Techniques to Produce Optimal Web Service Compositions

Eduardo Blanco     Yudith Cardinale     María-Esther Vidal
Jesús Graterol
Universidad Simón Bolívar, Departamento de Computación y T. I., Apartado 89000, Caracas 1080-A, Venezuela
{eduardo,yudith,mvidal,jesus.graterol}@ldc.usb.ve

Abstract

As Web Services proliferate, it becomes more difficult to find a service that can perform a given task, and a coordination of several services may be required. We present two algorithms to identify orderings of Web Service compositions. These algorithms follow different strategies to prune the space of possibilities while minimizing the evaluation cost. The first one, \textit{DP-BF}, combines a best first strategy with a dynamic-programming technique and produces good Web Service compositions by exploring a small portion of the search space. The second one, \textit{PT-SAM}, adapts a petri-net unfolding algorithm and tries to find a desired marking from an initial state. We conducted an experimental study to evaluate the behavior of \textit{DP-BF} and \textit{PT-SAM} compared to \textit{SAM} and to the exhaustive solution. Our experiments show that the quality of the compositions identified by our algorithms are close to the optimal solution produced by the exhaustive algorithm, while the optimization time is close to the time required by \textit{SAM} to identify a solution.

1. Introduction

Transparent service composition has a great potential to facilitate the integration of applications developed among different organizations. However, sufficiently-rich, machine-readable descriptions will be required to create a solution composed of a diversity of heterogeneous Web Services. Existing solutions are focused on the inherent challenges associated with service discovery and composition tasks [3, 11, 12]. However, during the exploration of the search space, information about the cost of evaluating the solution is not considered in a large-scale Web Service environments (e.g., in the rank of 1,000 to 100,000 available services); in consequence, the final solution may be costly.

In this paper we consider the problem of identifying optimal evaluation plans for large-scale Web Service compositions, and we propose cost-based strategies to select good plans for a given Web Service request. We refer to this problem as the \textit{WSC} problem.

We illustrate the problem of efficiently identifying an optimal Web Service composition in the context of the \textsc{Semantic Web} [9] as follows:

Consider a scientist who needs to process and combine some data-sets in order to satisfy a particular requirement. The scientist knows that there are enough available services that can be used to solve her problem but these services need to be properly coordinated. Additionally, a service can solve more than one task simultaneously, while others are only able to perform just one of them. Finally, these services may be associated with particular costs, that describe time required to transfer data across the network and the delays to be paid according to their real-time performance.

To avoid a costly evaluation, this scientist must solve two complex tasks: first, she has to identify a set of services that fulfill her requirements; second, she needs to combine the selected services in a way that the evaluation cost is minimized.

Note that usually users have to manually perform these two tasks. This may be not too difficult if the Web service request is simple or it can be directly decomposed into a moderate number of services. However, if the number of services is large or they are replicated and located on different sites with different performance behavior, the task ahead of this scientist may be extremly complex. Thus, it is clear that in such scenarios, the \textit{WSC} problem cannot be solved manually, and scientists need tools to automatically generate good Web Service compositions.

In [3], the \textit{SAM} algorithm is presented; it receives as input a query and a set of ontological service descriptions and returns a coordination of services that produces the desired result. \textit{SAM} traverses the space of possibly composite Web Services by using a greedy approach, and it focuses on finding a solution regardless of its quality. In consequence, as we will show in section 5, the cost of the compositions iden-
tified by SAM tend to be very far from the optimal.

To overcome this limitation, we propose two cost-based algorithms: DP-BF and PT-SAM. These algorithms assign a cost to each service, and estimate the overall evaluation cost of a particular combination of services. DP-BF extends SAM by combining a best first strategy with a dynamic-programming technique to produce good Web Service compositions; only intermediate solutions that may lead to an optimal composition are considered during the search. It resembles the Best First query optimization algorithm presented in [6]. On the other hand, PT-SAM adapts a petri-net unfolding algorithm [2] and tries to find a desired marking from an initial state. Both algorithms identify a composition of capability descriptors so that the input restrictions of each used descriptor are satisfied.

To evaluate the behavior of DP-BF and PT-SAM, we implemented these two search strategies, and conduct an experimental study. We compare DP-BF and PT-SAM to SAM and to the exhaustive solution, and report on the quality of the plans identified by each approach. Our experiments show that the quality of the compositions identified by the proposed algorithms are close to the optimal solution produced by the exhaustive algorithm. Additionally, DP-BF and PT-SAM outperform the solutions generated by SAM, while the optimization time remains in the same order of magnitude.

2. Related Work

The problem of identifying a good combination of services is related to problems that have been studied in query optimization, service matching and discovery, and Web Service composition. In this section we consider the main related approaches in each of these areas.

2.1. Query Optimization

In the context of the Web, several strategies have been presented to identify good evaluation plans where sources have limited query processing capabilities; then, the optimizer task is to identify a good ordering of the sub-goals of a query where limited processing capabilities of each considered source, are satisfied [10, 18].

Typically, existing approaches achieve the challenge of identifying good plans by representing query capabilities as binding patterns and using these patterns and meta-heuristics to traverse the space of source plans. Each sub-goal in the query is rewritten in the plan by using sources that define the sub-goal; limited processing capabilities of the sources are satisfied in the plan with query bindings or attributes projected out by previous sources in the plan.

DP-BF resembles the dynamic-programming based optimization algorithm presented in [6, 7]. DP-BF also traverses the space of possibly good service combinations in iterations and prunes the search space based on a cost metric that estimates the evaluation cost of a particular solution. Thus, only solutions with the lowest cost seen so far, survive at the end of each iteration. Additionally, DP-BF uses information about the capability of the available services to satisfy the restrictions represented in the user query.

In DP-BF a capability descriptor indicates the input arguments that the service needs to bound to produce a certain output. Similarly, in [6], a capability descriptor represents the set of attributes that a source is able to project out when specific input attributes are bound. The main difference between DP-BF and the best first solution in [6], is that the latter identifies good compositions of the capability descriptors that implement only the sub-goals in a given query; thus, the length of the solutions are limited by the length of the query. In contrast, DP-BF only receives as a query a set of bound arguments and the ones that need to be produced, and identifies compositions of any length that fulfill the query. In consequence, DP-BF uses only information about the descriptors of each service and the restrictions satisfied so far, to guide the search in the space of composite Web Services.

2.2. Service Matching and Discovery

The first matchmaking algorithms were based on the information published in the Service Profile to identify matches between services [17]. A Service Profile is part of an OWL-S ontology, where a service can be associated with zero or more Service Profiles. A Service Profile specifies the service functionality in terms of inputs, outputs, preconditions, and effects. Inputs and outputs refer to OWL classes describing the types of instances to be sent to the service and the type of the expected answer. The main disadvantage of these algorithms arises from the limited capability of the Service Profiles to describe the properties of a service. Information of the control structures used in a service is not available; thus, incorrect matches can be identified [1].

In [1], extensions to previous matchmaking algorithms are presented. In addition to consider Service Profile, this solution takes into account information specified on the Process Model, i.e., the restrictions and control structures that are used to define the service and to decide which attributes can be produced as output during the evaluation of the service. Similarly, our proposed algorithms PT-SAM and DP-BF relay the matchmaking task on the Process Model to overcome the limitations of the Service Profile.

2.3. Web Service Composition - The WSC Problem

The WSC problem has been extensively treated in the literature, and diverse solutions that take advantage of AI
techniques and Search Meta-Heuristics, have been proposed. First, in the context of AI, the WSC problem has been represented as a planning problem where actions to be taken by the planner are defined in terms of service preconditions and effects.

The description of a planning domain includes a set of planning operators and methods that establish the way a task can be decomposed into smaller subtasks. The description of a planning problem contains an initial state as in classical planning. Instead of a goal formula, there is a partially-ordered set of tasks to accomplish. Planning proceeds by decomposing tasks recursively into smaller and smaller subtasks, until primitive tasks, which can be performed directly by using one planning operator, are reached. For non-primitive tasks, the planner chooses an applicable method, instantiates it to decompose the task into subtasks, and then chooses and instantiates other methods to decompose the subtasks even further. If the constraints on the subtasks or the interactions among them prevent the plan from being feasible, the planning system backtracks and tries other methods.

As any planning problem, the approach presented in [8], requires of the formalization of the domain-dependent control knowledge in the planner. Thus, a domain expert is needed in order to achieve good performance in real-world domains.

An approach that uses Answer Set Programming is presented in [19]. It shows that service descriptions can be expressed in a rule based language that allows to search a repository efficiently and to build solutions that solve a goal with respect to soft and hard constrain. The author reports that the solution performs very well in this rather simple domain. It defines a simple cost function instead of a utility function. Its strength is that it provides means to gain all solutions for a given problem despite the cost of computation time and space. He states that dedicated software employing fast heuristic algorithms could rapidly find a good solution for user requests in much reasonable times. We plan to compare PT-SAM and DP-BF with this solution to evaluate the performance in the search.

In the context of Search Meta-Heuristics, the SAM (SERVICE AGGREGATION MATCHMAKING) algorithm is defined [3]. It makes use of an OWL-S ontology, and explicitly returns a sequence of atomic processes that need to be executed in order to achieve the desired result. SAM follows a greedy approach in which only one sub-plan is generated in each iteration. In each sub-plan, a sub-set of the output attributes is produced considering some of the bindings given in the query. The algorithm ends when all the output attributes are produced. In terms of time, SAM is able to scale up in environments with a moderate number of services (e.g., in the rank of 100 to 200 services). However, since SAM does not consider any cost metric or optimization criteria to compose the services, plans produced by SAM may be costly. To exacerbate this problem, SAM may add processes to the plan that are not needed to produce the output required by the user. Thus, the quality of generated plans may be far from optimal. PT-SAM and DP-BF incorporate optimization techniques to SAM, which improve the performance and the quality of the identified solutions.

3. Problem Definition

In this section we define the framework of our approach.

Definition 1 (Query) A query \( Q \) is a pair \((I, O)\), where \( I \) is a set of provided data types and \( O \) is a set of data types that needs to be produced.

Definition 2 (Service Graph) A service graph\(^1\) \( G = (V, E) \) is a DAG where nodes in \( V \) correspond to either data types or service processes. Edges in \( E \) represent relationships between types and service processes, such that, if \( T \) is a data type and \( P \) a service process then:

- \((T, P) \in E \text{ iff } T \text{ corresponds to an input parameter of the } P \text{ or}
- \((P, T) \in E \text{ iff } T \text{ is produced by } P.
- \text{ Each } P \text{ is associated with a real number, that corresponds to an estimate of the evaluation cost of the process.}

3.1. Problem

To date, multiple approaches have been proposed to identify Web Service compositions [4, 13, 14, 15, 16] however, to the best of our knowledge, none of them have considered cost-based techniques or are able to produce solutions that minimize the evaluation cost in scenarios of large-scale Web Services (e.g., in the rank of 1,000 to 100,000 services); thus, the final solution may be costly. In this paper we propose cost-based strategies to approach the WSC problem; we consider large-scale Web Service scenarios in which a good plan has to be generated for a given request.

We hypothesize that if there is a large number of Web Services published in different sites then, services’ performance may vary. Thus, it is imperative that the composition process considers an estimate of the service evaluation cost. This cost is used to prune the space of possibly costly plans of services, and to identify the service composition with the lowest estimated cost.

We define the WSC problem as follows:

\(^1\)We use a similar definition as the one presented in [3].
**Definition 3** (Web Service composition-WSC) Given a query \( q = (I, O) \), a service graph \( G = (V, E) \), the WSC problem is to generate a minimal service graph \( G' = (V', E') \) so that, \( V' \subset V \) and \( E' \subset E \), and the cost associated with the execution of \( G' \) is minimized.

## 4. Solutions to the WSC problem

We present two algorithms, DP-BF and PT-SAM to the WSC problem (see Definition 3). They use different criteria to prune the space of possibilities while minimizing evaluating costs. The following sections describe both algorithms.

### 4.1. DP-BF: Dynamic-Programming Best First

DP-BF follows a greedy best first strategy using a dynamic-programming technique and expands the top sub-compositions of services identified so far, until a Web Service composition that fulfill the query constraints is identified. It produces good Web Service compositions by traversing a small portion of the search space.

DP-BF resembles the dynamic-programming based optimization algorithm presented by Florescu et al in [6]. They proposed a dynamic-programming approach to solve the problem of query optimization in the presence of limited access patterns. Similarly, DP-BF traverses the space of possibly good Web Service combinations in iterations and prunes the search space based on a cost metric that estimates the evaluation cost of a particular solution. Thus, only solutions with the lowest cost seen so far, survive at the end of one iteration.

In each step of the algorithm, the sub-combination associated with the highest quality is combined with those sub-plans that can improve or maintain that quality level in a more complete sub-plan.

The quality of a plan can be defined in different ways depending on which parameter should be minimized. In this work, we have chosen to optimize the estimated evaluation time. In this approach, the plan quality is defined by the following utility metric:

\[
\text{Quality}(p) = \left[ \frac{\text{maxCost}}{\text{Cost}(p)} \right] \times \text{outputsReachable}
\]

where:

1. \( \text{maxCost} \) is the maximum allowed cost
2. \( \text{Cost}(p) \) is an estimated cost of plan \( p \).
3. \( \text{outputsReachable} \) is the number of outputs in the query which are already reachable from the plan \( p \).

Once the most useful plan is found, DP-BF chooses, from the sub-compositions generated so far, a list containing the Web Service compositions that are combinable. Thus, a Web Service compositions represented by a service graph is added to this list if it satisfies at least one of the following conditions:

1. At least one input free of the firable plan matches with an output bound of \( G \).
2. At least one input free of \( G \) matches with an output bound of the firable plan.
3. The amount of outputs reachable from the firable plan is smaller than the outputs reachable from the combination.
4. The estimated cost to the outputs of the query is bigger in both \( G \) and the firable plan in comparison with the estimated cost to the outputs in the combination.

This process is repeated until a combination that satisfies the query is found.

### 4.2. PT-SAM: Petri-Nets SAM

The petri-nets directed unfolding algorithm, presented in [2], is a modification to the algorithm ERV [5]. It is used to solve the problem of whether a marking is reachable from initial marking in the petri-net. It orders a set of transitions that are to be fired according to an heuristic that estimates how close it is to the desired marking.

In the same way, PT-SAM adapts that petri-net unfolding algorithm and tries to find a desired marking from an initial state. The initial state is the marking where the data types bound in the query are fired. The final marking correspond to the state where all free data types in the query were all fired. In order to find a final marking, PT-SAM keeps a list of all processes that can be added to the plan. This list is ordered according to a utility metric much alike as the one defined by DP-BF.

## 5. Experiments

In this section we present the results of our experimental study; we report on the time required to evaluate our solutions and the quality of the identified plans.

We compare DP-BF and PT-SAM to SAM and to a more exhaustive dynamic-programming solution, called DP-First. The latter is called DP-First because it expands all sub-plans in each iteration but it stops when the first execution plan \( p \) is found for a query \( Q \). We also implemented an algorithm (DP-All) that finds, using a dynamic-programming approach, all possible solutions to each query.
5.1. Experiment Design

5.1.1. Services

We created a base ontology to describe the concepts of our domain. We also create a set of OWL-S Web Service definitions. The atomic processes are described by the data types that correspond to their inputs and outputs arguments. These services were replicated with different costs as if they were placed in other sites.

5.1.2. Queries

A set of queries classified according to their size. The size measures the number of processes in the optimal plan in the whole space of solutions for each query. The sizes of queries range from one to twelve processes. There are twenty queries for each size, and a total of 240 queries. These queries were randomly generated following a uniform distribution.

5.1.3. Hardware

The five programs were run in a SUN workstation with 2 GBytes of memory, two Dual Core AMD Opteron processors 180 with 2.4 GHz and running Ubuntu 5.10 operating system. JDK 1.5.0_06 virtual machine was used to develop and run the programs. The OWL-S API was used to parse the Web Services definitions and to deal with the OWL classification reasoning process.

5.2. Results

We present the results of our experiments considering three measures: optimization time, the estimated cost of the plan and total response time.

5.2.1. Optimization Time

In Figure 1, we note that SAM better scales up. It produces solutions to user queries in less than 2 seconds. Times needed by DP-BF go up to 10 seconds but PT-SAM can produce answers in less time than SAM. This could be explained by the fact that SAM adds services to the plan even if the services are not required to answer the query.

5.2.2. Plan Quality

In Figure 2, we present the estimated evaluation cost of the plans identified by each approach. We do not report SAM because the SAM’s costs are too high that it makes difficult to observe the differences between the estimated cost of plans identified by the other algorithms. We see that these algorithms show solutions that are close to those produced by the DP-First algorithm.

5.3. Percentile analysis

In Table 1, we present the percentile analysis of the costs of plans generated by the four approaches compared to the costs of the plans produced by DP-All, i.e., we rank the solutions produced by the proposed strategies with respect to the optimal solution.

These results clearly show that, DP-BF and PT-SAM are able to produce plans that outperform the ones produced by SAM, while the optimization time remains in the same order of magnitude. The percentile analysis shows that our algorithms produce plans that are close to the optimal plan produced by DP-All.
Table 1. Percentile Range of estimated costs

<table>
<thead>
<tr>
<th>Query Size</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAM</td>
<td>46.3</td>
<td>71.8</td>
<td>49.4</td>
<td>51.6</td>
<td>45.6</td>
</tr>
<tr>
<td>DP-BF</td>
<td>88.3</td>
<td>90.0</td>
<td>93.2</td>
<td>93.9</td>
<td>96.1</td>
</tr>
<tr>
<td>PT-SAM</td>
<td>93.2</td>
<td>94.1</td>
<td>96.6</td>
<td>93.9</td>
<td>95.6</td>
</tr>
<tr>
<td>DP-First</td>
<td>95.7</td>
<td>99.0</td>
<td>95.4</td>
<td>94.60</td>
<td>95.8</td>
</tr>
</tbody>
</table>

6. Conclusions and Future Work

We presented two algorithms DP-SAM and DP-BF, for the WSC problem in large-scale Web Services scenarios. DP-BF extends SAM by combining a best first strategy with a dynamic-programing technique to produce good Web Service compositions; only intermediate solutions that may lead to an optimal composition are considered during the search. On the other hand, PT-SAM adapts a petri-net unfolding algorithm and tries to find a desired marking from an initial state. Both algorithms identify a composition of capability descriptors so that the input restrictions of each used descriptor are satisfied.

We are planning to enrich DP-BF, PT-SAM with costs models adapted to the dynamic characteristics of Grids platforms. In particular, we address the problem of generating good service coordinations that satisfy some of the constraints present in the GRID.

References


