Genetic Programming for Design Grammar Rule Induction

Julian R. Eichhoff and Dieter Roller

Institute of Computer-aided Product Development Systems, University of Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
{julian.eichhoff,dieter.roller}@informatik.uni-stuttgart.de
http://www.iris.uni-stuttgart.de

Abstract. The knowledge engineering effort associated with defining grammar systems can become a barrier for the practical use of such systems. Existing grammar and rule induction algorithms offer rather limited support for discovering context-sensitive graph grammar rules as required by some applications in the domain of engineering design. For this task the present work proposes a rule induction method grounded on Genetic Programming. Specializations regarding the representation and evaluation of rule candidates are discussed. Results from preliminary experiments with a prototype implementation demonstrate the feasibility of the suggested approach.

Keywords: Genetic Programming, Rule Induction, Graph Grammar, Machine Learning, Design Graph, Functional Decomposition

1 Introduction

There is a growing interest in using grammar-based systems in the field of computer-aided design. Sridharan and Campbell [10], for instance, use a context-sensitive graph grammar to explore functional descriptions of product designs. Here, grammar rules capture common moves for generating such descriptions, i.e. they represent knowledge on making design decisions. So far the process of formalizing this knowledge into rules has been subject to manual knowledge elicitation. Our research is targeted towards answering the question: *What are adequate machine learning processes for reducing the knowledge engineering burden of human experts in the context of automated engineering design and in which settings do they apply?*

In this paper focus is put on learning an additional grammar rule p_i in presence of an existing but incomplete rule set $[p]^A$ (sets are depicted in square brackets). The new rule set $[p]^B = [p]^A \cup p_i$ will be used in a graph-rewriting system that shall be capable of reproducing a set of desired design graphs $[\widehat{G}_n]$ by means of derivation from an initial graph G_0. All rules are allowed to be context-sensitive, i.e. left-hand side (LHS) and right-hand side (RHS) may contain both terminal and non-terminal (NT) elements. Considering the practical application for engineering two additional restrictions apply:

Restriction 1: There is no explicit information available on how the rules of $[p]^A$ are defined (LHS and RHS are hidden from the learner). This restriction stems from an application scenario where different engineering organizations, each working on different subsystems or disciplines, contribute to a commonly used graph-rewriting system. It is likely that these organizations want to share results with their partners, whilst being reluctant to reveal insight on *how* these results were established, i.e. their design rationale should be kept secret.

Restriction 2: As training data we are only given the initial graph G_0 (input) and desired design graphs $[\widehat{G}_n]$ (output). The data does not provide any negative examples of undesired graph-rewriting behavior. This restriction suits common engineering practice where positive milestones and final results are more likely to be documented than unfinished states and failures.

In light of this, a machine learning process based on Genetic Programming (GP) is proposed and studied using the following methodology: First, the feasibility of using GP for rule induction in the given context is validated by constructing a prototype implementation. Then, its applicability is tested with respect to an existing graph grammar from engineering design (pilot case) and a comparative study with an existing rule induction algorithm is undertaken.

Accompanying materials, including an application example and a discussion on extending the proposed method to iteratively learn multiple rules, are provided online at `http://ouky.de/accompanying-materials/ruleml-2015`.

2 Related Work

Besides various contributions addressing the induction of context-free string grammars, there only has been relatively little research on learning context-sensitive graph-grammars [1, 6, 4, 2, 5, 3]. None of these approaches is readily applicable under the mentioned restrictions as they are bound to several limitations: First, LHSs were limited to rule templates (cf. [1, 4]) or LHSs and RHSs were limited to single components. Hence, existing approaches are not capable of accepting multiple isolated subgraphs in their LHS (e.g. to link these components with new edges). Second, all grammars have been induced from scratch. Completing a set of given (unknown) rules has not been considered. This complicates the problem of rule induction as existing rules may produce or require nodes, edges, or application conditions that cannot be inferred from the training examples.

In order to address these limitations, machine learning can be conceptualized as search over a space of possible hypotheses [9]. In this case, hypotheses are candidates for rule p_i and the search space is defined by the vocabulary and syntax used for formulating rules. The goal of the learning algorithm is to search for rule candidates that best describe the training data. GP [7] is a well known evolutionary search heuristic. It has proven suitable for large search spaces and is robust against local optima. Wyard [11] first used a genetic approach to the induction of context-free string grammars. However, as shown herein, GP can also be employed for inducing rules in context of an existing rule set.

3 Proposed Solution

This section describes a GP-based rule induction algorithm for learning rules that comply with an existing context-sensitive graph grammar. The learner's task is to determine definitions for rule p_i in order to enable a graph-rewriting system to produce the desired design graphs $[\widehat{G}_n]$ from the initial black-box design G_0.

Consider a predefined rule sequence $S = p_1, p_2, ..., p_{i-1}, p_i, p_{i+1}, ..., p_{n-1}, p_n$ of length n that can be used to derive $[\widehat{G}_n]$ from G_0. Subsequences $S_1 = p_1, ..., p_{i-1}$ and $S_2 = p_{i+1}, ..., p_n$ are known to the graph-rewriting system (reasoning process) but unknown to the learner, and p_i is unknown to both. The proposed learner is capable of learning definitions for the LHS and RHS of p_i given the following is provided: G_0, $[\widehat{G}_n]$, and access to the graph-rewriting system in order to gather answer sets $[G_{i+1}], [G_{i+2}], ..., [G_n]$ for possible candidate definitions of p_i.

The proposed learner's *main* procedure (see algorithm 1) learns a rule candidate for one desired design graph at a time and tests if the found candidate is applicable to more than one graph of the desired answer set. Its sub-procedure *evolve* is a GP algorithm specially configured for the rule induction task.

Algorithm 1 *Main* procedure of proposed GP-based rule induction algorithm.

```
Load $G_0$ and $[\widehat{G}_n]$
Initialize graph-rewriting system with rules $p_1, p_2, ..., p_{i-1}$ and $p_{i+1}, ..., p_{n-1}, p_n$
while $[\widehat{G}_n]$ is not empty do
  Get next $\widehat{G}_{n,j}$ from $[\widehat{G}_n]$
  Use evolve to learn a rule $p_{i,j}$ that enables derivations $G_0 \overset{S}{\Longrightarrow} \widehat{G}_{n,j}$
      where $S = p_1, p_2, ..., p_{i-1}, p_{i,j}, p_{i+1}, ..., p_{n-1}, p_n$
  Remove $\widehat{G}_{n,j}$ and all graphs of $[\widehat{G}_n]$ that can be derived using $p_{i,j}$
  Add $p_{i,j}$ to the set of learned rules $[p_i]$
end while
return $[p_i]$
```

During GP, candidates for p_i are sampled from the space of possible rule definitions by means of evolutionary principles. Initially, the set of considered candidates, called population, is chosen randomly. Then, with each GP iteration, a new generation of this population is produced by applying mutation, crossover and selection operators. The selection operator ranks every produced candidate with respect to evaluation criteria and determines what candidates are considered for the next generation (survival of the fittest). Mutation and crossover are used to produce new candidates from the members of the current population. Crossover recombines parts of two candidates to form a new candidate, whereas mutation randomly changes parts of an existing candidate.

As a preparatory step, all possible host graphs $[G_{i-1}]$, on which p_i may be applied, are computed. Then, from this set a subset $[\widehat{G}_{i-1}]$ is formed that only contains host graphs which are feasible for deriving the current desired design graph $\widehat{G}_n$. This is realized by a filter that checks whether all monotonic elements present in G_{i-1} are also present in $\widehat{G}_n$. In this case only NTs are non-monotonic. Once added, terminal nodes and edges are preserved throughout the remaining direct derivations. Hence, there must be a subgraph isomorphism that maps all terminal nodes and edges of G_{i-1} to G_n. If this is given, then $G_{i-1} \in [\widehat{G}_{i-1}]$.

3.1 Representation

The representation of candidates is crucial for the rule induction task. In this case a candidate denotes, which host graph G_{i-1} is chosen from $[\widehat{G}_{i-1}]$, and what graph operations will be performed on G_{i-1} to yield G_i:

Regarding the first point, all graphs in $[\widehat{G}_{i-1}]$ are considered as possible bases for deriving $\widehat{G}_n$. Since the learner is not allowed to inspect the LHS of the following rules, different $G_{i-1} \in [\widehat{G}_{i-1}]$ need to be tested by executing the graph-rewriting system in a manner of trial-and-error. Which G_{i-1} is chosen is reflected by an index parameter within the rule candidate representation that is subject to the evolutionary optimization.

The second point addresses the RHS of the rule. Essentially each rule candidate is a tree structure (the standard representation for GP individuals) that specifies how the chosen G_{i-1} will be modified. The tree consists of *operation nodes* and *index nodes. Operation nodes* represent the graph operations used for modifying G_{i-1}. For the problem at hand the considered operations are *add-node*, *add-edge*, *add-non-terminal*, and *remove-non-terminal. Index nodes* parametrize these operations with respect to G_{i-1} and $\widehat{G}_n$. Further, every operation has a cost factor associated that is used for evaluation.

In order to evaluate a rule candidate, the set of considered host graphs $[\widehat{G}_{i-1}]$ is passed to the GP-tree's root node. Using an *index node* attached to the root, some $G_{i-1} \in [\widehat{G}_{i-1}]$ is chosen. Then, G_i is initialized as a copy of this graph and passed to the root's children in order to be propagated through the whole tree. At each *operation node* the graph is modified with respect to the parameter defined by its *index node* child (see list below). An operation is bypassed if it is not applicable. Having passed this way through all nodes of the tree, the modified graph G_i is considered to be the direct derivation result of rule candidate p_i, just as if it had been applied in the graph-rewriting system.

Add-node: For all terminal nodes in $\widehat{G}_n$ that are not yet present in G_i, add the one specified by the *index node* to G_i. The operation is not applicable if the set of addable nodes is empty. The cost for this operation is 2.

Add-edge: For all edges in $\widehat{G}_n$ that are not yet present in G_i, add the one specified by the *index node* to G_i. Incident nodes that are not present in G_i are added as well. The operation is not applicable if the set of addable edges is empty. Adding an edge costs 2 and each added node adds up with an extra 2.

Add-non-terminal: For all edges in $\widehat{G}_n$ that are not yet present in G_i, but where exactly one incident terminal node already exists in G_i, select the edge specified by the *index node.* Add a NT and the selected edge to G_i, where one end of the edge marks the existing terminal node, and the other is the newly added NT. The operation is not applicable if the set of edges to be considered is empty. The cost for this operation is 1.

Remove-non-terminal: For all NTs in G_i, remove the one specified by the *index node*. The operation is not applicable if the set of NTs is empty. The cost for this operation is 1.

3.2 Evaluation and Selection

Three criteria, termed *match*, *brevity*, and *variety*, are considered for determining the fitness of a candidate. *Match* targets the main goal of finding an applicable rule that enables the derivation of $\widehat{G}_n$ under consideration of the remaining rule sequence $p_{i+1}, ..., p_{n-1}, p_n$. The other criteria have corrective purposes for improving the quality of the produced rule candidates.

Match: After G_i has been prepared, the learner tries to apply the remaining rule sequence S_2 using the graph-rewriting system. The graphs produced during derivation are compared with G_n in terms of similarity. Match is considered at maximum (1) if one of the produced graphs is isomorphic to $\widehat{G}_n$, or if $\widehat{G}_n$ is completely "included" in one of the produced graphs (subgraph isomorphism). Otherwise match is measured using a graph similarity algorithm which compares the topology and labeling of both graphs and returns a value within $[0, 1]$. The most similar graph is taken as reference for the rule candidate's *match.* This has been implemented using the *Neighbor Matching* algorithm proposed by [8].

During the GP's evolutionary search the rule candidates are allowed to contain operations that are not directly needed to achieve the goal of producing the desired design graph: First, there may be operations that are not applicable. These are simply ignored during the generation of G_i. Second, it is sufficient if the desired graph could be matched in terms of a subgraph isomorphism, i.e. superfluous operations do not prevent from reaching maximum *match.* Leaving these elements in the rule candidates is a deliberate design choice, as it increases the genetic variation in the population. Thereby, a once superfluous or inapplicable operation of one candidate actually can become useful when it is passed to an individual of a later generation. In consequence of crossover and mutation this operation (or gene) may be put into a different context where it is mandatory for derivation. Just at the very end of the GP search process all unnecessary operations are removed from the best rule candidate.

Brevity: Taking only *match* into account could lead to the undesired behavior of discovering a rule that replaces existing rules of the following rule sequence instead of enabling their use. In order to penalize such rules *brevity* is defined as the inverted sum of costs of all applicable operations. The associated costs are also used to rank different strategies for graph manipulation. For instance, consider a specific edge that can be added either directly by the candidate or by advising a following rule to do so using a NT. Here, the second strategy is preferable, since it does not compromise the concerns of that existing rule. This is reflected by the lower score of *add-non-terminal* compared to *add-edge.*

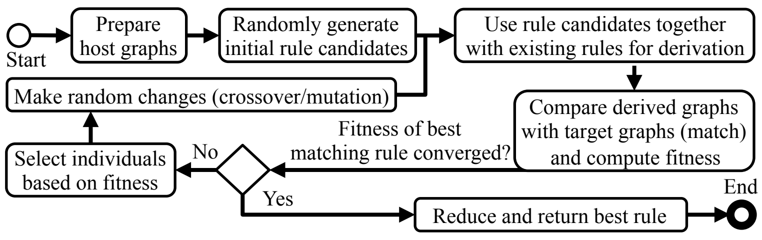
Start
Prepare host graphs
Randomly generate initial rule candidates
Use rule candidates together with existing rules for derivation
Make random changes (crossover/mutation)
Compare derived graphs with target graphs (match) and compute fitness
Fitness of best matching rule converged?
No
Select individuals based on fitness
Yes
Reduce and return best rule
End

Fig. 1. Flow chart of procedure *evolve*. Invoked by procedure *main* (see above).

Variety: This score targets special semantic properties of our application domain. Typically, a rule focuses on the conversion of some specific types of flows (edges) through functions (nodes). It would be atypical if a wide variety of flows were subject the modification by the rule candidate. Hence, *variety* is defined as the inverse number of distinct flow labels used by the rule.

Match, *brevity* and *variety* are multiplied to gain an overall fitness score for each rule candidate. These fitnesses are the basis for selecting individuals that will be used for "breeding" the next generation. Separately from the GP selection process, for each generation, individuals achieving maximum match (1) are ranked with respect to their fitnesses. If the fitness of the best candidate from this set converges, the best candidate found so far is reduced to its necessary operations and returned as result of the procedure *evolve*. The LHS and RHS of the rule are deduced from the remaining operations. Fig. 1 provides a summarizing flow chart for the *evolve* procedure.

4 Results

The hereinafter described preliminary experiments were conducted with a prototype implementation. The system uses a rule set consisting of 11 rules. These rules largely correspond to those of a graph grammar for functional engineering design, which was hand-crafted by a group of experts [10]. $[\widehat{G}_n]$ was generated by means of derivation over the complete rule set. The derivation process largely corresponds to the example described in [10].

In each experiment one rule was taken out of graph-rewriting system and the learner was given the task to learn it. The learned rule was compared to the original rule with respect to the nodes and edges each rule added to the produced graphs. Every node or edge of a desired graph in $[\widehat{G}_n]$ is classified as follows. True positive: node/edge is both added by the learned and the original rule. False positive: node/edge is only added by the learned rule. True negative: Neither learned nor original rule add the node/edge. False negative: node/edge is only added by the original rule. Accuracy (ACC) and the F1 score – two common measures for classifier evaluation – have been employed to measure the similarity of learned rules with original rules. Accuracy denotes the relative amount of correctly assigned nodes/edges. F1 puts a stronger bias on the elements that

Table 1. Comparison of Subdue with proposed GP-based approach. Rules are listed in order of application. Non-zero standard deviations are shown in parentheses.

	Subdue				Proposed GP Approach					
Rule	ACC	PRC	RCL	F1	ACC	PRC	RCL	F1	GEN	MIN
4	.62	.15	1	.27	1	1	1	1	5.36(1.63)	4.54(1.29)
29	.79	.25	1	.40	1	1	1	1	5.11(1.09)	19.67(3.40)
24	1	1	1	1	1	1	1	1	4.72(0.88)	20.38(3.83)
27	.79	.25	1	.40	1	1	1	1	4.60(0.91)	20.40(3.24)
5	.38	.31	1	.47	.97(.03)	.92(.08)	.92(.08)	.92(.08)	6.15(3.11)	7.35(2.89)
25	.94(.03)	.60(.22)	1	.73(.15)	1	1	1	1	4.46(0.81)	7.63(1.74)
20	.79	.25	1	.40	1	1	1	1	4.65(0.8)	5.74(1.36)
3	.76	.13	1	.22	.99(.02)	.80(.25)	1	.87(.16)	6.02(4.57)	8.83(10.29)
6	.76	.13	1	.22	1	1	1	1	4.52(0.65)	7.02(1.56)
17	.79	.20	.33	.25	.89(.06)	.84(.09)	.84(.09)	.84(.09)	5.04(1.53)	10.44(9.12)
33	.86	.20	1	.33	.90(.06)	.42(.36)	1	.52(.30)	4.00	1.12(0.13)

should not have been part of the learned rule. It is the harmonic mean of precision (PRC) and recall (RCL). PRC: How many added nodes/edges of the learned rule are also part of the original rule? RCL: How many added nodes/edges of the original rule are also added by the learned rule?

As a baseline, another rule induction algorithm based on the Subdue method [2] was given the task to learn the complete rule set from scratch. Subdue iteratively adds one rule at a time. With each iteration, it searches for subgraphs that frequently appear in the graphs of $[\widehat{G}_n]$ and forms rules from those subgraphs. Subdue generated a total of 16 rules, which were assigned to original rules with respect to maximum ACC.

Table 1 shows the mean ACC, PRC, RCL, F1 score for the learned rules. Since Subdue is a deterministic method, it is only executed once, and its mean scores are computed over the graphs of $[\widehat{G}_n]$. The proposed GP method is non-deterministic, thus every test run, i.e. the execution of the *main* procedure, has been repeated 25 times. The mean has then been computed over every final rule candidate of *evolve*. GEN and MIN denote the mean number of generations and minutes needed by *evolve* to converge (tested on a PC with 2.5 GHz 4-core CPU and 16 GB RAM).

The rules learned by the GP approach generally show a high similarity with the original ones. As expected they most often outperform Subdue's rules, as the rules used in the preceding sequence S_1 are kept and the learner tries to avoid replacing rules of S_2. Subdue's strategy is to make the rules' RHSs as large as possible. Hence, parts of multiple original rules are covered by the induced rule resulting in a PRC-drop.

A drawback of the GP method is that the trial-and-error evaluation affords significantly more computational resources than the Subdue method. On the test machine (see above), Subdue computes the new ruleset within seconds. Depending on the complexity of the rule and its position in the rule sequence, an execution of *evolve* takes several minutes. Further, in its current form the GP method may learn rules that produce graphs which are only subgraph isomorphic to the target graphs $[\widehat{G}_n]$.

5 Conclusion

The results show the principle feasibility of inducing rules by means of GP in context of an existing rule set. Further, it has been shown that the proposed method is applicable to a concrete example from the field of engineering design. It is capable of learning rules that are considerably similar to those of an existing graph grammar for functional decomposition. In comparison with a state-of-the-art rule induction algorithm that learned the complete ruleset from scratch, the proposed method achieves a higher similarity with the original rule definitions.

References

1. Bartsch-Spörl, B.: Grammatical Inference of Graph Grammars for Syntactic Pattern Recognition. In: Ehrig, H., Nagl, M., Rozenberg, G. (eds.) Graph Gramars and Their Application to Computer Science, 4th Intl. Workshop, Haus Ohrbeck, Germany, October 48, 1982. pp. 1–7. LNCS, Springer, Berlin/Heidelberg (1983)
2. Cook, D.J., Holder, L.B.: Substructure Discovery Using Minimum Description Length and Background Knowledge. J. Artif. Intell. Res. 1(1), 231–255 (1994)
3. Costa, F., Sorescu, D.: The Constructive Learning Problem : an efficient approach for hypergraphs. In: Constructive Machine Learning, Workshop at the 2013 Conf. on Neural Information Processing Systems (NIPS'13), Lake Tahoe, NV, USA, December 10, 2013. pp. 1–5 (2013)
4. Fürst, L., Mernik, M., Mahnič, V.: Graph Grammar Induction as a Parser-Controlled Heuristic Search Process. In: Schürr, A., Varró, D., Varró, G. (eds.) Applications of Graph Transformations with Industrial Relevance, 4th Intl. Symposium (AGTIVE'11), Budapest, Hungary, October 4-7, 2011, Revised Selected and Invited Papers. pp. 121–136. LNCS, Springer, Berlin/Heidelberg (2012)
5. Inokuchi, A., Washio, T., Motoda, H.: An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. In: Zighed, D.A., Komorowski, J., Zytkow, J. (eds.) Principles of Data Mining and Knowledge Discovery, 4th European Conf. (PKDD'00), Lyon, France, September 1316, 2000. pp. 13–23. LNCS, Springer, Berlin/Heidelberg (2000)
6. Jeltsch, E., Kreowski, H.J.: Grammatical lnference Based on Hyperedge Replacement. In: Ehrig, H., Kreowski, H.J., Rozenberg, G. (eds.) Graph Gramars and Their Application to Computer Science, 4th Intl. Workshop, Bremen, Germany, March 59, 1990. pp. 461–474. LNCS, Springer, Berlin/Heidelberg (1991)
7. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
8. Nikolić, M.: Measuring similarity of graph nodes by neighbor matching. Intell. Data Anal. 16(6), 865–878 (2012)
9. Sammut, C.: Learning as Search. In: Sammut, C., Webb, G.I. (eds.) Encyclopedia of Machine Learning, pp. 572–576. Springer, New York (2010)
10. Sridharan, P., Campbell, M.I.: A Grammar for Function Structures. In: Proc. of the ASME 2004 Intl. Design Engineering Technical Conf. and Computers and Information in Engineering Conf. (IDETC/CIE'04), Salt Lake City, UT, USA, September 28-October 2, 2004. vol. 3a, pp. 41–55 (2004)
11. Wyard, P.: Context Free Grammar Induction Using Genetic Algorithms. In: Proc. of the IEE Colloquium on Grammatical Inference: Theory, Applications and Alternatives, Colchester, UK, April 22-23, 1993. pp. P11/1–P11/5. IET (1993)