Available online at www.sciencedirect.com

ScienceDirect

IERI Procedia 10 (2014) 119 – 130

Procedia IERI

www.elsevier.com/locate/procedia

2014 International Conference on Future Information Engineering

# Scripting Language for Java Source Code Recognition

Tomáš Bublík*, Miroslav Virius

*Faculty of Nuclear Sciences and Physical Engineering Czech Technical University in Prague, Trojanova 13, Prague, 120 00, Czech Republic*

**Abstract**

This paper presents general results on the Java source code snippet detection problem. We propose the tool which uses graph and subgraph isomorphism detection. A number of solutions for all of these tasks have been proposed in the literature. However, although that all these solutions are really fast, they compare just the constant static trees. Our solution offers to enter an input sample dynamically with the Scripthon language while preserving an acceptable speed. We used several optimizations to achieve very low number of comparisons during the matching algorithm.

*Keywords:* AST; Java; tree matching; Scriphon; source code recognition

## 1. Introduction

It is usual that programs, consisting of a large source code, are becoming chaotic, and many times described illnesses start to appear (code duplicity, weak reusability, etc.). Maintaining a source code is a serious issue. There are a lot of tools and guides on how to approach this issue. The tool, described in this work, serves to programmable Java source code scanning. This tool is based on the Scripthon language which was developed for these purposes in the scope of this work. A script, which describes a source code structure and its properties, can be written in this language. This allows defining dynamic properties of searching

* Corresponding author. Tel.: +420605155484.
*E-mail address:* tomas.bublik@gmail.com.

doi:10.1016/j.ieri.2014.09.100

requirements. Next, an abstract syntax tree (hereinafter AST) is created dynamically from this script. Meanwhile, a similar tree is created from given Java source, and these two trees are matched by a graph matching algorithm. However, not only graph shapes are compared, also trees properties are considered. To obtain results faster, several graph optimizations are used during this process. And it is possible to scan higher amount of source code classes during a relatively short time. Typical usage of this approach is as follows: A user wants to find something, not easily describable, in a large program. But he or she knows that it could be there. He or she can use our tool and try to find at least the similar snippet of indented code. Therefore, a user performs a searching procedure, and specifies the input script based on the received results. By repeating this procedure, he or she filters the unintended results, and finally gets the desired code snippet. Our tool is useful not just for the fast Java sources scanning, but for example, for a better definability of search conditions. Another usage area of our tool can be a clone's detection problem. By using other clones detection tools, a material for further research can be gathered. For example, the non-ideal clones are difficult to detect and the output of such programs isn't unequivocal in many cases. However, it can be classified by Sripthon, and a common searching script based on such output can be.

## 2. Theory

A graph is an ordered pair $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ where $\boldsymbol{V}$ is a finite, non-empty set of objects called vertices, and $\boldsymbol{E}$ is a (possibly empty) set of unordered pairs of distinct vertices i.e., 2-subsets of $\boldsymbol{V}$ called edges. The set $\boldsymbol{V}$ is called the vertex set of $\boldsymbol{G}$, and $\boldsymbol{E}$ is called the edge set of $\boldsymbol{G}$. If $\boldsymbol{e} = \{\boldsymbol{u}, \boldsymbol{v}\} \in \boldsymbol{E}(\boldsymbol{G})$, we say that vertices $\boldsymbol{u}$ and $\boldsymbol{v}$ are adjacent in $\boldsymbol{G}$, and that $\boldsymbol{e}$ ***joins*** $\boldsymbol{u}$ and $\boldsymbol{v}$. We'll also say that $\boldsymbol{u}$ and $\boldsymbol{v}$ are the ***ends*** of $\boldsymbol{e}$. The edge $\boldsymbol{e}$ is said to be ***incident*** with $\boldsymbol{u}$ (and $\boldsymbol{v}$), and vice versa. We write $\boldsymbol{uv}$ (or $\boldsymbol{vu}$) to denote the edge $\{\boldsymbol{u}, \boldsymbol{v}\}$, on the understanding that no order is implied. Two graphs are ***equal*** if they have the same vertex set and the same edge set. But there are other ways in which two graphs could be regarded the same. For example, one could regard two graph as being "the same" if it is possible to rename the vertices of one and obtain the other. Such graphs are identical in every respect except for the names of the vertices. In this case, we call the graphs ***isomorphic***. Formally, graphs $\boldsymbol{G}$ and $\boldsymbol{H}$ are isomorphic if there is a $\boldsymbol{1-1}$ correspondence $\boldsymbol{f}: \boldsymbol{V}(\boldsymbol{G}) \rightarrow \boldsymbol{V}(\boldsymbol{H})$ such that $\boldsymbol{xy} \in \boldsymbol{E}(\boldsymbol{G}) \leftrightarrow \boldsymbol{f}(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{y}) \in \boldsymbol{E}(\boldsymbol{H})$. This function $\boldsymbol{f}$ is called an isomorphism.

A tree is a connected graph that has no cycles (i.e., a connected acyclic graph).

The graph matching problem is actually the same as the problem of finding the isomorphism between the graphs. Moreover, matching the parts of a graph with a pattern is the same challenge as the finding the isomorphic subgraph. There are many approaches to this topic in Irniger et al., 2005.

Subgraph isomorphism is useful to find out if a given object is part of another object or even of a collection of several objects. The maximum common subgraph of two graphs $\boldsymbol{g_1}$ and $\boldsymbol{g_2}$ is the largest graph that is isomorphic to a subgraph of both $\boldsymbol{g_1}$ and $\boldsymbol{g_2}$. Maximum common subgraph is useful to measure the similarity of two objects. Algorithms for graph isomorphism, subgraph isomorphism and maximum common subgraph detection have been reported in McKay, 1981, Ullmann, 1976, Levi, 1972, McGregor, 1982..

A more general method to measure the similarity of two graphs is graph edit distance. It is a generalization of string edit distance, also known as Levenshtein distance Stephen 1994.

Another approach measuring the similarity of two graphs is a distance measure based on the maximum common subgraph between $\boldsymbol{g_1}$ and $\boldsymbol{g_2}$. With increasing work being done in the field of maximum common subgraph detection, these measures are growing in popularity. In Bunke et al., 1998, a graph distance measure based on the maximum common subgraph of two graphs is introduced.

It is shown that the well-known concept of maximum common subgraph distance is a special case of graph edit distance under particular edit costs. Consequently, algorithms originally developed for maximum common subgraph detection can be used for edit distance computation and vice versa for the considered edit