October 2020

# Efficient Extraction of n-grams From a Grammar

Alexandre Boulgakov



# Efficient Extraction of n-grams From a Grammar

ABSTRACT

N-grams are a technique used in document processing to summarize the content of a document as a set of text fragments that it contains. N-grams are used for document processing across a wide range of applications such as indexing, clustering, and machine learning. This disclosure describes techniques to efficiently extract n-grams of a given length from a grammar, specified as a nondeterministic finite automaton (NFA) with ε-moves. The algorithm described here uses O(N) graph traversals to compute n-grams of length N from a grammar.

KEYWORDS

- N-grams
- Document summarization
- Breadth first search
- Non-deterministic finite automaton (NFA)
- Malicious code detection

## BACKGROUND

N-grams are a technique used in document processing to summarize the content of a document as a set of text fragments that it contains. N-grams are used for document processing across a wide range of applications such as indexing, clustering, and machine learning. N-grams can also be applied to domains that have document-like content, generalizable to a grammar that generates arbitrary sets of valid documents. Some examples of such domains include malware detection, formal verification, generation of execution traces of a software application, and interactive fiction. For example, given a loop in an application (e.g., a software program), a

grammar can succinctly represent a set of execution traces with an arbitrary number of iterations of the loop.

DESCRIPTION

Techniques for document processing and handling document-like content can use a grammar as a data point in domains where n-grams are used. Knowledge of the complete list of n-grams allowed by the grammar is a prerequisite for using grammars as data points in existing document-handling algorithms that use n-grams.

A finite grammar can generate an arbitrary number of valid documents, but only a finite number of n-grams of a given length. For a grammar with S symbols, the number of n-grams of length N would be at most $S^N$. To generate all valid n-grams from a grammar, the trivial approach of generating all the valid documents and extracting the n-grams from each is infeasible, since the number of valid documents is arbitrarily large and potentially infinite. Efficient extraction of n-grams of a given length from a grammar specified as a nondeterministic finite automaton (NFA) with epsilon-moves (or equivalently, graphs with optional edge labels) is an important technical problem.

This disclosure describes a technique to efficiently extract n-grams of a given length from a grammar, specified as a nondeterministic finite automaton (NFA) with ε-moves. The algorithm described here uses O(N) graph traversals to compute n-grams of length N from a graph with optional edge labels (equivalently, a nondeterministic finite automaton (NFA) with ε-moves, with ε denoting missing edge labels).

A recursive algorithm as described herein can be used to efficiently annotate each node with its incoming n-grams using dynamic programming and O(n) graph traversals.

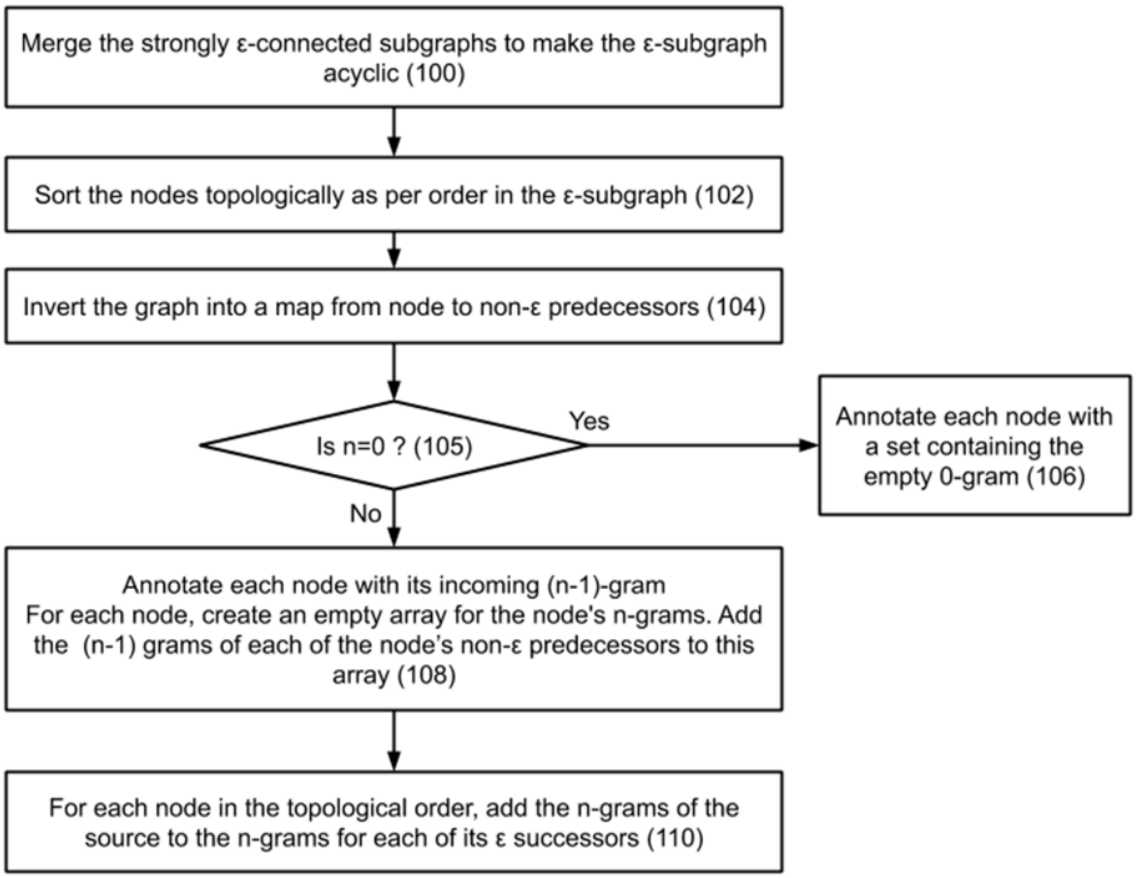


**Fig. 1: Extracting n-grams from a grammar**

Referring to Fig. 1, the first step of the algorithm is merging the strongly-ε-connected components to make an ε-subgraph acyclic (100). The next step is topologically sorting the nodes by their order in the ε-subgraph (102) followed by inverting the graph into a map from node to its non-ε predecessors (104). In this step, given a set of edges such as (source, destination, label), a map is created from destination->(source, label).

For the recursive base case (105), when n = 0, each node is annotated with a set containing the empty 0-gram (106). Else, each node is annotated with its incoming (n-1)-grams. For each destination node, an empty array is first created for its n-grams. The (n-1) grams of each of the node's non-ε predecessors are added to this array and the destination node is

annotated with this list (108). In the next step, each source node in the topological sort (upstream to downstream), for each of the ε successors (destination nodes), the array holding n-grams of the source node is appended to the n-grams of each successor (110).

In the above, steps 100-104 are preparatory steps, while the remaining procedure is recursive. The recursion works by annotating each node, extending n-grams by 1, and propagating the n-grams through ε-transitions. In the extending n-grams step, each n-gram (which includes n edge labels, which may be long) is stored in a table. This saves space when multiple nodes have the same n-grams and also eliminates the requirement for a final pass through the graph to collect the annotations for each node.

The described techniques can be used in many application domains, e.g., to detect malware or unwanted software based on application programming interface (API) call graphs that are extracted from native code by static analysis. Efficient queries of such graphs, e.g., for manual code analysis, clustering, or for code signature matching, can be supported by creating an index of n-grams extracted from the API call graphs. The techniques can also be used to detect code similarity to detect repackaging (where proprietary applications are modified and marketed by a third party), to detect software development kits (SDKs) present in an application, to find malicious code indicators in large applications, etc. n-grams can also be used to detect click fraud and trojans.

The use of grammars/graphs allows efficient evaluation of all potential executions of an application, uniformly across static and dynamic analysis. The use of n-grams obviates custom solutions that can be difficult to develop and maintain, and are harder to automate and scale.

## CONCLUSION

This disclosure describes techniques to efficiently extract n-grams of a given length from a grammar, specified as a nondeterministic finite automaton (NFA) with ε-moves. The algorithm described here uses O(N) graph traversals to compute n-grams of length N from a grammar.

## REFERENCES

1. Gibson-Robinson, Thomas, Philip Armstrong, Alexandre Boulgakov, and A. William Roscoe. "FDR3: a parallel refinement checker for CSP." *International Journal on Software Tools for Technology Transfer* 18, no. 2 (2016): 149-167.
2. Boulgakov, Alexandre, "Matching a graph with a non-deterministic finite automaton", Technical Disclosure Commons, (June 14, 2018) https://www.tdcommons.org/dpubs_series/1246
3. Boulgakov, Alexandre and Ren, Chuangang, "A Machine-Learned Model To Detect Malicious Code Using API-call N-grams From Static Analysis", Technical Disclosure Commons, (September 10, 2020) https://www.tdcommons.org/dpubs_series/3590