Masaryk University
Faculty of Informatics

Similarity of programming problems

Bachelor's Thesis

Dominik Gmiterko

Brno, Spring 2018

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Dominik Gmiterko

Advisor: Radek Pelánek

Acknowledgements

These are the acknowledgements for my thesis, which can span multiple paragraphs.

Abstract

This is the abstract of my thesis, which can span multiple paragraphs.

Keywords

similarity, metrics, programming, keyword2, …

Contents

VB000

Predbežné zadanie:
Goal of this thesis is to define metrics for measuring similarity of programming problems for single task (Robotanik) from Problem Solving Tutor (tutor.fi.muni.cz). Specific goals:

- Provide an overview of relevant related research.
- Propose metrics for measuring similarity of programming problems, analyze differences between them.
- Verify possibility of using similarity in management of large problem pool.

Predpokládaný vedúci: Radek Pelánek

Generovanie bibliografickych zaznamov ponechavam na BibLaTeXu.

Introduction

Tutoring systems are computer-based systems designed to introduce users into various domains. They usually have large amount of items which enables them to provide personalized experience. To maintain this large pool of items efficiently we need to be able to decide which items are useful and which are not. Also large portion of tutoring systems are focusing on teaching elementary programming. Learning of programming is a complex activity. This makes it important to choose best problems for users to solve. This are reason why it is useful to be able to measure similarity of items in tutoring systems as similarity of items can help us improve tutoring system.

Goal of this thesis is to define metrics for measuring similarity of programming problems for introductory programming. Results of this first part can be then used to determine clusters of similar problems and provide visualization that show structure of problem pool.

When developing metrics we used data collected from students who solved problems in Robotanik (single task from Problem Solving Tutor developed at Faculty of Informatics at Masaryk University).

Besides Introduction and Conclusion chapters, this thesis is structured into three additional chapters. First chapter talks in general about problem of measuring similarity of programming problems. It explains difference between program and programming problem, which data we have available and techniques used for measuring similarity of problems. Second chapter advances level deeper and describe everything what is specific to data we used. First part of chapter describes programming environment of Robotanik and data from it. Second part focuses in detail on metrics we used in experiments. Last chapter gives overview of implementation and usage of metrics and their evaluation.

1 Similarity

In this chapter we will talk in general about similarity and where it is used in related fields. Then we will define context of programming environments and problems. Which allows us talk about possible usage of results of this work. Most of the chapter focuses on explaining what kinds of data are available when comparing programming problems in different programming environments and techniques to do so. Last section describes goals of the thesis.

A lot of research has been dedicated to similarity in many different fields computer science like bioinformatics (sequence alignment, similarity matrix of proteins), information retrieval (document similarity), plagiarism detection and many more. On other hand currently there is almost no research focused directly at comparing programming problems.

One closely related area is recommender systems which differs from problem similarity only slightly. Both areas are distinguishing users and items. Only difference is that we know how well user did while solving specific item and recommender systems use rating of the items.

Another related areas make use of measuring code similarity. For example it can be used in plagiarism detection [1] and analysis of source code quality (detection of redundant functions). Main difference is that we can use more data about problem. We also have some problem statement and data about performance of students when solving problem.

1.1 Programming environments

In this section we would like to define what programming environment and programming problems are for us.

When we talk about programming problems we mean problems created in interactive programming environments. Programming environments are tutoring systems dedicated to teaching introductory programming. Some examples of programming environments are Robotanik[1], Karel[2], Lightbot.

Problems in this environments are specified by board divided to grid which is filled with tiles where each type of tile has different meaning. Users are supposed to write program for some entity to fulfill given task. Most common task is to visit all „goal" tiles. In Robotanik student are asked to build program guiding robot to collect all flowers.

Some programming environments try to constrain student in order to produce more creative solutions. One possible limitation is allowing student to use only subset of all available commands. Another approach is to limit length of program. Limiting length prevents student from manually placing each step into program and forces him to think of more general solution using loops or recursion.

1.2 Possible usage

After defining metrics for measuring similarity of programming problems we can use them for different purposes in programming environments. First, most direct, usage is recommendation of problems for student to solve. We do not want to recommend very similar problems to those that were solved without any problems. However when student struggled system should recommend more of similar problems.

Another possible usage is generating hints by selecting similar example from database of example source codes. Similarity of user solutions was used by Hosseini; Brusilovsky [2] for this purpose.

Previous use cases were using problem similarity automatically inside tutoring system. Another approach is to bringing human beings into the decision-making loop [3]. This approach provides authors of tutoring system with visualizations which should inform them what changes may be useful.

Authors of systems can use this metrics for gaining insight of problem pool. It is possible to detect redundant problems, even tell which problems are missing. One way of achieving this is plotting problems to plane and displaying it to author. Large amount of problems close

1. `<https://tutor.fi.muni.cz/>`
2. `<http://stanford.edu/~cpiech/karel/ide.html>`

to each other suggests there is lot of similar problems. When there is some problem standing alone it suggests that author of system may want to add more of similar problems.

Last idea we are quite interested in is automatic construction of user interface. When we have large amount of problems without any present categorization we can use problem similarity to construct categories for student to select from. Even when system already has problems categorized author can use our metrics to verify that groups are formed correctly and refine them. [4]

1.3 Metric types

When comparing two items there is a lot of choices to make. We can choose different information about items and techniques of measuring similarity. In first part we will describe data which are usually available in programming environments. This data can be then processed using multiple techniques. We will describe them in second part.

1.3.1 Data

We can define different metrics for comparing programming problems. It is possible to divide metrics into 4 categories based on data used.

Problem statement in interactive programming environments can be used to collect features used in problem. Similarity is mostly computed using distance of vectors describing some features of problem. This category includes usage of tiles in problem, allowed commands, size of level, and more.

Manual labeling provides us with information not directly related to problem statement but specific for each problem. Some examples of this data are division of problems to categories or levels, difficulty of problem. We are not interested in using this kind of data alone, but they can by used to aid in some other metric.

Solution based metrics differ greatly with method used by users to input solutions. We can divide them to textual and visual programming languages. Commonly used programming languages like Python or Java are text-based. However some programming environments for the sake of simplicity use alternative approach. E.g. Robotanik uses commands (blocks) that can be placed in rows representing functions. Programming enthronements Robomise and Scratch use Blockly[3] (like) user interface where user builds trees from predefined nested blocks.

When dealing with solutions we usually have more than one example solution. This means we may want to somehow extended metrics to use solutions from all students their attempts. Simplest way is to average of calculated features. We will talk more about computing features in next section.

Performance based metrics requires having collected data about students while solving problems. Example of data that falls in this category is solution time, number of attempts, hints used. Metrics using performance data are widely applicable to many systems [4]

Combined metrics may produce more accurate results with smaller amount of data. But it is harder to construct and evaluate them. We will talk more about them in later chapters.

1.3.2 Similarity techniques

Following section will discuss different techniques used to compute similarity of two problems using data described in previous section.

Text/token-based techniques can be applied to data describing problem statement and solution. We can use different techniques known from information retrieval. Sometimes it is required to modify this standard techniques when using with data from visual programming environments.

One of simplest methods is using bag-of-words. In our problems we can either count occurrences of each tile type used in problem statement or commands used in solution. Many of text-based techniques can be applied to multiple sources of data. In either case we will end up with vector of features describing data. This means we have to apply additional step to retrieve similarity of two problems. Applying some standard similarity metric like Euclidean similarity or Cosine similarity results in final problems similarity.

3. `<https://developers.google.com/blockly/>`

Edit distances can also be applied to different kind of data. When analyzing text we can use Levenshtein distance which can be used to directly compare two problems.

Also techniques mostly used in plagiarism detection like w-shingling and winnowing [5] may be used.

There are many other more complex information retrieval and natural language processing techniques. However wont be using them in this work because solutions in Robotanik are really short and we think more complex techniques can't give us any significant benefit.

Abstract nature of source code limits usefulness of text-based processing techniques. Most programming languages allow to write many ambiguous programs which are doing same thing. Most common reasons are identifier renaming, line reordering, variable whitespace count. This is reason to implement more abstract techniques using different aspects of programs. We will list some of them in the rest of this section.

Many similarity techniques are based on edit distance. Similarity is then typically specified as $1/(1+d)$ where d is edit distance. Where edit distance is minimum number of operations necessary to transform one instance of structure to another.

Tree-based techniques are often used in measuring source code similarity using Abstract syntax trees (AST). Similarity of two trees is computed as their tree edit distance.

Graph-based techniques may also use edit distance or some other metrics describing graphs. For solutions we can generate program dependence graphs (PDG) [6], control flow graphs (CFG).

Correlation-based – for some types of data (e.g. user solving time) there is no need to preprocess them and some similarity metric like euclidean similarity can be used directly.

1.4 Goal

In the rest of the thesis we will try to answer several questions:

- What types of metric can be used on data available from tutoring system Robotanik.

- Do different metrics correlate? Do they measure different aspects of problem similarity or is there one underlying common similarity? If there are multiple aspects, what is the relation between them? How do we use them in practical applications?
- Implement simple tool which will be able to ease measuring similarity of grid based programming problems.

A An appendix

Here you can insert the appendices of your thesis.

Bibliography

1. BETH, Bradley. *A Comparison of Similarity Techniques for Detecting Source Code Plagiarism*. 2014.
2. HOSSEINI, Roya; BRUSILOVSKY, Peter. A study of concept-based similarity approaches for recommending program examples. *New Review of Hypermedia and Multimedia*. 2017, pp. 1–28.
3. BAKER, Ryan S. Stupid tutoring systems, intelligent humans. *International Journal of Artificial Intelligence in Education*. 2016, vol. 26, no. 2, pp. 600–614.
4. PELÁNEK, Radek; ŘIHÁK, Jiří. Measuring Similarity of Educational Items Using Data on Learners' Performance. 2017.
5. SCHLEIMER, Saul; WILKERSON, Daniel S; AIKEN, Alex. Winnowing: local algorithms for document fingerprinting. In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 2003, pp. 76–85.
6. WANG, Tiantian; WANG, Kechao; SU, Xiaohong; MA, Peijun. Detection of semantically similar code. *Frontiers of Computer Science*. 2014, vol. 8, no. 6, pp. 996–1011.