A New Approach for Web Services Composition

Ujas Patel[1] Govind Patel[2]
[1]P.G. Student [2]Assistant Professor
[1]Department of Computer Engineering [2]Department of Information Technology
[1,2]Sankalchand Patel College of Engineering, Visnagar, India

Abstract— Semantic web services composition is about finding services from a repository that are able to accomplish a specified task if executed. The task is defined in a form of composition request which contains a set of available input parameters and a set of wanted output parameters. Instead of parameter values, concepts from an ontology describing their semantics are passed to the composition engine. The composer then finds a sequence of services, called a composition. Here three different approaches to semantic web services composition are formally defined and compared with each other: an uninformed search in form of an IDDFS algorithm, a greedy informed search based on heuristic functions, and a multi-objective genetic algorithm. Due to the increasing number of available web services, the search space for finding semantically equivalent services for best service composition is growing exponentially. Here a combination of multi-objective genetic algorithm and greedy informed search based on heuristic functions is proposed for service composition. The greedy algorithm is utilized to generate valid and locally optimized individuals to populate the initial generation for genetic programming (GP), and to perform mutation operations during GP. Here in this context, this work present web service composition which also consider Quality of Service (QoS) parameters. This new proposed hybrid approach can be applied with good performance to the QoS aware service composition problem to find optimal solution for larger service repositories of all sizes and also strongly supporting the knowledge-base.

Key words: Cascaded multilevel inverter, developed H-bridge, multilevel inverter, voltage source inverter

I. INTRODUCTION

Web of meaningful data that can be processed and interpreted by machines directly & indirectly which allows data to be shared & reused across application, enterprise & community boundaries with enabling machines & people to work in cooperation is called semantic web or web 3.0. It extends the network of hyperlinked human-readable web pages by inserting machine-readable metadata about pages & how they are related to each other, enabling automated agents to access the web more intelligently & perform tasks on behalf of users. [wiki]

Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located or invoked over the network to create products, processes and supply chains. These applications can be local, distributed or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML and XML. Services perform functions that can range from answering simple requests to executing sophisticated business processes requiring peer-to-peer relationships between service consumers and providers.

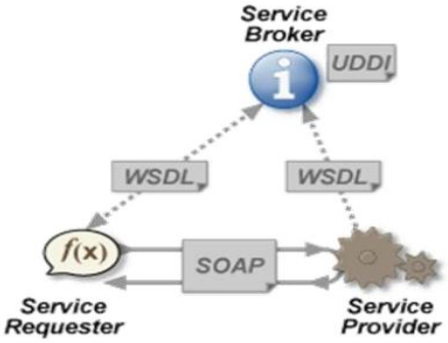

Fig. A: Web Service Architecture [wiki]
WSDL–Web Services Description Language
SOAP–Simple Object Access Protocol
UDDI–Universal Description, Discovery & Integration

Service registry is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearing house for companies and their services.

Web services can be combined in a loosely coupled way to achieve complex operations. Programs providing simple services can be interact with each other to deliver sophisticated added-value services.

This paper is organized in seven sections. Section 2 reviews and compares the web service composition models, namely, the semantic and syntactic models. Section 3 reviews approaches to the semantic web service composition model and presents a comparison of these approaches. Section 4 & 5 present the new proposed hybrid approach. Section 6 states QoS awareness and finally section 7 states the conclusions and suggestions for future research.

II. WEB SERVICES COMPOSITION MODELS

Web service composition can be achieved by one of two models: semantic (dynamic) and syntactic (static) Web service composition models. Web service composition taxonomy including models, approaches and languages is illustrated in Figure I. [1]

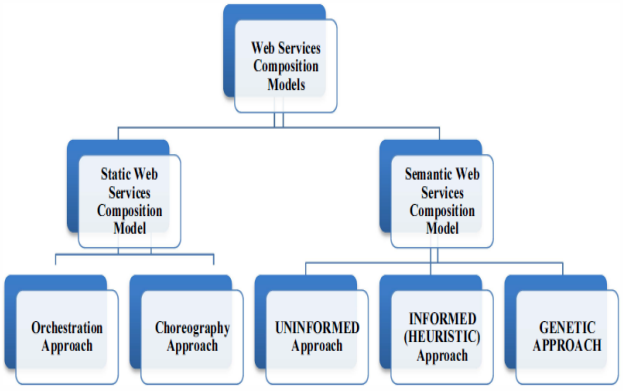

Fig. 1: Taxonomy of Web service composition models and approaches [1]

Table I. A comparison ofsyntactic and semantic composition models

	Syntactic	Semantic
Driven	Industry driven	Academic driven
Composition model	Static	Dynamic
Dynamic topology	Not supported	Supported
Control	Centralized executor	Distributed (peer to peer)
Language	Process execution language	Description language
Connectivity	Supported	Supported
Exception and compensation handling	Strongly supported	Supported
QoS	Indirectly supported	Supported
Composition Place	Design-time	Runtime
Recursive composition	Not supported	Supported
User interaction at runtime	Not supported	

Table I: summarizes the characteristics of the various Web service composition models: [1]

III. Approaches for Semantic Web Service Composition Model

In semantic model of web service composition three main motivating tasks are included in the process of semantic composition, that is, automatic discovery, invocation, composition, and interoperation. [1] In order to discuss the idea of semantic service composition properly, we need some prerequisites. Therefore, let us initially define the set of all semantic concepts M. All concepts that exist in the knowledge base are members of M and can be represented as nodes in a wood of taxonomy trees. [2]

Definition 1 (subsumes): Two concepts A,B ∈ M can be related in one of four possible ways. We define the predicate subsumes: (M × M) _→ {true, false} to express this relation as follows: [2]

1) Subsumes (A,B) holds if and only if A is a generalization of B (B is then a specialization of A).
2) Subsumes (B,A) holds if and only if A is a specialization of B (B is then a generalization of A).
3) If neither subsumes (A,B) nor subsumes(B,A) holds, A and B are not related to each other.
4) Subsumes (A,B) and subsumes(B,A) is true if and only if A = B.

The subsumes relation is transitive, and so are generalization and specialization.

If a parameter x of a service is annotated with A and a value y annotated with B is available, we can set x = y and call the service only if subsumes(A,B) holds (contra variance). This means that x expects less or equal information than given in y. [2]

From the viewpoint of a composition algorithm, there is no need for a distinction between parameters and the annotated concepts. The set S contains all the services *s* known to the registry. Each service $s \in S$ has a set of required input concepts $s.in \subseteq M$ and a set of output concepts $s.out \subseteq M$ which it will deliver on return. We can trigger a service if we can provide all of its input parameters. [2]

Similarly, a composition request *R* always consists of a set of available input concepts $R.in \subseteq M$ and a set of requested output concepts $R.out \subseteq M$. A composition algorithm discovers a (topologically sorted) set of *n* services $S = \{s1, s2, \ldots, sn\} : s1, \ldots, sn \in S$. As shown in Equation 1, the first service (s0) of a valid composition can be executed with instances of the input concepts *R.in*. Together with *R.in*, its outputs (*s1.out*) are available for executing the next service (*s2*) in *S*, and so on. The composition provides outputs that are either annotated with exactly the requested concepts *R.out* or with more specific ones (*covariance*). For each composition solving the request *R*, *isGoal*(*S*) will hold: [2]

$$isGoal(S) \Leftrightarrow \forall A \in s1.in\ \exists B \in R.in : subsumes(A,B) \wedge \forall A \in si.in,\ i \in \{2..n\}$$
$$\exists B \in R.in \cup si{-}1.out \cup \ldots \cup s1.out : subsumes(A,B) \wedge \forall A \in R.out\ \exists B \in s1.out \cup \ldots \cup sn.out \cup R.in : subsumes(A,B) \quad (1)$$

The search space that needs to be investigated in web service composition is the set of all possible permutations of all possible sets of services. Let us therefore define the operation *promising* which obtains the set of all services $s \in S$ that produce an output parameter annotated with the concept *A* (regardless of their inputs). [2]

$$\forall s \in promising(A)\ \exists B \in s.out : subsumes(A,B) \quad (2)$$

A. Uninformed Approaches – IDDFS:

The most general, easy to understand and straightforward approach to web service composition is the uninformed search. [1][2] It comprises an uninformed search algorithm that does not take advantage of or utilize any information other than the goal predicates as defined in Equation 1. [1][2] We can build such an composition algorithm base on iterative deepening depth-first search. It is only fast in finding solutions for small service repositories but optimal if the problem requires an exhaustive and a comprehensive search. [1][2]

Algorithm 1: $S = IDDFSComposition(R)$
Input: R the composition request
Data: $maxDepth$, $depth$ the maximum and the current search depth
Data: in, out current parameter sets
Output: S a valid service composition solving R

```
1  begin
2      maxDepth ← 2
3      repeat
4          S ← dl_dfs(R.in, R.out, ∅, 1)
5          maxDepth ← maxDepth + 1
6      until S ≠ ∅
7  end
8  dl_dfs(in, out, composition, depth)
9  begin
10     foreach A ∈ out do
11         foreach s ∈ promising(A) do
12             wanted ← out
13             foreach B ∈ wanted do
14                 if ∃ C ∈ s.out : subsumes(B, C) then
                       wanted ← wanted \ {B}
15             foreach D ∈ s.in do
16                 if ∄ E ∈ in : subsumes(D, E) then
                       wanted ← wanted ∪ {D}
17             comp ← s ⊕ composition
18             if wanted = ∅ then
19                 return comp
20             else
21                 if depth < maxDepth then comp ←
                       dl_dfs(in, wanted, comp, depth + 1)
22                 if comp ≠ ∅ then return comp
23     return ∅
24 end
```

Fig. 2: Algorithm IDDFS

Algorithm 1 builds a valid web service composition starting from the back. In each recursion, its internal helper method dl dfs tests all elements *A* of the set *wanted* of yet unknown parameters. It then iterates over the set of all services *s* that can provide *A*. For every single *s*, *wanted* is recomputed. If it becomes the empty set $\emptyset$, we have found a valid composition and can return it. If dl dfs is not able to find a solution number of services in the composition), it returns $\emptyset$. The loop in Algorithm 1 iteratively invokes dl dfs by increasing the depth limit step by step, until a valid solution is found. [2]

B. An (Informed) Heuristic Approach:

The IDDFS algorithm just discussed is slow and memory consuming for bigger repositories since it does not utilize any additional information about the search space. If we use such information, we can increase the efficiency of the search remarkably. In an informed search, a heuristic c helps to decide which nodes are to be expanded next. [2] Therefore, with an infonned heuristic approach, a heuristic value is defined and used to choose which nodes in the search space will be expanded at the next step and which nodes will be ignored. In other words, this heuristic value requires selecting the best nodes from the search space in each step. [1] If the heuristic is good, such algorithms may dramatically outperform uninformed strategies. As second composition method we therefore define a greedy algorithm that internally sorts the list of currently known candidate compositions in descending order according to a heuristic in form of a comparator function c : Sn ∈ R. The comparator function c(S1, S2) will be below zero if S1 seems to be closer to the solution than S2 and greater than zero if S2 is a more prospective candidate. Thus, the best elements will be at the end of the list *X* in Algorithm 2. [2]

Algorithm 2: $S = greedyComposition(R, c)$
Input: R the composition request
Data: X the sorted list of compositions to explore
Output: S the solution composition found, or $\emptyset$

```
1  begin
2      X ⟵ ⋃_{∀A∈R.out} (promising(A))
3      while X ≠ ∅ do
4          X ⟵ sort(descending, X, c)
5          S ⟵ popLastElement(X)
6          if isGoal(S) then return S
7          foreach A ∈ wanted(S) do
8              foreach s ∈ promising(A) do
9                  X ⟵ appendList(X, s ⊕ S)
10     return ∅
11 end
```

Fig. 3: Heuristic Approach

We can easily derive different comparator functions for web service composition. In our experiments and real-world experiences, the function c_{cmp} has proven to be most efficient. It combines the size of the set unsatisfied parameters $\forall A \in wanted(S) \Rightarrow \exists s \in S : A \in s.in \wedge A \_ \in known(S)$, the composition lengths, the number of satisfied parameters $\forall B \in eliminated(S) \Rightarrow \exists s \in S : B \in s.in \wedge B \in known(S)$, and the number of known concepts $known(S) = R.in \cup \forall s \in S\ s.out$ as defined in Algorithm 3. [2]

First, it compares the number of wanted parameters. If a composition has no such unsatisfied concepts, it is a valid solution. If both, *S*1 and *S*2 are valid, the solution involving fewer services wins. If only one of them is complete, it also wins. Otherwise, both candidates still have unsatisfied concepts. Only if both of them have the same number of satisfied parameters, we again compare the wanted concepts. If their numbers are also equal, we prefer the shorter composition candidate. If even the compositions are of the same length, we finally base the decision of the total number of known concepts. [2]

Algorithm 3: $r = c_{cmp}(S_1, S_2)$
Input: S_1, S_2 two composition candidates
Output: $r \in \mathbb{Z}$ indicating whether S_1 $(r < 0)$ or S_2 $(r > 0)$ should be expanded next

```
1  begin
2      i1 ⟵ |wanted(S1)|
3      i2 ⟵ |wanted(S2)|
4      if i1 = 0 then
5          if i2 = 0 then return |S1| − |S2|
6          else return −1
7      if i2 = 0 then return 1
8      e1 ⟵ |eliminated(S1)|
9      e2 ⟵ |eliminated(S2)|
10     if e1 > e2 then return −1
11     else if e1 < e2 then return 1
12     if i1 > i2 then return −1
13     else if i1 < i2 then return 1
14     if |S1| ≠ |S2| then return |S1| − |S2|
15     return |known(S1)| − |known(S2)|
16 end
```

Fig. 4: Composition

C. A Genetic Approach:

Evolutionary algorithms (EAs) are generic, population-based meta-heuristic optimization algorithms that use biology inspired mechanisms like mutation, crossover, natural selection and survival of the fittest or fitness function. [2][1] This genetic algorithm is slower than the greedy algorithm defined in the informed approach, but is also a good solution for all sizes of knowledge-bases and service registries. [1] The advantage of evolutionary algorithms compared to other optimization methods is that they make only few assumptions about the underlying fitness landscape and therefore perform consistently well in many different problem categories. [2]

All evolutionary algorithms proceed in principle according to the scheme described below. [2]

- Initially, a population of totally random individuals is created.
- All of them are tested for their utility as solution.
- Based on this evaluation, fitness values are assigned to them.
- A subsequent selection process filters out the individuals with low fitness and allows those with good fitness to enter the mating pool with a higher probability.
- In the reproduction phase, offspring is created by varying or combining solution candidates.
- If the termination criterion is met, the evolution stops here. Otherwise, it continues at step 2.

For the reproduction of solution candidates, EAs employ two different operators: [2]

- Mutation slightly modifies one existing individual and
- Crossover combines two solution candidate, creating offspring with features of both.

D. IDDFS vs. HIDDF vs. GA:

Table II summarizes the characteristics of the various approaches to the semantic web services composition model.

	IDDF	HIDDF	GA
Finding solution for larger service repositories	Very slow	Slow	Very slow
Finding optimal solution	Not good	Good	Very good
Size of repositories	Small	All sizes	All sizes
Knowledge-base	Not supported	Supported	Strongly supported
Algorithm	Depth-first search	Greedy algorithm	Genetic algorithm

Table II: Comparison of approaches [1]

IV. THE NOVEL GP-BASED APPROACH

The goal of this paper is to propose a novel GP-based approach to web service composition that overcomes shortcomings of previous GP-based approaches. Instead of starting with an initial population of service compositions that are randomly generated from the huge number of atomic web services in the repository, we apply a greedy search algorithm to pre-filter the repository for those atomic web services that are exclusively related to the given service composition task. We have examined our proposal using the public web service repositories of OWL-S TC [4], WSC2008 [5], and WSC2009 [6] as benchmarks. Further, we adapt our GP-based greedy algorithm enhanced approach to QoS-aware web service composition, and propose two fitness functions to guide the GP-based evolution. We have examined these fitness functions using the service repositories of WSC2008 and WSC2009 extended with QoS properties. We can demonstrate the effectiveness and efficiency of our GP-based greedy algorithm enhanced approach to QoS aware service composition. Specifically, we have investigated the following objectives:

1) Whether the new method can achieve reasonably good performance, and in particular outperforms existing GP-based approaches.
2) Whether the greedy algorithm can effectively discover atomic web services that are exclusively related to the service composition task.
3) Whether the evolved program (the solution to the given service composition task) is interpretable.
4) Whether the new method is suitable for QoS-aware service composition, and which fitness function to apply during GP evolution.

The existing GP-based approaches often start with a low quality population at the initial stage. To overcome this matter, we propose a GP-based greedy algorithm enhanced approach that uses a greedy algorithm to be combined with our GP-based service composition approach. A greedy algorithm can help to search locally optimal solutions, though using a greedy algorithm by itself cannot generate globally optimal solutions for web service composition in general. Rather, we plan to employ a greedy algorithm to generate locally optimal solutions such that the performance of generating a globally optimal solution by GP can be improved. In particular, we propose to use a greedy algorithm to generate individuals that can form the initial population of our GP-based service composition algorithm.

A. Variables in Genetic Programming:

To apply GP to the service composition problem, the first major step is to define the variables in GP, i.e., to identify the terminal set, the function set, and the fitness function. We will discuss how tree representations of web service compositions will be used for our tree-based GP approach. Now, we define the variables commonly used in GP, i.e., the terminal set, the function set, and the fitness function [7]. A service composition task is defined by an input concept I, an output concept O, and a repository R of atomic web services. We use the atomic web services in the given repository as the function set in GP, i.e., we regard the atomic web services as functions that map inputs to outputs. GP uses the tree representation discussed above: the internal nodes correspond to functions, all terminal nodes to the input concept I, and the root to the output concept O.

1) Terminal Set:

A service composition task is defined by an input concept I, an output concept O, and a repository of atomic web services. In our approach, all terminal nodes of the tree represent the given input concept I of the service composition task.

2) Function Set:

The atomic web services may be regarded as functions that map their inputs to their outputs. We can directly use the atomic web services in the given repository as the function set in GP. In our approach, all nodes of the tree represent functions, except for the terminal nodes that represent the input concept I, and for the root node that represents the output concept O.

3) Fitness Function:

A fitness function is used to measure the quality of candidate compositions. How to measure the quality of service composition depends on the task of web service composition. If we do not consider QoS requirements we use the unduplicated number of atomic web services used in a service composition to measure the fitness of a service composition. The fewer atomic web services used in the service composition, the better its performance will be. In addition, we also use the tree depth to measure the fitness of service compositions. The tree depth corresponds to the length of the longest path from the input concept to the output concept.

B. Genetic Operators:

GP uses the operations crossover, mutation, and reproduction to evolve individuals, i.e., service compositions in our case. To perform crossover, we stochastically select two random individuals and check if there is one node representing the same atomic web service in both individuals, and then swap the node together with their subtrees between the two individuals. This guarantees

that the matching rules stay satisfied. In Fig. 2, for example, the S3 nodes in the two individuals are swapped together with the subtrees rooted at them. As usual, the two new individuals generated as offsprings from the two individuals from the previous generation are then included into the next generation.

The mutation operator is normally used to replace a node together with its subtree in a selected individual, or to replace only the node. In our approach, we perform mutation by stochastically selecting one node in a randomly chosen individual and replacing its subtree with a new subtree generated by applying a greedy algorithm.

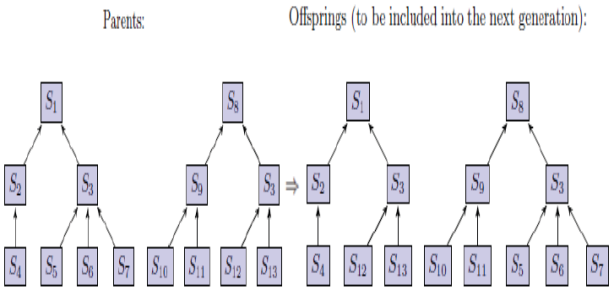

Fig. 2: An example for our crossover.

In Fig. 3, for example, assume S3 is selected for mutation and ca and cb are properties that S3 receives from S5 and S6, respectively. Then, the mutation operator replaces the subtree of S3 with a new subtree to generate a new individual as an offspring.

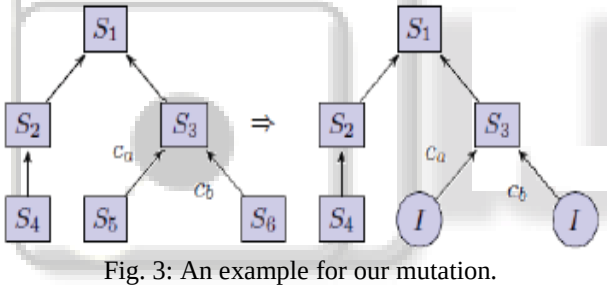

Fig. 3: An example for our mutation.

The fitness of the offspring generated by crossover or mutation can be smaller than its parents' one. To avoid a decrease of fitness of the fittest individuals we choose a top percentage of individuals from the old generation for mere reproduction and include them into the next generation without any modification.

C. GP-based Algorithm for Service Composition:

We now present a GP-based algorithm for service composition. The fitness function to be used depends on the requirements of the particular service composition task.

The existing GP-based approach start with random generated initial populations and therefore take long time to discover near-optimal solutions. In the section below to present a greedy algorithm for web service composition to overcome this matter.

V. RANDOM GREEDY SEARCH FOR INITIALIZATION AND MUTATION

Next we propose a random greedy algorithm for computing locally optimal solutions for a service composition problem, see Algorithm 2. Its inputs are the input concept I, the output concept O, and the repository R of the service com-position task to be solved. The algorithm generates the tree representation of a service composition S that is formally correct for the composition task at hand.

Input: P /* a set of initial service compositions
Output: an optimal service composition solution S
Evaluate each individual i in P using the fitness function
while g < g_{max} do
Perform reproduction with the rate r
Select two parents from the population P
Perform crossover with rate c
Perform mutation with rate m
Generate a new population P'
Evaluate each individual i in P' using fitness function
end while

A. Algorithm 1: genetic programming for web services composition

Input: I,O,R
Output: a service composition S, S_{list}

1) Csearch←I;
2) S_{list}←{}; /* discover a shrunk set of atomic web services
3) S_{found} ← DiscoverService();
4) while $|S_{found}| > 0$ do
5) S_{list} ← S_{list} U S_{found};
6) Csearch← Csearch U Coutput of S_{found};
7) S_{found}←DiscoverService();
8) end while
9) if Csearch ⊇ O then
10) ConnectNodes();
11) Report solution;
/* generate a web service composition
12) else
13) Report no solution;
14) end if

B. Algorithm 2: A Greedy Algorithm for Service Composition.

In the algorithm, Csearch denotes the concept used for searching the repository R, and S_{found} denotes the set of all those atomic web services whose inputs match Csearch. To begin with, Csearch is initialized by the input concept I. The discovered atomic web services are added to S_{found}, the outputs of these services are adjoined to Csearch. Steps 4 to 8 are repeated until no new atomic web service is discovered. This is in particular the case when Csearch is no longer extended. Afterwards it is checked whether Csearch subsumes the required output concept O of the composition task. If so, then the composition task has a solution. By applying the matching rule, the nodes of the tree are then stochastically connected to generate the arcs of the tree. Otherwise, there is no solution.

We use the random greedy algorithm as an auxiliary to our GP-based approach. It generates a set of locally optimal individuals to populate the initial generation for GP. By construction, all of them are formally correct solutions for the composition task at hand. By using the search concept Csearch the algorithm only considers services that are related to the composition task. The locally optimal individuals constitute an initial population that is already of high quality, thus overcoming a weakness of previous GP-based approaches.

Moreover, we apply our greedy algorithm to perform mutation in our GP-based approach. We use it to

generate the new subtree rooted at the selected node. This time, the output of the corresponding atomic web service serves as O, and R is restricted to the atomic web services that occur in the composition.

VI. QOS-AWARENESS

Table III describes the QoS properties.

QoS property	Description
Response Time	Time between receiving request and sending respond
Execution cost	Execution cost per request
Availability	$\frac{\text{UpTime}}{\text{UpTime+DownTime}}$
Reputation	$\frac{\sum \text{Rep}_i}{\text{Total Number Of Usage}}$
Successful Execution Rate	$\frac{\text{Numer of Successful Request}}{\text{Total Number of Request}}$

Table III: QoS Properties [3]

VII. CONCLUSIONS AND FUTURE RESEARCH SUGGESTIONS

This paper provides an analysis of web service composition models and their approaches. The uninformed search proofed generally unfeasible for large service repositories. However, in case that a request is sent to the composer which cannot be satisfied, even heuristic approaches have to test all valid service combinations and thus cannot be better than IDDFS. Superior performance could be measured by utilizing problem-specific information encapsulated in a fine-tuned heuristic function to guide a greedy search. Evolutionary algorithms are much slower, but were also always able to provide correct results to all requests. Here, they cannot cope with the greedy search. We anticipate that, especially in practical applications, additional requirements will be imposed onto a service composition engine. Such requirements will include quality of service (QoS), the question for optimal parallelization, or the generation of complete BPEL processes. In this case, heuristic search will most probably become insufficient but EAs will still be able to deliver good results.

In this paper we presented a hybrid approach for performing web service composition using a combination of genetic programming and greedy search. The random greedy algorithm is an auxiliary to GP. It generates locally optimal individuals for populating the initial generation for GP, and to perform mutations during GP. Moreover, it guarantees that the generated individuals are formally correct and thus interpretable for web service composition. We have applied the GP-based greedy algorithm enhanced approach to service composition without QoS requirements, and also to QoS-aware service composition. Our approach is efficient, effective and stable for computing near-optimal solutions. The initial greedy heuristic search helps to reduce the search area and to shrink the number of atomic web services to be considered by GP later on. The new proposed hybrid approach of multi-objective genetic algorithm and greedy informed search based on heuristic functions can be applied with good performance to the QoS-aware service composition problem to find optimal solution for larger service repositories of all sizes and also strongly supporting the knowledge-base with efficiency and effectiveness in terms of speed and accuracy.

In future, the cloud services composition can be performed using the proposed hybrid approach with privacy and security enhancements.

VIII. ACKNOWLEDGMENTS

This work was partially supported by Sankalchand Patel College of Engineering, Visnagar.

REFERENCES

[1] Hassan Mathkour, Sofien Gannouni, Mutaz Beraka "Web Service Composition: Models and Approaches" 2012 IEEE
[2] Thomas Weise, Steffen Bleul, Diana Comes, Kurt Geihs "Different Approaches to Semantic Web Service Composition" The Third International Conference on Internet and Web Applications and Services, 2008 IEEE
[3] Mahmood Allameh Amiri, Hadi Serajzadeh "QoS Aware Web Service Composition Based On Genetic Algorithm" 5th International Symposium on Telecommunications, 2010 IEEE
[4] U. Kuster, B. Konig-Ries, and A. Krug "An Online Portal to Collect and Share SWS Descriptions" IEEE Int. Conf. on Semantic Computing, pp. 480-481, 2008
[5] A. Bansal, M. Blake, S. Kona, S. Bleul, T. Weise, and M. Jaeger "WSC-08: Continuing the Web Services Challenge" In IEEE Conf. on E-Commerce Technology, pp. 351-354, 2008
[6] S. Kona et al. "WSC-2009: A Quality of Service-oriented Web Services Challenge" In IEEE Int. Conf. on Commerce and Enterprise Computing, pp. 487-490, 2009
[7] J. Koza "Genetic Programming" MIT Press 1992

