Appendix 1: Algorithms

Algorithm A-1: Identifying and aggregating the sequence pattern

Sequential Pattern *(Matrix A, cComp.Q)*
  set Flag_seq to zero
  While(1)
    Set Flag_Seq to zero
    ∀ Cur_node in Matrix
      if Cur_node has the Next_node and the ***equation(12)*** *is true:*
      Call Cal_Perform_Sequential(A,Cur_node, Next_node, cComp.Q)
        set Flag to 1
        set Flag_Seq to 1
      end
    end
  end
  If Flag_Seq is zero
    Break the loop
  End
End
Function Cal_Perform_Sequential*(Matrix A,Cur_node, Next_node, cComp.Q)*
  $\text{Availability}_{\text{Cur}_{\text{node}}}$, $ResponseTime_{Cur\_node}$, $Cost_{Cur\_node}$=$Q_{1:3}$*(cur node)*
  $\text{Availability}_{\text{Next}_{\text{node}}}$, $ResponseTime_{Next\_node}$, $Cost_{Next\_node}$=$Q_{1:3}(Next\ node)$
  Set $Probability_{Cur\_node} = Probability_{Next\_node}$
  Update the Matrix A(Remove all of connection of next_node to other and vice versa)
  Set $Availability_{Cur\_node} = Availability_{Cur\_node} \times Availability_{Next\_node}$
  Set $ResponseTime_{Cur\_node} = ResponseTime_{Cur\_node} + ResponseTime_{Next\_node}$
  Set $Cost_{Cur\_node} = Cost_{Cur\_node} + Cost_{Next\_node}$
  $Q_{1:3}\text{Cur}_{\text{node}}=\text{Availability}_{\text{Cur}_{\text{node}}}$, $ResponseTime_{Cur\_node}$, $Cost_{Cur\_node}$
  Delete $Q_{1:3}$*(next node)*
End

Algorithm A-2: Identifying and aggregating the parallel pattern

Function Parallel Pattern *(Matrix A,* Q*)*
  set Flag_Paral to zero
  While(1)
    ∀ Cur_node in Matrix A
      If Cur_node has the concurrent node and the ***equation(13)****or* ***equation(17)*** *is true*
      Call Cal_Perform_Parallel(A,Cur_node, Concurrent_node, Q)
        set Flag_Paral to 1
        Set Flag_Paral to 1
      end
    end
  End
  If Flag_Paral is zero
    Break the loop
  End
End

Cal_Perform_Parallel*(Matrix A,Cur_node,Concurrent_node, Q)*
  $Availability_{Cur\_node}$, $ResponseTime_{Cur\_node}$, $Cost_{Cur\_node}$=$Q_{1:3}$*(cur node)*
  $\text{Availability}_{\text{Concurrent\_node}}$, $ResponseTime_{Concurrent\_node}$, $Cost_{Concurrent\_node}$=$Q_{1:3}$*(Concurrent_node)*
  Set $Probability_{Cur\_node} = 1$
  Update the Matrix A(Remove all of connection of Concurrent_node to other and vice versa)
  Set $Availability_{Cur\_node} = Availability_{Cur\_node} \times Availability_{Concurrent\_node}$
  Set $ResponseTime_{Cur\_node}$=*Max{* $ResponseTime_{Cur\_node}$,$ResponseTime_{Concurrent\_node}$ *}*
  Set $Cost_{Cur\_node} = Cost_{Cur\_node} + Cost_{Concurrent\_node}$
  $Q_{1:3}\text{Cur}_{\text{node=Availability}_{\text{Cur}_{\text{node}}}}$, $ResponseTime_{Cur\_node}$, $Cost_{Cur\_node}$
  Delete $Q_{1:3}$*(Concurrent_node)*
End

Algorithm A-3: aggregating the loop pattern

1. ***Function Loop Pattern*** *(Matrix A, Q)*
2. *Set Flag_Loop to zero*
3. *Loop_array=**Call Identifying Loop Pattern**(Matrix A)*
4. *If (Loop_array!=null)*
5. *Set Flag_Loop to one*
6. *For i=1:length(Loop_array)*
7. *Set Array_node =Loop_array[i]*
8. *If **equation(16)** or **equation(18)** or **equation(19)** are true in Array_node:*
9. ***Call Cal_Perform_Loop(A, Array_node, Q)***
10. *end*
11. *end*
12. *end*
13. ***End***

14. ***Function Cal_Perform_Loop**(Matrix A, Array_node, Q)*
15. $Availability_{Array\_node}$, $ResponseTime_{Array\_node}$, $Cost_{Array\_node}$=*Q(Array_node)*
16. *Probability of every node in loop calculated by formula in table(2-2)*
17. *Update the Matrix A(Create connections from the v_1 to the external nodes of loop and remove the connections of v_2 ,…, v_n to the external nodes of loop)*
18. *Set $Availability_{Array\_node}$ by formula in table(2-1)*
19. *Set $ResponseTime_{Array\_node}$ by formula in table(2-1)*
20. *Set $Cost_{Array\_node}$ by formula in table(2-1)*
21. $Q_{1:3}$Array-node(1)=Availability$_{Array_{node}}$, $ResponseTime_{Array\_node}$, $Cost_{Array\_node}$
22. *Delete $Q_{1:3}$(Array_node(2:length(array_node))*
23. ***End***

Algorithm A-4: Identifying the loop pattern

1. ***Loop_array =Function Identifying Loop Pattern(Matrix A)***
2. *Add first node of graph to stack*
3. *set Visited-array to empty*
4. *while(1)*
5. *if the stack is empty*
6. *Break;*
7. *end*
8. *if the last member of stack is in visited array*
9. *Remove last member from the end of stack*
10. *Pos_last member=find position last member of stack in visiited array*
11. *∀ Cur_node in visited array from Pos_last member to the end of visited array*
12. *if Cur_node has connection with the last member of stack*
13. *Loop= [last member,…,Cur_node]*
14. *Add Loop to Loop_array*
15. *end*
16. *else if last member has at least one child*
17. *Add last member to visited array*
18. *Remove last member of end of stack*
19. *Add last member's children to the end of stack*
20. *else*
21. *Break*
22. *end*
23. *end*
24. *Sort the loops in Loop_array in order from small to large*
25. *Return loop_array*
26. ***end***

Algorithm A-5: Extracting paths from the aggregated composition

1. **[*NPath,Path]=ExtractingPath-graph(Matrix A,IPath)** */Path is an array whose each member contains web services of a graph path*
2. *i=1;*
3. *array[1]=Start_node; /* put the graph start_node in the first place of the array*
4. *while(~isempty(array))*
5. *If (j has children) /* j=the latest node of array[1]*
6. *find the children of j*
7. *Foreach child k of the children of j, {*
   a. *create vector;*
   b. *vector= array[1],k*

c. *Extend the array by adding the vector to the array*
d. *}*

8. else
9. Path[i]=array(1)
10.i=i+1;
11.end
12.delete the array(1)
13.end / end while*
14.NPath=length(Path)
15.Return (Path[IPath],NPath)
16.End