Journal of Computers Vol. 31 No. 1, 2020, pp. 208-217
doi:10.3966/199115992020023101018

A Web Service Composition Model k^2-MDD-WS

Fengying Li, Xianqiang Wu, Rongsheng Dong*

Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China
lfy@guet.edu.cn, 530766583@qq.com, ccrsdong@guet.edu.cn

Received 13 January 2019; Revised 3 March 2019; Accepted 12 March 2019

Abstract. Web Service Dependency Graphs are the abstract model for the Web service composition that are facing great challenges in both increasing size and finite state search space. Compact representation and efficient operation of graph data are urgent topics, have been attracting more and more attentions. A Web service composition model based on multi-valued decision diagram, namely k^2 Multi-valued Decision Diagram of Web Service Dependency Graph (k^2-MDD-WS), is proposed to reduce the storage space. The vertices of the Web Service Dependency Graph are encoded, and the edge codes are then encoded according to the codes of the vertices; so, the k^2-MDD-WS is constructed based on the code sets of the edges. The k^2-MDD-WS can combine redundant web service nodes caused by a large number of isomorphic sub-trees existing in the k^2-tree to greatly reduce the number of nodes generated. Additionally, the operations of the Web Service Dependency Graph are transformed into Boolean operations under the k^2-MDD-WS representation to further improve the efficiency of finding a Web service composition solution that meets user needs. The experimental results indicate that compression ratio in k^2-MDD-WS is only 3.5%–7.5% of the in k^2-tree.

Keywords: k^2-MDD-WS, multi-valued decision diagram, web service composition, web service dependency graph

1 Introduction

With the increasing number of Web services, a large number of simple and easy Web services are shared on the Internet, and the needs of users are increasingly complex. It is difficult for a single Web service to meet users' needs, so it is necessary to combine multiple Web services to complete the complex needs of the users. How to combine the results of service composition that can meet users' needs in available mass Web services has become one of the research hotspots [1-5].

The concept of the Web Service Dependency Graph is proposed, and it is introduced into the solution of a Web service composition problem in the research of the graph-based Web service composition method [1]. By using the planning algorithm to solve the problem of Web service composition on the basis of the Web Service Dependency Graph [2], Hoffmann et al. Construct a Web Service Dependency Graph from the initial state to the target state and then find a solution path through backward search. A Web service combination method based on Dijkstra algorithm and the graph programming algorithm is used to solve Web services composition problem [3]. A method of automatic Web service composition based on dynamic programming algorithms is proposed [4]. Under the condition of large-scale Web service, the above planning algorithm are limited by the search space, and it is difficult to get a feasible solution. To solve the problem of restricted search space, the Web Service Dependency Graph model based on k^2-tree was proposed [5], and the k^2-tree was introduced into the research of Web service composition problem. However, when one uses k^2-tree to represent a large-scale Web service composition diagram, there are still a large number of isomorphic sub-trees, which have some limitations [6-7].

Based on the Web Service Dependency Graph, we introduce the k^2-MDD into the research of the Web service composition problem [8-10]. The current study proposes a new Web service composition model

* Corresponding Author

called k^2-MDD-WS; it is based on multi-valued decision diagram and can merge a large number of isomorphic sub-trees in the Web Service Dependency Graph model based on k^2-tree to further reduce the search state space.

The formal definition of the k^2-MDD-WS model is as follows.

k^2-MDD-WS = < Request < R_{in}, R_{out} >, C_I, C_O, W_j, V, E, k, f, S >

The model is defined as a tuple with the following components:

(1) Request < R_{in}, R_{out}>: A user's given request; R_{in} is an input concept of the service, and R_{out} is an output concept of the service;

(2) C_I: C_I is a finite set of typed input parameters of the Web service [11]. Among these parameters, C_{I_i} represents the ith input concept parameter in the Web service library; OWL-S (Web Ontology Language for Web Services) files are used to define relationships between services and concepts. $i \in N$;

(3) C_O: C_O is a finite set of typed output parameters of the Web service [11]. Among these parameters, C_{O_i} represents the ith output concept parameter in the Web service library. $i \in N$;

(4) W_j: W_j is a finite set of the Web services, $j \in N$;

(5) V: V is the node set of the Web Service Dependency Graph; W_j, C_{I_i}, $C_{O\_i} \in V$;

(6) E: E is the edge set of the Web Service Dependency Graph, < C_{I_i}, W_j >, <W_j, C_{O_i} >$\in E$, and $E_{<C_{I\_i},W_j>}$ represent the set of the corresponding edges of C_{I_i} and W_j; $E_{<W_j,C_{O\_i}>}$ represents a set of the corresponding edges of W_j and C_{O_i};

(7) k: k represents a number of power blocks recursively divided into k by a Web service dependency graph corresponding to the original adjacency matrix;

(8) f: $D_{1\_i} \times D_{2\_i} \times \ldots \times D_{n\text{-}1\_i} \times D_{n\_i} \rightarrow S$. The multivalued decision diagram of Web service is constructed from a discrete multi-valued function f that contains n variables. $N = \lceil log_k|V_{nums}| \rceil, i \in \{1,2,\ldots,k^2\}$, $(f_{<W_j,C_{I\_i}>} \,||\, f_{<C_{I\_i},W_j>}) = 1$, which means that there is a dependency between C_{I_i} and W_j; $(f_{<W_j,C_{O\_i}>} \,||\, f_{<C_{O\_i},W_j>}) = 1$ and that there is a dependency between W_j and C_{o_i}, where S represents the range of the discrete multi-valued function f, $S \in \{0,1\}$;

(9) $\forall$ $C_{I\_i} \in C_I(W)(0 < I \leq |C_I(W)|)$; if and only if the input concepts of the corresponding Web service are all provided, then W_j can be invoked.

To understand the above formal models easily, an example is provided in Example 1 to illustrate the model definition above.

Example 1: By supposing W_1, $W_2 \in W$; A, B, $C \in C_I$; $D \in C_O$; A, B, C, D, W_1, $W_2 \in V$; <A, W_1>, <B, W_1>, <B, W_2>, <C, W_2>, <W_1, D>, <W_2, D>$\in E$, the Web Service Dependency Graph is illustrated in Fig. 1:

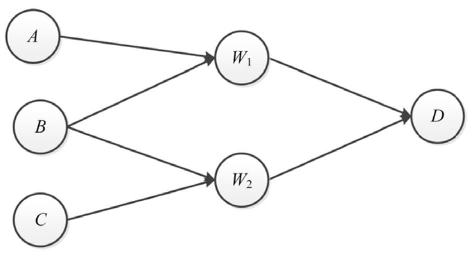


Fig. 1. Web service dependency graph

The corresponding adjacency matrix (*adjMatrix*(*in*), *adjMatrix*(*out*)) is determined based on the edge relation <C_{I_i}, W_j> and <W_j, C_{O_i}> in the Web Service Dependency Graph, as presented in Table 1 and Table 2.

Table 1. *adjMatrix (in)*

	W_1	W_2
A	1	0
B	1	1
C	0	1
D	0	0

Table 2. *adjMatrix (out)*

	A	*B*	*C*	*D*
W_1	0	0	0	1
W_2	0	0	0	1

The original adjacency matrix *adjMatrix*(*in*) has four rows and two columns, by adding two columns of 0 data, the adjacency matrix can be divided into a k^2 sub-adjacency matrix, as presented in Table 3. The original adjacency matrix *adjMatrix*(*out*) has two rows and four columns; by adding two rows of 0 data, the adjacency matrix can be divided into a k^2 sub-adjacency matrix, as presented in Table 4.

Table 3. The *adjMatrix* (*in*) repaired with 0

	W_1	W_2		
A	1	0	0	0
B	1	1	0	0
C	0	1	0	0
D	0	0	0	0

Table 4. The *adjMatrix*(*out*) repaired with 0

	A	*B*	*C*	*D*
W_1	0	0	0	1
W_2	0	0	0	1
	0	0	0	0
	0	0	0	0

The corresponding k^2-tree and k^2-MDD-WS are obtained according to the adjacency matrix in Table 3, as illustrated in Fig. 2.

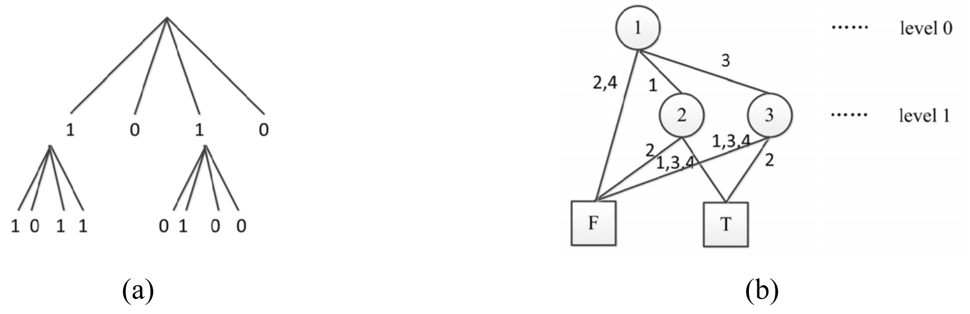


Fig. 2. The k^2-tree and k^2-MDD-WS of Table 3

The corresponding k^2-tree and k^2-MDD-WS are obtained according to the adjacency matrix in Table 4, as illustrated in Fig. 3.

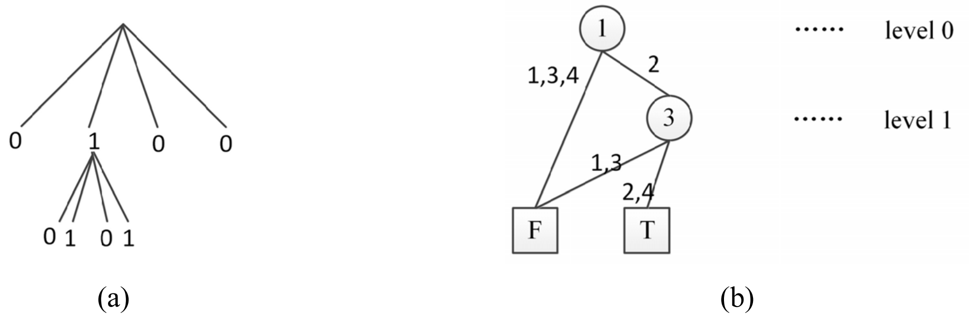


Fig. 3. The k^2-tree and k^2-MDD-WS of Table 4

A user request is given as Request $<R_{\text{in}}, R_{\text{out}}>$, $A,B \in R_{\text{in}}$, $D \in R_{\text{out}}$. $A,B \in C_{\mathrm{I}}$; $A,B,D \in V$; one can get W_1 and W_2 according to $f_{<C_{\mathrm{I}\_A},W_1>} = 1, f_{<C_{\mathrm{I}\_B},W_1>} = 1$, $f_{<C_{\mathrm{I}\_D},W_2>} = 1$, because $f_{<W_1,C_{\mathrm{I}\_A}>} = 1$, $f_{<W_1,C_{\mathrm{I}\_B}>} =1$, $f_{<W_2,C_{\mathrm{I}\_B}>} = 1, f_{<W_2,C_{\mathrm{I}\_C}>} = 1$, A, $B \in R_{\text{in}}$, $C \notin R_{\text{in}}$, and the rule $\forall$ $C_{\mathrm{I}\_i} \in C_{\mathrm{I}}(W)$ $(0 < I \leq | C_{\mathrm{I}}(W)|)$ is meet. One can then get W_1; based on the above steps one can get W_1 and get D, then the Request $<R_{\text{in}}, R_{\text{out}}>$ is satisfied.

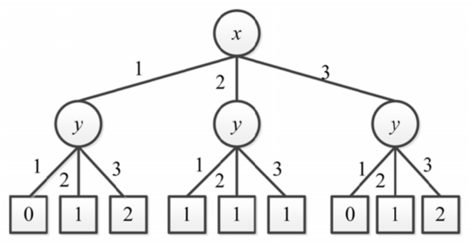
The nodes of k^2-MDD-WS include terminal nodes and nonterminal nodes. The nonterminal nodes are represented by x_i and contain k^2 pointers to other nodes, which correspond to the function f. The formal description is $f_{x_i=c} = f(x_1, x_2, \ldots, x_{i-1}, c, x_{i+1}, \ldots, x_n)$. When a set of values of the multi-valued variables x_1 to x_n is given, a unique terminal node value can be obtained. The simplification rules of k^2-MDD-WS can be summarized as follows:

Rule 1: Remove duplicate terminal nodes. Eliminate all but one terminal node with a given value and redirect all pointers to the eliminated nodes to the remaining one.

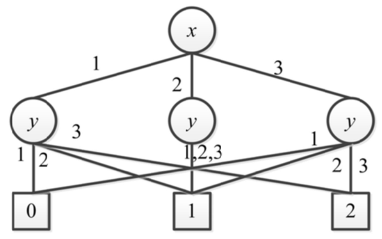
Rule 2: Remove duplicate nonterminal nodes. If nonterminal nodes m and n are in the same layer, $\text{pointer}_m(1) = \text{pointer}_n(1), \ldots, \text{pointer}_m(k^2) = \text{pointer}_n(k^2)$, then eliminate one of the two nodes and redirect all incoming pointers to the other node.

Rule 3: Remove redundant nodes. If all pointers of nonterminal node m point to the same node n, then eliminate m and redirect all incoming pointers to n.

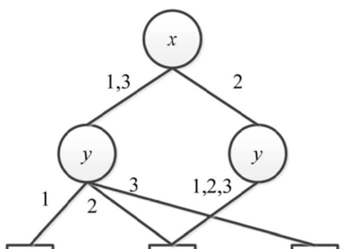
Example 2: Suppose $f = x \times y$ is a multi-valued discrete function, $x \in \{1, 2, 3\}$, $y \in \{1, 2, 3\}$, the range of the discrete function f is set $\{0, 1, 2\}$, the initial multi-valued decision graph corresponding to discrete function f as illustrated in Fig. 4(a), according to simplification Rule 1, the simplified result of Fig. 4(a) is illustrated in Fig. 4(b); according to simplification Rule 2, the simplified result of Fig. 4(b) is illustrated in Fig. 4(c); and according to simplification Rule 3, the simplified result of Fig. 4(c) is illustrated in Fig. 4(d).



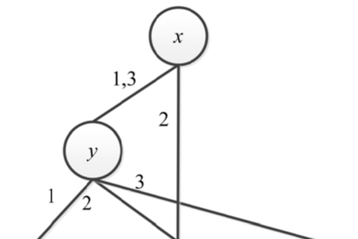
(a) The initial multivalued decision graph of decision function $f = x \times y$



(b) The result after used Rule 1



(c) The result after used Rule 2



(c) The result after used Rule 3

Fig. 4. The simplified rules of k^2-MDD-WS

2 The Construction Process of k^2-MDD-WS

In the Web Service Dependency Graph, the Web service W_j and the Web service concept parameter C_{I} or C_{O} are abstracts into vertex set V, and then the dependency relationships between the Web services and

Web service concept parameter are abstracted into edge set *E*. The construction process of k^2-MDD-WS includes binary coding of vertex set *V* of the Web Service Dependency Graph, encoding of edge set *E* according to the codes set of vertices, and constructing k^2-MDD-WS based on the codes set of edges. The specific algorithm implementation steps are as follows:

2.1 Encoding the Vertices of a Web service Dependency Graph

When the vertices of a Web Service Dependency Graph $G = (V, E)$ are binary coded, the number of bits in the binary code is $N=\lceil log_k|V\_nums|\rceil$, where *V_num* is the number of vertices of the Web Service Dependency Graph, and $k = 2$. Algorithm 1 illustrates the pseudo-code algorithm process of binary coding for the vertices of the Web Service Dependency Graph. Assuming $(A, B, C, W_1, W_2) \in V$, $V\_num = 5$, and the number of bits in the binary code is $n = 3$, according to *Algorithm* 1. The result of binary encoding of the vertices is illustrated in Fig. 5.

```
Algorithm 1. Binary encoding the vertices of a Web service dependency graph.
Input: the vertex number V_num that needs a binary code
Output: the vertex binary array V_Code[]
1. Binary_V_Encoding (V_Code[], V_num)    /* The vertex V_num of the Web
   Service Dependency Graph is encoded in binary form, and the encoding
   exists in the V_Code[] array. */
2. n ←logk|V_nums|   /* V_num is the number of vertices of the Web Service
   Dependency Graph, and n is the number of bits in the binary code.*/
3.   left ← 1
4.   right ← Pow (2, n)     /* Take a number of powers that are not less than
   2 to V_nums */
5.   i ← 1
6.   While (left <= right)  Do
7.          mid ← (left + right) / 2
8.          If   V_num <= mid Then
9.                V_Code [i] ← 0
10.               right ← mid - 1
11.         Else
12.               V_Code [i] ← 1
13.               left ← mid + 1
14.  i ← i + 1
15.  End While
```

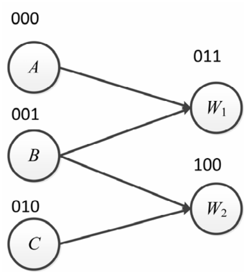


Fig. 5. The code of vertices

2.2 Encoding the Edges of a Web Service Dependency Graph

The edge relationship between one vertex and another represents the dependency between the Web service and the input/output concept parameters, and the edge of the vertex to the vertex can be described by the feature discrete function $E_{<V_0,V_1>}$.

Assuming $\boldsymbol{X} = (x_1, \ldots, x_n)$ and $\boldsymbol{Y} = (y_1, \ldots, y_n)$ is the coding vector for the vertices in the Web service graph, When $k = 2$, the feature discrete function of the edge of vertex *X* to vertex *Y* is represented as $E(X, Y)$: $\{0, 1\}^n \times \{0, 1\}^n \rightarrow \{1, 2, 3, 4\}^n$.

A code of edge is generated by the binary encoding of two vertices; the value of each binary bit of the vertex code is in {0, 1}, and the value of each bit of the edge code is in {1, 2, 3, 4} according the value of the vertex code. The encoding of the corresponding edges illustrated in Fig. 6 can be obtained by the vertex encoding *A*, *B*, *C*, W_1, W_2 in Fig. 5. The specific *Algorithm* 2 is as follows:

```
Algorithm 2. Encoding the edges of a Web Service Dependency Graph
Input: The number of start vertex start_V_num and the number of end vertex
      end_V_num
Output: The corresponding edge of the start vertex and the end vertex is an
       encoded array E_Code[]
1. Binary_E_Encoding (E_Code[], start_V_num, end_V_num)  /*The encoding of
   the corresponding start vertex and the encoding of the end node.*/
2. Binary_V_Encoding (V_Code1[], start_V_num) /*Get the code of the start
   vertex*/
3. Binary_V_Encoding (V_Code2[], end_V_num) /* Get the code of the end vertex
   */
4. For i ← 1…codeNum Do
5.   E_Code[i] ← V_Code1 [i] * 2 + V_Code2[i] + 1
6. EndFor
```

$A \rightarrow W_1$:	122
$B \rightarrow W_1$:	124
$B \rightarrow W_2$:	213
$C \rightarrow W_2$:	231

Fig. 6. The code of edges

2.3 Constructing k^2 - MDD-WS According to the Set of Edge Codes

k^2-MDD-WS is constructed by using *apply*() operation on the code set of the edges, the k^2-MDD-WS has *n* variables for the dependency graph of the Web service, and the values of the *n* variables are True and False. For example, according to the edge encoding in Fig. 6, the k^2-MDD-WS structure is illustrated in Fig. 2(b). In the current study, the MEDDLY [12] (multi-terminal and edge-valued decision diagram library) discrete function library is used as a reference in constructing k^2-MDD-WS. The MEDDLY function library is a C++ open-source project for manipulating MDD; it was developed by Iowa State University under the Linux platform, which provides rich MDD constructs and operational functions. The *createVariablesBottonUp*() function is applied to define the number of the variables and the value of each variable in the MDD. The MDD of the edge is created according to the *createEdge*() function, and then merge the MDD of the two edges according *apply*() function and the UNION operation. The specific *algorithm* 3 of constructing k^2-MDD-WS is as follows:

```
Algorithm 3. Constructing k2-MDD-WS
Input: A set of the edge codes
Output: k2-MDD-WS
1. CreateK2MDD-WS()  /*Construct the k2-MDD-WS according the set of the edge
   codes*/
2. For i ← 1…codeNum Do  /*set the number of the variables and set the range
   of the variables */
3.   Bounds[i] ← 4
4. EndFor
5. createVariablesBottomUp(bounds[], codeNum)  /*create variable*/
6. Binary_E_Encoding(edgeCode[], start_V_num[1], end_V_num[1])  /*encoding
   the first edge*/
7. createEdge(edgeCode[], 1, all)  /*create the first MDD from the first edge
   code*/
```

```
8. For i ← 2…edgeNum Do
9.   Binary_E_Encoding(edgeCode[], start_V_num[i], end_V_num[i])  /* encoding
   the ith edge */
10.  createEdge(edgeCode[], 1, rest)  /*create the MDD of the ith edge code*/
11.  apply(UNION, all, rest, all)  /*merge these created MDD according the
   UNION operation*/
12. EndFor
13. k2-MDD-WS ← all  /*constructing the k2-MDD-WS according the all edge
   MDD*/
```

If the Request <R_{in},R_{out}> is given, $R_{\text{in}} \in C_{\text{I}}$; $R_{\text{out}} \in C_{\text{O}}$;$C_{\text{I}\_i} \in R_{\text{in}}$; R_{in},$R_{\text{out}} \in V$. If the discrete function $f_{<C_{\text{I}\_i},W_j>}$ = 1 is true, the corresponding edge $E_{<C_{\text{I}\_i},W_j>}$ represents that there is correlation between the concept parameter C_{I_i} and the Web service W_j; otherwise, there is no dependency relationship. The MEDDLY function library provides the INTERSECTION operation that can compute the intersection of two MDDs. The edge query can be done by appling the INTERSECTION operation between k^2-MDD-WS and the MDD of the edge, if the value of the INTERSECTION operation is true, that indicate the edge exist in the k^2-MDD-WS. The specific *algorithm* 4 of the edge query is as follows:

```
Algorithm 4. Querying whether there is an edge relationship algorithm between
             two given vertices
Input: The start vertex numbers start_V_num and end vertex numbers end_V_num
       of the edge
Output: Indication whether the corresponding edge exists
1. edgeQuery(start_V_num, end_V_num)
2. Binary_E_Encoding(edgeCode[], start_V_num, end_V_num)
3. createEdge(edgeCode[], 1, tmp)
4. apply(INTERSECTION, K2MDD-WS, tmp, res)
5. If  res.getNode() = 0 Then
6.     Print false
7. Else
8.     Print true
9. End if
```

Suppose the assignment of a vertex parameter C_{I} is V_1 and all service vertices W_j are assigned to V_2; if the discrete function $f_{<V_1,V_2>}$ = 1, then V_2 is the service that corresponds to the parameter V_1; otherwise, it is not. The specific *algorithm* 5 is as follows:

```
Algorithm 5. Querying the end vertex of a given start vertex
Input: The number of the start vertex start_V_num
Output: The number of the end vertex end_V_num
1. outNeighborQuery(start_V_num,end_V_num)  /*The end vertex query according
   to the start vertex*/
2. Binary_V_Encoding (nodeCode[], nodeID)  /* Encode the vertices */
3. For i←1… nodenum Do
4.   EdgeQuery(start_V_num, node i)  /* Determine whether there is an edge
   between the start vertex and the query vertex.*/
5.   If res.getedge() = 0. then  /*if res.getedge() = 0, then there is no
   edge between the start vertex and the query vertex. */
6.      Print false
7.   Else
8.      Print(node i)
9.   End if
10.End for
```

In this model system, edges between services and concepts are compressed and represented as two types of k^2-MDD-WS: one for services and input concept parameters and the other for services and output concept parameters (k^2-MDD-WS$_{\text{in}}$ and k^2-MDD-WS$_{\text{out}}$, respectively). We search the k^2-MDD-WS for matching services and concepts. According to user's Request <R_{in},R_{out}> and the rules $\forall\ C_{\text{I}\_i} \in C_{\text{I}}\ (W_j)$ $(0 < I \leq |\ C_{\text{I}}\ (W_j)\ |)$, traversed k^2-MDD-WS$_{\text{in}}$ and k^2-MDD-WS$_{\text{out}}$ are to find a set of the invoke Web service sequences that meet the needs of users. In k^2-MDD-WS$_{\text{in}}$, $R_{\text{in}} \in C_{\text{I}}$, $R_{\text{out}} \in C_{\text{O}}$, $C_{\text{I}\_i} \in R_{\text{in}}$, if $f_{<C_{\text{I}\_i},W_j>}$ = 1, then get the set of the edge $E_{<C_{\text{I}\_i},W_j>}$, and $E_{<C_{\text{I}\_i},W_j>} \Rightarrow W_j$, $i \in$ N,$j \in$ N; if $f_{<W_j,C_{\text{I}\_i}>}$ = 1, then get the set of the edge $E_{<W_j,C_{\text{I}\_i}>}$, $i \in$ N, $j \in$ N, $E_{<W_j,C_{\text{I}\_i}>} \Rightarrow C_{\text{I}\_i}$ and $C_{\text{I}\_i} \in R_{\text{in}}$; it is determined that W_j can be invoked through the rule $\forall\ C_{\text{I}\_i} \in C_{\text{I}}\ (W_j)$ $(0 < I \leq |\ C_{\text{I}}\ (W_j)\ |)$. In the k^2-MDD-WS$_{\text{out}}$, if the W_j can be invoked and the discrete function $f_{<W_j,C_{O\_i}>}$ = 1, then get the set of the edge $E_{<W_j,C_{\text{O}\_i}>}$, $E_{<W_j,C_{\text{O}\_i}>} \Rightarrow C_{\text{O}\_i}$ and $C_{\text{O}\_i} \in$ R_{out},R_{out}= 1− ($R_{\text{out}} \cap C_{\text{O}\_i}$); if the $R_{\text{out}} = \varnothing$, then it satisfies the conditions, and the combination is done; otherwise, $R_{\text{in}} = C_{\text{O}\ \_i} \cup R_{\text{in}}$, and we repeat the above steps. The specific *algorithm* 6 is as follows:

Algorithm 6. Solving the problem of Web service composition

```
Input: Request <Rin,Rout>
Output: solution: a set of solution services
1. Conceptset ← initial
2. Serviceset ← initial
3. While  Rout ⊄ Conceptset
4.    For i←1...Rin_num Do
5.         Conceptset ← Rin
6.    End For
7.    For i←1...Conceptsetnum Do
8.       start_V_num ← Conceptset
9.       Addservice ← outNeighborQuery(start_V_num,end_V_num)      /*The
   services of the corresponding to the known input concept parameters are
   obtained according to the function outNeighborQuery
   (start_V_num,end_V_num) in k2-MDD-WSin */
10.   End For
11.   For i←1…addservicenum Do
12.       end_V_num ← addservice
13.       Temp ← inNeighborQuery(start_V_num, end_V_num)   /* Query all input
   concept parameters of the service through the function
   inNeighborQuery(start_V_num, end_V_num) and determine whether the service
   satisfies the rule ∀ CI_i∈CI (Wj)  (0 < I ≤ | CI (Wj) |)  in k2-MDD-WSin
   */
14.       Temp1 ← outNeighborQuery(start_V_num, end_V_num)
15.       If Temp ⊆ Conceptset && Temp1   Conceptset
16.           Serviceset ← addservice
17.       End If
18.   End For
19.   For i←1…servicenum
20.       start_V_num←serviceset
21.       Conceptse t← outNeighborQuery(start_V_num, end_V_num)      /* The
   input concept parameters of the corresponding to the known service is
   obtained according to the function outNeighborQuery
   (start_V_num,end_V_num) in k2-MDD-WSout*/
22.   End For
23.End while
```

3 Time Complexity Analysis of the Algorithm

The construct of k^2-MDD-WS is constructed from the edge codes set involving the MDD initialization, generation, and simplification and other operations. When k^2-MDD-WS$_{\text{in}}$ and k^2-MDD-WS$_{\text{out}}$ are constructed, the time complexity of a single vertex code for a web service dependency graph is $O(\log_k |V|)$. Similarly, the time complexity of encoding a single edge for the Web service dependency graph is

$O(\log_k |V|)$, so the time complexity of encoding all edges of the Web service dependency graph is $O(|E| \cdot \log_k |V|)$. According to Request<R_{in}, R_{out}> given by the user, finding a Web service composition solution that meets the needs of the users' requirements in the k^2-MDD-WS model and the external neighbor query and inner neighbor query operations are involved in the implementation process. Because the height of k^2-MDD-WS is $n=\lceil \log_k|V| \rceil$, the time complexity of the outer neighbor query and the inner neighbor query is equal to that of the k^2-tree.

4 Acknowledgements

The current study paper used the C++ language and the MEDDLY library to implement the proposed algorithm. The experimental machine configuration software platform is an Ubuntu14.04 LTS 64-bit operating system (kernel Linux 3.19.0-61-generic), and the hardware platform is 8 GB of memory operated under the device.

Table 5. The WSC'08 test data set

	Test01	Test02	Test03	Test04	Test05	Test06	Test07	Test08
Number of Concept	457	558	3087	3136	3068	12469	3076	12338
Number of Service	158	1665	604	1041	1090	2198	4113	8119

The data of the experiment are all from the data set of the Web Service Challenge competition. In the WSC'08 data set, the WSDL file describes the information about the Web service, and ontology describes the semantic relationship of all input/output object instant concept parameters in an XML file. The data set includes eight sub Web service libraries that include 457-12,469, the service corresponding input/output concept parameters, and 158-8119 Web services. Table 6 lists the results of experiments using k^2-tree and k^2-MDD-WS, The CR represents the compression rate and is expressed as formula (1):

$$\text{CR} = \frac{k^2 - \text{MDD} - \text{WS}\ \textit{structure}}{\textit{Services} \times \textit{Concepts}}. \tag{1}$$

Table 6. Experimental results

		Test01	Test02	Test03	Test04	Test05	Test06	Test07	Test08
k^2-tree	CR	16%	4.8%	5.7%	3%	3%	0.9%	3.1%	1%
	Search (ms)	153	426	2406	778	1594	10300	2570	11266
k^2-MDD-WS	CR	1.2%	0.3%	0.2%	0.15%	0.16%	0.04%	0.13%	0.04%
	Search (ms)	221	673	6289	1921	4308	28364	6992	31293

Table 6 presents the experimental results based on k^2-MDD-WS and k^2-tree. From the data of the experimental results in Table 6, we can see that the CR of nodes in k^2-MDD-WS in the selected eight web service libraries is only 3.5% to 7.5% of that in k^2-tree. The increase in the number of the concept parameters and the number of Web services from the original eight service libraries indicates that more nodes in k^2-MDD-WS is reduced when the number of concept parameters and the number of Web services are larger. It can be concluded that using k^2-MDD-WS to represent Web service composition issues reduces the number of nodes and is more suitable for large-scale Web service composition issues. In a Web service dependency graph of same scale, the corresponding k^2-MDD-WS is the same height as the k^2-tree structure, so the complexity of traversal time is the same when the Web service composition problem is implemented. From the combined time in Table 6, it can be seen that the combined time is at the same time complexity, and the desired effect is achieved.

5 Conclusions

Based on multivalued decision graph, this paper presents a new Web service composition model k^2-MDD-WS, and which is constructed according to Web Service Dependency Graph. In the construction process of k^2-MDD-WS, the redundant Web service nodes resulting from the large number of isomorphic subtrees existing in the k^2-tree are merged to reduce the number of Web service nodes, thus reducing the search state space in the process of finding the Web service satisfying the user's requirements. The k^2-MDD-WS transforms the Web service composition problem into the search problem of the multivalued decision graph, and the Boolean logic operations are used to improve the efficiency of searching the Web service that satisfies the user's requirement.

Acknowledgements

This work is supported in part by the National Natural Science Foundation of China under Grant 61762024, and in part by the Natural Science Foundation of Guangxi Province under Grant No. 2017GXNSFDA198050, Grant No. 2016GXNSFAA380054.

References

[1] S.-V. Hashemian, F. Mavaddat, A graph-based approach to web services composition, in: Proc. IEEE the 2005 Symposium on Applications and the Internet, 2005.

[2] J. Hoffmann, P. Bertoli, M. Pistore, Web service composition as planning, revisited: in between background theories and initial state uncertainty, in: Proc. the 22nd National Conference on Artificial Intelligence, 2007.

[3] Y. Yan, M. Chen, Y. Yang, Anytime QoS optimization over the Plan Graph for web service composition, in: Proc. the 27th Annual ACM Symposium on Applied Computing, 2012.

[4] M. Kuzu, N.-K. Cicekli, Dynamic planning approach to automated web service composition, Applied Intelligence 36(1) (2012) 1-28.

[5] J. Li, Y. Yan, D. Lemire, A web service composition method based on compact K2-trees, in: Proc. IEEE International Conference on Services Computing, 2015.

[6] N.-R. Brisaboa, S. Ladra, G. Navarro, K2-trees for compact web graph representation, in: Proc. the 16th International Symposium on String Processing and Information Retrieval, 2009.

[7] N.-R. Brisaboa, S. Ladra, G. Navarro, Compact representation of web graphs with extended functionality, Information Systems 39(1)(2014) 152-174.

[8] R.-S. Dong, X.-K. Zhang, H.-D. Liu, T.-L. Gu, Representation and operations research of K2-MDD in large-scale graph data, Journal of Computer Research and Development 52(12)(2016) 2783-2792.

[9] T.-L. Gu, The novel abstract data type: ordered binary decision diagrams, Journal of Guilin University of Electronic Technology 30(5)(2010) 374-388.

[10] Web ontology language for web services.

[11] D. Martin, M. Burstein, J. Hobbs, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, K. Sycara, OWL-S: Semantic Markup for Web Services. <http://www.w3.org/submission/owl-s/>, 2018 (accessed 30.10.18).

[12] Iowa State University Research Foundation. MEDDLY: Multi-terminal and edge-valued decision diagram library. New in Version 0.11.486, February, 2014.