AN APPROACH FOR GRAPH BASED PLANNING AND QUALITY DRIVEN COMPOSITION OF WEB SERVICES

S. Justin Samuel[1]

Research Scholar of Sathyabama University, Sathyabama University,
Rajiv Gandhi Road, Chennai
Tamil Nadu. 600119, India
chennaisjs@gmail.com

Dr. T. Sasipraba[2]

Professor and Dean, Sathyabama University,
Rajiv Gandhi Road, Chennai,
Tamil Nadu, 600119, India
tsasipraba@gmail.com

Abstract

Service Oriented Architectures enable a multitude of Web services to provide loosely coupled and interoperable services at different Quality of Service level in the recent few years. Dynamic Web services composition offers the opportunity for creating new web services at runtime from those already published in UDDI registry. However, this composition requires necessarily quantifying criteria for Web service selection to build a more appropriate one. As the result, the problem of synthesizing web services of high quality is a prominent issue. To resolve the problem of finding the best composition, quality based graph planning method is used. First we construct the composition planning graph and secondly by applying non-functional quality parameters the best composition schema is found by using iterative method. It provides a unique search space and also it guarantees to find a best solution to a solvable problem.

***Keywords*:** Web Service Composition; Composition Schema; Composition Planning.

1. Introduction

The Web Services Composition (WSC for short) can be done in a static or dynamic way. The static composition allows the requestor to create an abstract model that should be respected during the execution of this Web service. While the dynamic composition enables selecting the atomic Web services automatically and combines them to create an unlimited number of new Web services. Web service composition focuses on synthesizing individual, outsourced web services to create a new service. It aims to provide more value added service than single web services. By choosing appropriate web services offered by different web service providers, specifying their coordination plan, and implementing the plan through an orchestration engine, the composite web service can provide more valuable and complete service than a single web service. It can also reuse individual web services more efficiently. Also web service composition creates new functionalities by aggregating different services based on a specific workflow [1]. When there are more than one candidate web services for each task or process, there will be various combinations of web services having the same functionality with different quality. For instance, if there are m tasks and n candidate web services, the number of all possible plans is n^m. Problem of web service composition is usually an NP-hard [2]. We have to find a suitable composition plan according to the client's constraints without checking all combinations even if there are a few services and tasks in the workflow.

In this paper, we consider a simple version of the above composition problem, as a problem of receiving some inputs and returning some outputs as the result. Therefore, the order of actions is not important, except that we assume inputs happen before the outputs. Many information systems are coming under this category and the inputs and outputs can be retrieved from the respective wsdl files. Here we consider the impact of preconditions and effects, and also nonfunctional requirements. It is important to note that dealing with preconditions and effects is a necessary step in solving the composition problem. We use a specific graph structure, called dependency graph, to build an index for the available web services and their input-output information. It can be considered as a model for the specification repository of Figure 1, as it is accessed by the composition planner to answer a request. This paper presents a web service composition based on weighted planning graph by which the composition can be found in polynomial time and in the heterogeneous dynamic environment. First we find construct the composition planning graph and secondly the best composition schema is found by using iterative method. It provides a unique search space and experiments show it guarantees to find a solution to a solvable problem. In the next section, we present some of the related works on web services composition. The composition planning architecture and the composition schema are described in detail in Section 3. In Section 4, we briefly report about the planning graph and the related definitions. In Section 5 we present our composition approach. Section 6 we present the result of experimental studies. Finally, section 7 concludes the paper.

2. Related Work

Many composition papers are available to study about the problem presented in this paper. However, they are normally different based on the type of the web services they consider, information they capture from each web service, and the level of abstraction or concreteness of their solution. In this section, we briefly mention some of these works. Benatallah et al. in [3] propose an approach for facilitating large-scale interoperation of web services. As the main feature of their work, they take advantage of service communities as the repositories of web services with similar functionality. The workflow, preconditions and effects of each web service are modeled by a state chart, each state of which is one of the constituting web services. In [4] the authors model the semantic web service composition problem as a kind of simplified planning graph. It finds a solution in polynomial time with possible redundant services and does not deal with the quality of services. Services for a request are found using the graph and process algebra in [5], and also it explains how to compose them to obtain the expected behavior. This paper introduces process algebra to specify the behavior of composite web services based on the behavior of simpler ones. Jong, Chang, Ick-Hyun have proposed an approach for the web service composition problem in [6]. They have designed an execution plan optimizer which runs the web service composition algorithm in order to generate a QoS-oriented composition plan. The performance of the algorithm has been tested in a simulated environment. In [12], Huan and Bang have proposed a web service composition approach which is similar to our work with different implementation. Mohamed, Maroua and Abdel-Illah have proposed a Graph plan algorithm in [13] to solve simple problems but not for complex problems. In [14], authors Wenqiang Li, Xuemei, Hao have used service matching degree as quality weights but they have not used the non-functional parameters of web services as the quality weights.

3. Composition Architecture

All the composition problems have to be solved by building a web service composition engine in practice. A web service composition engine is supposed to process available web services and extract their behavioral information, accept requests for new services, and build a valid composite service for the request, if possible. The architecture of such a web service composition engine is depicted in Figure 1.

Plan Generator: It is the heart of the composition engine. When a request for a new service is submitted to the engine, the composition planner extracts the corresponding behavior and tries to satisfy that behavior by composing some of the available web services whose information are stored in the repository. If successful the schema is stored in the Schema repository for reference.

Composition Schema: It is a workflow of services which consists of atomic processes, sub-composite processes if any, and their control flows. The composition schema repository is the knowledge base which contains the necessary information about the web services and their schemas. A composite plan is a realized composition schema in which each abstract service is replaced with the suitable concrete service.

Execution engine: When the composite plan with quality constraints is ready for the given request, this component becomes responsible for executing the composition found. It simply manages the order of execution of involved repository services along with any necessary data passing between them.

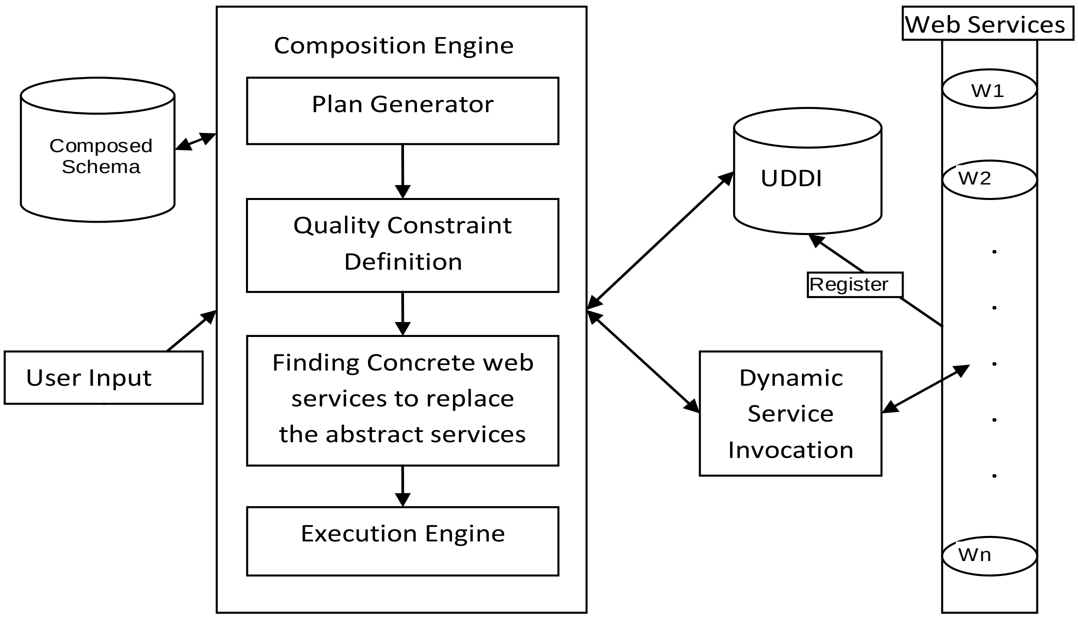
Composition Engine
Plan Generator
Quality Constraint Definition
Finding Concrete web services to replace the abstract services
Execution Engine
Composed Schema
User Input
UDDI
Register
Dynamic Service Invocation
Web Services
W1
W2
Wn

Fig.1 Web Service Composition Architecture Diagram

There are two ways to get the concrete service. One is, by UDDI where all the service providers would have registered their web services. Second, during execution if the execution engine want to modify any concrete service in the composition plan it can dynamically look up for services from the web and can be used to complete the execution. Developing a full-fledged OWL-S execution engine that supports the dynamic invocation of a service is not an easy task. At present, it is possible to process the OWL-S service model directly provided that the service grounding is pre-defined.

3.1 Composition Workflow and Instance sub-graphs

Fig.2 shows the composition work flow of 19 web services. In general, a workflow consists of a set of instance sub graphs which have different routing probability by workflow cases. To deduce the method of automatically computing routing probability of instance sub-graph, we extend notation of workflow by adding routing probability to every arc pointing out from places.

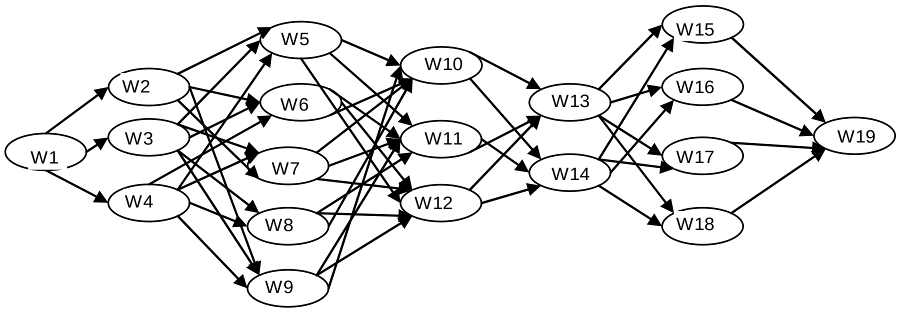

Fig. 2 Web Services Composition Workflow

An instance sub-graph [7] represents a subset of workflow tasks (transitions) that may be executed for a particular instance of a workflow. More exactly, the subset should include related arcs and places. From the composite workflow there can be many sub graphs generated for a given user request. But finding selecting the right execution path is the research problem. In the given example there are 19 web services. Now some sample sub graphs (which are also known as the composition schemas) generated from this workflow is given below in Fig.3

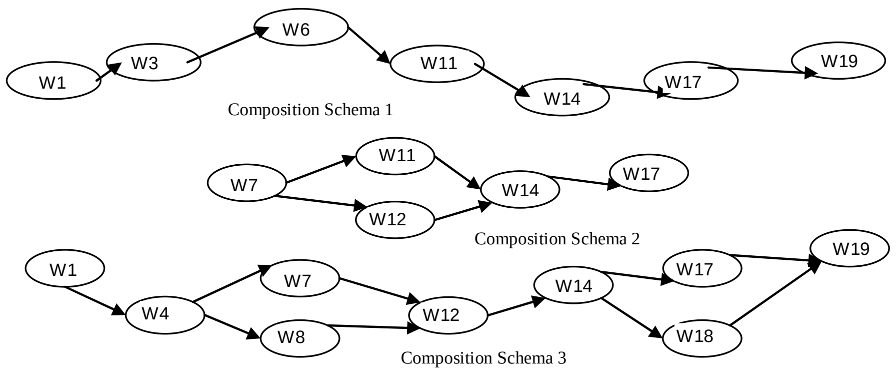

Fig 3. Instance Sub-graphs

4. Web service QoS requirements

The major requirements for supporting QoS specified by IBM [8] are as follows:

- Availability: Availability is the quality aspect of whether the Web service is present or ready for immediate use. Availability represents the probability that a service is available. Larger values represent that the service is always ready to use while smaller values indicate unpredictability of whether the service will be available at a particular time. Also associated with availability is time-to-repair (TTR). TTR represents the time it takes to repair a service that has failed. Ideally smaller values of TTR are desirable.
- Accessibility: Accessibility is the quality aspect of a service that represents the degree it is capable of serving a Web service request. It may be expressed as a probability measure denoting the success rate or chance of a successful service instantiation at a point in time.
- Integrity: Integrity is the quality aspect of how the Web service maintains the correctness of the interaction in respect to the source. Proper execution of Web service transactions will provide the correctness of interaction.
- Cost: It is the cost of executing a web service. The cost is usually fixed but may be changed according to the web service provider’s business policy. The execution cost is registered by the service provider.
- Response time: It is the average time expected for executing a web service. Individual execution times upon the requests of the clients vary because the servers load changes. Therefore, the average execution time should be updated continuously by the service provider.
- Reliability: Reliability is the quality aspect of a Web service that represents the degree of being capable of maintaining the service and service quality. The number of failures per month or year represents a measure of reliability of a Web service.

4.1 QoS Constraints

For a given composition request if there are ‘n’ no. of abstract services in the composite set then there may be ‘m’ no. of concrete services for each abstract service. For each abstract service we have to select exactly one candidate service which can fulfill all the QoS requirements. At any specific given time, only one service can be bound to any abstract service as given below.

$$\sum_{k=1}^{m} w_{i,k} = 1 \qquad \forall i \, . \tag{1}$$

i refers the abstract service index , (i,k) refers the kth concrete service of ith abstract service. $Wi_{,k}$ is set to 1, if any suitable service with specified quality found else it is 0. In our problem, we have three quality constraints namely, Availability, cost and response time. The quality attributes for each service can be retrieved from the corresponding WSDL-S document of the service. WSDL-S is an extended version of WSDL designed to specify more semantic information on web service [9]. All constraints are associated with their QoS property. Let us denote the permissible value of the kth QoS property as S_k (k = 1,2,. . . , 6). Once a composition plan is generated, ie., when the candidate web services are assigned to all abstract services of the composition schema, the kth QoS property value of the composition plan is evaluated through the aggregation of the QoS property values of the component web services considering their execution order. Let us denote the aggregated value as T_k (k = 1,2,. . . , 6). Then, each constraint is represented as follows.

$$\begin{aligned} T_k &\le S_k \quad if\ k=1,2 \\ T_k &\le S_k \quad if\ k=3,4,5,6 . \end{aligned} \qquad (2)$$

Aggregated values for cost and Response time should be less than or equal to their maximum values and aggregated values for availability should be larger than or equal to their minimum values.
Table 1 shows the QoS attributes and the aggregation function that have been specified by ([10], [11]) Given a concretization of a composite service, i.e., a composite service description where each abstract service has been bound to one of its corresponding concrete services, the overall QoS can be computed by applying the rules. These functions are recursively defined on compound nodes of the workflow. Namely, for a Sequence construct of tasks $\{t_1, . . . , t_m\}$, the *Response Time* and *Cost* functions are additive while *Availability* and *Reliability* are multiplicative.

Table 1 : Functions involved to Construct workflow and QoS attributes

Quality Parameters	Sequence	Switch	Flow	Loop
Response Time(T)	$\sum_{i=1}^{m} T(t_i)$	$\sum_{i=1}^{n} p_{ai} * T(t_i)$	$\text{Max}\{T(t_i)_{i \in \{1..p\}}\}$	K*T(t)
Cost(C)	$\sum_{i=1}^{m} C(T_i)$	$\sum_{i=1}^{n} p_{ai} * C(t_i)$	$\sum_{i=1}^{p} C(T_i)$	K*C(t)
Availability(A)	$\prod_{i=1}^{m} A(t_i)$	$\sum_{i=1}^{n} p_{ai} * A(t_i)$	$\prod_{i=1}^{p} A(t_i)$	$A(t)^k$
Reliability(R)	$\prod_{i=1}^{m} R(t_i)$	$\sum_{i=1}^{n} p_{ai} * R(t_i)$	$\prod_{i=1}^{p} R(t_i)$	$R(t)^k$

The Switch construct of Cases 1, . . . , n, with probabilities $p_{a1}, . . . , p_{an}$ such that $\sum_{i=1}^{n} p_{ai} = 1$, and tasks $\{t_1, . . . , t_n\}$ respectively, is always evaluated as a sum of the attribute value of each task, times the probability of the Case to which it belongs. The aggregation functions for the fork (named Flow in BPEL4WS) construct, are essentially the same as those for the Sequence construct, except for the Time attribute where this is the maximum time of the parallel tasks $\{t_1, . . . , t_p\}$. Finally, a Loop construct with k iterations of task t is equivalent to a Sequence construct of k copies of t.

5. Planning Graph

Planning Graph is a directed layered graph. It contains nodes and edges. Nodes can be referred as two types. One is propositions (collection of pre and post conditions of web services) referred as P and another is Actions (collection of web services) referred as A. Arcs are the edges which connects one layer with another. Quality attributes can be assigned over the arc as weight. For a query if we want to form the workflow then it involves with different levels referred as L where each level consists of a proposition and an action. For example, level 1 consists of an action level, A_1, and a proposition level, P_1. In the domain of AI, A_1 is the set of actions in which preconditions are nodes in P_1, and P_2 is the union of P_1 and the sets of positive effects of actions in A_1. An action node in A_1 is connected by incoming precondition arcs to preconditions in P_1, and it is connected by outgoing arcs to positive or negative effects in P_2. According to AI, state s is a subset of a finite set of proposition symbols L. An action a can be written as a = (precond(a); effects^-(a); effects+(a)). precond(a) is the preconditions of a. effects-(a) and effects^+(a) are negative effects and positive effects of a respectively. The action, a, is applicable to a state, s, if precond(a) belongs to s. A goal is reachable only if it's parameters appears in some proposition sets of the planning graph.

5.1 Modeling WSC problem as a Quality based Planning Graph

Consider a composite service containing k tasks $t_1, t_2, \ldots, t_k$. For each task $t_i (1 \le i \le k)$, there are n candidate services that can perform the task. Users may impose some constraints on the resources to be consumed (e.g. execution time, price). Let's assume a web service, w, has a set of input parameters $w_{in} = \{i_1, \ldots, i_n\}$ and a set of output parameters $w_{out} = \{o_1, \ldots, o_n\}$. When w is invoked with all input parameters, w_{in}, the output parameters, w_{out}, are returned. For a composite request R, we have to find out a set of web services which are connected satisfying the preconditions and post conditions need. If one service's output is the input of another service then we say that the former service is consumed by the later.

5.2 Definitions

Definition 1 : *Two web services are said to be semantically connected iff* $\forall k^1 \in w^1_{in}, \exists m^0 \in w^0_{out}$ *such that* $m^0 \subseteq k^1$*, which can be denoted as* $w^1_{in} \supset w^0_{out}$

Definition 2 : *A composition request R can be satisfied if there exists a set of web services {w1,w2,….,wn} such that* $w1in \supset Rin \cup w0out \cup w1out \ldots \cup wi\text{-}1out$ *and* $Rout \supset w0out \cup w1out \ldots \cup wnout$ *, where i=1,2,…n.*

Here, we define a graph as G and the goal proposition as g and every web service, w, is mapped to an action, a, of the planning graph. The input parameters of the Web service, w_{in}, are mapped as the action's preconditions, and its output parameters, w_{out}, are mapped to the action's effects, effects (a). Weight is based on the non-functional quality parameters (cost, response time and availability) and the weight of an action is given as weight (a). All input and output parameters are positive atoms, and each Web service only contains positive effects, since parameters can be used many times. Fig. 4 shows an example of service composition which has 7 web services w1,w2….w7. [a,c] and [e,p] are the input and output of the composite set respectively.

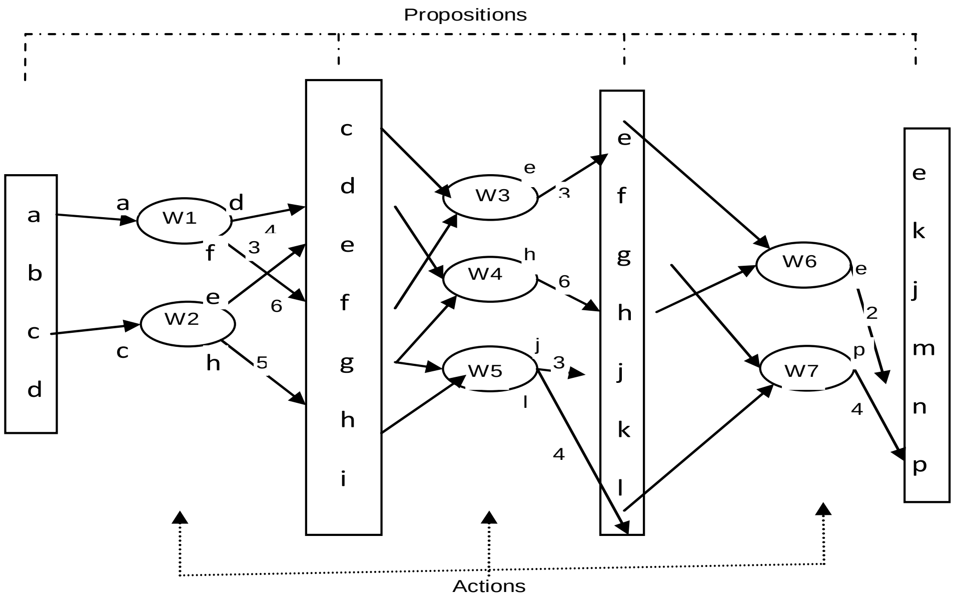

Fig.4 Example for Web Service Composition

There are four conditions to be followed when the planning graph with quality weight is expanded. This is to reduce the search space of the process and to reduce the size of the expansion plans and to enhance the efficiency.

Condition 1: *If a web service w, produce the same outputs of a web service w1 which is in the action set already, must be neglected. But if the weight (w) < weight (w1) then w must be replaced by w1. Here, weight refers cost, response time as minimum and availability as maximum. // According to formula (2)*

Condition 2 : *A web service is not added to the action set if its outputs already available in the previous proposition set.*

Condition 3 : *A web service which is not used in the next action layer will not be added but it may be considered to be added later.*

Condition 4 : *When a proposition layer has reached the goal proposition, stop adding the planning graph.*

5.3 Composition Algorithm

Algorithm 1 is the web service composition algorithm based on the planning graph with non-functional quality criteria (cost, availability, response time) as weights. The notations used in the algorithm are given in Table. 2

Table.2 Notations used in the algorithm

Symbol	**Description**
Inc(w)	Service w also called as action can be included in the graph or marked as valid
Rej(w)	Service w is already used and marked as invalid
Pr(w)	preconditions of w
Po(w)	Post conditions or effects of w
Wt(w)	Weight assigned by its quality parameters cost, availability, response time
g	contains the composite set output parameters
A	Action set which contains the selected service set in each action layer
P	pre and post conditions set in each proposition layer

Algorithm 1 : Composition(A,P_1,g)

```
{ i=0
Repeat {
For each w in A
        { If Pr(w) $\subseteq$ Pi && Inc(w) then
                $A_{i+1} = A_{i+1}$ U w U wt(w) // satisfying formula (1)
                Rej(w)
        $P_{i+1} = P_i$ }
For each w in $A_{i+1}$
        { if po(w) $\not\subset P_{i+1}$ then
        $P_{i+1} = P_{i+1}$ U po(w)
        else Rej(w) from $A_{i+1}$}
If g $\subseteq P_{i+1}$ then exit
For each w in $A_i$
        If po(w) $\cap$ {pr(w) | w $\in A_{i+1}$} = null then
                Re(w) from $A_i$
                Inc(w)
i=i+1}
Until (g $\subseteq P_{i+1}$)
}
```

In this Algorithm, first it generates a new action layer A_{i+1} and a new proposition layer $Pi_{+1.}$ Then according to condition1 for every action, the output is compared with the parameters in the proposition and it keeps the action within the action set with its corresponding quality weight and the effects in the proposition set. For every action w in A_i if the parameter is existing within the proposition set then it is discarded. Finally output parameters in g are checked within the last proposition set and if both matched then algorithm terminates.

6. Experiment and Result

The algorithm is executed using the programming language Java and is executed on desktop PC with Pentium 2.2 GHz dual core CPU and 3 GB RAM. The first experiment was performed with 30, 50 and 50 tasks. For each task we 100 web services. As shown in experiments show In this method, the plans are generated until the suitable plan is founded. From the Fig. 5, it can be inferred that the time of execution increases exponentially when number of tasks increase linearly. Also the complexity of the graph based web service composition algorithm with quality is polynomial, because it has finite distinct levels and the number of elements in the proposition set is n and number of elements in action set is m.

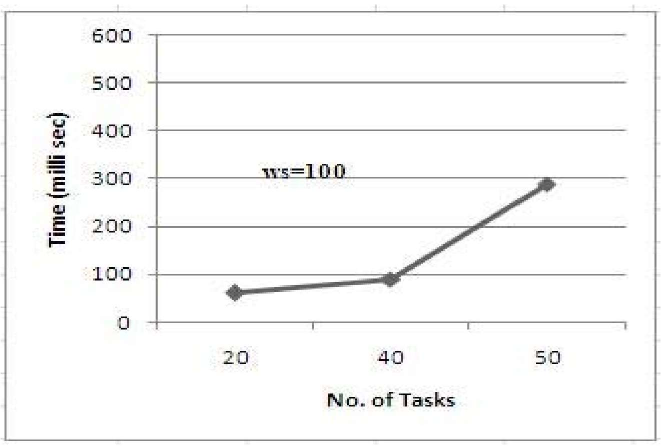

Fig.5 Result of the algorithm for 100 candidate services for different set of tasks

7. Conclusion and Future Works

In this paper we discussed about QoS based web service composition. Because the number of web services that offer the same service has proliferated, web service composition has become an important problem. Quality parameters in web services are one of the most important factors in selecting web services. We formulated this problem as AI graph plan to construct the web service model. Non-functional Quality parameters are included in the composition algorithm. Further improvements can be done to reduce the time complexity by using the process ontologies.

References

[1] Y. Chen, L. Zhou and D. Zhang, “Ontology-supported web service composition: an approach to service-oriented knowledge management in corporate services”, Database Management vol. 17, pp. 67–84, 2006.

[2] G. Canfora, M.D. Penta, R. Esposito and M. L. Villani, “An approach for QoS-aware service composition based on genetic algorithms”, Proc. of conf. on Genetic and evolutionary computation (GECCO ’05), New York, USA, pp. 1069–1075, 2005.

[3] B. Benatallah, Q. Sheng, A. Ngu, and M. Dumas, “Declarative composition and peer-to-peer provisioning of dynamic web services,” in Proceedings of the 18th International Conference on Data Engineering (ICDE), 2002, pp. 297–308.

[4] Yuhong Yan, Xianrong Zheng,, "A Planning Graph based Algorithm for Semantic Web Service Composition" IEEE 2008

[5] Seyyed, Farhad, "A Grah-based Framework for Composition os Stateless Web Services" IEEE 2006

[6] Jong Myoung, Chang, Ick-Hyun, "Quality-of-service oriented web service composition algorithm and planning architecture" The Journal of Systems and Software published by Elsevier 2008

[7] W. Sadiq and M. E. Orlowska, Analyzing process models using graph reduction techniques. Inf. Syst., vol. 25, no. 2, pp. 117–134, 2000.

[8] http://www.ibm.com/developerworks/webservices/library/ws-quality/index.html

[9] Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A., Verma, K., 2005. Web service semantics: WSDL-S, A joint UGA-IBM technical note, version 1.0. Available from: <IBM alphaWorks web site>.

[10] J. Cardoso. Quality of Service and Semantic Composition of Workflows. PhD thesis, Univ. of Georgia, 2002.

[11] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. IEEE Transactions on Software Engineering, 30(5), May 2004.

[12] Huan, Bang “An Approach for QoS-aware Web Service Composition based on Improved Genetic Algorithm” IEEE 2010

[13] Mohamed, Maroua, Abdel-Illah “Automated Web Service Composition Using Extended Representation of Planning Domain”, International Conference on Web Services, IEEE 2008

[14] Wenqiang Li, Xuemei, Hao “Web Services Composition Based on Weighted Planning Graph", International Conference on Networking and Distributed Computing, IEEE 2010