Discovering New Designs through Stochastic Context-free Grammar Formulation

Prakhar Jaiswal
MAE505 Project Report
University at Buffalo
prakharj@buffalo.edu

Abstract

Design exploration using design patterns has proven useful and influential in many domains, such as architecture, geometric modeling, software engineering, Web design, etc. However, creating such patterns is a time-consuming and manual process, typically formulated and compiled by few experts in any given domain. The objective of this paper is to algorithmically learn design patterns directly from data and to extract general principles from the provided exemplars of any given domain to synthesize novel designs in that domain. This problem is formulated as grammar induction, i.e., given a set of exemplar designs, a probabilistic formal grammar is induced over the exemplars. The root of this induction lies in learning how to generalize beyond the exemplar set without extrapolating patterns that are not supported by the data. The use of grammar-based models for this task provides a data-driven way to learn design patterns that are neither too specific nor overly general.

1. Introduction

During conceptual design phase, designers often attempt to converge at a single solution path. Especially, novice designers have tendency to adopt the first idea that comes to mind. They rarely realize that the adopted approach might not be the best one. Diligently identifying a proper solution from a broad range of possible solution alternatives is the best way to ensure that the final solution is relatively a good one. To obtain a vast range of alternatives, often attempts are made to formulate a set of rules for designers in a particular domain, which serves as a general guidelines and principles for a specific type of problem. Such design patterns are widely used and have proven beneficial in domains such as architecture \[2\], geometric modeling \[11, 13\], software engineering \[21\], interaction \[4\], and Web design \[20\].

Even though the design patterns are widely used, they are inefficient and ineffective in some sense. Users have to firstly decide and look for suitable design patterns for their specific needs. Often, they need to carefully combine multiple resources in coherent manner and apply it to their own problems. This process might not lead to accurate solutions in efficient fashion. Another problem with design patterns are that they are formulated for specific problems by few experts in that particular field. This process is also tedious and time consuming for the experts, and the resulting design pattern might not be accurate and comprehensive. It also suffers from an inclination of being biased towards the opinions of experts \[19\].

To overcome such inefficiencies and inaccuracies, we propose to algorithmically learn design pattern directly from data and codify them in a manner so that they can be used effortlessly. For this purpose, we use the concept of formal grammar. We formulate the problem as a grammar induction problem, wherein a set of input exemplars from a particular domain is used to learn a probabilistic formal grammar. The input exemplars are encoded in form of labeled trees, which are fed into the algorithm to obtain consistent and novel design solution alternatives as outputs.

Grammar induction process can be viewed as learning general principles and guidelines from a given set of exemplars and generalizing them such that they are neither too specific nor overly general. This ensures that the new design solution suggested by the algorithm is reliable and consistent with the provided input set. It is made possible by formulating grammar induction problem in probabilistic framework and performing an iterative search using simulated annealing approach to obtain Bayesian optimal grammar. The grammars are scored based on its ability to generate input exemplars, generalization capacity and compactness. These factors play an important role in allowing users to obtain desired results for their specific problem domain with ease.

The results of the algorithm are demonstrated using examples of 2D pictorial representation of houses and 3D model of tables. The next section discusses related work.
2. Related Work

The work presented in this paper is aligned with design pattern learning algorithm presented by Talton et al. [19]. However, some minor changes have been incorporated in this paper related to the structure of the grammar and the definition of the proposal distribution used for simulated annealing in search of global optima of posterior probability distribution. Furthermore, this paper presents the algorithm in a manner that could prove to be comprehensible by larger audience, including the researchers from probability theory, stochastic processes, and 3D modeling community.

Grammars have been used extensively as generative models in design field. Domain such as architecture [5, 7, 12], product design [1, 8, 13], document layout [6, 22], and 3D modeling [10, 11, 12] are some of the domains where the usage of grammar for procedural modeling is well known. Even though grammars have proved to be effective and valuable in such fields, grammars are formulated manually in majority of the work. This could be very tedious and cumbersome task even for experts from particular field. Few attempts have been made to learn deterministic rule from patterns [16] and adapting texture synthesis techniques for geometric modeling [3, 9]. However, these works have not presented a principled way to learn grammars from examples. In this paper, we use grammar induction, first introduced by Solomonoff [15], using a techniques called Bayesian Model Merging [18].

3. System Overview

This section provides a brief overview of the system architecture. Figure 1 presents a flowchart for the complete process. The system takes as an input a set of exemplar designs in the form of labeled trees. The trees are formed by assigning labels to each of the component used for creation of the exemplar set. The labels for the components form the nodes of the trees. An edge is included between two nodes in a tree corresponding to the spatial adjacency of corresponding components in the design. The set of labeled trees is inputted into the algorithm, along with type compatibility. Type compatibility defines the compatibility between nodes that can be interchanged.

The algorithm begins with forming a grammar for the input exemplar trees. The grammar is created by traversing through each tree in the exemplar set and constructing production rules for every unique subtree. Each label (or component) corresponds to one terminal in the grammar. The grammar generated in this process is called least general conforming grammar (LGCG). This is because the grammar is conforming as it generates each of the elements in the exemplar set and has no generalization capacity as it generates no other element which is not included in the exemplar set.

After construction of LGCG, the parameters which define the probabilities associated with each production rule are calculated. This is done by an iterative process of assigning higher probabilities to production rules which has higher expected count of usage in generating the exemplar set. Once the parameters are obtained, the posterior probability of the grammar given the input exemplar set is evaluated. The process of calculation of parameters and posterior probability is repeated for every grammar that is obtained by merging or splitting operation during the Markov chain Monte Carlo search.

The Markov chain Monte Carlo search is used to explore the space of more general conforming grammar. More general grammars are obtained by merging and splitting operation on the nonterminals in the grammars. The merging operation assigns a new common nonterminal to two type compatible nonterminals in the grammar and unions there production rules. The split operation is reverse of merge operation, as it assigns two new nonterminals replacing one nonterminal in the grammar. Each instance of the nonterminal being replaced is changed to one of the two new nonterminals, and the production rules of the nonterminal are randomly distributed into the new nonterminals.

Simulated annealing approach is used for Markov chain Monte Carlo search. The posterior probability of the grammar given the input exemplar set is used as the invariant probability distribution in this approach. Of course, the support of the posterior probability is the discrete set of grammars, which includes all possible combination of type compatible merge and split operation on LGCG. The support or the sample space spans from least general to most general grammar. At each iteration of the search process, a merge or split operation is chosen randomly and a new grammar is generated. Bayes-optimal grammar is sought in the search process, which is the grammar with highest posterior probability. At the end of the process, the grammar with the highest posterior in all iteration is returned. This grammar is then used to derive trees, which can be interpreted as designs based on the nodes and edges, which denotes the components and spatial adjacency, respectively.

4. System Description

This section describes the steps involved in the pattern learning from the set of input exemplar designs.

4.1. Stochastic Context Free Grammar

To obtain new design suggestions based on a given set of input designs, formal grammars have been used in the algo-
Figure 1. System Architecture.

4.2. Initial Grammar Creation

Given an input set $M$ of exemplars, i.e. a set of labeled trees, the objective of this step is to create a grammar $G$ whose language $\mathcal{L} = M$. This grammar is called least general conforming grammar (LGCG) and is used to initialize the search process. To create this grammar, the algorithm traverses through each exemplar tree one by one and generates a nonterminal for each unique tree or subtree. The production rules are also simultaneously added for each unique subtree (or tree) $\tau$ with its corresponding nonterminal $v$ as predecessor and $t v_1 v_2 \ldots v_k$ as the successors, where $t \in T$ is the terminal symbol at the starting node of subtree $\tau$ and $v_1, v_2, \ldots, v_k$ are the nonterminals corresponding to the $k$ child subtrees $\tau_1, \tau_2, \ldots, \tau_k$ of $\tau$. Each of the production rules are assigned a probability of 1, because in LGCG, there is only a single rule for each predecessor $v \in V$. Finally, the nonterminal symbol corresponding to each of the input tree in exemplar set is added to the axiom set $\omega$.

Along with the LGCG, a type compatibility between nonterminals for merging operation is also determined in this step. From the user input on type compatible nodes/components, which is essentially the type compatibility of the terminal symbols in the grammar, type compatibility of nonterminals is obtained. This can be done by making all the nonterminals corresponding to subtrees (or trees) compatible which have compatible starting terminal nodes. That is, if a set $T' \subseteq T$ consists of type compatible terminals with type $T$, then all the $v_1, v_2, \ldots, v_k$ corresponding to $\tau_1, \tau_2, \ldots, \tau_k$ with starting node $t \in T'$ is assigned type $T$.
### 4.3. Merging and Splitting Operations

For exploration of the space of more general grammars, the nonterminals are merged and split to obtain a new state of grammar \( G' \) from current state \( G \). The merge or split operation only changes the structure of the grammar, however, the parameters \( \theta_G \) for the new grammar \( G' \) is estimated after every merge or split operation as described in Section 4.3. Each merge operation selects two type compatible nonterminal symbols \( v_i, v_j \in V_G \), and assigns a new common nonterminal symbol \( v_k \notin V_G \) to it (\( V_G \) is the set of nonterminals of \( G \)). Every instance of \( v_i \) and \( v_j \) in production rules \( R_G \) of \( G \) is replaced by \( v_k \) to obtain \( R_{G'} \). This process occasionally creates repetition of rules in \( R_{G'} \), which should be eliminated. Similar steps are adopted for axiom set \( \omega_G \), as well, if \( v_i \) and/or \( v_j \) appear in \( \omega_G \). Also, \( v_i \) and \( v_j \) are removed from set \( V_G \) and \( v_k \) is added as an element to obtain \( V_{G'} \).

A merge operation makes the grammar more general, i.e. it increases the cardinality of the language of grammar. This means that a merge operation decreases the likelihood of obtaining the input exemplars. It also decreases the description length of grammar which is used to obtain the posterior probability of grammar given the input exemplar set. If all the possible merge operations are performed on a grammar, we obtain the most general grammar. Possible merge operations include the merging of all type compatible nonterminals. Since, the terminal symbols in the grammar represents actual components in a particular design domain, the semantics of the domain might restrict the interchangeability of the components. These constraints translate into restrictions on merging of two type incompatible nonterminals. With each merge operation, the type compatibility of the nonterminals is also updated by assigning the same type to the new nonterminal as the type of two nonterminals being merged.

A split operation is the reverse of merging, where a nonterminal symbol \( v_i \in V_G \) is chosen at random, such that there are more than one rules with predecessor \( v_i \). Two new nonterminals \( v_{i1}, v_{i2} \notin V_G \) are created. Then, the rules with predecessor as \( v_i \) are distributed among the two new nonterminals. It should be ensured, in this process, that both \( v_{i1} \) and \( v_{i2} \) are assigned as predecessor to at least one production rule. Every instance of \( v_i \) in successors of \( R_G \) is replaced by randomly selecting one of the new nonterminals for every instance. If \( v_i \in \omega_G \), then it is replaced, and both \( v_{i1} \) and \( v_{i2} \) is included to obtain \( \omega_{G'} \). Also, the set of nonterminals is modified accordingly. The new nonterminals are assigned the same type as \( v_i \). Note that, a split operation is not possible on LGCG, as each nonterminal appears only once as predecessor in \( R \).

### 4.4. Bayesian Inference for Grammars

Given a set \( M \) of exemplar designs, a posterior probability distribution is formulated over the space of potential grammars.

\[
P(G|M) \propto L(M|G)\pi(G),
\]

where \( L(M|G) \) is the likelihood of the obtaining the input set of designs \( M \) given the grammar \( G \) and \( \pi(G) \) is the prior probability of the grammar \( G \). Bayes-optimal grammar is the grammar with highest posterior probability among all possible grammars given the input set \( M \). The set of possible grammars serves as the sample space for the posterior probability. It is used as the search space for the algorithm, and it spans from least general to most general grammar. The input exemplar set is a subset of the language of each grammar in the sample space. As the grammar with highest posterior is sought, the search process reduces to maximizing the product of likelihood and prior.

**Design Likelihood** : Given a grammar \( G \), the probability of obtaining the exemplar set \( M \) can be calculated as follows:

\[
L(M|G) = \prod_{m \in M} P(m|G),
\]

where \( P(m|G) \) is the probability of deriving the design \( m \) from grammar \( G \). It can be calculating by summing up all the probabilities of possible sequences of rules resulting in \( m \).

\[
P(m|G) = \sum_{\gamma \Rightarrow m} P(\gamma|G)
\]

The probability of a sequence \( \gamma \) can be calculated by multiplying the probabilities associated with each of the rules in the sequence.

\[
P(\gamma|G) = \prod_{\rho \in \gamma} \theta_G(\rho)
\]

**Grammar Prior** : Since a grammar can be decomposed into its structure and its parameters, by using chain rule the prior of a grammar can be formulated as follows:

\[
\pi(G) = P(S_G, \theta_G) = P(\theta_G|S_G)\pi(S_G),
\]

where \( P(\theta_G|S_G) \) is the probability of parameters of grammar given its structure, and \( \pi(S_G) \) is the prior probability of grammar structure. The probability of grammar’s parameter given its structure can be computed by multiplying the probability of the subsets of the parameters, where each subset \( \theta_v \) consists of the parameters of the rules with predecessor \( v \in V_G \).

\[
P(\theta_G|S_G) = \prod_{v \in V_G} P(\theta_v|S_G)
\]
The probability of a subset $\theta_v$ given the structure of grammar can be formulated as symmetric Dirichlet distribution,
\[ P(\theta_v|S_G) = \frac{1}{B(\alpha)} \prod_{\rho \in R_v} (\theta_G(\rho))^\alpha - 1, \]  
where $R_v \subseteq R_G$ is the set of rules with predecessor $v \in V_G$, $\alpha > 0$ is the concentration parameter and $B(\cdot)$ is the multinomial beta function, which can be expressed in form of gamma function as follows:
\[ B(\alpha) = \frac{\Gamma(\alpha)^K}{\Gamma(K\alpha)}, \]  
where $K$ is the cardinality of set $R_v$. Note that the support of the Dirichlet distribution is the set $\theta_v$ with cardinality $K$, where $\sum_{k=1}^K (\theta_v)_k = 1$, and $(\theta_v)_k \in (0,1]$. By choosing value of the concentration parameter $\alpha = 1$, a uniform distribution over all possible production probability distributions is obtained. With $\alpha < 1$, a distribution in which most of the probability mass is concentrated in few production probabilities is favored, and with $\alpha > 1$, the distributions with uniform production probabilities are preferred.

To compute structure prior, the description length of the grammar $\ell(S_G)$ is used. The structure prior can be formulated as:
\[ \pi(S_G) = \left( \exp\left( - \ell(S_G) \right) \right)^\lambda, \]  
where $\lambda$ is the weighing term which is used to tune the algorithm’s preference towards general grammars. The description length of the grammar is computed by counting the total number of symbols in the grammar’s axiom and its production rules’ predecessors and successors. Each symbol contributes $\log_2(|V| + |T|)$ bits to the description length. Here, $|\cdot|$ denotes the cardinality of the set. With each merge operation, as the grammar becomes more and more general, the description length of grammar will decrease, increasing the structure prior probability. Consequently, since $0 < \exp(\ell(S_G)) < 1$, higher values of $\lambda$ favors less general grammar and lower values of $\lambda$ favors more general grammar.

### 4.5. Parameter Estimation

After each merge or split operation, the structure of the new grammar is obtained. However, to evaluate the posterior probability, the parameters $\theta$ also need to be determined for the new grammar. These are obtained by maximizing the likelihood of the parameters given the grammar structure and the input exemplar set. This is essentially maximization of likelihood of exemplar set given the structure and parameters of the grammar:
\[ \theta_G = \arg\max_{\theta} L(\theta|S_G, M) \]  

This problem can be solved by iteratively updating the parameters using the expected count of the rules. At each iteration, given the current parameter values, the updated parameter values $\hat{\theta}$ for a rule $\rho \in R_v$, is calculated as follows:
\[ \hat{\theta}(\rho) = \frac{\sum_{m \in M} c_{\varphi}(\rho; m)}{\sum_{m \in M} \sum_{q \in R_c} c_{\varphi}(q; m)}, \]  

where $c_{\varphi}(\cdot; \cdot)$ is the expected count of number of times a particular rule is used in derivation of a given design. The expected count is calculated using the current values of the parameters. Since, the expected count depends on the current value $\theta$, the process has to be repeated iteratively until the parameters converge. At the start of this iterative process, the parameters are initialized with uniform probabilities for all the rules with predecessor as a particular nonterminal.

In this process, higher probabilities are assigned to the rules which are used more number of times in derivation of input exemplar designs. This, in turn, maximizes the likelihood of the input exemplar set given the grammar.

### 4.6. Markov chain Monte Carlo Search

Markov chain Monte Carlo (MCMC) search is used to explore the space of possible grammars. Simulated annealing approach is used for global optimization of posterior probability function over the set of possible grammar. The support of the posterior probability is the finite and discrete set of grammars spanning from least general to most general grammar. The Bayes-optimal grammar corresponds to the maxima of the posterior probability function. To obtain the global maxima of this probability function over the search space/support, simulated annealing approach is used. For this purpose, the proposal distribution $q(G'|G)$ is defined as the probability of sampling grammar $G'$ given the current state of grammar $G$. In the state space, the edges between the states of grammar represent a merge or a split operation. Pair of grammars which are not interchangeable by a single merge/split operation are not connected in the state space and the associated proposal probability is zero. The LGGC is used as the starting node to initialize the state and the algorithm is run for a fixed number of iterations.

At each iteration $i$, a merge or split move is selected at random and the current grammar $G$ modified accordingly to obtain a new grammar $G'$. A random sample $u_i \sim U(0,1)$ is also drawn from uniform distribution with 0 and 1 as lower and upper bounds. This new grammar is then accepted as the current search state if the following acceptance probability is larger than $u_i$:
\[ A(G'|G) = \min\left( 1, \frac{P^{1/T_i}(G'|M)q(G|G')}{P^{1/T_i}(G|M)q(G'|G)} \right), \]  

where $P(\cdot|M)$ is the posterior probability of a grammar given the input exemplar set $M$ defined in Eq. 3 and $T_i$ is...
the decreasing cooling schedule with \( \lim_{i \to \infty} T_i = 0 \). After the algorithm has been run for pre-specified fixed number of iterations, the grammar with highest posterior probability among all the iterations is returned as Bayes-optimal grammar.

The proposal probability \( q(\cdot | \cdot) \) for merge and split operation is defined in reverse manner. Consider a merge operation on grammar \( G \), where firstly a type \( T \), which includes at least two nonterminals, is selected. Then, from among all the nonterminals of type \( T \), two nonterminal \( v_i \) and \( v_j \) are selected at random to be replaced by a new nonterminal symbol \( v_k \not\in V_G \) to obtain a new grammar \( G' \). For the reverse probability distribution, consider the grammar \( G' \) on which a split operation is used to get grammar \( G \). In this case, firstly a nonterminal \( v_k \in V_{G'} \), which is predecessor to at least two production rules, is selected at random. Then, the production rules with \( v_k \) as predecessor is split into two new nonterminals \( v_i \) and \( v_j \), with the constraint that at least one production rule is assigned to each of the new nonterminals. Finally, all the instances of \( v_k \) in the successors of production rules of \( G' \) is randomly replaced by either \( v_i \) or \( v_j \) to obtain the grammar \( G \). So, the proposal probabilities for merge operation can be defined as:

\[
q(G'|G) = \frac{2}{a_G b_G (b_G - 1)}, \quad (14)
\]

\[
q(G|G') = \frac{1}{c_{G'}(2^{d_{G'}} - 2)2^{e_{G'}}}, \quad (15)
\]

where,

- \( a \): number of types which include at least two nonterminals,
- \( b \): number of nonterminals of selected type \( T \),
- \( c \): number of nonterminals which are predecessor of at least two production rules,
- \( d \): number of production rules with \( v_k \) as predecessor as selected nonterminal \( v_k \),
- \( e \): total number of instances of selected nonterminal \( v_k \) in the successors of all production rules,
- subscript: reference grammar for these parameters.

The proposal probability for split operation can be defined similarly as follows:

\[
q(G'|G) = \frac{1}{c_{G'}(2^{d_{G'}} - 2)2^{e_{G'}}}, \quad (16)
\]

\[
q(G|G') = \frac{2}{a_{G'} b_{G'} (b_{G'} - 1)}, \quad (17)
\]

5. Results

To evaluate the presented framework, experiments were performed on two distinct domains. The first experiment involved 2D pictorial representation of houses. The input exemplar set consisted of three models of houses constructed using ten different components. At the start of the experiment, these components where assigned different labels, and a hierarchical labeled tree was formed for each of the input exemplar design of house. Type compatibility was also defined for the components. This tree was fed into the algorithm to first create the least general conforming grammar, which was later used as the starting state for simulated annealing approach in search of Bayesian-optimal grammar. After obtaining the Bayes-optimal grammar, it was used to synthesize new designs for houses in form of labeled trees, which was then interpreted back into 2D pictorial form. Figure 2 shows the input components and exemplars, and few random samples of output designs for the house.

Similar approach was adopted for 3D models of tables. It consisted of three table models as an input, formed using seven different components. After obtaining the Bayes-optimal grammar, random samples were drawn from it, and the results are shown in Fig. 3. From the results, it can be deduced that this approach of learning grammar using induction process is effective in creating novel, sensible and
conforming design suggestions. The Bayes-optimal grammar was able to generalize beyond the input exemplar set while being consistent with them. However, it was observed that few of the suggested models were unable to capture the high level semantic attributes of the domain. For example, some of the results shown for house models doesn’t have doors, even though all the input exemplars had them. Similarly, some of the table models have multiple drawers stacked together, whose combined height goes beyond the height of table legs, which makes these model unstable and infeasible.

6. Conclusions and Future Work

The presented work shows that grammar induction could be used as a powerful tool to formulate design patterns using given exemplars and can be used to create novel design suggestions during conceptual design phase. This can benefit not only novice designers, but also expert designers by eliminating the time consuming, manual and tedious task of compilation of design patterns. This framework helps in automatically learning a probabilistic formal grammar based design pattern from input exemplars. This grammar could then be efficiently used to produce vast number of novel design solution alternatives. The generated alternatives are consistent and conforming with the input design exemplars.

Although, few design alternatives created might fail to capture some of the high level semantic attributes of the input exemplars, the approach lays foundation for some interesting possibilities for future work. One apparent future step is to use this approach in different domain and perform user studies to judge various aspects of the alternatives generated, such as, reliability of the method for different domains, novelty of the designs created, consistency among the output designs and input exemplars, etc. Another exciting work would be to use vast number of exemplars for induction process and evaluate the performance of the method. It is believed that using vast number of input exemplars will allow the algorithm to learn complex relationship between components and semantic attributes of designs.

An important direction of future work is to introduce a measure of feasibility and novelty of designs, so that the output alternatives can be evaluated automatically based on these measures. Furthermore, the probabilistic formal grammar, though being a powerful tool, has certain limitations. It works better for domain with hierarchical design structure, and it is biased towards assigning higher probabilities to shorter derivations. Moreover, it is incapable of capturing high level relationships such as symmetries or other distant relationships between disjoint sub-trees in a derivation [19]. These limitations provoke designers to seek for more powerful tools. It would be interesting to investigate the use of other probabilistic frameworks for automatically learning design patterns.

Acknowledgments

I would like to thank Dr. Sonjoy Das and Dr. Rahul Rai for their valuable guidance, suggestions and feedback throughout the course of this project.

References


