Kaufmann, San Francisco, CA, 257–26.

FRYDENBERG, M. 1990. The chain graph Markov property. *Scandinavian Journal of Statistics 17*, 4, pp. 333–353.

GEMAN, S., AND GEMAN, D. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-6*, 721–741.

GOODMAN, N. D., MANSINGHKA, V. K., ROY, D. M., BONAWITZ, K., AND TENENBAUM, J. B. 2008. Church: a language for generative models. *Uncertainty in Artificial Intelligence 2008*, 220–229.

GRAVES, R., AND CORNISH, G. 1998. *Golf course design*. J. Wiley.

GREEN, P. J., AND MIRA, A. 1999. Delayed rejection in reversible jump metropolis-hastings. *Biometrika 88*, 1035–1053.

GREEN, P. J. 1995. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika 82*, 711–732.

HASTINGS, W. K. 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika 57*, 1, pp. 97–109.

KOLLER, D., AND FRIEDMAN, N. 2009. *Probabilistic graphical models*. MIT Press.

KSCHISCHANG, F. R., FREY, B. J., AND LOELIGER, H.-A. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory 47*, 498–519.

LIU, J. S., LIANG, F., AND WONG, W. H. 2000. The multiple-try method and local optimization in Metropolis sampling. *Journal of the American Statistical Association 95*, 449 (Mar.), 121+.

LUNN, D. J., THOMAS, A., BEST, N., AND SPIEGELHALTER, D. 2000. Winbugs - a Bayesian modelling framework: Concepts, structure, and extensibility. *Statistics and Computing 10*, 4, 325–337.

MCCALLUM, A., SCHULTZ, K., AND SINGH, S. 2009. FACTORIE: Probabilistic programming via imperatively defined factor graphs. In *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds., 1249–1257.

MERRELL, P., SCHKUFZA, E., LI, Z., AGRAWALA, M., AND KOLTUN, V. 2011. Interactive furniture layout using interior design guidelines. *ACM Trans. Graph. 30* (August), 87:1–87:10.

MILCH, B., MARTHI, B., RUSSELL, S., SONTAG, D., ONG, D. L., AND KOLOBOV, A. 2005. BLOG: Probabilistic models with unknown objects. In *IJCAI'05: Proceedings of the 19th international joint conference on Artificial intelligence*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1352–1359.

MIRA, A. 2001. On Metropolis-Hastings algorithms with delayed rejection. *Metron 59*, 3–4.

NEAL, R. 1994. Sampling from multimodal distributions using tempered transitions. *Statistics and Computing 6*, 353–366.

PFEFFER, A. 2007. Sampling with memoization. In *AAAI'07*, 1263–1270.

RICHARDSON, M., AND DOMINGOS, P. 2006. Markov logic networks. *Machine Learning 62*, 1, 107–136.

TALTON, J. O., LOU, Y., LESSER, S., DUKE, J., MĚCH, R., AND KOLTUN, V. 2011. Metropolis procedural modeling. *ACM Trans. Graph. 30* (April), 11:1–11:14.

TJELMELAND, H., AND HEGSTAD, B. K. 1999. Mode jumping proposals in MCMC. *Scandinavian Journal of Statistics 28*, 205–223.

YU, L.-F., YEUNG, S. K., TANG, C.-K., TERZOPOULOS, D., CHAN, T. F., AND OSHER, S. 2011. Make it home: Automatic optimization of furniture arrangement. *ACM Transactions on Graphics 30*, 4, 86.