Learning Design Patterns with Bayesian Grammar Induction

Jerry O. Talton
Intel Corporation
jerry.o.talton@intel.com

Lingfeng Yang
Stanford University
lfyg@stanford.edu

Ranjitha Kumar
Stanford University
ranju@stanford.edu

Maxine Lim
Stanford University
maxinel@stanford.edu

Noah D. Goodman
Stanford University
ngoodman@stanford.edu

Radomír Měch
Adobe Corporation
rmech@adobe.com

ABSTRACT

Design patterns have proven useful in many creative fields, providing content creators with archetypal, reusable guidelines to leverage in projects. Creating such patterns, however, is a time-consuming, manual process, typically relegated to a few experts in any given domain. In this paper, we describe an algorithmic method for learning design patterns directly from data using techniques from natural language processing and structured concept learning. Given a set of labeled, hierarchical designs as input, we induce a probabilistic formal grammar over these exemplars. Once learned, this grammar encodes a set of generative rules for the class of designs, which can be sampled to synthesize novel artifacts. We demonstrate the method on geometric models and Web pages, and discuss how the learned patterns can drive new interaction mechanisms for content creators.

ACM Classification Keywords

H.1.2. [Models and Principles]: User/Machine Systems – Human factors.

General Terms

Algorithms, Human Factors.

Author Keywords

Design patterns; grammar induction.

INTRODUCTION

As creative fields mature, a set of best practices emerge for design. Often, attempts are made to codify these practices into a set of formal rules for designers which set out principles of composition, describe useful idioms, and summarize common aesthetic sensibilities. Such *design patterns* have proven popular and influential in fields such as architecture [3], software engineering [14], interaction [7], and Web design [13].

Despite their popularity, design patterns are also problematic. For one, they are difficult to operationalize: users bear the burden of locating reputable sources of design knowledge, assimilating that knowledge, and applying it to their own problems. For another, patterns must be painstakingly formulated and compiled by experts, resulting in guidelines that may be less descriptive of actual practice than prescriptive of a particular designer's point of view.

A more attractive proposition is to learn design patterns directly from data, and encapsulate them in a representation that can be accessed algorithmically. In this paper, we address this problem for one common class of designs: those comprising a hierarchy of labeled components. Such hierarchies abound in creative domains as diverse as architecture [39], geometric modeling [35], document layout [21], and Web design [25].

To learn patterns in a principled way, we leverage techniques from natural language processing and structured concept learning. In particular, we cast the problem as *grammar induction*: given a corpus of example designs, we induce a probabilistic formal grammar over the exemplars. Once learned, this grammar gives a design pattern in a human-readable form that can be used to synthesize novel designs and verify extant constructions.

The crux of this induction is learning how to generalize beyond the set of exemplars: we would like to distill general principles from the provided designs without extrapolating patterns that are not supported by the data. To this end, we employ an iterative structure learning technique called Bayesian Model Merging [41], which formulates grammar induction as Bayesian inference. The method employs an inductive bias based on the law of succinctness, also known as Occam's razor, searching for the simplest grammar that best fits the examples. Since compactness and generality are inexorably linked in grammar-based models, the method provides a data-driven way to learn design patterns that are neither too specific nor overly general.

We demonstrate the method on two distinct classes of designs: geometric models (based on scene graphs), and Web pages (based on Document Object Model trees). We report on statistical analyses of the grammars induced by our technique, share the results from a small user study, and discuss how these sorts of probabilistic models could lead to better tools and interaction mechanisms for design.

UIST 12, October 710, 2012, Cambridge, Massachusetts, USA.

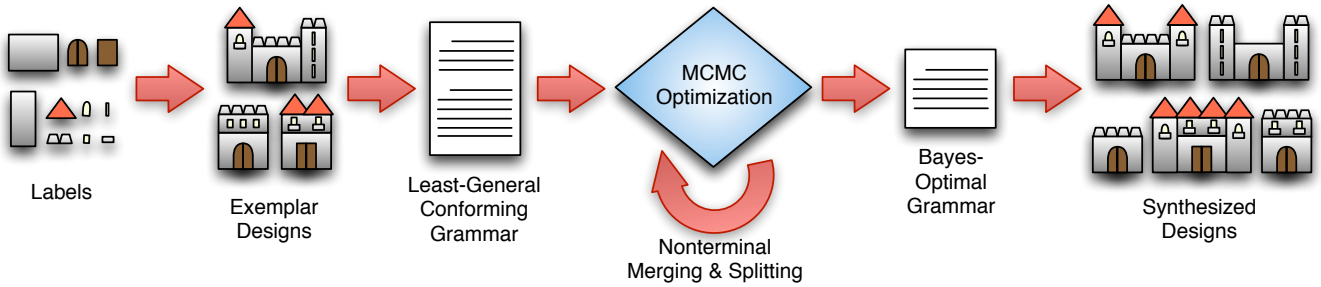


Figure 1. The pipeline for learning design patterns with grammar induction. A set of example designs, each comprising a hierarchy of labeled components, are used to produce an initial, specific grammar. Then, Markov chain Monte Carlo optimization is employed to explore a space of more general grammars, balancing the descriptive power of each one against its representation complexity. At the end of this process, the best scoring grammar is returned, and can be sampled to generate new designs.

BACKGROUND

The use of grammars as generative models has a long history in design. Grammar-based procedural models have seen extensive use in architecture [39, 8, 12], product design [24, 1, 34], document layout [9, 44], and 3D modeling [33, 31, 29]. However, despite the proven utility of grammars as a computational mechanism for design, the overwhelming majority of these models are coded painstakingly by hand. Although a few attempts have been made to learn deterministic rules from patterns [38] and adapt texture synthesis techniques to geometric modeling [28, 6], relatively little work has been done on learning grammars from designs in a principled way.

In formal language theory, the problem of grammar induction was first introduced by Solomonoff [37], who posed the fundamental question of language learning: given a sequence of words from a formal language, is it possible to learn an automaton capable of recognizing that language? The classical result, due to Gold [15], is negative, and states that no superfinite language class is learnable in the limit from positive examples. This means that none of the languages in the Chomsky hierarchy—regular, context-free, context-sensitive, or recursively enumerable—can be learned in this way, regardless of how many samples from the language the induction algorithm is allowed to inspect, or how long it is allowed to process them.

Horning [19] showed that things are not quite so grim for *probabilistic* languages: in fact, stochastic context-free grammars can—in theory—be induced from positive examples. Learning these grammars in practice, however, is a challenging problem: even deciding whether or not there exists an n-state automaton which agrees with some finite set of data is known to be $\mathcal{NP}$-complete [16]. Accordingly, no general algorithm for learning stochastic context-free grammars in the limit has been proposed [10], although several authors have demonstrated natural language induction from large corpora by leveraging domain-specific linguistic features [22, 23].

One popular strategy for making the grammar induction problem tractable is the introduction of an inductive bias. In this paper, we use a technique called Bayesian Model Merging [41] which employs an inductive bias based on the cognitive principle of Occam's razor: specific, complex models are deemed less likely than simple, general ones [26]. In particular, we formulate a posterior probability distribution over the space of possible grammars, and then attempt to maximize this posterior via Markov chain Monte Carlo (MCMC) optimization. This gives a principled, flexible method for inducing design patterns from data.

ALGORITHM OVERVIEW

The method takes as input a set of designs in the form of labeled trees, where each label is drawn from a discrete dictionary $\mathbf{C}$. The algorithm begins by traversing each tree and creating a production rule for every node to generate a *least-general conforming grammar* (LGCG). The grammar is conforming in the sense that every exemplar is a valid derivation from it; it is the least-general such grammar because it derives only the exemplars, with no additional generalization capacity.

Once this grammar is constructed, Markov chain Monte Carlo optimization is used to explore a series of more general conforming grammars by merging and splitting nonterminal symbols. Each merge operation takes two nonterminals, rewrites them to have a common name, and unions their productions; each split operation is the reverse of a merge.

To judge the quality of each candidate grammar, we adopt a Bayesian interpretation that balances the likelihood of the exemplar designs against the description length of the grammar. At each step in the optimization, we randomly select a split or merge move to apply, and evaluate the posterior of the resultant grammar. This search procedure is run until it exhausts a predetermined computational budget and the *maximum a posteriori* estimate is returned. This process is outlined in Figure 1.

GRAMMAR FORMULATION

In order to describe the grammar induction framework, we must first define some basic concepts. A stochastic, context-free grammar (SCFG) is a tuple

$$\mathbf{G} = \langle V, T, \omega, R, \theta \rangle,$$

where V is the set of *nonterminals*, T is the set of *terminals*, $\omega \in (V \cup T)^+$ is the *axiom*, $R \subset V \times (V \cup T)^+$ is a finite set of *production rules*, and $\theta : R \to (0, 1]$ is a *probability*

function. Each production $\rho \in R$ is written as $A \to \chi$, where $A \in V$ is called the *predecessor*, and $\chi \in (V \cup T)^+$ is called the *successor*. For any $A \in V$, the sum of the probabilities for all the rules in R with predecessor A is 1. In the context of structure learning, the set $S_{\mathbf{G}} = \langle V, T, \omega, R \rangle$ is called the grammar's *structure*, and $\theta_{\mathbf{G}}$ is said to comprise its *parameters*.

The set $\mathcal{L}(\mathbf{G})$ of all strings which the grammar can generate is called the *language* of $\mathbf{G}$. Given a string $m \in \mathcal{L}(\mathbf{G})$, there is at least one *derivation tree* τ for m. Each such tree is rooted on ω, and every subsequent level in τ consists of a string produced by applying a matching production to each nonterminals in the level above. The last level of the tree contains m.

BAYESIAN INFERENCE FOR GRAMMARS

Given a set of example designs $\mathbf{M}$, we formulate a posterior probability distribution over the space of possible grammars

$$p(\mathbf{G}|\mathbf{M}) \propto P(\mathbf{M}|\mathbf{G})\pi(\mathbf{G}) \qquad (\star)$$

where $P(\mathbf{M}|\mathbf{G})$ is the likelihood of the set of example designs given the grammar and $\pi(\mathbf{G})$ is the grammar prior. Finding the optimal grammar for the exemplars then reduces to maximizing this posterior.

From a strict Bayesian perspective we should formulate this inference as a search for the optimal grammar *structure*, integrating over the associated parameters to compute

$$P(\mathbf{M}|S_{\mathbf{G}}) = \int_{\theta_{\mathbf{G}}} P(\theta_{\mathbf{G}}|S_{\mathbf{G}})P(\mathbf{M}|S_{\mathbf{G}}, \theta_{\mathbf{G}})d\theta_{\mathbf{G}}. \qquad (\diamond)$$

Since this integral has no analytic solution, it would have to be approximated numerically via a costly Monte Carlo simulation for each candidate grammar structure, making inference intractable. Instead, we make a Viterbi assumption that the maximum likelihood parameters for any given structure will dominate in ($\diamond$), which simplifies the inference procedure by identifying a unique set of parameters with each candidate grammar.

Design Likelihoods

With this assumption, given a grammar $\mathbf{G}$ comprising structure $S_{\mathbf{G}}$ and parameters $\theta_{\mathbf{G}}$, we can calculate the likelihood of the exemplar set under the grammar

$$P(\mathbf{M}|\mathbf{G}) = \prod_{m \in \mathbf{M}} P_{\mathbf{G}}(m),$$

where $P_{\mathbf{G}}(\cdot)$ is the probability of deriving a design. The probability of a model m is the sum over all possible derivation trees yielding m,

$$P_{\mathbf{G}}(m) = \sum_{\tau \Rightarrow m} P(\tau) = \sum_{\tau \Rightarrow m} \prod_{\rho \in \tau} \theta_{\mathbf{G}}(\rho),$$

where the probability of a tree is just the product of the probabilities of the rules used in its derivation.

Grammar Priors

To calculate the second term in the posterior ($\star$), the grammar prior, we decompose the grammar into structure and parameters, and apply the chain rule to write

$$\pi(\mathbf{G}) = P_G(\theta_{\mathbf{G}}|S_{\mathbf{G}})\pi_G(S_{\mathbf{G}}),$$

where $P_G(\cdot|\cdot)$ is the probability of the grammar's parameters given its structure, and $\pi_G(\cdot)$ is a structure prior.

To compute the probability of the grammar's parameters, we examine each nonterminal $v \in V_{\mathbf{G}}$ and observe that the subset of parameters $\theta^v \subseteq \theta_{\mathbf{G}}$ associated with it form a multinomial. Then, we write the probability as a product of Dirichlet distributions

$$P_G(\theta_{\mathbf{G}}|S_{\mathbf{G}}) = \prod_{v \in V_{\mathbf{G}}} P_G(\theta^v) = \prod_{v \in V_{\mathbf{G}}} \frac{1}{B(\alpha)} \prod_{\rho \in R|v} \left(\theta^v_\rho\right)^{\alpha - 1},$$

where $B(\cdot)$ is the multinomial beta function, and α is a concentration parameter. By setting $\alpha = 1$, we obtain a uniform distribution over all possible production probabilities; when $\alpha < 1$ we prefer grammars in which most of the probability mass is concentrated in a few productions; when $\alpha > 1$ we give preference to distributions in which all of the probabilities are equal.

To compute the prior probability of the grammar structure, we take

$$\pi_G(S_{\mathbf{G}}) = \exp\left[-\ell(S_{\mathbf{G}})\right]^{\lambda},$$

where $\ell(\cdot)$ is an approximation of the description length of the grammar and λ is a weighting term. We encode the grammar as a string of productions separated by a special termination character, starting with the axiom and continuing with the predecessor and successor of the remaining rules. Each symbol in the string then contributes $\log_2(|V| + |T|)$ bits to the length. Since our formulation measures model complexity syntactically and fit to data semantically, we use λ to adjust the balance between these two intrinsically incommensurable quantities.

MODEL MERGING

With this formulation, we use Markov chain Monte Carlo optimization to seek a grammar that maximizes the posterior. The key idea behind this optimization is that we can explore a family of related, conforming grammars by merging and splitting nonterminal symbols. While the original Bayesian Model Merging algorithm used only merge operations in a beam search, we employ both merges and splits in a full MCMC framework to prevent the search procedure from getting "stuck" in local maxima.

Initial Grammar Creation

The algorithm begins by processing the exemplars to create a least-general conforming grammar, which is used to initialize the optimization. To produce this grammar, one nonterminal symbol is created for each instanced label used in $\mathbf{M}$, and a counter i is initialized. Then, each hierarchical design is traversed recursively. As each label g with children $c_1, \ldots, c_k$ is encountered, the traversal generates a new production $A_i \to gA_{i+1} \ldots A_{i+k}$, pushes productions

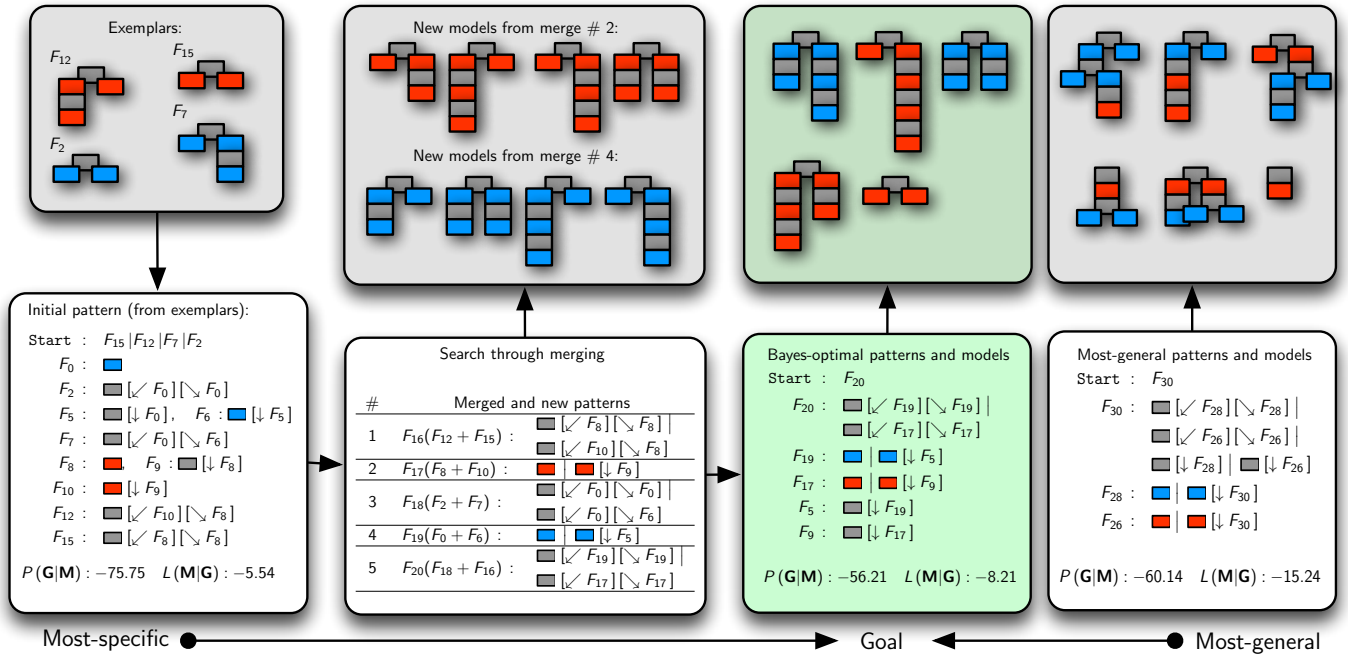


Figure 2. The grammar induction pipeline applied to a simple design space of blocks in a pattern. (left) The initial exemplars and the least-general conforming grammar generated from them. (mid left) The set of merges used to collapse the grammar. (mid right) The optimal grammar, and a set of random samples drawn from it. (right) The most-general conforming grammar, and a set of random samples drawn from it. The design pattern learned with our method generalizes beyond the exemplars in a reasonable way.

$A_{i+1} \rightarrow \ldots$ through $A_{i+k} \rightarrow \ldots$ onto the stack, and increments i by $k+1$. At the end of this process, the starting nonterminals for each of the distinct designs are set as successors of the axiom S. The resultant grammar generates input example m with probability $1/|\mathbf{M}|$.

Merging and Splitting

From the LGCG, the method explores a space of more general grammars by merging and splitting nonterminal symbols. Each merging operation adds to the grammar's generalization capacity and decreases its description length. With successive merges, the cardinality of $\mathcal{L}(\mathbf{G})$ increases: accordingly, each merging operation can only decrease the likelihood of the examples. Occasionally, a merge will create a loop in the grammar structure, making the resultant language superfinite.

Each merge operation takes two nonterminals $A_i, A_j \in V$ and their productions

$$A_i \rightarrow \chi_1 \mid \ldots \mid \chi_k \quad \text{and} \quad A_j \rightarrow \psi_1 \mid \ldots \mid \psi_l,$$

and collapses them to a single, new nonterminal A_{ij} whose successors are the union of the productions of the original symbols

$$A_{ij} \rightarrow \chi_1 \mid \ldots \mid \chi_k \mid \psi_1 \mid \ldots \mid \psi_l.$$

A split operation is the reverse of a merge: it takes a single nonterminal A_i and randomly splits its productions between two new nonterminals A_{i1} and A_{i2}. Then, every instance of A_i in the grammar is replaced by one of these new nonterminals selected at random.

Type Restrictions

In general language learning applications, any two nonterminal symbols can be merged. In particular design domains, however, the semantics of the symbols can be used to restrict the set of potential merge operations and reduce the size of the search space. Since every merge operation represents a determination by the learning algorithm that two symbols can be used interchangeably and identically, we can improve performance by identifying symbols that are fundamentally incomparable *a priori*. In essence, we impose a type system for nonterminals, and only consider merges amongst type-compatible symbols. This type system comprises a discrete set of types $\mathcal{T}$, and an assignment for each variable $\mathbf{t} : V \rightarrow \mathcal{T}$.

Parameter Estimation

While each merge or split operation defines a new grammar structure, to evaluate the posterior we must also have corresponding parameters. In particular, given a grammar structure $S_{\mathbf{G}}$, we wish to find maximum likelihood values for $\theta_{\mathbf{G}}$ such that

$$\theta_{\mathbf{G}} = \underset{\theta}{\operatorname{argmax}}\, P(\mathbf{M}|S_{\mathbf{G}}, \theta).$$

This problem is solved using expectation maximization via the Inside-Outside algorithm [5], a generalization of the classical Baum-Welch algorithm for hidden Markov models. Given a current estimate for the probabilities θ, we update

Domain	$\lvert\mathbf{M}\rvert$	$\lvert\mathbf{C}\rvert$	λ	Least-General		Most-General			Bayes-Optimal		
				DL	L	DL	L	$\mathbf{p_{200}}$	DL	L	$\mathbf{p_{200}}$
Spaceship	5	51	1.15	402	3.2×10^{-4}	369	1.0×10^{-27}	.34	389	1.9×10^{-11}	.85
Castle	6	37	1.05	739	2.1×10^{-5}	662	1.1×10^{-88}	.0045	718	6.9×10^{-18}	.92
Seussean	9	29	1.2	376	2.6×10^{-9}	341	4.2×10^{-32}	.62	346	1.2×10^{-27}	.77
Sakura	8	19	1.2	463	6.0×10^{-8}	406	6.5×10^{-73}	1.3×10^{-7}	433	3.0×10^{-29}	.14
Web page	30	30	1.0	733	4.9×10^{-45}	67	1×10^{-1546}	-	602	8.8×10^{-101}	.37

Table 1. For each of the example domains, the number of exemplars; the number of components; the parameter from the grammar prior; the description length; and the likelihood for the least-general, most-general, and Bayes-optimal grammars. For these latter two grammars, we also report the probability mass occupied by the two-hundred highest-probability designs as a measure of the grammar's generalization capacity.

the probability of a rule $A \to \chi$ by

$$\hat{\theta}(A \to \chi) = \frac{\sum_{m \in \mathbf{M}} c_\theta(A \to \chi; m)}{\sum_{m \in \mathbf{M}} \sum_\gamma c_\theta(A \to \gamma; m)},$$

where $c_\theta(\cdot;\cdot)$ is the expected count of the number of times that a particular rule is used in the derivation of a given model. Since each $c_\theta(\cdot;\cdot)$ depends on the current value of θ, the algorithm is iterative: we initialize the probabilities to be uniform for all rules associated with a given nonterminal, and then update and recompute the counts until θ converges.

Parsing

To compute the expected count of the number of times a particular rule is used, we must find derivation trees for each of the designs in $\mathbf{M}$. In linguistic theory, this process is known as *parsing*. Because the number of possible derivations for any particular design can be exponential in the size of the tree, we cannot enumerate these derivations and count the number of occurrences of each rule. Instead, we employ a bottom-up, chart-based parser similar to the CYK algorithm [2], but adapted for labeled trees instead of strings. Once constructed, the chart can be used to efficiently compute both the the total probability of deriving a model $P_{\mathbf{G}}(\cdot)$ and the expected count of a rule in a derivation $c_\theta(\cdot;\cdot)$ via dynamic programming [27].

Markov chain Monte Carlo

With this machinery in place, the Bayes-optimal grammar can be sought via Markov chain Monte Carlo search using the Metropolis-Hastings algorithm [4]. The state space is a graph where every node is a grammar, and adjacencies between nodes represent merging and splitting operations. The score of each node is the posterior probability ($\star$), and the starting node is the LGCG.

At the ith iteration of the optimization, a split or merge move is chosen at random in the current grammar $\mathbf{G}$, and a new grammar $\mathbf{G}'$ is generated. This new grammar is then accepted as the current search state with probability

$$\alpha(\mathbf{G}'|\mathbf{G}) = \min\left(1, \frac{p^{1/T_i}(\mathbf{G}'|\mathbf{M}) q(\mathbf{G}|\mathbf{G}')}{p^{1/T_i}(\mathbf{G}|\mathbf{M}) q(\mathbf{G}'|\mathbf{G})}\right),$$

where $q(\cdot|\cdot)$ is the probability of splitting or merging one grammar to generate the other and T_i is a decreasing cooling schedule with $\lim_{i\to\infty} T_i = 0$. After the algorithm has been run for a fixed number of iterations, the grammar with the highest posterior is returned.

To calculate $q(\mathbf{G}'|\mathbf{G})$, consider a split move where nonterminal A_i is broken into two new nonterminals A_j and A_k. First, A_i is selected uniformly from amongst the $|V_{\mathbf{G}}|$ nonterminals. Then, each of the $|R_{A_i}|$ productions that have A_i as their predecessor are assigned randomly to A_j or A_k. Finally, each of the f instances of A_i in the grammar are randomly replaced by one of A_j or A_k. In this case, the reverse distribution $q(\mathbf{G}|\mathbf{G}')$ measures the probability of selecting A_j and A_k from the set of $|V_{\mathbf{G}'}|$ nonterminals to merge. In the absence of type restrictions, this gives

$$q(\mathbf{G}'|\mathbf{G}) = \frac{1}{|V_{\mathbf{G}}| 2^{|R_{A_i}|+f}},\ q(\mathbf{G}|\mathbf{G}') = \frac{2}{|V_{\mathbf{G}'}|\left(|V_{\mathbf{G}'}|-1\right)}.$$

EVALUATION

To test our framework for inducing design patterns, we experimented with two distinct classes of hierarchical, labeled designs: Web pages constructed from Document Object Model (DOM) trees, and geometric models constructed from scene graphs.

Web Pages

We built a corpus of thirty Web pages, harvested from design blogs and popular business sites. We used the Bento page segmentation algorithm [25] to decompose each page into a visual information hierarchy bootstrapped by the DOM. Then, we assigned each element in the hierarchy one of thirty distinct labels describing its semantic properties. These labels were chosen in the spirit of the semantic tags included in HTML 5, comprising concepts like page, text, image, header, navigation, container, subheading, link, hero image, and logo. In the induction, we use the type system to prevent the "page" label from being merged with any other labels.

Geometric Models

We also constructed collections of geometric models for several different classes of objects: alien spaceships, Japanese castles, sakura trees, and Seussian architecture. Each class is built with a collection of modular 3D components, which comprise the labels in our framework [32]. These components are assembled into a scene graph, which imposes a tree topology over them, specifying parent/child relationships and relative transformations between nodes. In these examples, we employ the type system to disallow merges between components with differing labels, which helps preserve geometric semantics.

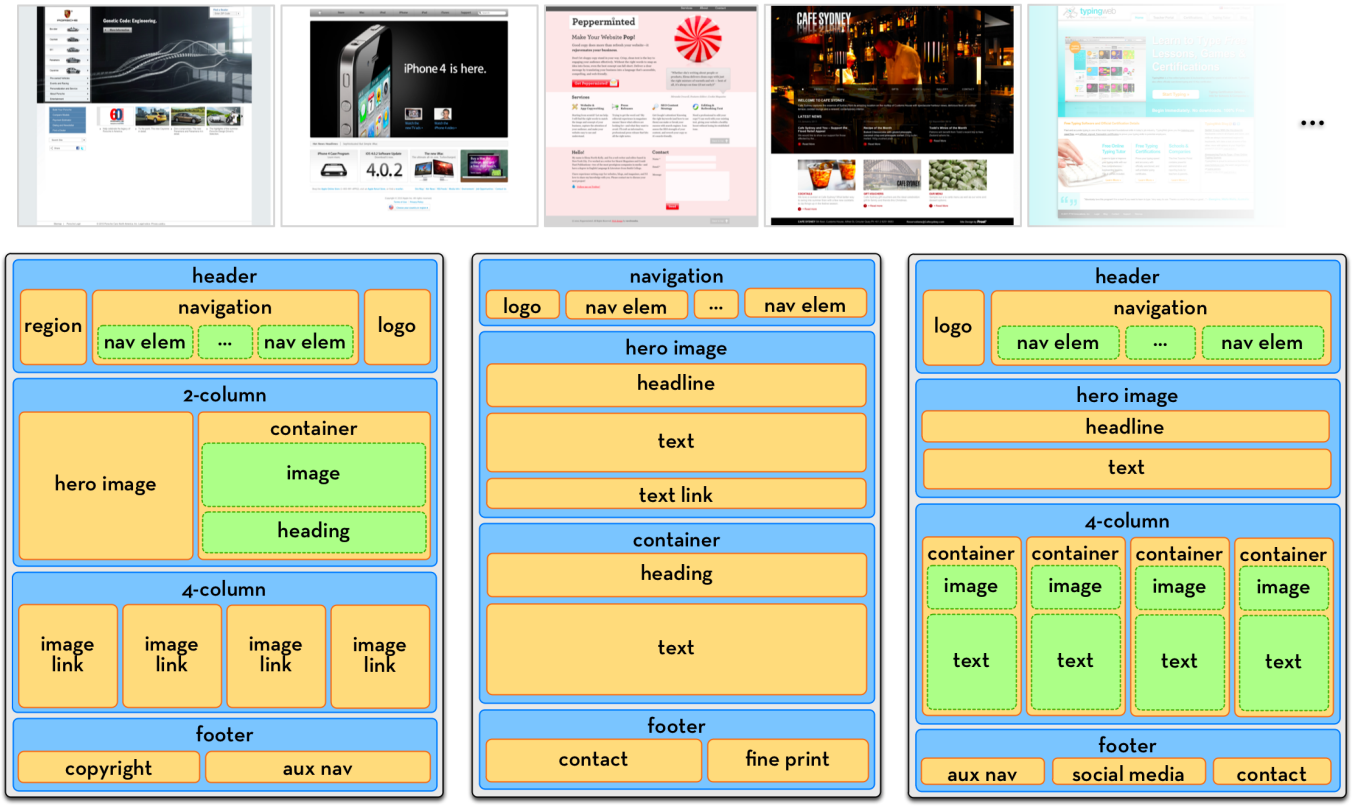


Figure 3. (top) A small sampling of the Web pages in our corpus. (bottom) Three random derivations from the grammar induced over that corpus.

RESULTS

Figure 2 illustrates the algorithm applied to a simple geometric block space, showing the initial examples, least-general conforming grammar, set of merges chosen in the search, the optimal grammar, and some random derivations sampled from the induced design pattern. In this example, the algorithm is able to generalize beyond the initial exemplar set in a sensible way.

The last column in the figure shows the *most-general conforming grammar* (MGCG) over the exemplars. This grammar provides a useful benchmark by which to judge the quality of the induction. The MGCG is obtained in the limit of the merging process, when all type-compatible symbols have been merged. Accordingly, the MGCG's description length is small and its generalization capacity is large, but the likelihood of the exemplar set is low. In essence, the MGCG makes a Markov assumption of stationarity that is not well-founded in design domains. In fact, the MGCG is precisely the grammar that is generated by model synthesis techniques that are popular in visual computing applications today [28, 6].

By varying the parameter λ in the grammar prior, we can smoothly interpolate along the space of grammars between the LGCG and the MGCG. Good design patterns will naturally fall somewhere in the middle of these two extremes. Table 1 gives statistics for the grammars induced in this paper, including the description length, likelihood of the exemplar set under the grammar, and cumulative probability of the two-hundred most likely designs generated by the grammar. This latter quantity is useful as a measure of the capacity of the grammar to generalize beyond the set of exemplars on which it was trained.

Figure 4 shows typical samples from the MGCG and the optimal grammar induced from a set of spaceship models. Models produced from the MGCG resemble the exemplar models only locally; conversely, the models synthesized with our technique exhibit similar global structure. Figure 5 shows fifty distinct random samples from a design pattern induced from six Japanese castle models; Figure 6 shows fifty distinct random samples from a design pattern induced from eight different sakura tree models.

Figure 7 shows modes from the distribution defined by the grammar of Seussian architecture, along with their probabilities. While the majority of the produced designs are plausible, these samples also highlight some of the limitations of our framework (highlighted in red). Because we induce context-free grammars, it is not possible for these design patterns to reliably learn high-level semantic constraints like "every building must have at least one door." Similarly, since our models are specified as hierarchies, relationships between subtrees which are very far apart in the derivation are difficult to capture: thus, some models have stairways that are not connected to the ground.

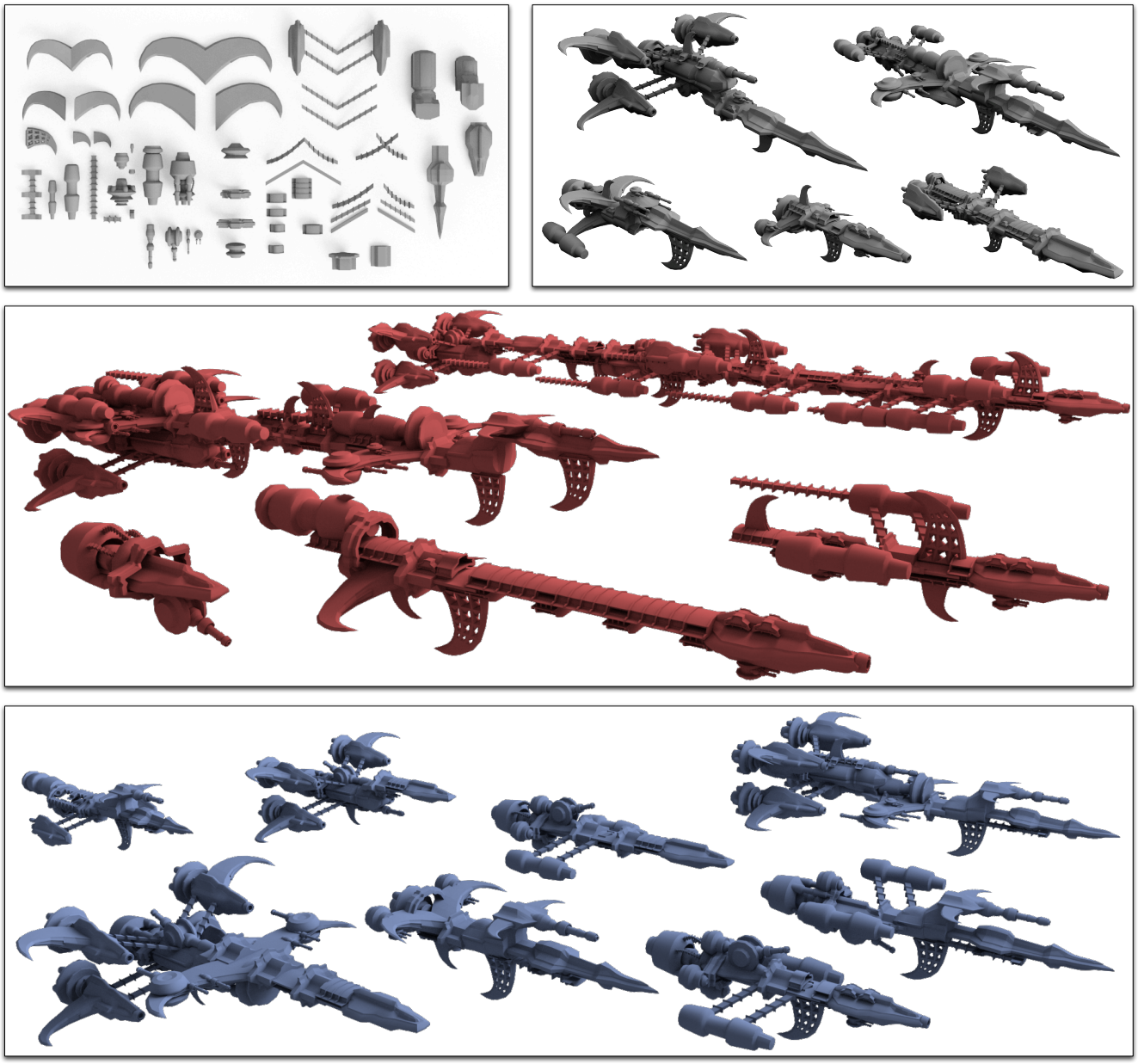


Figure 4. Spaceships. (top left) The set of components. (top right) The exemplars. (mid) Random samples from the MGCG: observe how these designs resemble the exemplars only locally. (bottom) Random samples from the induced design pattern: note the more plausible global structure.

We can also gain insight about the patterns induced by our technique by inspecting the grammar itself. For instance, when we examine the grammar we induce over Web pages, we see one rule that differentiates between two classes of designs. Each consists of a header and a common center structure comprising either two or four columns; one class has a hero image, while the other does not. Figure 3 shows a small portion of our page corpus, as well as a few random derivations from the learned model of page structures.

User Study

To assess how effectively the presented technique is able to generalize beyond the set of exemplars, we conducted a cognitive evaluation on Amazon's Mechanical Turk for three of the geometric domains. In each experiment, we sampled a collection of two-hundred designs from our induced grammars and the MGCGs. One-hundred test subjects were then shown the set of exemplar designs and a single sampled design below, and asked to answer the question "How similar in style is this design to the ones above?" on a four-point Likert scale ranging from "not at all similar" to "very similar." This process was repeated until each participant had rated twenty random designs. Responses were normalized to $[0, 1]$.

Table 2 shows the results from this study. We see higher average ratings for the designs sampled from our learned grammars by proportions of between 7% and 21%, all with $p < 0.001$ two-tailed using a Mann-Whitney-Wilcoxon test.

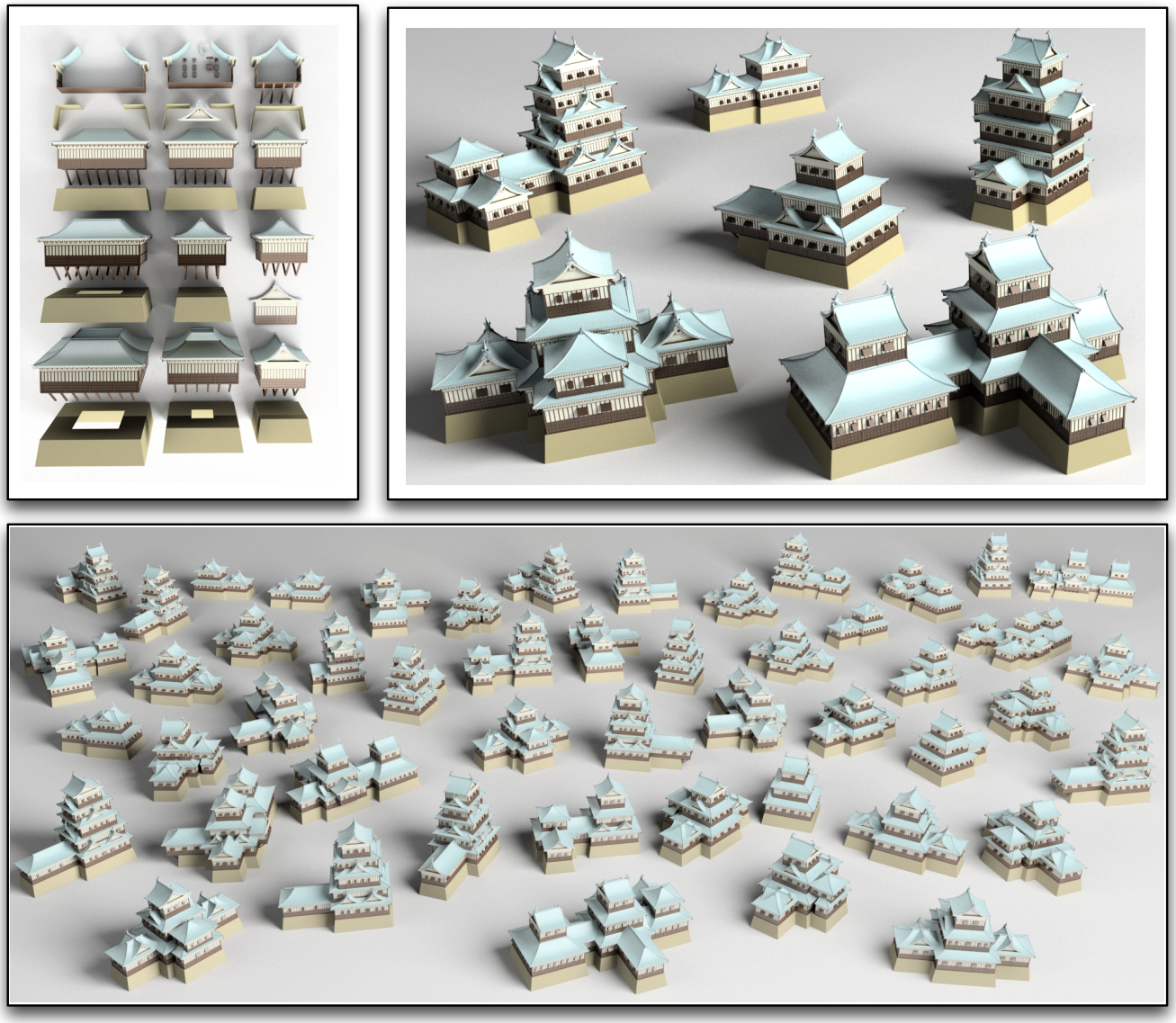


Figure 5. Japanese castles. (top left) The set of components. (top right) The exemplar designs. (bottom) Fifty distinct random derivations from the induced design pattern.

Although this is only a preliminary validation of our approach, this study seems to indicate that the grammars induced by our technique are better predictors of stylistic similarity than existing model synthesis methods.

DISCUSSION AND FUTURE WORK

Although the method we describe in this paper is just a first step towards learning design patterns from data in a principled way, it raises several interesting avenues for future work. Inducing patterns for other design domains is one obvious next step: even for classes of data that lack explicit hierarchical structure, it may be possible to infer hierarchies as part of the learning process [36]. Similarly, extending the induction to scale efficiently to training sets of thousands or millions of examples is another important direction for future research [40]: imagine learning a generative model of page designs that is trained over the entire Web.

	MGCG	Bayes-optimal	U
Seuss	.69	.76	15288
Sakura	.54	.75	8990
Spaceship	.58	.67	14777

Table 2. Average ratings for the three domains in our user study, along with the Mann-Whitney U value.

Data-Driven Design Tools

Another area ripe for further investigation is leveraging the induced patterns to build better tools and interaction mechanisms for content producers and designers. We hope that general, principled techniques for learning design patterns will reinvigorate the study of data-driven design in HCI. A few directions seem particularly promising.

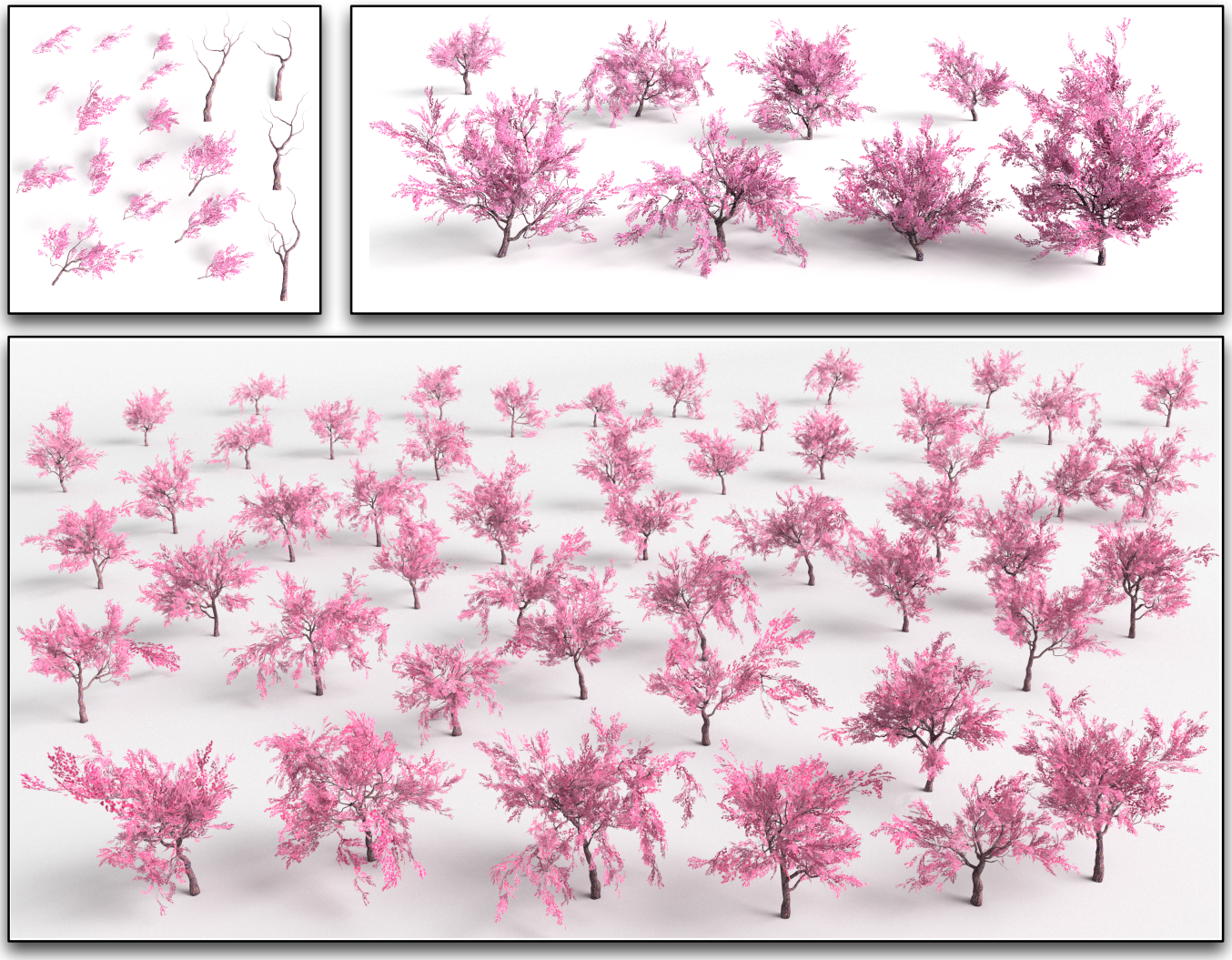
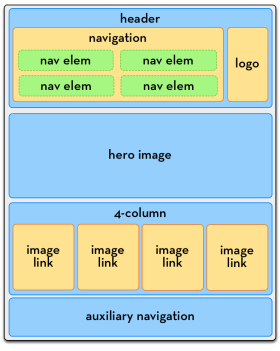


Figure 6. Sakura trees. (top left) The set of components. (top right) The exemplar designs. (bottom) Fifty distinct random derivations from the induced design pattern.

For one, generative models like the ones learned in this paper provide a simple algorithmic mechanism for producing design alternatives. Given a small set of representative examples, techniques like grammar induction can be used to quickly and easily generate a diverse selection of new designs that are stylistically similar—but not identical—to the exemplars. Furthermore, by varying the weighting parameter λ in the induction process, the degree to which the grammar generalizes can be controlled. In this manner, it seems likely that a whole host of tools for automatically populating scenes, galleries, and virtual environments with content could make use of design pattern learning.

In addition, it has been well established that considering alternatives in the design process itself facilitates discussion, improves the rate of convergence, and increases the quality of productions [43, 18, 11]. One can imagine a workflow à la active learning in which patterns are induced in the background during design and sampled to suggest new constructions which are then assimilated by users in turn.

Perhaps most intriguing is the potential for exploiting the rich mathematical structure of generative probabilistic models in tool-building. In particular, it seems likely that many common design tasks can be formulated as probabilistic inference problems over a particular design pattern, and solved with Monte Carlo methods [42, 45]. For instance, we can write down a smooth function that scores page designs based on how closely they conform to a particular specification, and perform MAP estimation via MCMC to find a page from our learned Web grammar that has a four-column layout, a single navigation bar with four navigation elements, a header, and a hero image (see inset).



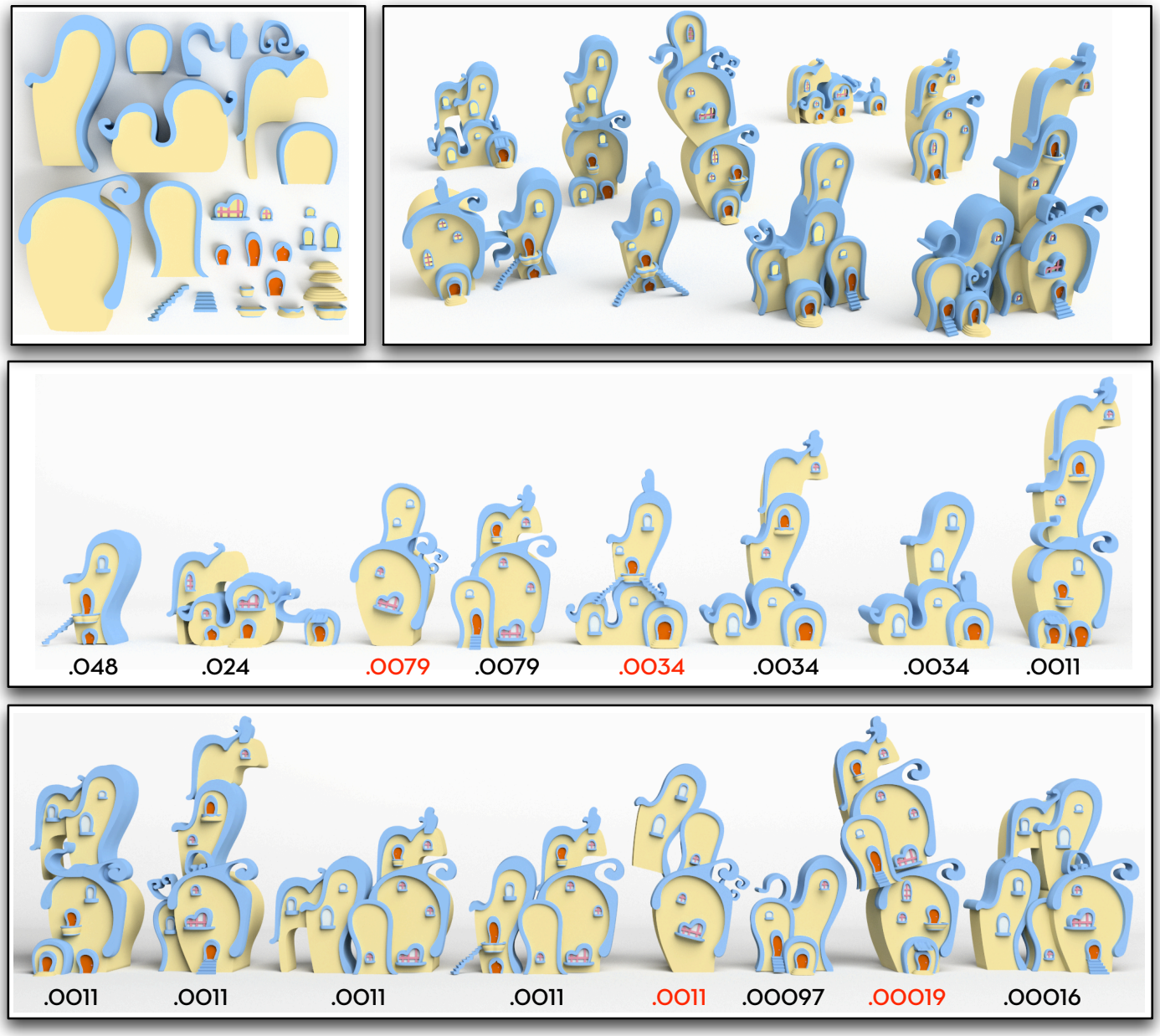


Figure 7. Seussian architecture. (top left) The set of components. (top right) The exemplar designs. (bottom) Modes from the distribution defined by the induced grammar, along with their probabilities. Models with questionable structure are highlighted in red.

New Computational Models

One final direction for investigation is learning more powerful computational models for design. Although stochastic context-free grammars provide a useful and compact generative representation, they are subject to a number of limitations which have led content creators to seek out more powerful graphical models [30]. For one, SCFGs automatically assign higher probabilities to shorter derivations, an assumption well-founded neither in natural language nor design, where model size typically peaks at some intermediate length. For another, the independence assumptions inherent in SCFGs prevent them from accurately representing models which have more general graph (rather than tree) structure, precluding them from capturing symmetries or other distant relationships between disjoint subtrees in a derivation. Similarly, CFGs are fundamentally discrete representations, and cannot easily encode continuous variability.

Recent work in probabilistic program induction suggests ways to overcome some of these limitations [20]: although the induction problem is more difficult with less structure *a priori*, universal probabilistic programming languages like Church [17] are capable of encoding any computable function.

ACKNOWLEDGMENTS

We thank Chris Platz for designing, building, and texturing all the 3D models used in this paper; Yu Lou for his work building an early version of our codebase; Tony Kaap for his invaluable Maya support; Haden Lee for his work on the Web examples; and Scott Klemmer—along with several members of the Stanford Graphics Laboratory—for providing valuable feedback on this research. This material is based upon work supported by the National Science Foundation under Grant No. 0846167, and a Google PhD Fellowship.

REFERENCES

1. Agarwal, M., and Cagan, J. A blend of different tastes: the language of coffee makers. *Environment and Planning B: Planning and Design 25*, 2 (1998), 205–226.
2. Aho, A. V., and Ullman, J. D. *The theory of parsing, translation, and compiling*. Prentice-Hall, 1972.
3. Alexander, C. *The Timeless Way of Building*. Oxford University Press, 1979.
4. Andrieu, C., de Freitas, N., Doucet, A., and Jordan, M. I. An introduction to MCMC for machine learning. *Machine Learning 50*, 1 (2003), 5–43.
5. Baker, J. K. Trainable grammars for speech recognition. In *Speech Communications for the 97th Meeting of the Acoustical Society of America.*, ASA (June 1979), 31–35.
6. Bokeloh, M., Wand, M., and Seidel, H.-P. A connection between partial symmetry and inverse procedural modeling. In *Proc. SIGGRAPH*, ACM (2010).
7. Borchers, J. *A Pattern Approach to Interaction Design*. John Wiley & Sons, 2001.
8. Buelinckx, H. Wren's language of City church designs: a formal generative classification. *Environment and Planning B: Planning and Design 20*, 6 (1993), 645–676.
9. Conway, A. Page grammars and page parsing: a syntactic approach to document layout recognition. In *Proc. ICDAR* (1993).
10. de la Higuera, C. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.
11. Dow, S. P., Glassco, A., Kass, J., Schwarz, M., Schwartz, D. L., and Klemmer, S. R. Parallel prototyping leads to better design results, more divergence, and increased self-efficacy. *ACM Trans. Comput.-Hum. Interact. 17*, 4 (2010), 18:1–18:24.
12. Duarte, J. P., Rocha, J. M., and Soares, G. D. Unveiling the structure of the Marrakech Medina: A shape grammar and an interpreter for generating urban form. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 21*, 4 (2007), 317–349.
13. Duyne, D. K. V., Landay, J., and Hong, J. I. *The Design of Sites*. Addison-Wesley, 2002.
14. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
15. Gold, E. M. Language identification in the limit. *Information and Control 10*, 5 (1967), 447–474.
16. Gold, E. M. Complexity of automaton identification from given data. *Information and Control 37*, 3 (1978), 302 – 320.
17. Goodman, N. D., Mansinghka, V. K., Roy, D. M., Bonawitz, K., and Tenenbaum, J. B. Church: A language for generative models. In *Proc. UAI* (2008), 220–229.
18. Hartmann, B., Yu, L., Allison, A., Yang, Y., and Klemmer, S. R. Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. In *Proc. UIST*, ACM (2008), 91–100.
19. Horning, J. J. *A study of grammatical inference*. PhD thesis, Stanford University, Stanford, CA, USA, 1969.
20. Hwang, I., Stuhlmüller, A., and Goodman, N. D. Inducing probabilistic programs by Bayesian program merging. *CoRR abs/1110.5667* (2011).
21. Jacobs, C., Li, W., and Salesin, D. H. Adaptive document layout via manifold content. In *Workshop on Web Document Analysis* (2003).
22. Klein, D., and Manning, C. D. A generative constituent-context model for improved grammar induction. In *Proc. ACL* (2002), 128–135.
23. Klein, D., and Manning, C. D. Corpus-based induction of syntactic structure: models of dependency and constituency. In *Proc. ACL* (2004), 478–487.
24. Knight, T. W. The generation of Hepplewhite-style chair-back designs. *Environment and Planning B: Planning and Design 7*, 2 (1980), 227–238.
25. Kumar, R., Talton, J. O., Ahmad, S., and Klemmer, S. R. Bricolage: Example-based retargeting for web design. In *Proc. CHI*, ACM (2011).
26. MacKay, D. J. C. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.
27. Manning, C., and Schutze, H. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
28. Merrell, P. Example-based model synthesis. In *Proc. I3D*, ACM (2007), 105–112.
29. Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. Procedural modeling of buildings. In *Proc. SIGGRAPH*, ACM (2006), 614–623.
30. Měch, R., and Prusinkiewicz, P. Visual models of plants interacting with their environment. In *Proc. SIGGRAPH*, ACM (1996), 397–410.
31. Parish, Y. I. H., and Müller, P. Procedural modeling of cities. In *Proc. SIGGRAPH*, ACM (2001), 301–308.
32. Perry, L. Modular level and component design. *Game Developer Magazine* (2002).
33. Prusinkiewicz, P., and Lindenmayer, A. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., 1990.
34. Pugliese, M. J., and Cagan, J. Capturing a rebel: modeling the Harley-Davidson brand through a motorcycle shape grammar. *Research in Engineering Design 13* (2002), 139–156.
35. Shuey, D., Bailey, D., and Morrissey, T. P. PHIGS: a standard, dynamic, interactive graphics interface. *IEEE Comput. Graph. Appl. 6*, 9 (Aug 1986), 50–57.
36. Socher, R., Manning, C., and Ng, A. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning* (2010).
37. Solomonoff, R. J. A formal theory of inductive inference. *Information and Control 7* (1964).
38. Stava, O., Beneš, B., Měch, R., Aliaga, D., and Kristof, P. Inverse procedural modeling by automatic generation of L-systems. *Computer Graphics Forum 29*, 2 (2010).
39. Stiny, G., and Mitchell, W. J. The palladian grammar. *Environment and Planning B: Planning and Design 5*, 1 (1978), 5–18.
40. Stolcke, A. *Bayesian Learning of Probabilistic Language Models*. PhD thesis, University of California at Berkeley, 1994.
41. Stolcke, A., and Omohundro, S. M. Inducing probabilistic grammars by Bayesian model merging. In *Proc. International Colloquium on Grammatical Inference and Applications* (1994), 106–118.
42. Talton, J., Lou, Y., Lesser, S., Duke, J., Měch, R., and Koltun, V. Metropolis procedural modeling. *ACM Trans. Graphics 30*, 2 (2010).
43. Tohidi, M., Buxton, W., Baecker, R., and Sellen, A. Getting the right design and the design right. In *Proc. CHI*, ACM (2006), 1243–1252.
44. Weitzman, L., and Wittenburg, K. Relational grammars for interactive design. In *IEEE Workshop on Visual Languages* (1993), 4–11.
45. Yeh, Y.-T., Yang, L., Watson, M., Goodman, N., and Hanrahan, P. Synthesizing open worlds with constraints using locally annealed reversible jump MCMC. In *Proc. SIGGRAPH* (2012).