Dynamic Service Composition

A Bidirectional Approach using forward and Backward Chaining

Sandip Khakhkhar[1], Vikas Kumar[2], Sanjay Chaudhary[3]

DA-IICT, Gandhinagar Gujarat, India

[1]sandip_khakhkhar@daiict.ac.in; [2]vikas_kumar@daiict.ac.in; [3]sanjay_chaudhary@daiict.ac.in

***Abstract*- Service is a network addressable software component to perform a specific task. A service discovery mechanism can be used to find services that can be executed to satisfy a service request. A service composition generates a composition plan and a composite service to satisfy a service request. Static composition process consumes considerable amount of time and effort. It is also vulnerable to changes in input/output of services. A dynamic composition algorithm can automatically select services involved in composite plan and generate a composite service on-the-fly. Composition time taken by the algorithm to generate a composite service is the main issue with dynamic composition algorithms. Dynamic composition algorithms presented in previous work mainly follow either forward chaining approach (FCA) or backward chaining approach (BCA) to generate a composite service. Their performance suffers for certain cases to generate a composite service where the number of services explored increases exponentially as number of iterations increases. This work proposes a dynamic composition algorithm that gives consistent performance across all the cases. Proposed algorithm approaches from two directions alternatively, one follows FCA and another follows BCA. Proposed algorithm explores less number of services and takes less composition time compared to algorithms FCA or BCA.**

Keywords- Service Composition; Forward Chaining Approach; Backward Chaining Approach

I. INTRODUCTION

Service is a network addressable software component to perform a specific task. Services like search engine, e-commerce, on-line shopping etc. are a few examples of services accessible on the web that performs certain tasks required by users [1]. Service interface is represented in terms of input parameters it consumes and output parameter it provides. User uses a service to perform a required task. A service request specifies required task in terms of input parameters that can be provided and output parameters that are required. A service discovery mechanism can be used to find services that can be executed to satisfy service request. A service request is satisfied if at least one service exists which provides output parameters required by service request, and, all its input parameters can be provided by service request [2]. Service and service request is matched by comparing their input/output parameters [5].

An individual service may not be able to satisfy the complete service request. It might be possible to execute a chain of services in a particular order to satisfy service request. This chain of services is referred to as composition plan and service offered by executing this composition plan is referred as composite service. Service request provides the input parameters to the service at the start of composition plan. A service at the end of composition plan provides output parameters required by service request. Intermediate services consume its input parameters from the output parameters provided by services executed before each one of them. Intermediate services provide their output parameters to be consumed by services to be executed after it. The aim of service composition algorithm is to generate a composition plan and generate composite service to satisfy service request [3], [4].

Service composition processes can be divided in two ways based on time when services involved in the composition plan are selected (i) in static composition process, services are selected at the time of designing the composite service, and (ii) in dynamic composition process services are selected at the time of execution of the composite service [6], [7], [8]. Composite service design by static composition process involves the services which exist at the time of its design. Composite services generated using static composition process is vulnerable to changes in service environment. Services existing at design time may be inaccessible due to various reasons, or replaced by other services at the time of execution of the composite service [7].

Dynamic composition process can deal with such changes in service environment as services involved in the composition plan are selected at the time of executing composite service [6]. Dynamic composition has been the hot topic both in academia and industry. There have been a lot of researches in the area of dynamic service composition. The main issue with dynamic composition algorithm is related to the composition time taken by algorithm to generate a composite service [10], [11], [12]. Composition time indicates duration of the time at which the service request was submitted to the algorithm till the algorithm generates a composite service that can satisfy service request. Composition time depends upon the number of services explored by a dynamic composition algorithm in order to find services that can take part in composite plan. This was the main focus of Web Service Challenge 2009 competition [14] and many researchers proposed their technique targeting to reduce composition time. [10] And [12] won the first rank in WWW competition for their performance and architecture respectively. [10] And [12] mainly follow forward chaining approach (FCA) that starts from the source; explore all paths until destination is reached. Another architecture [11] that won the second best architecture award in same WSC 2009 competition follows backward chaining approach (BCA) in which algorithm starts from destination, explore all the nodes from which destination is reachable until all the explored nodes can be reached from the source.

DOI: 10.5963/IJCSAI0203004

FCA and BCA are described in detail in Section III-A and Section III-B respectively. For dynamic web service composition the technique will depend on the criteria, which may be automation, quality of service (QoS) or composition time [15], [16], [17].

Researchers model the dynamic composite service problem in terms of graph searching problem. A service is represented by a node in the graph. There is an edge between two nodes if any of input parameters of one service matches with any of the output parameter of other services. In terms of graph, dynamic composition problem can be described as finding the path from source node to destination node. Source node represents the service whose input parameters are provided by service request whereas destination node represents service whose output parameters are required by service request. However, for an algorithm to qualify as dynamic it should act on services existing at the execution time. With reference to graph, it means that algorithm will have knowledge about only nodes, services, and should not assume any pre-existing knowledge of edges and how services are organized. Keyword match requires parameters to have the similar name to match, whereas in semantic matching parameters are considered to match if they are semantically interoperable. Proposed approach considers keyword match and semantic matching can be performed with very minimal changes. Result of experiment following proposed algorithm shows improvement compared to that of algorithm following FCA [10] and BCA [11]. Mathematical analysis also proves that proposed algorithm is scalable and improvement in performance increases as the length of composition plan increases and it is well supported by the result of experiments.

A. Problem Statement and Motivation

Performance of algorithms based on FCA or BCA depends upon the service organization - graph connectivity and service request. We intuitively thought that FCA and BCA will suffer for certain cases as mentioned in Section III-D. Results of experiment confirmed our intuition. Experiment 1 is one such case where performance of FCA degrades, whereas in Experiment 2 performance of BCA degrades. With FCA and BCA the number of services explored by an algorithm grows exponentially as the number of iterations required by an algorithm increases. They tend to suffer severely as the length of the composition plan increases. Mathematically, in a graph having branching factor of b, one service connected to b a number of other services, FCA or BCA will explore maximum of (b^n) services to find a composition plan of length n. Research has proposed solution to solve this exponential problem up to certain limit using heuristic. Heuristic nature helps FCA and BCA to narrow down its search space and follow only some paths that has high probability of leading to a solution. This reduces the number of services explored by an algorithm but it creates new problem of incompleteness. Heuristic algorithm does not ensure that solution will be found even if it exists. A heuristic solution looses the completeness of an algorithm in effort to reduce the composition time.

An algorithm is required that can give consistent performance and composition time in all the cases. We use the fact that service request specifies available inputs, that is starting node, and required outputs, that is destination node. In this scenario, we can approach from two directions to find the composite service. Motivation to proposed bi-directional algorithm based on finding the path by following two directions, one moves forward from starting point, named FCA and another moves backward from destination point. In this way, each of them has to perform only half the number of the iteration, as they are meeting in the middle, compare to the number of iterations needed if only one direction is followed. This will reduce the number of services to be explored. Mathematically, proposed bi-direction algorithm will explore maximum of ($b^{n/2} + b^{n/2}$) services, to find a composition plan of length n in a graph having branching factor of b. Thus, its space complexity is $O(b^{n/2})$ and since composition time directly depends on the number of services required to explore, time complexity is also $O(b^{n/2})$. As length of composition plan, n, is not known in advance, proposed algorithm follows alternative step in each direction. This ensures that algorithm performs n/2 iteration in both the directions to achieve the optimum results. Proposed algorithm follows the FCA approach from one direction and BCA in another direction. Proposed algorithm does not strictly force to use this strategy, one is free to follow any algorithms, apply heuristic, use semantic services but ensure implementation should allow algorithms to meet in the middle.

II. DEFINITION

We have defined some terms in this section that will allow the reader to grasp the remaining part of the paper.

Definition 1—Service: It is a 4-tuple S = <SN, IPSet, OPSet, QoS> where SN is a service name, IPSet = { IP 1 , IP 2 , IP 3 ...IP n } is a set of input parameters required by a service, OPSet = { OP 1 , OP 2 , OP 3 ...OP n } is a set of output parameters provided by a service, and QoS = {Attr 1 , Attr 2 , Attr 3 , ... } is a set of attributes that defines the quality of a service. Table I shows an example of a service related to an agriculture domain. 'GetStorageLocation' service accepts the 'crop variety' and 'quantity' as input parameters. It finds all the locations where specified quantity of given crop variety can be stored. Service returns back a list of storage locations that match these parameters. QoS parameter like ResponseTime indicates the amount of time required to execute the service and get the results.

TABLE I EXAMPLE OF A SERVICE

Description	Example
Service name	GetStorageLocation
Input Set	CropVariety, qantity
Output Set	StorageLocation
Quality of Service	ResponseTime (µs)(400)

Definition 2—Service Request (SR): It is a 2 -tuple SR = <IPSet, OPSet> where IPSet = { IP 1 , IP 2 , IP 3 ...IP n } is a set of input parameters provided with service request and OPSet = { OP 1 , OP 2 , OP 3 ...OP n } is a set of output parameters required by service request. A service request related to agriculture domain is shown in Table II. Service

DOI: 10.5963/IJCSAI0203004

requester provides 'crop variety' and 'quantity' as input and wants availability as output.

TABLE II EXAMPLE OF A SERVICE REQUEST

Tuple	Example
Input	CropVariety, Quantity
Output	Storage Offer

Definition 3—Usable Service (US): A Service becomes usable service when all its input parameters are provided directly in a service request or indirectly by other usable services.

Definition 4 — Relevant Service (RS): A Service becomes relevant service when it provides at least one of output required by a service request or other relevant service.

III. APPROACHES TO SERVICE COMPOSITION

A. Forward Chaining Approach (FCA)

Forward Chaining Approach for dynamic service composition is described in [9] and is followed by [10] and [12]. FCA starts with the available input as shown in Figure 1. Initially available input is same as input provided in service request. Procedure starts, where matching algorithm finds a set of services whose required input is present in the available input set. Services found by matching algorithm are marked as usable services (Definition 3) and are considered to be candidates for taking part in the composite service. A usable service also marks its output as usable parameters and some are added to currently available input set. Thus, available input set grows. Now available input set contains the inputs that are either provided directly in service request or indirectly provided by a usable service as its output. If any usable service provides the output required by service request then it would be present in available input set. Procedure is repeated again to execute matching algorithm with the updated available input set. Procedure stops when required output of a service request is present in the available input. This happens when composite service exists which can satisfy the required task. Procedure also stops when matching algorithm does not find any usable service. If a composite service exists, then backward search as mentioned in Section III-C is performed to get the composite service from the usable services.

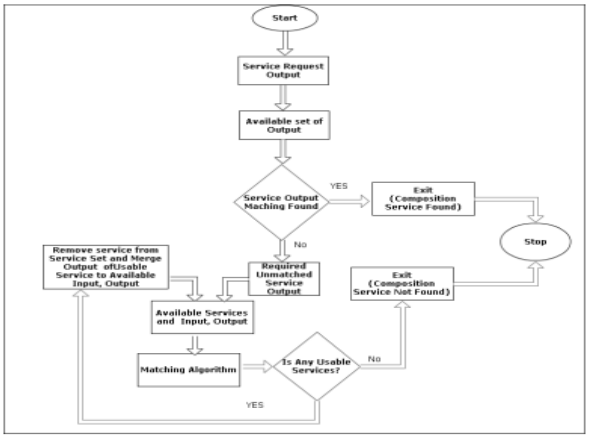

Fig. 1 Flow Chart of Forward Chaining Approach

B. Backward Chaining Approach (BCA)

Backward Chaining Approach starts with the goal and works backward to convert it into sub-goals until it can be achieved. Backward Chaining Approach for dynamic service composition is introduced by [9] and is followed by [11]. Figure 2 shows the flow chart of BCA. Initially required output is same as output required by service request. During the execution of procedure, matching algorithm finds services that can provide any of the output present in required output set. Services found by matching algorithm are marked as relevant services (Definition 4). If all input parameters required by relevant service are present in available input then it is also marked as usable service. Usable relevant services also mark its output as usable and add them to currently available input set. An unusable relevant service adds its unusable input (input which is not present in available input) to the required output set. Procedure is repeated again to execute matching algorithm with the updated required output set. Procedure stops when output required by service request is present in the available input. In this case, there exists a composite service which can satisfy the service request. A backward search (as mentioned in Section III-C) is performed to get the composite service from the usable services. Procedure also stops if matching algorithm cannot find any relevant service.

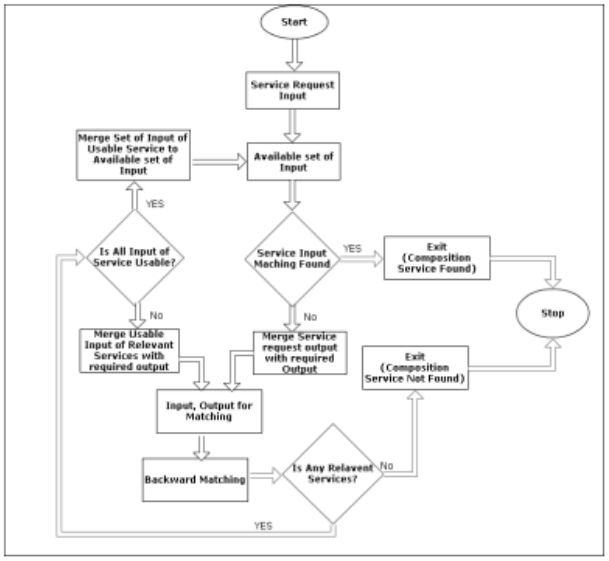

Fig. 2 Flow Chart of Backward Chaining Approach

C. Backward Search

Backward search is performed to generate composition plan and composite service [9], [10], [11], [12] from the candidate services. Candidate services are services which found to be relevant to service request and can be a part of the required composite service. Algorithms like BCA, FCA and proposed algorithm performs the main time consuming task of finding such candidate services.

Backward search only considers the services that are usable as candidate services to take part in composition plan. Initially outputs required by service request are added to the set of required outputs. Procedure starts by finding the services, from candidate services, that can provide a set of

DOI: 10.5963/IJCSAI0203004

required output. These services are added to composition plan. These services are usable and can be executed directly or indirectly using the input provided in the service request. This allows us to remove all the outputs provided by these services from the set of required output. This in turn will empty the required output set. This is required because when composite service exists then we would have at least one service for each of the entries in a set of required output. Now, we have to find the service that can provide the input required by services added to composition plan. This can be done by reinitializing the required output set by input required by services which were added to the composition plan. Only that input is added, which are not provided by the service request. Procedure is repeated with a current set of required output. Procedure stops when required output set becomes empty and composition plan contains a chain of existing services, which can satisfy the service requested by user.

A situation may occur where one of the required outputs is provided by multiple services. This situation is resolved by considering quality of service attribute specified in service description. In proposed work, response time (Quality of service attribute) is used. Service which provides the minimum response time will be selected.

D. Open Issues in FCA and BCA

FCA stores and processes all usable services. All usable services do not help to get the output required by a service request. These services are referred to as irrelevant services. Ideally FCA should process only usable and relevant services but it does not know which usable services are relevant until it performs last iteration where it finds a service that gives output required by service request. FCA performance suffers from the overhead of storing and processing irrelevant services. Number of usable and irrelevant services among them increase in each iteration. This overhead in turn increases the composition time required to find a composite service. Performance of forward chaining approach suffers significantly when usable services, and thus irrelevant services among them, increases exponentially with each iteration [13], [9]. Experiment 1 shown in Section VI-A shows such a case.

Similarly in BCA, all relevant services cannot be executed using the input provided by service request. These services are referred as unusable services. Ideally BCA should process only relevant and usable services, but it does not know which relevant services are usable until it performs last iteration where it finds services that can be executed using input provided by a service request. BCA performance suffers from overhead of storing and processing unusable services. Number of relevant services, and unusable services among them increase with each iteration. This overhead in turn increases the composition time required to find a composite service. Performance of backward chaining approach suffers significantly when relevant services, and thus unusable services among them, increase exponentially with each iteration [13], [9]. Experiment 2 shown in Section VII shows such a case.

IV. DOCUMENT SPECIFICATION FOR PROPOSED APPROACH

This section contains the required documents specification and formats for proposed service composition approach, which is based on the forward chaining approach and backward chaining approach. Section IV-A defines the data structure used by the proposed algorithm. Section IV-B introduces the document format for service and service request used by matching algorithm. Section V describes the proposed service composition algorithm.

A. Proposed Data Structure AND Definition

Data structure of parameters used by proposed algorithm is as shown in Table III. This data structure contains four types of information (i) parameter name (ii) its service consumers, i.e. services having parameter as its input (iii) and its service providers, i.e. services having parameter as its output and (iv) a variable, named IsUsableParameter, that indicates whether a parameter is usable or not according to Definition 5.

TABLE III DATA STRUCTURE OF A PARAMETER

```
Class Parameter
{
String ParameterName;
Service[] SCSet; // a set of service consumers
Service[] SPSet; // a set of service providers
Boolean IsUsableParameter = false;
}
```

For example, from Table I, parameter ‘CropVariety’ and ‘Quantity’ will have service named ‘GetStorageLocation’ mentioned in SCSet as one of its service consumers and ‘StorageLocation’ will have it in SPSet as one of its providers.

Definition 5—Usable Parameter (UP): A parameter becomes usable parameter if it is either directly provided with service request or indirectly provided by one of the services. This service is one among services, in service provider list of a parameter, who’s all input parameters are usable parameter. When a parameter becomes usable its IsUsableParameter variable is set to true.

Proposed algorithm maintains a list of parameters. This list is virtually divided in two parts: (i) available input (ii) required output. Parameters which are usable (Definition 5) are considered as a part of available input set. Parameters which are not usable are considered as a part of required output set. Initially input and output of service request are added to available input and required output respectively. Input is added by the user as its service provider and output is added by the user as its service consumer. Available input and required output are updated by the input/output of services, foundling each iteration of proposed algorithm.

B. Document Formats

1) Service Advertisement Document:

Service advertisement document for service illustrated in Table I is given in Table IV. Service name is mentioned in

DOI: 10.5963/IJCSAI0203004

the name attribute of <Service>. <Service> has <Input>, <Output> and <QoS> as child nodes corresponding to input parameter, output parameter, and a set of quality of service attributes of a service. <QoS> has several child tags named with corresponding quality of service attribute, e.g. Response Time.

TABLE IV SERVICE ADVERTISEMENT DOCUMENT

```
<Service name="GetStorageLoaction">
<Input>CropVariety</Input>
<Input>Quantity</Input>
<Output>StorageLocation</Output>
<QoS>
<ResponseTime>100</ResponseTime>
</QoS>
</Service>
```

2) Service Request Document:

Service request document for service illustrated in Table II is given in Table V. Service request description is encapsulated inside <ServiceRequest>. <ServiceRequest> has <Input> and <Output> as child nodes corresponding to input parameter, output parameter of a service request.

TABLE V SERVICE REQUEST DOCUMENT

```
<ServiceRequest>
<Input>CropVariety</Input>
<Input>Quantity</Input>
<Output>StorageOffer</Output>
</ServiceRequest>
```

V. PROPOSED ALGORITHM AND PROCEDURES USED

Service Composition algorithm (Algorithm 1), as shown in Table VI, performs the iteration in direction of forward chaining and backward chaining alternatively. A service request (Definition 2) contains the information about the input provided by the user and output required by the user. As previously discussed, information about the current status of available input parameters and required output parameters is maintained. Initially, parameters provided with the service request are considered as usable parameters and added to available input parameters.

Parameters required by the service request are considered as unusable parameters and added to required output parameters. We say that service request is satisfied when all the required output parameters of a service request are present in the available input set (line 5, 7 and 9). At this time, backward search as specified in section III-C is performed to find the composite service.

Service composition algorithm (Algorithm 1) first executes the iteration in forward chaining (Algorithm 2) direction as shown in Table VII. Forward chaining finds the services whose all input parameters are present in the current available input set. These services are set as usable services (Definition 3) by Algorithm 3 shown in Table VIII. Usable services add its output parameters to the available input set. Algorithm also inserts usable services as service provider of the current parameter according to Definition 5. This may cause some unusable services in the list of service consumer of a parameter to become usable service. This happens if all other input parameters required by a service consumer are already present in the available input set. These services are also marked as usable services by recursively calling by Algorithm 3 shown in Table VIII.

Service Composition Algorithm (Algorithm 1) checks whether service request can be satisfied after executing the iteration of forward chaining. Algorithm stops if service request is satisfied otherwise it continues to execute the next iterations. Once algorithm stops then backward search is performed to find out composite services from the current set of usable services.

In the next phase, Service Composition Algorithm (Algorithm 1) executes iteration in backward chaining (Algorithm 4) direction as shown in Table IX. Backward chaining finds the services which provide any of the parameters in the list of required output parameters. These services are marked as relevant services using Algorithm 5 shown in Table X.

If all the input parameters required by a relevant service are present in the available input set then that relevant service is also marked as usable service using Algorithm 3 shown in Table VIII otherwise input parameters which are not present in the available input set are added to the required output set.

TABLE VII ALGORITHM BASED ON FCA

ALGORITHM 2: Forward Chaining

INPUT : A list of available input and Service set

OUTPUT: Services who's all input parameter are present in list of available input and mark them as usable services. If any usable service found then return true else return false.

```
1. begin
2.   Find the services (from unusable services in service set)
       whose all input parameters are present in the list of
       available input
3.    if Any service found in step 2
4.          then
5.               for each Service S found in step 2
6.                    begin
7.                        Set Service Usable(S);
8.                end
9.          return true;
10.      else
11.          return false;
12.    end if
13. end
```


DOI: 10.5963/IJCSAI0203004

TABLE VIII ALGORITHM TO MARK USABLE SERVICES

```
ALGORITHM 3: Set Service Usable
GLOBAL : A list of available input and Service set
INPUT : A Service S to set usable
OUTPUT : Set  Service  S to usable
1. begin
2.      Mark the service S as usable service
3.        for each Parameter P in output of S
4.           begin
5.        if (P ∉ set of available input or required output)
6.            then
7.                Add P in available input
8.         end if
9.        Add S in service provider list of P
10.       for each Service SC in service consumer of P
11.          begin
12.               if all input of SC is present in available
input
13.               then
14.                    Set Service Usable(SC);
15.            end if
16.        end
17.     end
18. end
```

TABLE IX ALGORITHM BASED ON BCA

```
ALGORITHM 4: Backward Chaining
INPUT : A list of required output and Service set
OUTPUT: Services can provide any of the current required
output are found, and mark them as relevant service. If any
relevant service found then return true else return false.
1. begin
2.  Find the services (from irrelevant services in service
    set) which provides any of the parameter in the list of
    required output parameter
3.        if Any service found in step 2
4.           then
5.              for each Service S found in step 2
6.                  begin
7.                     Set Service Relevant(S);
8.                  end
9.              return true;
10.           else
11.              return false;
12.        end if
13. end
```

TABLE X ALGORITHM TO MARK RELEVANT SERVICES

```
ALGORITHM 5: Set Service Relevant
GLOBAL : A list of required output and Service set
INPUT : A Service S to set relevant OUTPUT : Set
        Service S to relevant
1. begin
2.      Mark the service S as relevant service
3.          for each Parameter P in input of S
4.              begin
5.                  if P does not exists in required
                    output or available input
6.                      then
7.                          Add P in required output
8.                  end if
9.                Add S in service consumer list of P
10.             end
11.         if all input of S is present in available input
12.             then
13.                 Set Service Usable(S);
14.         end if
15. end
```

After executing the iteration of backward chaining, service composition algorithm (Algorithm 1) checks whether service request can be satisfied. If service request is satisfied then backward search is performed to find the composite services from the current set of usable services. In this case, algorithm stops at this point, otherwise it continues to execute the next iterations.

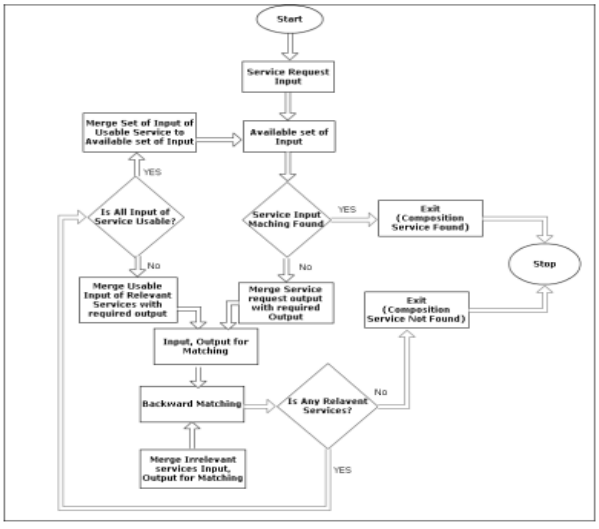

Fig. 3 Flow Chart of Proposed algorithm

Service composition algorithm is stopped, if either forward chaining does not find any usable services or backward chaining does not find any relevant services. This case can occur when there is no chain of existing services which can provide the required output from input provided by the user. Service composition performs iteration of forward chaining (Algorithm 2) and backward matching

DOI: 10.5963/IJCSAI0203004

(Algorithm 4) alternatively until service request can be satisfied. Figure 3 shows the entire procedure followed by proposed service composition algorithm in the form of flow chart.

VI. EXPERIMENTS

We have used the service set shown in Appendix-A and Appendix-B. In real world, each service is represented by a Web Service Definition Language (WSDL) file that contains lots of information about service. Parsing WSDL file to get the required information makes implementation of algorithm much complex and shifts the focus of algorithm from its main task of finding composite service. Hence a service advertisement document as mention in section IV-B1 containing only the information necessary for algorithm is prepared. Main focus of the experiment is to highlight issues with FCA and BCA, and how proposed algorithm performs efficiently in given scenarios. Results of proposed algorithm are compared with FCA algorithm given by [10] and BCA given by [11] as they achieved the first and second rank in WSC'2009 competition for their work on reducing composition time among all other algorithms.

A. Experiment 1

FCA, BCA and proposed algorithms are executed to find the composite service for the service request shown in Table XI. Service set used for this experiment is shown in Appendix A.

TABLE XI SERVICE REQUEST 1

Service Request-1 based on Service Set-1	
Input	CropVariety, Quantity
Output	StorageOffer

TABLE XII COMPARISON OF SERVICES (USABLE AND RELEVANT) EXPLORED IN EACH ITERATION TO FIND A COMPOSITE SERVICE FOR SERVICE REQUEST AS MENTIONED IN TABLE XI

Iteration No.	**FCA**	**BCA**		**Proposed Algorithm**	
	Usable Service	**Usable Service**	**Relevant Service**	**Usable Service**	**Relevant Service**
1	{1}		{18}	{1}	
2	{2,3}		{8,9}		{18}
3	{4,5,7}		{4}	{2,3}	
4	{8,9,10, 11,12,15, 16,17 }		{2}		{8,9}
5	18,19,20, 21,22,23, 24,25,30, 31,32,33}	{1,2,4, 8,9,18}	{1}	{4,5,7, 8,9,18}	
Total Services Processed	26	6		9	

Matching algorithm finds the services based on exact match between two words in keyword base composition. Services explored by FCA, BCA and proposed algorithm in each iteration are shown in Table XII. As discussed earlier, the proposed algorithm performs the iteration in direction of FCA and BCA alternatively. Thus, the usable and relevant services found by proposed algorithm in Iterations 1 and 3 are same as usable services found by FCA in Iterations 1 and 2 respectively. Output parameters of these services are added to available input with their service provider information (Definition 5). Similarly proposed algorithm follows BCA in Iterations 2 and 4 and the relevant services found are same as relevant services found by BCA in Iterations 1 and 2. Input parameters required by these services are added to required output and service is added to its service consumer set.

FCA stops when service number 18 is explored since it provides the required output. In case of BCA, required output is initialized with the output required by service request. BCA starts with the services that provide the required output and mark them as relevant services. It selects Service 18, which provides the required output. BCA finds that input parameters required by Service 18 are not present in available input, i.e. it is unusable service, so it adds it to the required output and BCA executes again with updated required output. BCA stops when it finds the service number 1 as input required by this service is provided by user. Hence, Service 1 becomes usable and adds its output to available input, which in turn causes Service 2 to become usable and so on Services 4, 8, 9 and 18 becomes usable. After that BCA stops since Service 18 provides required output by service request.

Proposed algorithm does not have to perform all the iterations performed by FCA or BCA. In the last iteration, proposed algorithm performs forward matching, where it finds Services 4, 5 and 7 (same as in 3rd iteration of FCA). Thus, output parameters of these services are marked as usable. As this parameter becomes usable, proposed algorithm checks its service consumers and finds the Services 8 and 9, so it checks whether these services become usable services due to transition of current parameters from unusable to usable. Algorithm finds that all input parameters required by Services 8 and 9 are usable parameters and they are usable services. Thus, Service 8 and 9 are marked as usable services, and their output parameters are marked as usable. This in turn makes the relevant Service 18 to become usable service, which provides the required output of service request. Here, required output by service request becomes usable and proposed algorithm stops.

After the required output becomes usable, backward search is performed. All the unusable parameters are removed from required output. The unusable services in the service provider and service consumer list of usable parameters are removed before backward search is applied. Backward search starts from the required output by service request and finds that Service 18 is in service provider list. It adds Service 18 to composition plan. Required output is reinitialized with input required by 18. Parameters which are provided by the input of service request are removed from the required output set since they are already usable and there is no need to search for its service providers. Backward search continues with the current required output set. It finds that Services 8 and 9 both provide the required output. Here, Service 8 is selected since response time of

DOI: 10.5963/IJCSAI0203004

Service 8 is less than Service 9. Service 8 is added to the composition plan and procedure continues until required output becomes empty. At the end, composition plan includes the Services {1, 2, 4, 8, 18}.

Table XII shows the number of services explored by FCA, BCA and proposed algorithm. A graph shown in Figure 4 indicates how the number of services explored increases as number of iterations increases. Proposed algorithm follows bidirectional approach where it follows half number of steps of FCA from one direction which finds usable services and other half number of steps of BCA from other direction which finds relevant services. For this experiment, proposed algorithm gets the advantage over BCA but suffers compared to FCA. Graph shown in Figure 4 shows that for Service Request 1, number of services explored by proposed algorithm is much less than FCA and comparable to BCA. This shows that performance of FCA suffers compared to proposed algorithm and BCA, in cases where the number of usable services increase rapidly in each iteration. BCA gives optimal performance since relevant services increase slowly in each iteration. In Experiment 2, we have shown that just like FCA, BCA also suffers in certain situations, where relevant services increases rapidly with each iteration. The proposed algorithm gives consistent performance across all the cases.

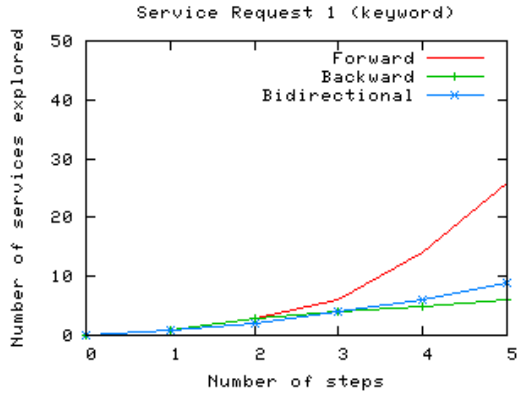

Fig. 4 Graph of number of services explored by FCA, BCA and proposed algorithm for service request mention in Table XI

A graph shown in Figure 5 compares the time required by FCA, BCA and proposed algorithm. It can be seen that proposed algorithm requires much less time than FCA and similar to that of BCA.

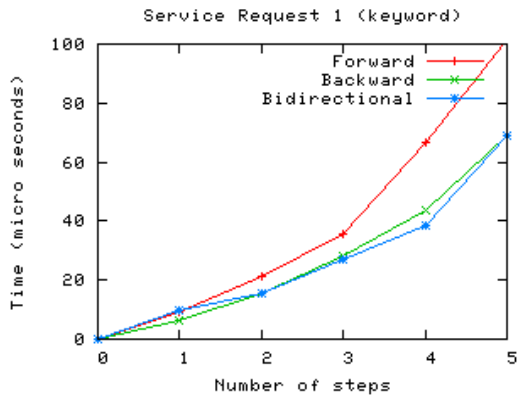

Fig. 5 Graph of composition time required by FCA, BCA and proposed algorithm for service request as mentioned in Table XI

B. Experiment 2

FCA, BCA and proposed algorithm are executed to find the composite service for the service request shown in Table XIII. Service set used for this experiment is shown in Appendix-B. Services explored by FCA, BCA and proposed algorithm in each iteration are shown in Table XIV. After finding the candidate services, backward search is applied, which generates composite services {17, 18, 19, 3, 1}.

TABLE XIII SERVICE REQUEST-2

Service Request - 2 based on Service Set - 2	
Input	CropVariety, Quantity
Output	TransportationPrice

TABLE XIV COMPARISON OF SERVICES (USABLE AND RELEVANT) EXPLORED BY FCA, BCA AND PROPOSED ALGORITHM IN EACH ITERATION TO FIND A COMPOSITE SERVICE FOR SERVICE REQUEST AS MENTIONED IN TABLE XIII

Iteration No.	**FCA**	**BCA**		**Proposed Algorithm**	
	Usable Service	**Usable Service**	**Relevant Service**	**Usable Service**	**Relevant Service**
1	{17,18, 19}		{1}	{17,18, 19}	
2	{5,6,7}		{2,3,5,7}		{1}
3	{1}	{17,18, 19,5,7, 1}	{8,9,10, 11,12,13, 14,15,17, 18,19}	{5,6,7, 1}	
Total Services Processed	7	16		7	

Graph, in Figure 6 and Figure 7, shows comparison of number of services explored and composition time required by FCA, BCA and proposed algorithm. It shows number of services explored and composition time required by proposed algorithm is less than BCA and comparable to FCA

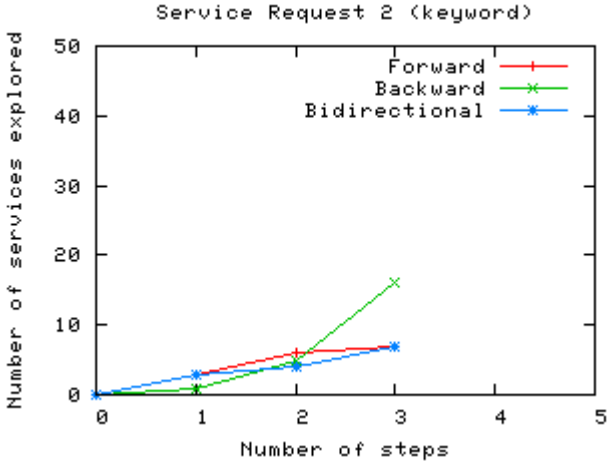

Fig. 6 Graph of number of services explored by FCA, BCA and proposed algorithm for service request as mentioned in Table XIII

DOI: 10.5963/IJCSAI0203004

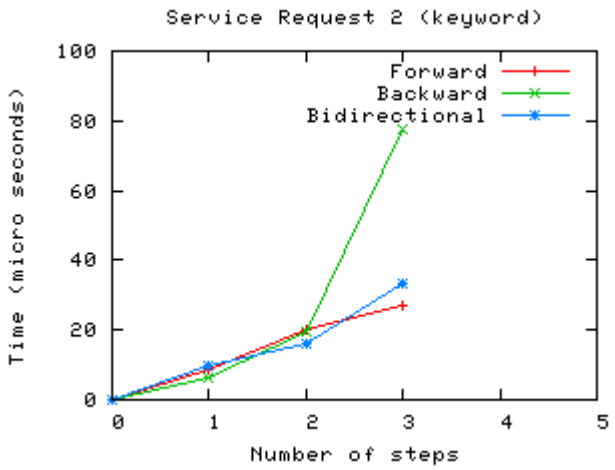

Fig. 7 Graph of composition time required by FCA, BCA and proposed algorithm for service request mention in Table XIII

VII. SUMMARY OF RESULTS

Results of the experiments performed on for service composition are summarized as under. Overall performance of the FCA, BCA and proposed algorithm based on keywords matching, to find a composite service is evaluated. Table XV shows the total number of services explored for experiment 1 and Experiment 2. It also shows the average number of services explored by each approach. It can be concluded from the Table XV that proposed algorithm explores less number of services than FCA or BCA.

TABLE XV TOTAL NUMBER OF SERVICES (USABLE AND RELEVANT) EXPLORED BY FCA, BCA AND PROPOSED ALGORITHM FOR EXPERIMENT 1 AND EXPERIMENT 2

Type of Approach	Composition Time (µs)			
	Experiment -1 (Figure 5)	Experiment -2 (Figure 7)	Total	Average
FCA	101	6	107	53
BCA	68	58	126	63
Proposed Approach	68	17	85	42

TABLE XVI COMPOSITION TIME BY FCA, BCA AND PROPOSED ALGORITHM IN EXPERIMENT 1 AND EXPERIMENT 2

Type of Approach	Number of services explored			
	Experiment-1 (Table XII)	Experiment-2 (Table XIV)	Total	Average
FCA	26	7	33	16
BCA	6	16	22	11
Proposed Approach	9	7	16	8

Table XVI shows the composition time required for Experiment 1 and Experiment 2. It also shows the average composition time taken by each of the approach. It can be concluded from Table XVI that overall composition time required by proposed algorithm is less compared to FCA or BCA. With BCA more number of relevant services become usable compare to our proposed algorithm (from Table XII), which consumes more processing time. So time required by BCA is more than proposed algorithm even if same number of services is explored.

VIII. CONCLUSIONS

Most algorithms present in the literature to find a composition plan dynamically either follows FCA or BCA. It can be seen from the experiment that FCA and BCA suffer severely in certain cases as shown in Experiment 1 and Experiment 2 respectively.

Performance of algorithm based on only FCA or BCA suffers due to the fact it only finds usable services or relevant services respectively which increases exponentially in each iteration. FCA does not know which of the usable services will help to get the output required by service request until it finally gets them in the last iteration. Similarly, BCA does not have knowledge of which relevant services can be executed using the input provided by service request until the last iteration.

Proposed algorithm follows iteration in two directions, one follows FCA and other follows BCA alternatively. Algorithm following only FCA or BCA performs all the iterations in one direction, where as the proposed algorithm requires only half number of iterations in both the directions. This improves the performance due to the fact discussed above that the number of services explored increases exponentially as number of iterations increases. Thus, proposed algorithm is better than FCA or BCA in terms of (i) Number of services to be explored and (ii) time required to discover a composition plan.

Experiments uses smaller service set but it can be very well seen that same scenarios will occur even in larger service set. Scalability of the proposed algorithm can be realized from the discussion made in Section I-A, there is no doubt on scalability. Performance increase may seem minimal in our results but as the length of composition plan increases the performance improvement increases. This can be realized by looking at improvement in composition time by the proposed algorithm with respect to FCA and BCA in Experiment 1 where length of composition plan is 5, is higher than in Experiment 2 where length of composition plan is 3.

IX. FUTURE WORK

Matching algorithm does not consider semantic or context while finding the usable services. A semantic and context based matching algorithm can be developed to identify services more effectively. Backward search takes the decision of selecting services, when multiple services provide the same required output based on the local information of quality of service attribute. This selection may not be correct for the overall composite service. Selection based on the global information of the entire composition plan can be considered as a future direction. An update in service should be propagated to all the composite

DOI: 10.5963/IJCSAI0203004

services that use corresponding service as a part of it. An approach of how to keep composite service up to date is also an interesting research area.

REFERENCES

[1] R. Hull, M. Benedikt, V. Christophides, J. Su, "E-services: A Look Behind the Curtain", in Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 1-14, 2003.

[2] P. Patel, S. Chaudhary, "Context Aware Semantic Service Discovery", in Services Part II, IEEE Congress on Services, pp. 1-8, 2009.

[3] G. Alonso, F. Casati, H. Kuno, V. Machiraju, "Web Services: Concepts, Architectures and Applications", in Springer Verlag, pp. 1-150, 2004.

[4] Z. Laliwala, R. Khosla, P. Majumdar, S. Chaudhary, "Semantic and Rules Based Event-Driven Dynamic Web Services Composition for Automation of Business Processes", in IEEE Services Computing Workshops, pp. 175-182, 2006.

[5] T. Broens, S. Pokraev,M. Sinderen,J. Koolwaaij, P. Costa, "Context-Aware, Ontology-Based Service Discovery", in EUSAI, pp. 72-83, 2006.

[6] A. Alamri, M. Eid, A. El Saddik, "Classification of the state-of-the-art Dynamic Web Services Composition Techniques", in The Semantic Web ISWC 2002, pp. 148-166, 2006.

[7] S. Dustdar, W. Schreiner, "A Survey on Web Services Composition", in International Journal of Web and Grid Services, pp. 1-30, 2005.

[8] J. Rao, X. Su, "A Survey of Automated Web Service Composition Methods", in Semantic Web Services and Web Process Composition, pp. 43-54, 2005.

[9] N. Milanovic, M. Malek, "Search Strategies for Automatic Web Service Composition", in International Journal of Web Services Research, pp. 1-32, 2006.

[10] P. Bartalos, M. Bielikov´a, "Semantic Web Service Composition Frame-work Based on Parallel Processing", in IEEE Conference on Commerce and Enterprise Computing, pp. 495-498, 2009.

[11] M. Aiello, E. el Khoury, A. Lazovik, P. Ratelband, "Optimal QoS-aware Web Service Composition", in Proceedings of the 2009 IEEE Conference on Commerce and Enterprise Computing-Volume 00, pp. 491-494, 2009.

[12] Z. Huang, W. Jiang, S. Hu, Z. Liu, "Effective Pruning Algorithm for QoS-Aware Service Composition", in Proceedings of the 2009 IEEE Conference on Commerce and Enterprise Computing-Volume 00, pp. 519-522, 2009.

[13] U. K¨uster, M. Stern, B. K¨onig-Ries, "A Classification of Issues and p- approaches in Automatic Service Composition", in International Workshop on Engineering Service Compositions, 2005.

[14] Web Services Challenge 2009, http://wschallenge.georgetown.edu/wsc09/index.html, IEEE Conference on Commerce and Enterprise Computing 2009, Vienna, Austria, July 20-23, 2009.

[15] Xudong Song, Wanchun Dou, Jinjun Chen, "A workflow framework for intelligent service composition 2010", in Future Generation Computer System, Elsevier, July 2010.

[16] Frederico G. Alvares de Oliveira, Jose M. Parente de Oliveira, "QoS- based Approach for Dynamic Web Service Composition", in Journal of Universal Computer Science, vol. 17, no. 5 (2011)

[17] Rui Wang, Chaitanya Guttula, Maryam Panahiazar, Haseeb Yousaf, John A. Miller, Eileen T. Kraemer and Jessica C. Kissinger, "Web Service Composition using Service Suggestions", in IEEE World Congress on Web Services 2011.

Sandip Khakhkhar is currently working in Cisco, before joining cisco he worked as a M.Tech. student at Dhirubhai Ambani Institute of Information and Communication Technology (DA-IICT), Gandhinagar, India. He has completed M.Tech. from DA-IICT in 2010. His areas of research interest are distributing computing and Service oriented architecture.

Vikas Kumar has done B.Sc. from MJP Rohilkhand University Barailly and MCA from UPTU Lucknow. Currently he is a Ph.D. candidate at Dhirubhai Ambani Institute of Information and Communication Technology (DA-IICT), Gandhinagar, India. His area of research interest are content mining, semantic web, service oriented architecture and social network analysis.

Sanjay Chaudhary is a Professor and Dean (Academic Programs) at Dhirubhai Ambani Institute of Information and Communication Technology (DA-IICT), Gandhinagar, India. His research areas are Distributed Computing, Service-Oriented Computing, and ICT Applications in Agriculture. He has authored four books and a number of book chapters. He has published a number of research papers in international conferences, workshops and journals. He is an active member of program committees of leading International conferences and workshops. He is also a member of review committees of leading journals. He has received research grants from leading companies including IBM and Microsoft.

DOI: 10.5963/IJCSAI0203004

Appendix A : Service Set 1

Service ID	Response Time	Service Name	Service Input			Service Output
1	400	GetStorageLocation	Variety	Quantity		StorageLocation
2	300	GetSellerLocation	Variety	Quantity		SellerLocation
3	800	GetTransportLocation	Variety	Quantity		TransportLocation
4	500	GetStorageID	StorageLocation	Variety		StorageID
5	800	GetSellerID	SellerLocation	Variety		SellerID
6	400	GetTransportID	TransportLocation			TransportID
7	500	GetSellerType	SellerID	Variety		SellerType
8	700	GetMechantType	MerchantID	Variety		MerchantType
9	400	GetStorageType	StorageID	Variety		StorageType
10	600	GetTransportationType	TransportID	Quantity		TransportType
11	400	GetDiscountFromSeller	SellerType	Variety	Quantity	SellerDiscount
12	400	GetVarietyPriceFromSeller	SellerType	Variety	Quantity	VarietyPrice
13	600	GetDiscountFromMechant	MerchantType	Variety	Quantity	MerchantDiscount
14	600	GetVarietyPriceFromMechant	MerchantType	Variety	Quantity	VarietyPrice
15	700	GetStorageDiscount	StorageType	Variety	Quantity	StorageDiscount
16	800	GetStoragePrice	StorageType	Variety	Quantity	StoragePrice
17	400	GetStorageCost	StorageType	Variety	Quantity	StorageCost
18	700	GetTransportCost	TransportType	Variety	Quantity	TransportCost
19	200	GetTransportDiscount	TransportType	Variety	Quantity	TransportDiscount
20	300	GetTransportPrice	TransportType	Variety	Quantity	TransportPrice
21	400	GetStorageOffer	StoragePrice	StorageDiscount		StorageOffer
22	600	GetMerchantOffer	VarietyPrice	MerchantDiscount		MechantOffer
23	500	VarietyPrice	SellerDiscount			SellerOffer
24	300	TransportPrice	TransportDiscount			TransportOffer

Appendix B : Service Set 2

Service ID	Response Time	Service Name	Service Input		Service Output
1	600	GetMarketTransportationPrice	TransportationPricePerUnit	Quantity	TransportationPrice
2	800	GetMarketTransportationPricePerUnit	MarketTransportationID1	TransportationType	TransportationPricePerUnit
3	400	GetMarketTransportationPricePerUnit	MarketTransportationID2	TransportationType	TransportationPricePerUnit
4	400	GetMarketTransportationPricePerUnit	MarketTransportationID3	TransportationType	TransportationCostPerUnit
5	700	GetMarketTransportationPricePerUnit	TransportationID	TransportationType	TransportationPricePerUnit
6	400	GetMarketTransportationPricePerUnit	TransportationID	TransportationType	TransportationCostPerUnit
7	800	GetMarketTransportationPricePerUnit	MarketTransportationID	TransportationType	TransportationPricePerUnit
8	400	GetTransportationID1	Variety1	Quantity1	TransportationID
9	500	GetTransportationType1	Variety1	Quantity1	TransportationType
10	600	GetMarketTransportationID1	Variety1	Quantity1	MarketTransportationID1
11	500	GetTransportationID2	Variety2	Quantity2	TransportationID
12	800	GetTransportationType2	Variety2	Quantity2	TransportationType
13	400	GetMarketTransportationID2	Variety2	Quantity2	MarketTransportationID2
14	600	GetTransportationID3	Variety3	Quantity3	TransportationID
15	300	GetTransportationType3	Variety3	Quantity3	TransportationType
16	700	GetMarketTransportationID3	Variety3	Quantity3	MarketTransportationID3
17	500	GetTransportationID	Variety	Quantity	TransportationID
18	400	GetTransportationType	Variety	Quantity	TransportationType
19	500	GetMarketTransportationID	Variety	Quantity	MarketTransportationID

DOI: 10.5963/IJCSAI0203004