A New Methodology for Search Space Reduction in QoS Aware Semantic Web Service Composition

Soumi Chattopadhyay, Member, IEEE, and Ansuman Banerjee, Member, IEEE

Abstract—The semantic information regulates the expressiveness of a web service. State-of-the-art approaches in web services research have used the semantics of a web service for different purposes, mainly for service discovery, composition, execution etc. In this paper, our main focus is on semantic driven Quality of Service (QoS) aware service composition. Most of the contemporary approaches on service composition have used the semantic information to combine the services appropriately to generate the composition solution. However, in this paper, our intention is to use the semantic information to expedite the service composition algorithm. Here, we present a service composition framework that uses semantic information of a web service to generate different clusters, where the services are semantically related within a cluster. Our final aim is to construct a composition solution using these clusters that can efficiently scale to large service spaces, while ensuring solution quality. Experimental results show the efficiency of our proposed method.

Index Terms—Semantic Service Composition, Quality of Service (QoS), Abstraction, Refinement

1 INTRODUCTION

With proliferation of network technologies, services computing has attracted a significant research attention. Web services have become a useful tool to facilitate many necessary activities throughout the world. A web service performs a specific task to satisfy a user requirement. The performance of a web service is measured by a set of QoS parameters like response time, throughput, reliability, availability etc. Sometimes, a single web service is sufficient to meet a user requirement, sometimes, multiple web services are required to do so. In the latter case, web service composition [1] is required. Though optimality [2], [3] is a primary concern in composition, it comes at the cost of more computation time, which is another key issue in recent times. With increase in the scale of web services [4], scalable service composition [5], [6] is becoming a challenge.

A significant body of research focuses on finding optimal QoS aware composition solutions [7], [8]. However, these methods often fail to scale for large service repositories in real time. Therefore, a number of heuristic methods [9], [10], [11] have been proposed in literature. These methods also have their own limitations. Many of these fail to deliver the solution with desired quality, since they do not navigate the entire search space. The trade-off between optimality and efficiency has been explored extensively in web service literature [12], [13], [14]. In this work, we handle the trade-off between optimality and efficiency from a different perspective. In a realistic setting, a number of services are often deployed to fulfill the same purpose. Though these services are intended to serve similar functionality, they may have syntactically different interfaces. Sometimes, identifying two semantically related services becomes extremely important to compute the composition solution efficiently. The aim of this work is to identify semantically related services obtained from an ontology [15] stored in a service repository along with the web service set. Using this information, we further aim to form service groups essentially required for generating efficient composition solution. The crux of our proposal is to reduce the search space of the underlying composition algorithm by clustering the semantically related services and thus, expedite its solution computation time. Once we cluster the services into multiple groups, each service group is then represented by a new service. Each new service is then assigned a set of QoS values that are the best representative in its respective group. The composition algorithm is first carried out in the new service space to find a solution satisfying all QoS constraints. Since the algorithm explores a smaller search space, it becomes more efficient than the original one. However, some information regarding service QoS parameters may be lost due to this clustering. Therefore, the composition algorithm, performed with the new representative service, may fail in producing a solution to a query satisfying all the constraints. In this paper, this clustering is termed as an abstraction. To deal with constraint violation issue, we further propose to refine an abstraction, which ensures a solution satisfying all the necessary constraints if one exists originally. Our framework is, therefore, sound and complete.

Novelty of our work: Our method is built on the foundation of abstraction refinement, a standard technique for handling large state spaces, originally proposed in [16] in the context of formal verification. Abstraction refinement based on syntactic similarity in terms of input-output parameters has been dealt with in our earlier work [17]. However,
### TABLE 1: DataSet $D$: Service Repository

| Method: | Service Name: | Input: | Precondition: | Output: | Postcondition: |
|---------|---------------|--------|---------------|---------|----------------|
| 1 | BWImageToBarcode | BWImage | BWImage must be in jpeg format | Barcode | - |
| 2 | BinImageToBarcode | BinImage | BinImage must be in jpeg format | Barcode | - |
| 3 | ImageToBarcode | Image | Image must be in jpeg format | Barcode | - |
| 4 | AnyImageToBarcode | Image | Image size must be less than 50KB | Barcode | - |
| 5 | ImageToIDConverter | Image | Image must be in jpeg format | ProductID | - |
| 6 | EuropeanBarcodeToProductID | EANBC | Image size must be less than 50KB | ProductNumber | - |
| 7 | BarcodeToProductID | Barcode | - | PID | - |
| 8 | BarcodeToEncoded9ProductID | Barcode | - | PID | - |
| 9 | BarcodeToEncoded14ProductID | Barcode | - | PID | - |
| 10 | GetReviewInEnglish | Barcode | - | Review | - |
| 11 | GetReviewAndRatingInEnglish | Barcode | - | Review, Rating | - |
| 12 | GetCheapShopAtPos | Barcode | - | Price, GPS | - |
| 13 | GetShopAtPos | Barcode | - | Price, GPS | - |

**Ontology**

Concept: PID, BinImage; $f(ProductID) = f(PID) = PID$; $PID_9$ $f(BWImage) = BinaryImage$; $f(BinImage) = BinaryImage$; EANBC $\sqsubseteq$ Barcode; BinaryImage $\sqsubseteq$ Image; $PID_9 \sqsubseteq PID$; $PID_{14} \sqsubseteq PID$
the notion of semantics of a web service has not been dealt with in that work. In this paper, we bring the notion of semantics and propose a methodology that provides a scalable way of pruning the composition search space. Our method can be applied on top of any service composition algorithm to improve its performance. In [15], the authors proposed functional clustering based on semantics of the services. However, the focus of [15] is on reducing the search space with the objective of improving the reliability and flexibility, while our focus is to reduce the search space and thus expedite solution composition time. Moreover, the clustering methods that we consider is different from the one in [15].

**Our Contribution:** In this work, we have the following contributions:

- We first propose different abstractions based on the semantics of the services in the service repository, that help to reduce the search space and thus to expedite solution computation time.
- In order to ensure that our algorithm always produce a valid solution (i.e., a solution satisfying all the constraints), we further propose two refinements corresponding to each solution.
- We have performed extensive experiments and it is evident from our experiments that on an average our framework is more efficient, when compared with the performance of the underlying composition algorithm without abstraction.

This paper is organized as follows. Section 3 presents some preliminary concepts. Section 4 demonstrates our proposal on the following running example. Section 5 includes a discussion on related literature, while Section 6 concludes our paper.

## 2 Working Example

We illustrate our proposal on the following running example as shown in Table 1 containing 32 services $S_1$ to $S_{32}$. We consider the following query: A user provides a binary image of the Barcode of of a product in jpeg format with size less than 50KB along with his current location to obtain the review of the product in English and the location coordinates GPS of the nearest shop and the product price. The query $Q$ is formally written as: $Q = \{\{\text{BinaryImage}, \text{GPS}\}, \{\text{Review}, \text{GPS}, \text{Price}\}, \{\text{Image in jpeg format, Image size less than 50KB}\}, \{\text{Review written in English}\}, \{\}, \{\})$. We begin with defining a few terminologies.

**Definition 3.1. [Web Service]:** A web service $S_i$ is a 6-tuple $(S_i^{ip}, S_i^{op}, M_i, \mathcal{P}_i, \mathcal{PR}_i, \mathcal{PO}_i)$, where, $S_i^{ip}$ and $S_i^{op}$ are the set of input and output parameters of $S_i$ respectively, $M_i$ is the method implemented by $S_i$, $\mathcal{P}_i = (P_{i1}, P_{i2}, \ldots, P_{im})$ are the values of the set of QoS parameters. $\mathcal{PR}_i$ represents a set of preconditions, defined over $S_i^{ip}$ and specified in quantifier free first order logic [18]. $\mathcal{PO}_i$ represents a set of postconditions, defined over the set of output parameters of $S_i$ and specified in quantifier free first order logic.

A web service is executed, once all its inputs are available and the set of preconditions are satisfied. If a web service is executed, it produces the corresponding set of outputs, while ensuring the set of postconditions. The pre and post conditions are optional for a service. Therefore, the pre or post condition / both of them can be null for a service.

**Example 3.1.** Example services can be found in Table 1. The table contains 32 services $S_1$ to $S_{32}$. For each service, a brief description of the method it implemented is provided in the table.

The QoS parameters of a web service are classified into two categories: (a) **Positive QoS**, for which a high value is desirable (e.g., reliability) (b) **Negative QoS**, for which a low value is desirable (e.g., response time).

Two input / output parameters may have syntactically different names. However, both of them may be semantically related. An ontology, which is stored in the service repository along with the web services, maintains all such information. To take into account the semantics of the parameters, a list of concepts and a mapping from each input / output parameter to a concept are maintained in the ontology.

**Definition 3.2. [Concept]:** Each input / output parameter corresponding to an ontology maps to a unique entity $C_i$ of the ontology, called a concept.

**Definition 3.3. [Parameter mapping]:** Parameter mapping $f$ is a mapping of elements in $\mathcal{IO}$ to $C_i$, where $\mathcal{IO}$ and $C_i$ are respectively the set of input-output parameters and the set of concepts corresponding to an ontology.

**Example 3.2.** Consider services $S_1$ and $S_3$ in Table 1. $S_1$ takes a BWImage as input, while $S_3$ takes a BinImage as input. According to the ontology, both refer to the same concept BinaryImage. Therefore, $f(\text{BinImage}) = f(\text{BWImage}) = \text{binaryImage}$.

Two concepts $C_1$ and $C_2$ in an ontology are related in either of the following ways: (a) $C_1 = C_2$: $C_1$ and $C_2$ are identical concepts. (b) $C_1 \sqsubseteq C_2$: $C_1$ is a sub concept of $C_2$, while $C_2$ is a super concept of $C_1$. (c) $C_1 \neq C_2$: $C_1$ and $C_2$ are not related.

**Example 3.3.** Consider 4 different concepts: BinaryImage, GrayImage, RGBImage, Image. The relationship mentioned in an ontology, is as follows: binaryImage $\sqsubseteq$ grayImage $\sqsubseteq$ Image; grayImage $\neq$ RGBImage; RGBImage $\sqsubseteq$ Image;
| Service Name: | ⋄₁ | ⋄₂ |
|--------------|-----|-----|
| Input:       | Image | ProductName |
| Precondition:| Image in jpeg/png format | ProductName in English |
| Method:      | GetImageToProduct | GetReview |
| Output:      | ProductName | Review |
| Postcondition:| ProductName in English | Review in local language |

**Definition 3.4.** [Query]: A query \( Q \) is a 6-tuple \((Q^p, Q^o, IS, OR, OP, CP)\), where, \( Q^p \) and \( Q^o \) are the set of given input and desired output parameters respectively. \( IS \) refers to the input specification, defined over \( Q^p \) and specified in quantifier free first order logic. \( OR \) represents the output requirement, defined over \( Q^o \) and specified in quantifier free first order logic. \( CP \) represents a set of QoS constraints, typically, the bounds on the worst case value of the QoS parameters. \( OP \) refers to the objectives of the query (e.g., maximizing throughput, minimizing response time).

**Example 3.4.** Consider a query. A user provides a binary image of the Barcode of a product in jpeg format with size less than 50KB to obtain the review of the product in English. The query \( Q \) is formally written as: \( Q = (\{BinaryImage\}, \{Review\}, \{Image in jpeg format, Image size less than 50KB\}, \{Review written in English\}, \{Maximum response time is 500ms\}, \{Maximize the reliability\}). The last two elements of the tuple represent QoS constraints and objective of the query respectively. We use this query throughout this paper.

Once a query comes to the system, a set of services are activated based on the query inputs and input specification. We now define the notion of service activation.

**Definition 3.5.** [Service Activation]: A service \( S_i \) is activated, while serving a query \( Q \), if the following are satisfied:

(a) All the inputs of \( S_i \) are available (either query inputs or the set of outputs of already activated services) in the system, i.e., \( \forall i \in S^p, \exists i' \in Q^p \cup S^o_{AC} \) such that \( f(i') \subseteq f(i) \).

(b) All preconditions of \( S_i \) defined over its set of inputs are also satisfied either by the query input specification or by the postconditions of the services that are activated by the query, i.e., \( IS \cup (\forall s, \exists s' \in S^o_{AC} : TS \rightarrow PR_{AC}) \rightarrow PR_i \), where, \( S_{AC} \) is the set of services activated by \( Q^p \) directly / transitively and \( S^o_{AC} \) is the union of the outputs of the services in \( S_{AC} \).

**Example 3.5.** Consider two services, as defined below and a query \( Q = (\{BWImage\}, \{Review\}, \{BWImage in jpeg format\}, \{Review in local language\}, \{}\), \{}\). According to the ontology, \( BWImage \) is a sub concept of \( Image \). Therefore, \( BWImage \) can be fed as an input to \( S_1 \). Moreover, according to the input specification, the BWImage is in jpeg format. Therefore, with the given input specification, the preconditions of \( S_1 \) is also satisfied. Hence, \( S_1 \) is activated first. Once \( S_1 \) is activated, from the output of \( S_1 \), \( S_2 \) is also activated, since \( PO_1 \) satisfies \( PR_2 \). Finally, \( S_2 \) provides \( Q^o \) while ensuring the output requirement.

It may be noted, in this paper, we use the symbol \( A \rightarrow B \) to denote \( B \) holds as well, however, the converse is not true. Once a query comes to the system, the services are activated directly / transitively. Gradually, a dependency graph is constructed to serve the query. We now show an example of a dependency graph on our running example.

**Example 3.6.** Considering our running example (shown in Table 1), the dependency graph, constructed in response to the query \( Q \) (shown in Example 3.4) is shown in Figure 1. Each rectangle presents a group of services having same inputs, outputs, preconditions and postconditions. Each number, situated at the left side of the rectangle, presents the number of services in that group. It may be noted, initially, based on the query inputs and the input specifications, \( S_1 \) to \( S_{10} \) are activated. From the outputs of \( S_1 \) to \( S_{10} \) and their postconditions, \( S_{11} \) to \( S_{17} \) are activated. Finally, \( S_{18} \) to \( S_{20} \), \( S_{24} \) and \( S_{25} \) are activated, that in turn produces the outputs that serve the query and ensure the output requirements. As shown in Figure 1, the dependency graph without abstraction consists of 20 services \((2 + 2 + 2 + 3 + 1 + 3 + 1 + 1 + 3 + 2)\) (rest are not activated). The total number of solutions (in terms of functional dependencies) is 173. The calculation is shown in the figure. Above each rectangle, the calculation shows the number of ways the outputs corresponding to the rectangle can be produced. Finally, the calculation above the query output shows the number of solutions.

We now formally discuss the service composition problem.

**Problem formulation**: We are given the following:

1. A set of web services \( S = \{S_1, S_2, \ldots, S_n\} \).
2. A set of concepts \( C \).
3. For each input-output parameter \( io \), the mapping to its concept \( f(io) \).
4. A query \( Q \).

The aim of service composition is to find a set of services to serve the query with appropriate objectives as specified therein. In this paper, our aim is to expedite the process of composition by reducing the search space as explained in the next section.

4 DETAILED METHODOLOGY

Our framework comprises two key steps: abstraction done at preprocessing time and refinement performed during execution time, i.e., when a query arrives into the system. In the preprocessing phase, a set of services are abstracted based on their functional semantics to obtain a new abstract service. The major advantage of this step is that it reduces the search space and therefore, expedites computation time. Once a query comes to the system, service composition
is performed on the abstract service space first. However, the abstraction has a major limitation due to the lack of information of all the services. In some cases, the solution to a query obtained from the abstract service space may not satisfy all the QoS constraints. Therefore, we further propose to refine an abstraction. Refinement is done in the execution phase. The main benefit of this step is that it makes the method complete [17]. However, an additional computational overhead may be added during composition, if refinement is applied. In the following subsections, we discuss 3 abstraction methods, each followed by the corresponding refinement methods.

4.1 First level Abstraction: Based on Equivalence Relation

We first define the notion of equivalence between services.

**Definition 4.1. [Service Equivalence]** Two services $S_1, S_2$ are functionally equivalent, expressed as $S_1 \simeq S_2$, if the following conditions are satisfied:

1. $\forall i \in S_1^{op}, \exists i' \in S_2^{op}$, such that, $f(i) = f(i')$.
2. $\forall i' \in S_2^{op}, \exists i \in S_1^{op}$, such that, $f(i) = f(i')$.
3. $\mathcal{PR}_1 \leftrightarrow \mathcal{PR}_2$, i.e., $\mathcal{PR}_1 \rightarrow \mathcal{PR}_2$ and $\mathcal{PR}_2 \rightarrow \mathcal{PR}_1$.
4. $\forall o \in S_1^{op}, \exists o' \in S_2^{op}$, such that, $f(o) = f(o')$.
5. $\forall o' \in S_2^{op}, \exists o \in S_1^{op}$, such that, $f(o) = f(o')$.
6. $\mathcal{PO}_1 \leftrightarrow \mathcal{PO}_2$, i.e., $\mathcal{PO}_1 \rightarrow \mathcal{PO}_2$ and $\mathcal{PO}_2 \rightarrow \mathcal{PO}_1$.

The notion of equivalence adopted here is different from the one in [17]. Here the equivalence is considered in the semantic space and not just based on syntactic names of input-outputs.

**Example 4.1. Consider two services $S_1$ and $S_3$ as shown in Table 1.** From the definitions of $S_1$ and $S_3$, $S_1 \simeq S_3$, since according to ontology, $f(BWImage) = f(binImage) = binaryImage$. Moreover, BWImage in jpeg format $\leftrightarrow$ BinImage in jpeg format and BWImage size must be less than 50KB $\leftrightarrow$ BinImage size must be less than 50KB.

Two services $S_1$ and $S_2$ satisfying the first three conditions of Definition 4.1 are called input equivalent, whereas, two services satisfying the last three conditions of the definition are called output equivalent.

It may be noted, the binary relation “$\simeq$” defined over a set of services is an equivalence relation. The first level of abstraction is done based on the equivalence relationship between services. Since “$\simeq$” is an equivalence relation, the service repository is partitioned based on this relation. Each equivalence class is then abstracted by a new representative service. The number of services required to serve a query reduces by this abstraction. Since the service repository is partitioned into a set of equivalence classes and the equivalence classes are mutually exclusive and collectively exhaustive, the number of equivalence classes in the service repository is less or equal to the number of services in the repository.

Consider $S' \subseteq S$ be a set of equivalent services. Using the above method, $S'$ is abstracted by a new service, say $S'$. To assign the inputs, outputs of the abstract service, we choose any member service ($S_i \in S'$) from its corresponding equivalence class (since each member service is functionally equivalent) and assign the following:

$S_1^{ip} = \{f(j) : \forall j \in S_1^{ip}\}$; $S_1^{op} = \{f(o) : \forall o \in S_1^{op}\}$

**Example 4.2.** Considering our running example (shown in Table 1), the dependency graph constructed in response to $Q$ (shown in example 3.4) with the first level abstract services is shown in Figure 2. After this abstraction, the dependency graph now consists of 9 services and the total number of solutions (in terms of functional dependencies) is now reduced from 173 to 13. The calculation is shown in the figure.

Once an abstract service $S_1$ is constructed for each equivalence class, it is also important to assign the QoS values to $S_1$. To do so, we choose a representative service from each corresponding equivalence class and assign the QoS values of the representative service to it. Consider $S_1$ is an abstract service corresponding to the equivalence class $S' \subseteq S$. Below, we discuss the method of choosing the representative service.

**Step 1:** For each service $S_i$ in an equivalence class, we first normalize the values of the QoS parameters between 0 and 1. To normalize, at first, the maximum ($P_i^{Max}$) and minimum ($P_i^{Min}$) values of each QoS parameter $P_i$ across the equivalence class are computed. The normalized value of the parameter is computed as:

$NV(P_i) = \begin{cases} P_i - P_i^{Min} & \text{if } P_i \text{ is a positive QoS} \\ P_i^{Max} - P_i & \text{if } P_i \text{ is a negative QoS} \end{cases}$

**Step 2:** Once the QoS parameters are normalized, we compute the weighted sum of the normalized QoS parameters:

$WS(S_1) = \sum_{i=1}^{m} (NV(P_i) \times w_i); \quad w_i \in [0, 1]$  

We assume that the weights of the QoS parameters can be provided externally by the user. If no weights are specified, all QoS parameters are provided equal weights, i.e., $\forall l \in \{1, 2, \ldots, m\}, w_i = \frac{1}{m}$.

**Step 3:** The service with maximum weighted sum is chosen as the representative service for each equivalence class.

4.2 Composition with First Level Abstract Services

Any service composition algorithm can be applied over the abstract service space to get a solution to a query. The major advantage of this abstraction method is that for a single QoS parameter, it preserves the solution quality. In other words, if an optimal algorithm is used to find a solution, it is guaranteed to find an optimal solution from the abstract
service space as well. Similarly, if a heuristic algorithm
is used to find a solution from the abstract space, it is
guaranteed that the solution quality does not degrade in
comparison to the solution that can be generated from the
original / un-abstracted service space by the same algo-
rithm. However, for multiple QoS parameters, we cannot
guarantee this guarantee. In some scenarios, it may not be
possible to generate any solution from the abstract service
space that satisfies all the QoS constraints, though one exists
in the original service space. In such scenarios, refinement is
required. In the next subsection, we discuss the refinement
techniques in detail.

4.3 First Level Refinement

In this paper, we consider two refinement techniques: partial
refinement and complete refinement.

Partial Refinement

If the composition algorithm returns a solution in terms
of the abstract services which does not satisfy all the QoS
constraints, we first apply the partial refinement on the
solution. It may be noted, one solution in terms of abstract
services consists of multiple solutions in terms of original
services. The main idea of the partial refinement technique
is to change the representative service corresponding to
each abstract service belonging to the solution and thus
to change the QoS values of the solution. As discussed
earlier, the representative service is chosen on the basis of
the weighted sum method. Here, we change the weights of
the QoS parameters on the basis of the constraint values
to choose a different service. We first consider a solution
s consisting of the set of abstract services \( S_1 \). We now discuss
how we modify the weights of the QoS parameters.

Step 1: For each abstract service \( S_1 \in S_1 \), we find the
maximum \( \text{Max}(S_1, \mathcal{P}_l) \) and the minimum \( \text{Min}(S_1, \mathcal{P}_l) \)
values for each QoS parameter \( \mathcal{P}_l \) of the solution. If the
constraint value of \( \mathcal{P}_l \) lies between the best and worst
aggregated value, we compute the normalized value of the constraint as:

\[
NV(\mathcal{P}_l) = \frac{\text{Aggr}_\text{max}(\mathcal{P}_l) - \text{Aggr}_\text{min}(\mathcal{P}_l)}{\text{Aggr}_\text{max}(\mathcal{P}_l) - \text{Aggr}_\text{min}(\mathcal{P}_l)}
\]

Step 2: For each abstract service \( S_1 \in S_1 \), we compute the maximum
aggregated value \( \text{Aggr}_\text{max}(s, \mathcal{P}_l) \) for each QoS parameter
\( \mathcal{P}_l \) considering \( \text{Max}(S_1, \mathcal{P}_l) \) as the value of \( \mathcal{P}_l \) of \( S_1 \),
for all \( S_1 \in S_1 \). Similarly, we compute the minimum
aggregated value \( \text{Aggr}_\text{min}(s, \mathcal{P}_l) \), for each QoS parameter
\( \mathcal{P}_l \) considering \( \text{Min}(S_1, \mathcal{P}_l) \) as the value of \( \mathcal{P}_l \) of \( S_1 \),
for all \( S_1 \in S_1 \).

Step 3: Among \( \text{Aggr}_\text{max}(s, \mathcal{P}_l) \) and \( \text{Aggr}_\text{min}(s, \mathcal{P}_l) \), we
consider one as the best value and the other as the worst
value of \( \mathcal{P}_l \) for the solution. If \( \mathcal{P}_l \) is a positive (negative)
QoS parameter, \( \text{Aggr}_\text{max}(s, \mathcal{P}_l) \) is considered to be the best
(worst) value and \( \text{Aggr}_\text{min}(s, \mathcal{P}_l) \) is considered to be the
worst (best) value of \( \mathcal{P}_l \).

Step 4: If the best value of any QoS parameter \( \mathcal{P}_l \) of the
solution does not satisfy the constraint, we do not apply
the partial refinement, since, no modification of the representa-
tive service corresponding to any abstract service can make
the constraint satisfiable.

Step 5: If the worst value of any QoS parameter \( \mathcal{P}_l \) of the
solution is better than the constraint value, we ignore the
QoS parameter \( \mathcal{P}_l \) from the weight computation. This is
mainly because, no matter what representative service is
chosen for an abstract service belonging to the solution, the
constraint value is always satisfied.

Step 6: If the constraint value of \( \mathcal{P}_l \) (say \( \delta(\mathcal{P}_l) \)) lies between the
best and worst aggregated value, we compute the
normalized value of the constraint as:

\[
NV(\mathcal{P}_l) = \begin{cases} 0 & \text{if Step 5 is satisfied} \\ \frac{\delta(\mathcal{P}_l) - \text{Aggr}_\text{min}(s, \mathcal{P}_l)}{\text{Aggr}_\text{max}(s, \mathcal{P}_l) - \text{Aggr}_\text{min}(s, \mathcal{P}_l)} & \text{if } \mathcal{P}_l \text{ is a positive QoS} \\ \frac{\text{Aggr}_\text{max}(s, \mathcal{P}_l) - \delta(\mathcal{P}_l)}{\text{Aggr}_\text{max}(s, \mathcal{P}_l) - \text{Aggr}_\text{min}(s, \mathcal{P}_l)} & \text{if } \mathcal{P}_l \text{ is a negative QoS} \end{cases}
\]

It may be noted, when we assign the QoS values to an
abstract service during the preprocessing phase, we do not
have any information regarding the QoS requirements of
a query, which can only be known when a query comes
to the system during the execution phase. Moreover, the
QoS constraints vary across queries. Therefore, in the partial
refinement phase, our intuition is to set the weights of the
QoS parameters according to the QoS requirements by the
query. Since the QoS parameters are disparate in nature, in
order to compare them, we first normalize a QoS constraint
value with respect to the maximum and minimum aggrega-
ted values of the parameter that can be provided by the
solution. It may be noted, a solution can provide maximum
normalized QoS value as 1 for any QoS parameter. However,
a QoS parameter \( \mathcal{P}_l \) requires the normalized value as
\( NV(\mathcal{P}_l) \) to satisfy its corresponding constraint. Therefore,
we set the weight of the QoS parameters proportionate to
their normalized constraint values.

Step 7: The weight for each QoS parameter is finally com-
puted as:

\[
WT(\mathcal{P}_l) = \frac{\sum_{k=1}^{m} NV(\mathcal{P}_k)}{m}
\]

Once we compute the weights of the QoS parameters,
we reiterate the algorithm for choosing the representative
service as discussed above and assign the QoS values of the
representative service to its corresponding abstract
service. Finally, we compute the aggregated QoS values of
the solution with the modified abstract services. If the
solution now satisfies all the QoS constraints, we return
the solution. Otherwise, we apply complete refinement as
discussed below.

Example 4.3. Consider a solution to a query as shown in Figure
3 which is a sequential combination of two abstract services
\( S_1 \) and \( S_2 \), each corresponding to 3 un-abstract services.
Each service has 2 QoS parameters: response time and reliability (as
shown by the tuple in the figure). The calculated weighted sum
for each service is also shown in figure. For each abstract service,
the corresponding representative service is shown by gray color.
The QoS values of the solution is \((80, 0.76)\). Now consider the
following two cases:

Case 1: The query has the following constraints: \((200, 0.8)\).
Clearly, the solution violates the reliability constraint. Therefore,
we apply partial refinement. The maximum and minimum values of the response time corresponding to \( S_1 \) are 70 and 30ms respectively and the same for \( S_2 \) are 90 and 30ms respectively. Therefore, the maximum aggregated response time of the solution is 160ms (90 + 70) and the minimum aggregated value for the solution is 60ms (30 + 30). Since the response time constraint is 200ms, we can ignore response time for the weight calculation. This is because whatever representative service is chosen for an abstract service, the response time constraint remains satisfied. Similarly, the maximum and minimum values of the reliability corresponding to \( S_1 \) are 0.95 and 0.8 respectively and the same for \( S_2 \) are 0.99 and 0.7 respectively. Therefore, the maximum aggregated reliability of the solution is 0.94 (0.95 * 0.99) and the minimum aggregated value for the solution is 0.56 (0.8 * 0.7). The constraint value lies between 0.94 and 0.56. Therefore, we first calculate the normalized constraint value for reliability \( \frac{0.8-0.56}{0.94-0.56} = 0.63 \). Following the above algorithm, the weights for response time and reliability are 0 and 1 (0.63/0.63). Using the calculated weights, we reiterate the algorithm for choosing the representative service. \( S_2 \) is now the representative service for \( S_1 \) and \( S_2 \) is the representative service for \( S_2 \), since they have now maximum weighted sum compared to the other members belonging to their corresponding equivalence classes. Hence, the solution now has the QoS values \((160, 0.94)\), which satisfies all the constraints.

Case 2: The query has the following constraints: \((50, 0.8)\). In this case, the response time constraint can never be satisfied using these two abstract services, since the best response time value for the solution is 60ms. Therefore, complete refinement is required.

Complete Refinement

If the composition algorithm does not return any solution or the partial refinement technique fails to generate a solution, we apply the complete refinement technique. The main idea of the complete refinement is to revert the abstract services to the original services and apply the composition algorithm on the original service space once more to obtain the solution.

4.4 Second level Abstraction: Based on Dominance Relation

This abstraction is done based on the notion of dominance. In this abstraction, we cluster the set of abstract services formed after the step above into further groups, where one service dominates the rest of the services.

Definition 4.2. [Dominance]: A service \( S_1 \) dominates another service \( S_2 \), expressed as \( S_1 \succ S_2 \), if the following conditions hold:

- \( \forall i \in S_1^{ip} \exists i' \in S_1^{ip} \), such that \( i \equiv i' \).
- \( \forall o' \in S_2^{op}, \exists o \in S_1^{op} \), such that \( o \equiv o' \).
- \( PR_1 \rightarrow PR_1 \), \( PO_1 \rightarrow PO_1 \).
- \( S_1 \) and \( S_2 \) are not output equivalent.

The notion of dominance essentially indicates that the functionality of \( S_2 \) is covered by the functionality of \( S_1 \). In other words, \( S_1 \) can be activated, if \( S_2 \) gets activated and \( S_1 \) provides more functionality than \( S_2 \) (either provides more outputs or provides more specific outputs, i.e. at least one output of \( S_1 \) is a sub concept of \( S_2 \) or the postcondition of \( S_1 \) implies the postcondition of \( S_2 \)). The last condition is used for the next abstraction level.

Example 4.4. Consider two services \( S_9 \) and \( S_{11} \), as shown in Figure 2. From their definitions, \( S_{11} \succ S_9 \), since according to ontology, \( PID_9 \subset PID \), \( PR_{19} \rightarrow PR_{11} \) and \( \{Review\} \subset \{Review, Rating\} \). It may be noted, \( S_{11} \) must be activated, if \( S_9 \) gets activated and \( S_{11} \) provides more outputs than \( S_9 \).

Consider \( S_1 \) is the set of first level abstract services. If a service \( S_1 \in S_1 \) is not dominated by any other service in \( S_1 \) (i.e., \( S_1 \) is non-dominated), \( S_1 \) forms a group consisting of all the first level abstract services that are dominated by \( S_1 \). The group is finally abstracted by a new service, say \( S_{21} \). The input, output parameters, preconditions and postconditions of \( S_2 \) are same as \( S_1 \). The QoS values of \( S_1 \) are assigned to \( S_{21} \). It may be noted, though, every dominated service \( S_{21} \) belongs to at least one group, however, no such service forms its separate group containing all the first level abstract services that are dominated by \( S_2 \). The set of services corresponding to \( S_2 \) and \( S_{21} \) (\( i \neq j \)) are not always mutually exclusive, since one service can be dominated by multiple services. However, the number of services after the second level of abstraction still reduces. This is because the maximum number of non-dominated services is at most equal to the number of services after the first level.

Example 4.5. Considering our running example (shown in Table 1), the dependency graph constructed in response to \( Q \) (shown in Example 3.4) with the second level abstract services is shown in Figure 4. The dependency graph now consists of 7 services and the total number of solutions (in terms of functional dependencies) is now reduced from 13 to 7. The calculation is shown in the figure.

As earlier, any service composition algorithm can be applied over the abstract service space to get a solution to a query. However, unlike the first level abstraction, the second level abstraction is not quality preserving. Therefore, refinement may be required even for a single QoS parameter.

4.5 Second Level Refinement

In this level as well, we have two different refinement techniques: partial refinement and complete refinement. The
partial refinement here is similar to the previous level. The only difference is that for a second level abstract service $S_2$, all its corresponding first level abstract services do not participate in selecting the representative service for $S_2$. This is because of the following reason: consider $S_1$ is the non dominated first level service belonging to the group corresponding to $S_2$. First of all, it is not necessary, when $S_1$ gets activated all other services in the group are also activated with the available inputs in the system. Furthermore, $S_1$ provides more functionality than any other first level abstract services in that group, which may require serving the query. In this case, no other service can participate in serving the query. Therefore, we have to first select the set of services corresponding to a second level abstract service that can serve the query.

Consider a second level abstract service $S_2$ belonging to the solution corresponding to the set of first level abstract services: $S_1' \in S_1$, where $S_1$ is the set of first level abstract services. A first level abstract service $S_1 \in S_1'$ participates in the representative service selection method, if the following conditions are satisfied: (1) $S_1$ can be activated from the available input set, when $S_2$ is activated. (2) $S_1$ produces all the parameters (in the form of identical / sub concept) that are required to be produced by $S_2$ only, i.e., no other service in the solution produces them. More formally, $\forall i_{o_1} \in IO, \exists i_{o_2} \in S_1^{op}, i_{o_2} \subseteq i_{o_1}$, where $IO$ is the set of parameters that are required to be produced by $S_2$ only. (3) The post conditions of $S_1$ must satisfy the set of conditions that are required to be ensured by $S_2$. More formally, $PO_{i_1} \rightarrow PC$, where, $PC$ is the set of conditions that are required to be ensured by $S_2$.

Once the representative services are chosen, the input-output parameters, pre and postconditions and the QoS values are changed according to the chosen representative services. If a valid solution is obtained, we return the solution. Otherwise, we apply the complete refinement technique.

4.6 Abstraction Using Input Implication Output Equivalence

This is the third and the final level of abstraction. The abstraction is done based on the notion of input equivalence and output implication. Here, we focus on the set of services that produce equivalent set of outputs and ensure the same postconditions, however, one service dominates the other service in terms of inputs and preconditions.

Definition 4.3. [Input Implication Output Equivalence Relation (IIOE)]: IIOE is a relation between two services $S_2$ and $S_2'$ (expressed as $S_2 \xrightarrow{\text{IIOE}} S_2'$), which ensures $S_2$ is output equivalent to $S_2'$ and $S_2'$ implies $S_2$ in terms of input parameters satisfying the following conditions:

- $\forall i \in S_2' \exists i' \in S_2^{op}$, such that $i \subseteq i'$.
- $PR_{21} \rightarrow PR_{22}$. $\blacksquare$

The above definition means, though both the services $S_2$ and $S_2'$ provide same output and ensure same functionality, however, $S_2$ is more generic than $S_2'$ in a sense that $S_2$ can be activated with less number of inputs or with weaker preconditions.

Example 4.6. Consider two services $S_2$ and $S_3$, as shown in Figure 4. From their definitions, $S_2 \xrightarrow{\text{IIOE}} S_3$, since both the services are output equivalent and according to the ontology, $\text{BinaryImage} \sqsubset \text{Image}$ and $PR_{21} \rightarrow PR_{22}$.

We first construct an IIOE graph based on the IIOE relation between the services.

Definition 4.4. [IIOE Graph]: An IIOE graph $G_{IIOE} = (V, E)$ is a directed acyclic graph, where $V$ is the set of nodes and $E = \{(u_1, u_2); u_1, u_2 \in V\}$ is the set of edges. Each node $u_i \in V$ contains a first-level abstract service $S_2$. An edge from $u_i$ to $u_j$ exists, if $S_2 \xrightarrow{\text{IIOE}} S_2'$. $\blacksquare$

We first create all possible nodes and edges in the IIOE graph. Finally, we remove all the transitive edges to make the composition algorithm efficient, as discussed later. If $S_2 \rightarrow S_2'$ and $S_2' \rightarrow S_2''$, we remove the edge from $u_i$ to $u_k$.

Since, each node $u_i \in V$ contains a second-level abstract service $S_2$, the number of nodes in $G_{IIOE}$ is equal to the number of second-level abstract services. Now consider a node $u_i \in V$ in $G_{IIOE}$ containing an abstract service $S_2$. It may be noted, if $S_2$ is activated in the system, all the services corresponding to the tree rooted at $u_i$ are also activated at the same time, and provide same functionality as $S_2$. Therefore, the entire tree is abstracted by a new service, say $S_3$. In other words, each final level abstract service $S_3$ corresponds to a tree rooted at the node $u_i$ in the implication graph. Therefore, the number of final level abstract services is equal to the number of second level abstract services. The input, output parameters, pre and postconditions of $S_3$ are exactly same as $S_2$, and the QoS assignment technique for this abstraction level is same as the first level abstraction, i.e., for each final level abstract service $S_3$, we choose a representative service from its corresponding set of second level services and assign its QoS values to $S_3$.

Example 4.7. Consider three services $S_2$, $S_2'$ and $S_3$, as shown in Figure 5. From their definitions, $S_2 \rightarrow S_2' \rightarrow S_3$, since according to ontology, $\text{BinaryImage} \sqsubset \text{Image}$ and $PR_{21} \rightarrow PR_{23}, PR_{22} \rightarrow PR_{23}$. $G_{IIOE}$ constructed from the services is shown in Figure 5. Each third level abstract service is shown with a different color in the figure. Considering our running example (shown in Table 1), the dependency graph constructed in response to $Q$ (shown in Example 3.4) with the final level abstract services is shown in Figure 5. After the final level of abstraction, the
dependency graph now consists of 5 services and the total number of solutions (in terms of functional dependencies) is now reduced from 7 to 3 \((((\{S_3\}) \times \{S_6, S_7\}) \cup \{S_4\}) \times \{S_8\}) = (1 \times 2 + 1) \times 1 \times 1 = 3\). ■

Though the number of services in this step is exactly same as in the previous step, the number of nodes in the dependency graph reduces from the previous level of abstraction. During composition, if one final-level abstract service \(S_3\), is chosen in the dependency graph, no other final level service is chosen for which its corresponding second level service belongs to the tree rooted at the node containing \(S_2\), where \(S_3\), corresponds to the tree rooted at \(S_2\). Therefore, the composition algorithm becomes efficient, if it starts traversing the graph top down (i.e., starting from the root node of any connected component of the IIOE graph) during dependency graph construction.

Example 4.8. As shown in Figure 5, if \(S_3\) is chosen to serve a query, \(S_3\) and \(S_3\) are not chosen. Similarly, if \(S_5\) is chosen, \(S_6\) and \(S_7\) are not chosen.

The final level refinement techniques are exactly same as the first level refinement techniques.

4.7 Solution Reconstruction and Analysis

Once a solution is generated in terms of the abstract services, we revert the solution from the abstract space to the original service space. The abstract service in each level is replaced by its previous level service until we reach the original service space. Our framework provides the following guarantees.

Lemma 1. Soundness: A QoS constraint satisfying solution containing abstract services produced by any composition algorithm is a valid solution satisfying all QoS constraints on the original service space.

Proof. Once a QoS constraint satisfying solution is generated in terms of the abstract services, the solution is reverted from the abstract space to the original service space. It may be noted, each \(i^{th}\) \((i = 1, 2, 3)\) level abstract service of the solution, containing the \(i^{th}\) level abstract services, is replaced by its corresponding previous level service having the same QoS values. Therefore, the overall QoS values of the solution remains unchanged on the original service space. Hence, we always get a valid solution satisfying all QoS constraints on the original service space.

Lemma 2. A solution (in terms of functional dependencies) to a query can always be constructed by composing the abstract services, if and only if there exists a solution to the query in the original service space.

Proof. We first consider the following cases:

- If any service \(S_i\) is eventually activated from the query inputs and input specification, the abstract service \(S_1_j\) containing \(S_i\) must be activated, since both of them are functionally equivalent and will produce the set of outputs and identical postconditions, that further producing the query outputs eventually with desired output requirements, if there exists a solution to the query.

- If any service \(S_1\) is eventually activated from the query inputs and input specification, the abstract service \(S_2_j\) containing \(S_1\) must be activated, since the representative service of \(S_2_j\), either \(S_1\), itself or dominates \(S_1\). Therefore, \(S_2\), will produce the set of outputs containing the output set of \(S_1\), having stronger postconditions, that further producing the query outputs eventually with desired output requirements, if there exists a solution to the query.

- For each service \(S_2\), there exist a third level abstract service \(S_3\), having identical set of inputs, outputs, preconditions and postconditions. Therefore, the same argument holds for this case as well.

Therefore, if there exists a solution to a query in the original service space, we can always construct a solution in terms of the functional dependencies by composing any level of abstract services.

Therefore, if there exists a solution to a query in the original service space, we can always construct a solution in terms of the functional dependencies by composing any level of abstract services.

Our entire framework provides the following guarantee.

Lemma 3. Our framework is always able to find a valid solution, if one exists.

If no solution is found from an abstraction level, we can always go back to the previous level by applying the complete refinement and search for the solution. In the worst case, we may end up searching for a solution in the original service space. Therefore, we never miss a solution, if one exists.

5 Experimental Results

Our proposed algorithms were implemented in Java. We implemented our framework on top of two composition algorithms [11] and [20]. We first applied our framework on the WSC-2008 datasets [21]. Since the services in the WSC dataset are generated complete randomly [15], we did not find significant reduction in any abstraction step. For the 2nd dataset, we found 1 pair of equivalent services, for the 7th dataset 1 pair of equivalent services and 1 pair of dominant services and finally for the 8th dataset, we found 2 pairs of dominant services. For rest of the datasets, we could not find any reduction due to abstraction. Hence, our framework could not provide significant performance gain for WSC-2008 datasets. Therefore, we demonstrate our method on a synthetically generated dataset.

Our synthetic dataset is an extension of our example dataset as shown in Table 1. Our example dataset is constructed from the dataset described in [15]. We now discuss the construction of our synthetic dataset. We first generated a random number of concepts and assigned a random relationship (identical / sub / super / unrelated) between the concepts. We then randomly generated the number of input-output parameters and randomly associated them with the concepts. However, the services were also generated in a semi random manner. We increased the number of services in a random manner maintaining the relationship (equivalence / dominance / IIOE / unrelated) between the services as discussed above and accordingly assigned the input-output parameters, preconditions and postconditions. Finally, we generated the QoS parameters of the services.
In this paper, we considered 4 QoS parameters: response time, throughput, reliability, availability and modeled them assuming they follow normal distribution \cite{22}. Once the service repository was created, we generated a random query having random inputs, outputs, input-specifications, output-requirements and QoS constraints. We now analyze the results.

We present our experiment in two parts. In the first part, we show the performance gain achieved by our framework at runtime, when implemented on top of \cite{20}. Figure 6 shows the reduction in the number of services in the service repository by applying each abstraction level for 5 random datasets (RDS1 to RDS5). Figure 7 shows the reduction in the number of services in the dependency graph constructed to respond to a query by applying each abstraction level for 5 random datasets. On an average, there are 6x, 10x and 16x reductions in the number of services in the dependency graph for abstraction levels 1, 2 and 3 respectively with respect to the number of unabstracted services. Table 2 shows the computation time to construct the dependency graph generated in response to the query for each of the 3 abstractions. Columns 2 to 5 of the table show the computation time required by the underlying algorithm for 4 levels of services (the 0th level corresponds to without abstraction). Columns 6 to 8 of the table show the speed up \((SU_i)\) for \(i\)th \((i = 1, 2, 3)\) abstraction level respectively with respect to the un-abstracted service space (i.e., Level 0). As evident from the table, we achieved, on an average, 19x, 30x, 70x speed up for abstraction levels 1, 2 and 3 respectively. Figure 8 shows the response time \((RT)\) obtained by Algorithm \cite{20} for 4 levels of services. The response time obtained by our framework, on an average, degrades 4x, 5x, 6x for abstraction levels 1, 2 and 3 respectively, while the computation speed up achieved by our framework to compute response time is on average 40x, 43x, 126x (shown in Table 3). Similarly, the throughput obtained by our framework, on an average, degrades 1.28x, 1.33x, 1.36x respectively for abstraction levels 1, 2 and 3 (shown in Figure 9), while the speed up achieved by our framework to compute throughput is on average 62x, 64x, 142x (as in Table 4).

In the second part of our experiment, we show the performance of our framework implemented on top of \cite{11}. Here, we considered the query having QoS constraints. We conducted three sets of experiments (Exp. 1, Exp. 2 and Exp. 3 of Table 5) varying the number of services and query and recorded the computation time to serve a query. In the final experiment (Exp. 3 of Table 5), we varied the constraints of the QoS parameters (9th Column has tighter constraints than 8th, which has tighter constraints than 7th) for the same query and recorded the result. As can be seen from the table, in all 3 experiments, as the abstraction level increases, the number of services either remains same or decreases, which we have claimed in this paper. However, it is not guaranteed that the computation time to find a solution always improves as the abstraction level increases, because in some cases, complete refinement is necessary (as can be seen in the first experiment). In the first experiment,
TABLE 2: Computation Time (ms) and speed up for dependency graph construction

| DataSet | Abs. 0 | Abs. 1 | Abs. 2 | Abs. 3 | SU1 | SU2 | SU3 |
|---------|--------|--------|--------|--------|-----|-----|-----|
| RDS1    | 731    | 79    | 56     | 28     | 10.14 | 1.25 | 2   |
| RDS2    | 3493   | 109   | 89     | 19     | 32.05 | 1.22 | 4.68 |
| RDS3    | 6159   | 380   | 221    | 154    | 16.21 | 1.72 | 1.44 |
| RDS4    | 17930  | 893   | 532    | 334    | 20.08 | 1.68 | 1.59 |
| RDS5    | 15904  | 874   | 421    | 343    | 18.2  | 2.08 | 1.23 |

TABLE 3: Computation Time (ns) and speed up for optimal RT computation

| DataSet | Abs. 0 | Abs. 1 | Abs. 2 | Abs. 3 | SU1 | SU2 | SU3 |
|---------|--------|--------|--------|--------|-----|-----|-----|
| RDS1    | 3752358 | 2111230 | 1921228 | 884105 | 17.77 | 1.1 | 2.17 |
| RDS2    | 348775813 | 6014884 | 5817691 | 1079896 | 57.99 | 1.03 | 5.39 |
| RDS3    | 192147284 | 3180854 | 3021811 | 2012347 | 60.41 | 1.05 | 1.50 |
| RDS4    | 389731251 | 17119550 | 15407595 | 3599254 | 22.77 | 1.11 | 4.28 |
| RDS5    | 209139820 | 4896525 | 4651698 | 3370275 | 18.2  | 2.08 | 1.23 |

TABLE 4: Computation Time (ns) and speed up for optimal throughput computation

| DataSet | Abs. 0 | Abs. 1 | Abs. 2 | Abs. 3 | SU1 | SU2 | SU3 |
|---------|--------|--------|--------|--------|-----|-----|-----|
| RDS1    | 37098104 | 2127783 | 1978838 | 1027102 | 17.44 | 1.05 | 1.93 |
| RDS2    | 425637217 | 3029113 | 2998821 | 1040155 | 140.52 | 1.01 | 2.88 |
| RDS3    | 28199735 | 3893887 | 3777070 | 2154407 | 72.22 | 1.03 | 1.75 |
| RDS4    | 38649038 | 7178013 | 7106232 | 2382924 | 5.38  | 1.01 | 2.98 |
| RDS5    | 264711653 | 3455560 | 3386448 | 2291248 | 76.60 | 1.02 | 1.48 |

TABLE 5: Computation Time (ms) for constraint satisfying solution computation

| Level | Exp. 1 | Exp. 2 | Exp. 3 |
|-------|--------|--------|--------|
|       | # Services | Time | # Services | Time | # Services | Time |
| Abs. 0 | 3846 | 29862 | 9718 | 25896 | 98361 | - |
| Abs. 1 | 1297 | 11596 | 6389 | 13791 | 53846 | - |
| Abs. 2 | 1123 | 3276 | 5231 | 11563 | 19326 | 3695 |
| Abs. 3 | 201 | 9698 | 1226 | 2215 | 1856 | 1532 |

TABLE 6: Computation Time (ms) for constraint satisfying solution computation

| Level | Exp. 1 | Exp. 2 | Exp. 3 |
|-------|--------|--------|--------|
|       | # Services | Time | # Services | Time | # Services | Time |
| Abs. 0 | 3846 | 29862 | 9718 | 25896 | 98361 | - |
| Abs. 1 | 1297 | 11596 | 6389 | 13791 | 53846 | - |
| Abs. 2 | 1123 | 3276 | 5231 | 11563 | 19326 | 3695 |
| Abs. 3 | 201 | 9698 | 1226 | 2215 | 1856 | 1532 |

we have shown as the abstraction level increases, the time requirement to compute a solution decreases up to the 2nd level of abstraction. However, in the 3rd level of abstraction, the time requirement increases. It is due to the fact that in the 3rd level of abstraction no solution could be found. Therefore, complete refinement was performed. In the last experiment, "-" indicates no solution is generated due to time out. It may be noted, no solution was generated in the lower level of abstraction as well. However, at the higher level, we obtained a solution. It may be observed, on an average, the composition algorithm in the abstract service space outperforms the composition algorithm in the unabstract service space.

6 RELATED WORK

Research on service composition has been carried out in multiple directions. A significant amount of work considering semantic web service composition has been performed considering optimality being the primary concern (e.g., graph based approaches [2], [3], ILP based approaches [4], [5]). AI planning based approaches [6], [7]. However, in real time, optimality has proved to be an expensive requirement [8], [9]. Furthermore, since most of the optimal methods do not scale for large service repositories, a number of heuristic methods [9], [26], [27], [28] have been proposed, where computation time is considered as the main objective. In [29] and [30], authors have proposed the multiconstrained QoS aware service composition approach, instead of finding the optimal solutions. Dynamic binding is the main concern of [31], where authors propose to generate the skyline services for each task and cluster the services using the K-means algorithm. In [32], authors proposed an Integer Linear Programming (ILP) based approach, where ILP is used to divide the global constraints into a set of local constraints and then using the local constraints, the service selection is done for each task. In [33], authors proposed a planning graph based approach and an anytime algorithm [34] that attempts to maximize the utility. In [35], authors proposed an Integer Linear Programming (ILP) based method to maximize the utility. In [36], authors proposed an anytime algorithm [37], [38]. In [39], the authors proposed a planning graph based approach and an anytime algorithm [39] that attempts to maximize the utility. In [40], authors proposed an ILP based method to maximize the utility. In [41], the authors analyze the relation between multiobjective service composition and the Multi-choice, Multi-dimensional 0-1 Knapsack Problem (MMP). There are few methods [42], [43] in literature, that enlighten the issue of search space reduction of the composition algorithm. However, they often fail to generate a solution with desired quality.

In contrast to existing literature, we propose a framework for semantic web service composition based on abstraction refinement that aims to expedite the solution construction time by reducing the search space without compromising on the solution quality. Service clustering based
on syntactic similarity in terms of input-output parameters has been dealt with in our earlier work [17]. However, the notion of semantics of a web service has not been dealt with in that work. In this paper, we bring the notion of semantics and propose a methodology that provides a scalable way of pruning the composition search space. Our method can be applied on top of any service composition algorithm to improve its performance. To conclude, we do not propose any new service composition algorithm, rather a framework on top of existing algorithms. This adds a unique novelty to this work.

7 CONCLUSION
This paper presents an approach for reducing the search space for semantic web service composition based on abstraction refinement. For a large dataset, the approach can be quite efficient, since it greatly improves performance. As future work, we are currently working on designing more sophisticated refinement techniques. We also aim to perform more extensive experiments of this framework on real datasets.

REFERENCES
[1] C. Jatooth et al., “Computational intelligence based qos-aware web service composition: A systematic literature review,” TSC, vol. PP, no. 99, pp. 1–1, 2015.
[2] D. Schuller et al., “Cost-driven optimization of complex service-based workflows for stochastic qos parameters,” in ICWS, 2012, pp. 66–73.
[3] P. Rodriguez-Mier et al., “Hybrid optimization algorithm for large-scale qos-aware service composition,” TSC, 2015.
[4] Y. Wu et al., “A multilevel index model to expedite web service discovery and composition in large-scale service repositories,” IEEE TSC, vol. 9, no. 3, pp. 330–342, 2016.
[5] S. Chattopadhyay et al., “A fast and scalable mechanism for web service composition,” TWEB, vol. 11, no. 4, pp. 261–2636, 2017.
[6] F. Lecue and N. Mehandjiev, “Towards scalability of quality driven semantic web service composition,” in ICWS, 2009, pp. 469–476.
[7] D. Papadias et al., “An optimal and progressive algorithm for skyline queries,” in SIGMOD. ACM, 2003, pp. 467–478.
[8] S. Deng et al., “Service selection for composition with qos correlations,” TSC, vol. 9, no. 2, pp. 291–303, 2016.
[9] P. Rodriguez-Mier et al., “Automatic web service composition with a heuristic-based search algorithm,” in ICWS, 2011, pp. 81–88.
[10] R. Aggarwal et al., “Constraint driven web service composition in meteor-s,” in SCC, 2004, pp. 23–30.
[11] S. Chattopadhyay et al., “QSCAS: qos aware web service composition algorithms with stochastic parameters,” in ICWS, 2016, pp. 388–395.
[12] A. Klein et al., “Efficient heuristic approach with improved time complexity for qos-aware service composition,” in ICWS, 2011, pp. 436–443.
[13] M. Alrifai et al., “Combining global optimization with local selection for efficient qos-aware service composition,” in WWW, 2009, pp. 881–890.
[14] P. Hennig et al., “Highly scalable web service composition using binary tree-based parallelization,” in ICWS, 2010, pp. 123–130.
[15] F. Wagner et al., “Qos-aware automatic service composition by applying functional clustering,” in ICWS. IEEE, 2011, pp. 89–96.
[16] E. Clarke et al., “Counterexample-guided abstraction refinement,” in CAV, 2000, pp. 154–169.
[17] S. Chattopadhyay and A. Banerjee, “Qos constrained large scale web service composition using abstraction refinement,” IEEE TSC, vol. PP, no. 99, pp. 1–1, 2017.
[18] F. Lindström, “First order predicate logic with generalized quantifiers,” Theory, vol. 32, no. 3, pp. 186–195, 1966.
[19] L. Ju et al., “Tenant onboarding in evolving multi-tenant software-as-a-service systems,” in ICWS, 2012, pp. 415–422.
[20] Y-M. Xia et al., “Web service composition integrating qos optimization and redundancy removal,” in ICWS, 2013, pp. 203–210.
[21] A. Bansal et al., “Wsc-08: continuing the web services challenge,” in E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services, 2008 10th IEEE Conference on. IEEE, 2008, pp. 351–354.