Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)

Discovering Algorithmic Relationship Between Programming Resources on the Web

Guojin Zhu*, Kai Zhang and Jiyun Li
School of Computer Science and Technology
Donghua University
Shanghai, China
gjzhu.dhu@163.com, kaiyou1989@163.com and jyli@dhu.edu.cn

***Abstract*—Algorithmic relationships are discovered here for programming tutoring. There are two kinds of algorithmic relationships between programming resources on the web: associative relationship and structural similarity relationship. They can be organized as a hierarchical body. An algorithm can solve different programming problems and a programming problem also can be solved by different algorithms. Thus, there is such algorithmic relationship, or associative relationship, between these programming resources on the web. The algorithmic structures of source codes can be mined by neural computing. Different source codes may have a structural similarity relationship between them, meaning that they are similar in their algorithmic structures. A learner can learn algorithms from simple to complicated structures or from similarities in their structures. In our experiment, we use a tree structure to organize the algorithmic relationships.**

Keywords-algorithmic relationship; lobe component analysis; formal concept analysis; programming tutoring; nueral computing

I. INTRODUCTION

There are lots of programming problems and their source codes on the web. Thousands of algorithmic methods are hid in these source codes. However, it is difficult to provide the suitable algorithmic methods for programming learners. Consequently, Intelligent Computer-assisted Instruction (ICAI) has been arisen [1, 2]. It is suitable to deal with this issue by mining the algorithmic relationships in the source codes and organizing the algorithmic relationships into a reasonable structure. Our previous work has organized the programming knowledge on a basis of a predefined hierarchical body [3].

On one hand, a problem usually can be solved by several different algorithms. To learn different algorithms to solve one problem can expand learners' thinking. On the other hand, an algorithm can solve different problems. For example, a teacher who has just taught students an algorithm by solving one problem can assign the students with other problems that can be solved by the same algorithm. The students can understand the algorithm better by solving the assigned problems. This explains that each pair of these problems has such an associative relationship between them.

Different source codes may have a structural similarity relationship between them, meaning that they are similar in their algorithmic structures. A learner can learn a simple algorithmic structure before he learns a complicated one. A learner who has learnt an algorithmic structure can learn other similar algorithmic structures easily. It is easy to learn algorithmic structures by organizing the structural similarity relationships into a relationship hierarchy.

Lobe Component Analysis (LCA) is a theory that meets the Autonomous Mental Development (AMD) [4]. Our previous work has applied LCA to algorithm recognition [5]. Its vectors are tree-edit distances from which we cannot read algorithmic structures. We convert algorithmic structures into vectors to develop algorithmic templates by LCA. We can get structure patterns from algorithmic structures which are extracted from algorithmic templates. The developed algorithmic templates are used to recognize which algorithm hid in each source code. We proposed a hierarchical model to organize associative relationships and structural similarity relationships. Our other previous work has applied Formal Concept Analysis (FCA) to discovery mainstream knowledge in source codes [6]. It has proved that FCA is an efficient tool to mine the knowledge of source codes which are used to solve problems. Hence, we use it to generate the concept lattice to order the algorithms in the first three layers of the hierarchical body.

This paper proposed a way to mine algorithmic relationships from programming resources on the web. We employ the algorithmic relationships to organize the programming resources into a hierarchical body.

II. TERMINOLOGY

A. *Control Structures*

A control structure is to specify what has to be done by the program. Each control structure is a block. There are four kinds of control structures in C++: the *while* loop, the *for* loop, the conditional structure *if* and the selective structure *switch*. In Fig. 1, there is a control structure *while*. All source codes in the block *while* will execute circularly until the condition becomes *false*.

B. *Language Points*

A language point is the language knowledge of source codes. The relevant language points are language points that belong to one control structure. The left side of Fig. 1 is a source code. It has a control structure *while* and its relevant language points are *input*, *not equal to*, etc..

C. *Matrix for Source Code*

---

*Corresponding Author.

Published by Atlantis Press, Paris, France.

```
#include<stdio.h>
int main()
{
    int N;
    while(scanf("%d",&N)!=EOF&
        &N>=1&&N<10000)
          printf("%d\n",(1+N)*N/2);
    return 0;
}
```

$$\begin{bmatrix} 80 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 1. A source code and its matrix.

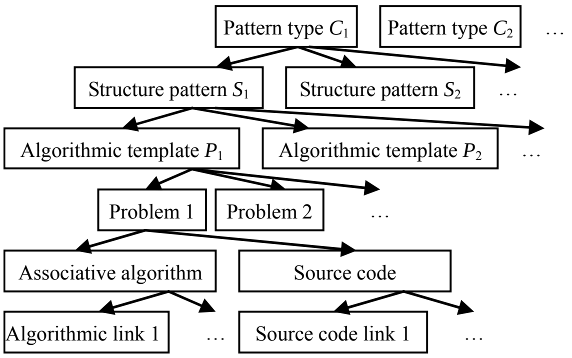

Figure 2. The hierarchical model.

A source code can be converted into a matrix through its control structures and language points. The right side of Fig. 1 is a matrix for the left source code.

D. Algorithmic Templates

An algorithmic template represents the common program plan for many source codes. An algorithmic structure can be extracted from its algorithmic template in this paper. Every algorithmic template can be represented by a matrix like the right side of Fig. 1.

E. Structure Patterns

A program may have more than one control structures, but has one structure pattern only. All control structures in one program form a structure pattern. A structure pattern can be extracted from an algorithmic structure. A structure pattern is denoted by a combination of control statements, pairs of parentheses "(" and ")". The structure pattern of the source code in Fig. 1 is denoted by *while*(). Control structures have two kinds of relations: nesting relation and sequence relation. For example, the structure pattern *while*(*if*()*if*()) has the two conditional structures, each denoted by *if*(), which have a sequence relation with each other and have nesting relations with the *while* loop.

F. Pattern Types

A pattern type is a category of structure patterns, denoted by a combination of control structures, pairs of parentheses "(" and ")", the symbol "*" and the symbol "+". The symbol "*" means that there are zero or more structure patterns. The symbol "+" means there are one or more structure patterns. For example, a pattern type *while*(*if*(*)+) represents a group of structure patterns which include *while*(*if*()*if*()), *while*(*if*()*if*()*if*()), *while*(*if*()*for*()*for*(*if*())), etc..

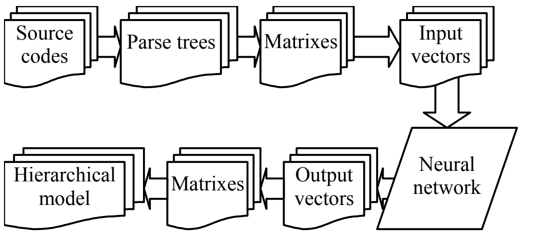

Figure 3. The nine steps.

III. THE HIERARCHICAL MODEL

We organize algorithmic relationships as a hierarchical model. The hierarchical body has six layers. The first layer C consists of pattern types. The second layer S consists of structure patterns. The third layer P consists of algorithmic templates. The fourth layer consists of problems. The fifth layer has two directories: associative algorithms and source codes. The sixth layer consists of associative algorithms and source codes. Associative algorithms are other algorithms to solve the same problem. There are source code links under the source code. For a triple $<C, S, P>$, we can search the algorithmic template by $C\text{->}S\text{->}P$. Then we can search its problems and then source codes and the associative algorithmic templates of the problems. The Fig. 2 shows the hierarchical model.

IV. METHODOLOGY

A. Discovery Procedure

There are nine steps to obtain algorithmic relationships. An adult algorithmic template P is an algorithmic template whose age is greater than or equal to a threshold. The discovery procedure is shown as follows.

1) Convert the source codes E into the depth-first traversals T of their parse trees.
2) Convert the depth-first traversals T into matrixes M_1.
3) Convert the matrixes M_1 into vectors N_1 column by column.
4) Develop every algorithmic template P from the vectors N_1 in a one-layer neural network which are based on LCA.
5) Get every one of developed vectors N_2 from every adult algorithmic template P.
6) Get developed matrixes M_2 from the developed vectors N_2 column by column.
7) Get every structure pattern S from the developed matrixes M_2.
8) Get associative relationships for algorithms through recognize relationships of algorithms and problems.
9) Organize algorithms by hierarchical body through every triple $<C, S, P>$ and associative relationships.

Fig. 3 shows the process to discovery algorithmic relationships between programming resources on the web. The left side in Fig. 1 is a source code to solve a problem that summarizes 1 to N. The right side in Fig. 1 is the matrix of matrixes M_1 for the left source code. Each row in the matrix is a control unit which consists of control structures and their relevant language points. We use controlling

```
#include<iostream>
using namespace std;
int main(void){
    int a,i,sum=0;
    while(cin>>a){
        for(i=1;i<=a;i++)
            sum+=i;
        cout<<sum<<endl;
        sum=0;
    }
    return 0;
}
```

```
#include<iostream>
using namespace std;
int main(){
    long long int a,b;
    while((cin>>a>>b)&&(a>
            =0&&a<10000)&&(
            b>=0&&b<10000))
            cout<<a*b<<endl;
    return 0;
}
```

Figure 4. Two source codes.

numbers 80, 110, 190 and 220 to represent control structures and use non-controlling numbers 0 and 1 to represent nonexistence and existence of relevant language points. In each row, the controlling number occupies several positions and every non-controlling number occupies one positions. In Fig. 1, the controlling number occupies one position and the non-controlling number occupies six positions. The first number is the control number 80 which represents the *while* loop. The next six are non-controlling numbers which correspond to six language points: *greater than*, *equal to*, *less than*, *input*, *output*. The left non-controlling numbers are 1s mean there are *equal to*, *less than* and *input*. For each input of vectors N_1, the neural network activates one neuron and updates its weight and adds one to its age.

We can get every structure pattern *S* and language points from every adult algorithmic template *P* whose age is greater than or equal to a threshold. A row which contains a number greater than a threshold will be treated as a control unit in each one of developed matrixes M_2. The position of the first number which greater than the threshold reveals the relation between this unit and the upper control unit. This unit which has an indentation with a similar position of the upper control unit has a nesting relation with the upper control unit, otherwise it has a sequence relation. We can obtain the all language points according to numbers that are greater than a threshold in the matrix. We use "0" and "1" to represent a language point is nonexistence or existence. Then we get the binary representation for language points. We change the binary representation to hexadecimal representation. Thus, we get the structure pattern *S* and the language points. An algorithmic name is obtained through analyzing source codes.

B. *Discovering Algorithmic Relationship*

We get the pattern type *C* of the structure pattern *S* according its representation. In a triple <*C*, *S*, *P*>, every pattern type *C* is defined by ourselves and the algorithmic template *P* is simplified by language points and algorithmic names. The structural similarity relationship can be implemented by organizing these triples.

TABLE I. Pattern type

ID	Pattern type
1	while()
2	while(if(*))
3	while(switch(*))
4	while(while(*))
5	while(for(*))
6	while(if(*)+)
7	while(while(*)+)

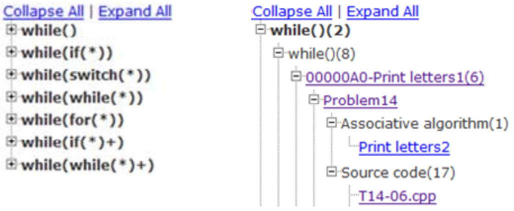

Figure 5. A part of the tree structure in the web page.

On the one hand, there are generally many algorithms to solve one problem. For example, the left side of Fig. 4 provides another algorithm to solve the problem that can be solved by the left side of Fig. 1. On the other hand, an algorithm commonly solves more than one problem. The right side of Fig. 4 shows a source code to computer a rectangle area for given width *a* and height *b*. Obviously, it has the same algorithm with left side of Fig. 1.

When recognizing which algorithm behind a source code, we should convert it into a vector as the same ways we develop algorithm templates. Thus, for source codes belong to a problem, we can active some neuron and get the algorithm that can solve the problem. The associative relationship can be implemented by last three layers in the hierarchical body. For every algorithmic template *P* that have the same pattern type *C* and structure pattern *S*, we sort them by FCA. We build the formal context by treating every algorithmic template *P* as objects and their language points as attributes. We use tool called ConExp to generate concept lattice. The sorting of them is the topological sorting of the concept lattice.

C. *Model Implementation*

We use a tree structure to implement the hierarchical model. Algorithmic names, problems and source codes have hyperlinks to link the web pages of algorithms, problems or source codes. A web page of algorithm lists all problems it can solve. It also lists the source codes after every problem. All they are hyperlinks. A web page of problem lists all algorithms can solve it and also list the source codes after every algorithm. All they are hyperlinks. A web page of problem also gives the description of the problem. The

TABLE II. The statistics for the first five algorithms

ID	Problem total	Source code total	Most source code	Threshold total
1	11	47	21	3
2	10	49	13	3
3	4	35	17	2
4	9	37	18	2
5	18	83	28	5

TABLE III. The statistics for the first five problems

ID	Algorithm total	Source code total	Most source code	Threshold total
1	14	32	7	4
2	12	30	7	3
3	5	49	40	2
4	8	48	21	3
5	8	38	17	2

hyperlinks of source codes only open source codes.

V. EXPERIMENT AND RESULT

A. Experimental Data

Our experiment data are 2341 C++ source codes that are used to solve 60 simple problems from an online judge system. All source codes in each problem are submitted by sixty college students and judged correct. The number of control statements in each source code is not greater than ten and the nesting level of control statements is not greater than six. All these source codes are filtered from 2706 source codes. It leads to 50th problem has none source codes and 51st only has one source code. All source codes are converted into vectors and each vector occurs 300 times. TABLE I is the seven pattern types. The neural network has developed 400 algorithmic templates. We choose 69 adult algorithmic templates whose ages are greater than or equal to 3000 for recognition. All algorithmic templates are numbered from 1 to 400 according to their ages from small to big.

B. Associative Relationship

TABLE II is the statistics for the first five algorithms. The problem total is the total of problems that can be solved by the corresponding algorithms. The source code total is the total of source codes that contains corresponding algorithms. The most source code is the most number of source codes for problems of algorithms. The threshold total is the number of problems whose number of source codes is more than a threshold. The threshold is the round for ten percent of source code total. The maximum source code total is 83 and the minimum source code total is 12. The maximum most source code is 40 and the minimum one is 2. We consider the threshold total. Every algorithm generally can solve more than one problem. Only the 46th algorithm just can solve zero problems. The maximum threshold total is 7. The average of threshold total is 2.609.

TABLE III shows the statistics for the first five problems. The algorithm total is the total of algorithms for each problem. The source code total is the total source codes of each problem. The most source code is the most total number for algorithms that can solve the problem. The threshold total is the algorithm total for all algorithms whose source codes are more than threshold. We choose the round for ten percent of source code total as threshold. We consider the threshold total. The maximum algorithm total for one problem is 5. The 50th problem has one algorithm and the 51st problem has zero algorithms. The minimum algorithm total is 1 except the 50th and 51st problems. The average of algorithm total is 3. The maximum source code total is 49 and the minimum source code total is 19 except the 50th and 51st problems. The maximum most source code total is 40 and the minimum most source code total is 4 except the 50th and 51st problems.

TABLE IV. The part of the tree structure

C	S	P
while()	while()	00000A0-Print letters1
		00001A0-Print letters2

C. Tree Structure

TABLE IV is the part of the tree structure. *C* is the pattern type, *S* is the structure pattern and *P* is the algorithmic template. The threshold is 50 when we convert matrixes into structure patterns. There are 8843 control statements. The average of control statements is $8843/2341 \approx 3.777$. The total control statements of 69 structure patterns is 245 and the average of them is $245/69 \approx 3.551$. The order of pattern type *C* follows the order of TABLE I. The order of structure pattern *S* is organized from small to big according the length of structure pattern. Fig. 5 is a part of the tree structure in the web page. The first level consists of pattern types. The number in the pair of parentheses shows how many structure patterns in this pattern type. They are hyperlinks for web pages of algorithmic template *P*.

VI. CONCLUSION

There are many algorithms behind source codes between programming resources on the web. We develop algorithmic templates of source codes by a neural network based on LCA. The algorithmic templates in the neural network can be used to recognize what algorithm in every source code. Associative algorithmic relationships between programming problems are discovered on a basis of the algorithmic templates. We can obtain structure patterns from the algorithmic templates. We organize the associative relationships and structural similarity relationships into a hierarchical body.

In this paper, we proposed a method to discovery algorithmic relationships that are implemented through a tree structure with hyperlinks on the web. The implementation can be used to program tutoring in the future.

ACKNOWLEDGMENT

This research is supported by the National Natural Science Foundation of China (NSFC) under Grant No. 60973121.

REFERENCES

[1] A. Kurnia, et al., "Online Judge," Computers &amp; Education, vol. 36, pp. 299-315, 2001.

[2] Tangjinjuan and Xiahongwen, "Intelligent tutoring system based on computing conceptual graphs," in 2010 3rd International Conference on Artificial Intelligence and Education, ICAIE 2010, October 29, 2010 - October 30, 2010, Hangzhou, China, 2010, pp. 60-62.

[3] G. Zhu and L. Fu, "Automatic Organization of Programming Resources on the Web," in Advances in Computer Science and Information Engineering. vol. 168, D. Jin and S. Lin, Eds., ed: Springer Berlin Heidelberg, 2012, pp. 675-681.

[4] J. Weng and M. Luciw, "Dually optimal neuronal layers: Lobe component analysis," IEEE Transactions on Autonomous Mental Development, vol. 1, pp. 68-85, 2009.

[5] G. Zhu and X. Zhu, "Autonomous mental development for algorithm recognition," in 2011 International Conference on Information Science and Technology, ICIST 2011, March 26, 2011 - March 28, 2011, Nanjing, China, 2011, pp. 339-347.

[6] G. Zhu and Z. Zhang, "Discovering mainstream knowledge in source codes of programming learners," Information Science and Technology, pp. 333-338, 2011.