QoS-aware Automatic Syntactic Service Composition problem: complexity and resolution

Virginie Gabrel, Maude Manouvrier, Kamil Moreau, Cecile Murat

QoS-aware Automatic Syntactic Service Composition problem: complexity and resolution

Virginie Gabrel, Maude Manouvrier, Kamil Moreau, Cécile Murat
E-mail: {gabrel,murat,manouvrier}@lamsade.dauphine.fr

Abstract—Automatic syntactic service composition problem consists in automatically selecting services, from a registry, by matching their input and output data. The composite service, resulting from this selection, allows producing a set of output data, needed by a user, from a set of input data, given by the user. Many approaches resolving the aforementioned problem do experimentations on the well-known Web Service Challenge (WSC) standard benchmark, which provides synthetic services. Since 2009, this challenge has extended the syntactic service composite problem to a Quality-Of-Service (QoS) based one, by describing the synthetic services with execution time and throughput values. In this article, we propose an original formulation of the QoS-aware automatic syntactic service composition problem in terms of scheduling problem with AND/OR constraints, using a directed graph structure. For the WSC-09 benchmark (considering execution time and throughput QoS criteria), our exact algorithm outperforms the related work. We also analyse the complexity of optimizing other QoS criteria (cost and reliability) and exhibit polynomial cases.

Index Terms—Service composition, Web Service Challenge, AND/OR constraints, QoS optimization.

1 Introduction

The QoS-aware automatic syntactic service composition problem consists in automatically composing services by matching their parameters (input and output data) such that the resulting composite service can produce a set of output data from a set of input ones while optimizing a Quality of Service (QoS) criterion.

Available services are grouped into a service registry. For example the *Jena Geography Dataset*[1] groups almost 200 geography services that have been gathered from different web sites. Moreover, it exists standard synthetic data benchmarks, as the *Web Service Challenge* (WSC) one. As shown in [24], WSC proposes an incremental synthetic data benchmark. In 2005, it only focuses on syntactic web service composition, then it integrates OWL for Taxonomy in 2007, and structured data types in 2008 [3], and finally it introduces QoS criteria (execution time and throughput) in 2009 [14]. These benchmarks are still used by recent approaches. For example, authors in [23] experiment their approach on WSC-08, authors in [17] on WSC-08 and WSC-09 and authors in [22] on WSC09. In the last benchmarks, a WSDL standard file [5] describes the input and output data of all the services available in the registry. Input and output data are hierarchically organized in terms of concepts using ontology recorded in a OWL [4] standard file. A WSLA [15] file describes the QoS criteria of all available services.

In this article, we propose an original formulation of the QoS-aware automatic syntactic service composition problem in terms of scheduling problem with AND/OR constraints. This formulation allows to apply very efficient exact algorithm for optimizing the execution time and throughput criteria (as shown by our experimental results on WSC-09). We analyse the complexity of optimizing other types of QoS criterion and exhibit polynomial cases.

This article is organized as follows. Section 2 presents the syntactic service composition problem. Related work is reviewed in Section 3. Considering execution time and throughput criteria, the service composition problem is modelled as a scheduling one with AND/OR constraints in Section 4. Section 5 analyses the theoretical complexity of service composition problem for cost and reliability criteria. Finally Section 6 concludes.

2 Syntactic Service Composition problem

2.1 Problem description: the feasibility part

As the syntactic description of functions in programming language, any (SOAP, RESTful or Web API) service s can be syntactically described by the set of its input data (i.e. the data needed by the service to be invoked), $in(s)$, and the set of its output data (i.e. the data produced by the service invocation), $out(s)$. We denote S the set of services and D the set of data.

Example 1. Let us consider an example with 9 services and 9 data:

s	1	2	3	4	5	6	7	8	9
$in(s)$	J	K,P	L,M	M	M	P	J	N,P	L
$out(s)$	M	J,M	N	P	P,Q	R,Q	L	T	N,P

Table 1
An example of a service registry

A service registry can easily be represented by a directed graph $G = (X, U)$, called in the following *ServiceData graph*:

- vertices represent services and data: $X = D \cup S$,
- directed edges in U represent two kinds of relation: (1) an edge from $i \in S$ to $j \in D$ represents the fact

1. http://fusion.cs.uni-jena.de/professur/jgd/

that service i produces data j ($j \in out(i)$), (2) an edge from $i \in D$ to $j \in S$ represents the fact that service j needs data i to be executed ($i \in in(j)$).

Let us remark that G is a bipartite graph (there does not exist any directed edge linking two vertices belonging to S or two vertices belonging to D).

Example 2. The ServiceData graph associated with the service registry given in Table 1 is represented in Fig. 1. Vertices representing data are represented with circles while vertices representing services are drawn with squares. For example, vertex P is a data produced by services represented by vertices 4, 5 or 9 and, is an input of services 2, 6 and 8. Service represented by vertex 3 needs data L and M to be executed and produces data N.

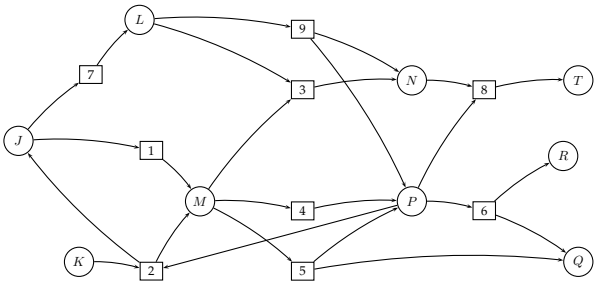

Figure 1. The ServiceData graph associated with registry of Table 1

Such graph can be, for example, easily built from the files provided by WSC-09 synthetic benchmark. This benchmark provides 5 test sets, each one containing one service registry described by 3 main files: `services.wsdl` describing services of the registry by their input/output, `taxonomy.owl` containing the ontology and `Servicelevelagreements.wsla` storing the QoS (response time and throughput) criteria values of each service. The first file, `services.wsdl`, contains all the services' identifier and their input and output, each one being defined as an instance of the ontology (i.e. as a leaf in a concept hierarchy) stored in the second file `taxonomy.owl`. A service is then represented in the ServiceData graph by a vertex s. Each input i of a service s is represented by vertex i. Each output o of a service s is represented by several vertices, as many as the concept of o has ancestors in the ontology.

A user query, denoted (I, O), is defined by a set $I \subseteq D$ of data provided by the user and a set $O \subseteq D$ of data required by the user. A service can be invoked only if all its input data are available. A data is available if it is computed by at least one service or if it is provided by user. In order to respond to a user query, the syntactic service composition problem is to select the set of services (representing the components of the resulting composite service) that provides, from the user input set I, the set O of all outputs needed by the user.

We can formulate this problem on the ServiceData graph transformed as follows:

- we add a fictitious vertex d_0, representing an "empty" data, d_0 is included in D and I; for each service s which does not need any input to be executed, we add an arc $(d_0, s) \in U$ (d_0 is included in $in(s)$),
- we add a fictitious vertex denoted s_0, which represents the beginning of the process; s_0 is included in S and for all $i \in I$, we add an arc $(s_0, i) \in U$ (thus $out(s_0) = I$),
- we add a fictitious vertex denoted End, which represents the end of the process; End is included in S and for all $o \in O$, we add an arc $(o, End) \in U$ (thus $in(End) = O$).

A solution denoted C of the aforementioned composition problem can be expressed in terms of sub-graph. More precisely, given a query characterized by I and O, a sub-graph $G^C = (X^C, U^C)$ of the ServiceData graph G corresponds to a composite service C if and only if:

- s_0 and End belong to X^C,
- if a vertex $i \in D$ belongs to X^C, at least one arc $(j, i) \in U$, with $j \in S$, belongs to U^C (it is due to the fact that each data i has to be produced by at least one service),
- if a vertex $i \in S$ belongs to X^C, all arcs $(j, i) \in U$, with $j \in D$, belong to U^C (it is due to the fact that each service i can be executed if and only if all its inputs data are available),
- G^C does not contain any directed cycle.

Example 3. Given the graph of Fig. 1 and the query described by $I = \{J, K\}$ and $O = \{Q, T\}$, we can propose the composite service $\{6, 7, 8, 9\}$. This solution is associated with the sub-graph represented in bold arcs in Fig. 2.

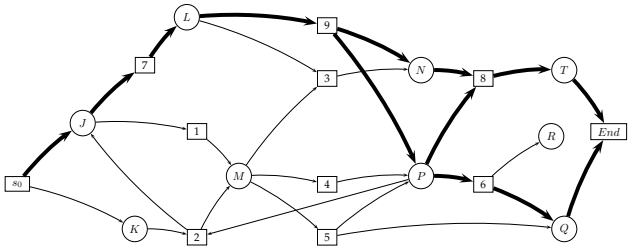

Figure 2. A composite service and the associated sub-graph

Services can also be described by their Quality of Service (QoS) criteria. Among QoS criteria, we can find the execution time (i.e. the time needed by a service to produce output data from the input ones) and the throughput (i.e. the average rate of successful service execution). A service with the highest throughput and the lowest execution time represents a good performing service. In the following section, we present the most used criteria and the way to compute them.

2.2 Problems description: the optimality part

The QoS of a composite service C depends on the QoS of each service belonging to the composite service. In the following, we choose to present 4 famous QoS criteria which are representative of 4 different ways to aggregate individual QoS services:

- **Execution time**. Let us denote $e(s) \geq 0$ the execution time of service s, $\mathcal{P}^{\mathcal{C}}$ the set of paths in G^C from s_0 to End and $v_e(C)$ the value of the composite service C on the execution time criterion. We have:
$$v_e(C) = \max_{\mu \in \mathcal{P}^{\mathcal{C}}} \sum_{s \in \mu} e(s)$$
This criterion has to be minimized and the associated problem is denoted ESC for Execution time Service Composition.
- **Throughput**. Let us denote $t(s) \geq 0$ the throughput of service s and $v_t(C)$ the value of the composite service C on the throughput criterion. We have:
$$v_t(C) = \min_{s \in C} t(s) = \min_{\mu \in \mathcal{P}^{\mathcal{C}}} \min_{s \in \mu} t(s)$$
This criterion has to be maximized and the associated problem is denoted TSC for Throughput Service Composition.
- **Cost**. Let us denote $c(s) \geq 0$ the cost of service s and $v_c(C)$ the value of the composite service C on the cost criterion. We have:
$$v_c(C) = \sum_{s \in C} c(s)$$
This criterion has to be minimized and the associated problem is denoted CSC for Cost Service Composition.
- **Reliability**. Let us denote $r(s)$, with $0 \leq r(s) \leq 1$, the reliability of service s and $v_r(C)$ the value of the composite service C on the reliability criterion. We have:
$$v_r(C) = \prod_{s \in C} r(s)$$
This criterion has to be maximized and the associated problem is denoted RSC for Reliability Service Composition.

The QoS values are modelized in the ServiceData graph as weight on vertices: a vertex representing a service has a weight equals to its QoS value, vertices associated to data and fictitious vertices (s_0 and End) have a neutral value (0 for the execution time and the cost values, 1 for the reliability value and $+\infty$ for the throughput value).

For the first two criteria, the value of a composite service C is equal to the value of a particular optimal path in G^C: the maximal path value for the execution time criterion and the minimal path value for the throughput criterion. The first criterion has to be minimized while the second one has to be maximized. That's why these two criteria, maxmin-type (or equivalently minmax-type) criteria, can be similarly processed by computing optimal path. Polynomial-time algorithm has been proposed in [12], [13], [16] for solving the service composition problem with a maxmin-type criterion. This composition problem is the subject of the WSC-09 and a lot of papers have proposed different approaches to solve it.

At the opposite, the last two criteria (sum-type and product-type respectively) cannot be expressed in terms of path value in G^C. They induce much more difficult optimization problems. Indeed in [10], [19], authors show that the service composition problem is NP-hard when a sum-type criterion (e.g number of services or cost) is minimized. In [10], the proof is based on a reduction to the set cover problem. A composition problem with a sum-type criteria is the subject of WSC-08 which appears to be much more difficult than WSC-09.

In this article, we analyse the four optimality problems and the main results are:

- We show that the minimal ESC problem is a well-studied project scheduling problem with AND/OR precedence constraints.
- The minimal ESC and maximal TSC problems can be solved with a polynomial-time algorithm (already proposed in the project scheduling context) directly applied on the associated ServiceData graph. This algorithm is more efficient on WSC-09 instances than already published algorithms [12], [13], [16].
- We analyse the particular difficulty of the NP-hard minimal CSC problem and we exhibit a polynomial case (associated with a particular class of ServiceData graph).
- We show that the maximal RSC problem is NP-hard and we exhibit also a polynomial case.

The next section presents the related approaches associated with the aforementioned problems.

3 Related work

QoS-aware automatic syntactic service composition problem has attracted a lot of attention from different fields in recent years. We choose to classify these approaches into four groups: search approaches, dependency graph approaches, planning graph approaches and integer linear programming ones.

3.1 Search approaches

In [11] and [13], the WSC-09 first run-up winners propose a polynomial-time algorithm to solve the minimal ESC and maximal TSC problems. Its worst case complexity, given in [13], is $O(a|S|^2)$, with a the average number of services' input data. This algorithm proceeds in two steps: in the first one, a forward search aims to eliminate useless services for satisfying the user's query and, at the same time, computing optimal QoS values of useful services; in the second stage, a backtrack search is executed to determine the optimal composite service.

In [16], authors propose a more efficient polynomial-time algorithm for solving the minimal ESC problem: the first part is a search procedure determining, at each iteration, the minimal accumulated execution time of one service (this search procedure stops when all user outputs are reached)

and, the second part is a backtrack search in order to determine the optimal composite service. No theoretical complexity results are given. This algorithm is the closest one from ours. However, authors claim that the backtrack search can be rather long. From our point of view, this is due to the chosen data structures and we propose a more efficient graph-based data structure.

3.2 Dependency graph approaches

In [10], [12], authors represent the QoS-aware automatic syntactic service composition problem with another directed graph, called *dependency graph*. In the dependency graph, denoted $H = (S, A)$, vertices only represent services. There is an arc in A from i to j if the intersection between the output of i and the input of j is not empty (i.e. service j needs a subset of the outputs produced by i to be executed, $in(j) \cap out(i) \neq \emptyset$). The arc is tagged with the subset of data belonging to the intersection.

Example 4. The dependency graph corresponding to the service registry given in Table 1 is represented in Fig. 3. Vertex s is drawn with a square divided into three columns: $in(s)$ is given in the first column, s in the middle and $out(s)$ in the latest one. For example, service 9 produces data N and P and service 6 needs data P, so there is an arc from 9 to 6 with tag P.

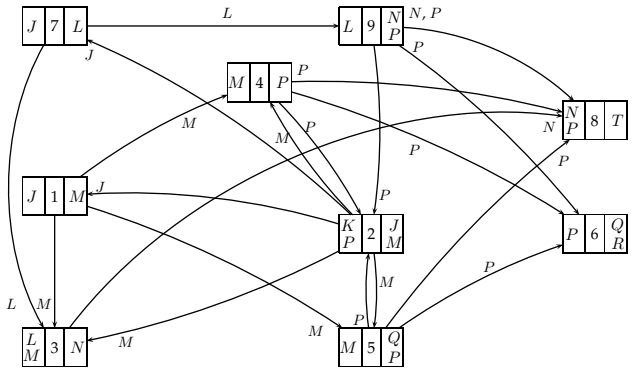

Figure 3. The dependency graph associated with registry of Table 1

Remark 1. For building the dependency graph, one needs to read the service registry and, for each service, one has to match its inputs with the outputs of all the other services, inducing an $O(|S|)$ complexity for each service (cf. [20]). Consequently, the building step of the dependency graph is $O(|S|^2)$.

In [12], authors propose a new approach: to apply a Dijkstra-like algorithm on the dependency graph to solve the minimal ESC and maximal TSC problems. In their experimental results, we remark that the computational time is mainly due to the building step of the dependency graph (for example, for instances with 10000 services, the building time is 6000ms while the resolution time is 60ms). Moreover, this article includes inexact results: authors claim that their algorithm can be also applied to determine the optimal composite service on product-type criterion (reliability). We show in section 5.2 that it is not true since RSC problem is NP-hard. In [22], authors mention the inexact result of [12] to establish that the optimal QoS can be calculated in polynomial time, which is right only for throughput and execution time criteria.

3.3 Planning graph approaches

IA planning and graph-based approaches are also used for solving service composition problem. In [26], the planning graph model is applied for solving the following feasibility problem: does there exist a feasible composite service for satisfying the user's request, without considering any QoS criterion? Considering QoS-aware composition problem, Chen and Yan in [6] propose a three steps algorithm to determine the best composite service considering maxmin (for example execution time) or sum-type (for example cost) criterion. First, they represent the service composition problem by a labelled planning graph. Second, they transform the labelled planning graph into a layered weighted graph. Third, an optimal service composite is found by applying Dijkstra shortest path algorithm. Steps 1 and 3 can be done by polynomial-time algorithms; the complexity of step 2 is not presented in [6]. Since the QoS-aware service composition with sum-type criterion is NP-hard, the step 2 is the critical non-polynomial part of the algorithm. Thus, this approach is not interesting for solving the service composition with maxmin-type criterion since a polynomial-time algorithm has been already proposed in [11]. The same criticism can be done for the algorithmic study presented in [7].

3.4 Integer linear programming approaches

Integer Linear Programming (ILP) model have also been proposed for QoS-aware service composition in [9], [21], [25]. In [21], [25], a composite service is decomposed into stages: a stage contains one service or several services executed in parallel. The associated ILP represents the problem of selecting one or several services per stages. Thus, the number of variables and constraints can be huge since there are proportional to the number of services and data times the number of stages. Moreover, the number of stages is not known; only upper bounds can be chosen (the worst one is to set the number of stages equals to the number of services). The size of the model does not allow to solve big size instances. In [9], authors propose a new ILP model containing a lower number of variables and constraints (proportional to the number of services and data). This model is able to optimize a QoS criterion while satisfying transactional requirements. In a large majority of the experimental instances of WSC-09, model of [9] finds a better solution more rapidly than models proposed in [21], [25].

In conclusion, for minimal ESC and maximal TSC problems, search approaches dominate. Planning graph and integer linear programming approaches become relevant for other QoS-aware problems. In this article, we show that the minimal ESC problem is exactly a well studied project scheduling problem with AND/OR constraints. In [18] authors propose a Dijkstra-like algorithm to solve the project scheduling problem with AND/OR constraints. This

algorithm is similar to the one proposed in [12] but can be directly applied on a ServiceData graph (without computing a Dependency graph) as shown in the next section.

4 Service Composition as a scheduling problem with AND/OR constraints

4.1 Scheduling with AND/OR precedence constraints

In a classical precedence-constrained scheduling problem, we have to determine the starting times of a set of jobs to be performed (with known execution time) while satisfying some precedence constraints of the form "a job i can be performed if and only if a job j is finished". Thus, a job is ready to be executed when all its precedence jobs are completed. The objective is to minimized the total execution time of all the jobs. This well-known problem can be solved in polynomial-time with critical-path algorithm (see e.g. [2]). In a scheduling problem with AND/OR precedence constraints, two kinds of jobs are considered: jobs with AND precedence constraints are "classical" ones which can be executed when all their precedence jobs are finished, and jobs with OR precedence constraints can be performed when at least one of their precedence jobs is terminated. This scheduling problem can also be solved in polynomial time : several algorithms have been proposed in [1], [8], [18].

The minimal ESC problem is exactly a scheduling problem with AND/OR precedence constraints in a ServiceData graph:

- both data and services are jobs,
- services are AND-constrained jobs since each service can be executed if and only if ALL its input data are available,
- data are OR-constrained jobs since each data is available when AT LEAST one service produces it.

We adapt the algorithm given in [18] to the service composition context and present it in the next subsection.

4.2 Polynomial-time algorithm for minimal ESC problem

In Algorithm 1, we denote $nd(j)$: the number of non-available data expected by service j to be executed (at the beginning $nd(j) = |in(j)|$). Data and services are labelled with variables λ representing their starting time. The labels are initialized (lines 2-8) as follows:

- $\lambda(s_0) = 0$ (line 4)
- $\lambda(i) = 0$ for all data i in I (lines 5-8),
- $\lambda(i) = +\infty$ for any other data and services (line 2).

For each data i, we have to record in $p(i)$ the index of the service that produces i and $p(i)$ is initialized by -1 (line 2).

At the beginning, all the labels are temporary. When the label of service j becomes definitive, it means that $\lambda(j)$ is equal to its earliest starting time, while for the label of data i, it means that the data i is available at the earliest time $\lambda(i)$. The set of vertices with definitive labels is denoted L in Algorithm 1. At the beginning, it only contains s_0.

The temporary finite labels of data are recorded in a heap H: the root of the heap contains the data with the minimal value temporary label. In Algorithm 1, the classical heap functions are called:

- root(H): removes and returns the root of heap H
- insert($\lambda(i), i, H$): inserts data i with key $\lambda(i)$ into heap H,
- decrease(v, i, H): modifies heap H due to the decreasing of the label of data i, now equals to v.

In a current iteration of the algorithm, the three following steps are performed (lines 10-24):

Step 1. Data i with the smallest temporary $\lambda(i)$ is chosen: its label becomes definitive (line 10).

Step 2. For each service j with data i as input, decrease $nd(j)$ of one unit. If the counter $nd(j)$ reaches 0, it means that the service j can be executed at the starting time of its latest available input data: $\lambda(j) = \max_{i \in in(j)} \lambda(i)$. Thus, the label of service j becomes definitive (line 14).

Step 3. For each service j whose label became definitive at the previous step, consider each data i belonging to $out(j)$ and refresh its label as follows (lines 15-22):

$$\lambda(i) = \min\{\lambda(i), \lambda(j) + e(j)\}$$

and update $p(i)$.

When Algorithm 1 terminates, two cases have to be considered:

- The fictitious vertex End has a definitive label. It means that all its input data (corresponding to set O) are available and, $\lambda(End)$ is equal to the minimum execution time. Using $p(i)$ ($\forall i \in D$), we can easily obtain the solution composite service, with a very fast backtrack procedure.
- The fictitious vertex End has not a definitive label and $\lambda(End)$ equals $+\infty$. It means that it does not exist any feasible composite service to satisfy the query.

Algorithm 1 is exact: the proof is given in [18]. Concerning the complexity, we obtain $O(|D| \log_2 |D| + \sum_{s \in S} |in(s)| + \sum_{s \in S} |out(s)| \log_2 |D|)$.

4.3 Polynomial-time algorithm for maximal TSC problem

Algorithm 1 can be modified in order to optimize the throughput criterion. The entire algorithm is presented in the appendix and we present here the main modifications. First, the labels representing throughput are now initialized as follows:

- $\lambda(s_0) = +\infty$,
- $\lambda(i) = +\infty$ for all data i in I,
- $\lambda(i) = 0$ for any other data and services.

Moreover, current iteration has to be updated as follows:

Algorithm 1 Minimal ESC.

1: $H \leftarrow \emptyset$
2: $\lambda(i) \leftarrow +\infty \; \forall i \in S \cup D$ and $p(i) \leftarrow -1 \; \forall i \in D$
3: $nd(j) \leftarrow |in(j)| \; \forall j \in S$
4: $\lambda(s_0) \leftarrow 0$ and $L \leftarrow \{s_0\}$
5: **for all** $i \in out(s_0)$ **do**
6: $\quad \lambda(i) \leftarrow 0$ and $p(i) \leftarrow 0$
7: $\quad$ insert$(\lambda(i), i, H)$
8: **end for**
9: **while** $End \notin L$ and $H \neq \emptyset$ **do**
10: $\quad i \leftarrow$ root(H) and $L \leftarrow L \cup \{i\}$
11: $\quad$ **for all** $j \in S \setminus L$ such that $i \in in(j)$ **do**
12: $\quad\quad nd(j) \leftarrow nd(j) - 1$
13: $\quad\quad$ **if** $nd(j) == 0$ **then**
14: $\quad\quad\quad L \leftarrow L \cup \{j\}$ and $\lambda(j) \leftarrow \max_{k \in in(j)} \lambda(k)$
15: $\quad\quad\quad v \leftarrow \lambda(j) + e(j)$
16: $\quad\quad\quad$ **for all** $k \in out(j) \setminus L$ **do**
17: $\quad\quad\quad\quad$ **if** $\lambda(k) == +\infty$ **then**
18: $\quad\quad\quad\quad\quad \lambda(k) \leftarrow v$, $p(k) \leftarrow j$ and insert(v, k, H)
19: $\quad\quad\quad\quad$ **else if** $\lambda(k) > v$ **then**
20: $\quad\quad\quad\quad\quad \lambda(k) \leftarrow v$, $p(k) \leftarrow j$, decrease(v, k, H)
21: $\quad\quad\quad\quad$ **end if**
22: $\quad\quad\quad$ **end for**
23: $\quad\quad$ **end if**
24: $\quad$ **end for**
25: **end while**
26: Return $\lambda(End)$

Step 1. Data i with the maximal temporary $\lambda(i)$ is chosen: its label becomes definitive.

Step 2. For each service j with data i as input, decrease $nd(j)$ of one unit. If the counter $nd(j)$ reaches 0, it means that the service j can be executed with a throughput value equals to the minimal throughput of its input data: $\lambda(j) = \min_{i \in in(j)} \lambda(i)$. Thus, the label of service j becomes definitive.

Step 3. For each service j whose label became definitive at the previous step, consider each data i belonging to $out(j)$ and refresh its label as follows:

$$\lambda(i) = \max\{\lambda(i), \min\{\lambda(j), t(j)\}\}$$

The temporary finite labels of data are recorded in a heap which root contains the maximal λ. We have to update the heap function by replacing the decrease function by an increase one.

When the service composition problem does not have any solution, Algorithm 2 cannot label End and terminates with $\lambda(End) = 0$.

Property 1. When Algorithm 2 terminates with $End \in L$, we have: for all $i \in S \cup D$, $\lambda(i) = v_t(\mu^*(i))$ with $\mu^*(i)$ the maximal throughput value path from s_0 to the vertex i.

Proof 1. The proof is by induction on the following property: **Rank** k**.** We denote D^k (resp. S^k) the subset of data i (resp. service i) belonging to L at the kth iteration of the while loop (lines 9-25). For all $i \in D^k \cup S^k$, $\lambda(i)$ equals to the optimal path value from s_0 to i on throughput criterion denoted $v_t(\mu^*(i))$.

Rank 0. We have $S^0 = \{s_0\}$ and $D^0 = I$. At the initialization step of the algorithm:
- $\lambda(s_0) = +\infty \Rightarrow \lambda(s_0) = v_t(\mu^*(s_0))$
- $\forall i \in I, \lambda(i) = +\infty \Rightarrow \lambda(i) = v_t(\mu^*(i))$

So, the property is satisfied at rank 0 and we assume that it is verified at rang k. Thus we have to show that the property is satisfied at rank $k+1$.

Rank $k+1$**.** The algorithm chooses the data d belonging to $D \setminus D^k$ such that $\lambda(d) = \max_{i \in D \setminus D^k} \lambda(i)$. Let us suppose that this choice is wrong: the optimal path is $\mu^*(d) = \{s_0, d_1, \ldots s_q, d_{q+1}, \ldots, s_l, d_{l+1} = d\}$ and we have $v_t(\mu^*(d)) > \lambda(d)$. If we scan $\mu^*(d)$ from s_0 to d, one necessarily encounters a service, denoted s_q with $1 \leq q \leq l$, belonging to S^k while $d_{q+1} \notin D^k$. Since the property is verified at rank k and $s_q \in S^k$, we have $\lambda(s_q) = v_t(\mu^*(s_q))$. By construction of the algorithm, we also have: $\lambda(d_{q+1}) \geq \min\{\lambda(s_q), t(s_q)\}$ implying:

$$\lambda(d_{q+1}) \geq v_t(\mu^*(s_q)) \quad (1)$$

Moreover, since $\mu^*(s_q) \subset \mu^*(d)$, we have:

$$v_t(\mu^*(s_q)) \geq v_t(\mu^*(d)) \quad (2)$$

Equations 1 and 2 and the hypothesis $v_t(\mu^*(d)) > \lambda(d)$ imply: $\lambda(d_{q+1}) > \lambda(d)$ which contradicts $\lambda(d) = \max_{i \in D \setminus D^k} \lambda(i)$. Thus, if the algorithm chooses data d with the maximal $\lambda(i)$, we have $\lambda(d) = v_t(\mu^*(d))$. And at the end, $\lambda(End) = v_t(\mu^*(End))$

In the next sub-section, we present the computational time taken by Algorithms 1 and 2 on WSC-09.

4.4 Experimental results

The experiments were carried out on a Dell PC with Intel (R) Core TM i7-2760, with 2,4 Ghz processor and 8 Go RAM, under Windows 7 and Python 2.7. We have tested our algorithm on the 5 Test Sets (TS) of WSC-09[2] [14] containing around 500, 4000, 8000 and 15000 WS (described by their response time and throughput QoS values) with respectively more than 1500, 10000, 15000 and 25000 data. Each test set corresponds to one query.

For each test set, in Table 2, we report:

- in the first line, the number of vertices associated with services, $|S|$,
- in the second line, the number of vertices associated with data, $|D|$,
- in the third line, the computational time (in ms) necessary for Algorithm 1 to compute the optimal solution on the execution time criterion and the computational time necessary to obtain the corresponding optimal solution with the backtrack procedure,
- in the fourth line, the computational time (in ms) necessary for Algorithm 2 to compute the optimal solution on the throughput criterion and the computational time necessary to obtain the corresponding optimal solution with the backtrack procedure.

Let us compare our results with the closest approaches already published. First, our results are better than those of [12], since we don't have to build the dependency graph.

2. Available at `http://www.it-weise.de/documents/files/wsc05-09.zip`.

	TS01	TS02	TS03	TS04	TS05
$\|S\|$	572	4129	8138	8301	15211
$\|D\|$	1330	9599	16263	16364	28337
ESC /back. (ms)	3/5	11/6	11/6	25/5	30/6
TSC /back. (ms)	2/5	9/5	15/5	26/6	24/7

Table 2
Sizes and computing times on WSC-09

Our results are also better than those of [16] (and consequently than those of [11], [13]) because our algorithm is based on a directed graph structure, which specially allows us to define a very efficient backtrack procedure. Unfortunately, as shown below, this polynomial-time algorithm can not be extended to other QoS criteria.

5 QoS-AWARE SYNTACTIC COMPOSITION PROBLEM: NP-HARD CASES

5.1 Minimal CSC problem: the case of sum-type QoS citerion

5.1.1 General case

The minimal CSC problem is proved to be NP-hard in [10]. Considering a service s (resp. a data d), we denote by C_s (resp. C_d) a composite service ending with s (resp. d) and by $v_c(C_s)$ (resp. $v_c(C_d)$) its associated cost. The difficulty comes from the fact that $v_c(C_s) = \sum_{s_k \in C_s} c(s_k)$ is not equal to $\sum_{d \in in(s)} v_c(C_d) + c(s)$. In order to illustrate this problem, let us consider the two following examples.

In the first typical situation, a service produces several data.

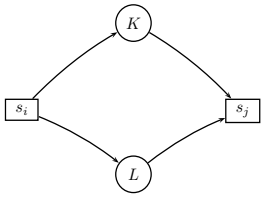

Figure 4. An example with a two-output data service

The service composite of Fig. 4 has a total cost $v_c(C_{s_j}) = c(s_i) + c(s_j)$. On the other hand, summing the cost values of s_j's inputs, we obtain:

$$v_c(C_K) + v_c(C_L) = 2c(s_i)$$

When adding $c(s_j)$, we do not obtain $v_c(C_{s_j})$ since $c(s_i)$ is counted twice (forgetting that K and L are both outputs of the same service s_i).

In the second typical situation, a data is an input of several services.

The composite service of Fig. 5 has a total cost $v_c(C_{s_l}) = c(s_i) + c(s_j) + c(s_k) + c(s_l)$. On the other hand,

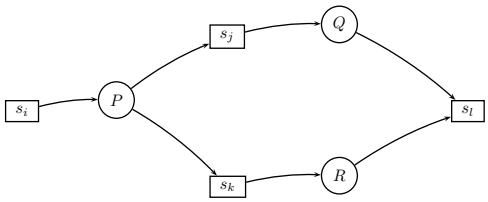

Figure 5. An example with a same input data of two distinct services

summing the cost values of the s_l's inputs, we obtain:

$$\begin{aligned} &= v_c(C_Q) + v_c(C_R) \\ &= v_c(C_{s_j}) + v_c(C_{s_k}) \\ &= 2v_c(C_P) + c(s_j) + c(s_k) \\ &= 2c(s_i) + c(s_j) + c(s_k) \end{aligned}$$

When adding $c(s_l)$, we do not obtain $v_c(C_{s_l})$ since the cost of P is counted twice.

In general case, the cost of a composite service cannot be computed by summing the cost of its input data and Algorithm 1 cannot be updated for solving minimal CSC problem. However, in the following section, we can exhibit a polynomial case.

5.1.2 A polynomial case for solving CSC problem

If we assume the two following hypotheses, then the minimal CSC problem becomes polynomial:
- (a) each service produces only one data,
- (b) each data is an input of only one service.

In this case, each feasible composite service is such that:

$$v_c(C_s) = \sum_{i \in in(s)} v_c(C_i) + c(s) \tag{3}$$

since $X^{C_i} \cap X^{C_j} = \{s_0\}, \forall i, j \in in(s)$. If $X^{C_i} \cap X^{C_j} \neq \{s_0\}$, then it exists a vertex $v \neq s_0$, $v \in C_s$ such it exits a path from v to i and it exists a path from v to j. Then $i \in in(s)$ and $j \in in(s)$ imply that there exist two disjoint paths from v to s which contradicts hypothesis (a) or (b). In particular, equation 3 is useful for the minimum cost value and allows us to adapt the Algorithm 1 for solving the minimal CSC problem.

5.2 Maximal RSC problem: the case of product-type QoS citerion

In this section, we show that the maximal RSC problem is NP-hard.

Theorem 1. The maximal RSC problem is NP-hard.

Proof 2. The proof is based on a reduction from the minimal CSC problem. Given an instance of the minimal CSC problem described by a graph $G = (X, U)$ and a cost $c(s) \geq 0$ associated to each service s, we build an instance of the maximal RSC problem described by the same graph G and a reliability $r(s) = e^{-c(s)}$ associated to each service s, inducing that each $r(s)$ belongs to $[0, 1]$. An optimal solution of such an instance of the maximal

RSC problem is equivalent to an optimal solution of the minimal CSC problem since:

$$\begin{aligned}
\max_{C\in\mathcal{C}} \prod_{s\in C} r(s) &\Leftrightarrow \max_{C\in\mathcal{C}} \ln\left(\prod_{s\in C} r(s)\right)\\
&\Leftrightarrow \max_{C\in\mathcal{C}} \sum_{s\in C} \ln(r(s))\\
&\Leftrightarrow \max_{C\in\mathcal{C}} \sum_{s\in C} \ln(e^{-c(s)})\\
&\Leftrightarrow \max_{C\in\mathcal{C}} \sum_{s\in C} (-c(s))\\
&\Leftrightarrow \min_{C\in\mathcal{C}} \sum_{s\in C} c(s)
\end{aligned}$$

Therefore, both maximal RSC and minimal CSC problems are NP-hard.

In [12], authors claim that their polynomial-time algorithm (similar to Algorithm 1 applied to the dependency graph) can be used for exactly solving maximal RSC problem. Theorem 1 implies that this claim is wrong. Here we give an example to illustrate that the algorithm provided in [12] does not give the optimal solution.

Example 5. In the dependency graph in Fig. 6, we consider the following reliability:

Services	1	2	3
Reliability	0.8	0.8	0.7

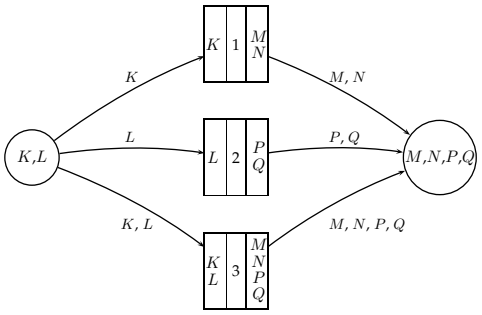

Figure 6. A counterexample for the reliability measure

At each current iteration, algorithm of [12] chooses a maximal temporary label for becoming definitive: the first label concerns service 1 and the second one service 2. Thus, the algorithm determines the composite service with services 1 and 2 to obtain data M, N, P and Q with global reliability of 0.64. The optimal solution is clearly the service 3 with reliability of 0.7 which is better.

Corollary 1. If the ServiceData graph $G = (X, U)$ verifies hypothesis (a) and (b), then the maximal RSC problem becomes polynomial.

6 Conclusion

The QoS-aware syntactic service composition problem has been the subject of numerous studies. The well-known benchmark WSC, allows experimenting approaches for such problem using synthetic services. In this article, we recall that the theoretical complexity of the challenges of 2008 and 2009 are different: the service composition problem presented in WSC-08 (without QoS) is NP-hard, while for the problem presented in WSC-09 (QoS-aware service composition), we propose a polynomial-time algorithm. This algorithm is an extension of a Dijkstra-like project scheduling algorithm, with AND/OR constraints, for exactly solving the service composition problem. Our algorithm is based on a simple directed graph structure and does not need to construct any additional structure, like a dependency graph or a planning graph as usually used in the related work. Thanks to this simple graph structure, our approach is the most efficient one for exactly solving, in polynomial time, the service composition problem with a maxmin-type criterion (like execution time or throughput criterion). Unfortunately, for sum-type criterion (like cost QoS criterion), polynomial time approaches can not be applied, since this problem has been shown as NP-hard. In this article, we highlight the particular difficulty of such optimization problem and exhibit polynomial cases. We also show that the QoS-aware syntactic composition problem is NP-hard, when optimizing a product-type criteria like the reliability criterion. Approximate approaches can be defined for solving NP-hard composition problems. This is left for future research.

References

[1] G. M. Adelson-Velsky and E. Levner, "Project Scheduling in AND/OR Graphs: A Generalization of Dijkstra's Algorithm," Mathematics of Operations Research, vol. 27, no. 3, pp. 504–517, 2002.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Inc., 1993.

[3] A. Bansal, M. Blake, S. Kona, S. Bleul, T. Weise, and M. Jaeger, "WSC-08: Continuing the Web Services Challenge," in IEEE CEC, July 2008, pp. 351–354.

[4] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "OWL Web Ontology Language Reference," W3C, Tech. Rep., February 2004, http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

[5] D. Booth and C. K. Liu, "Web services description language (wsdl) version 2.0 part 0: Primer," W3C, Tech. Rep., 2007, http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626.

[6] M. Chen and Y. Yan, "QoS-aware Service Composition over Graphplan through Graph Reachability," in IEEE International Conference on Services Computing (SCC),, 2014, pp. 544–551.

[7] S. Deng, B. Wu, J. Yin, and Z. Wu, "Efficient planning for top-k web service composition," Knowledge and information systems, vol. 36, no. 3, pp. 579–605, 2013.

[8] E. A. Dinic, "The Fastest Algorithm for the Pert Problem with AND-and OR-Nodes (The New-Product-New-Technology Problem)," in Proceedings of the 1st Integer Programming and Combinatorial Optimization Conference. University of Waterloo Press, 1990, pp. 185–187.

[9] V. Gabrel, M. Manouvrier, and C. Murat, "Optimal and Automatic Transactional Web Service Composition with Dependency Graph and 0-1 Linear Programming," in Service-Oriented Computing. Springer, 2014, pp. 108–122.

[10] S. C. Geyik, B. K. Szymanski, and P. Zerfos, "Robust dynamic service composition in sensor networks," Services Computing, IEEE Transactions on, vol. 6, no. 4, pp. 560–572, 2013.

[11] Z. Huang, W. Jiang, S. Hu, and Z. Liu, "Effective Pruning Algorithm for QoS-Aware Service Composition," in IEEE Conference on Commerce and Enterprise Computing, 2009, pp. 185–187.
[12] W. Jiang, T. Wu, S. Hu, and Z. Liu, "Qos-aware automatic service composition: A graph view," Journal of computer science and technology, vol. 26, no. 5, pp. 837–853, 2011.
[13] W. Jiang, C. Zhang, Z. Huang, M. Chen, and S. Hu, "QSynth: A Tool for QoS-Aware Automatic Service Composition," in IEEE International Conference on Web Services (ICWS), 2010, pp. 42–49.
[14] S. Kona, A. Bansal, M. B. Blake, S. Bleul, and T. Weise, "WSC-2009: A Quality of Service-Oriented Web Services Challenge," in IEEE CEC, 2009, pp. 487–490.
[15] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck, "Web Service Level Agreement (WSLA) Language Specification," W3C, Tech. Rep., 2003, http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.
[16] S. Luo, B. Xu, and Y. Yan, "An accumulated-qos-first search approach for semantic web service composition," in IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 2010, pp. 1–4.
[17] H. Ma, A. Wang, and M. Zhang, "A hybrid approach using genetic programming and greedy search for qos-aware web service composition," in Trans. on Large-Scale Data- and Knowledge-Centered Sys. XVIII, ser. Lecture Notes in Computer Science, A. Hameurlain, J. Kng, R. Wagner, H. Decker, L. Lhotska, and S. Link, Eds. Springer Berlin Heidelberg, 2015, vol. 8980, pp. 180–205.
[18] R. H. Möhring, M. Skutella, and F. Stork, "Scheduling with AND/OR precedence constraints," SIAM Journal on Computing, vol. 33, no. 2, pp. 393–415, 2004.
[19] S.-C. Oh, D. Lee, and S. R. Kumara, "Effective web service composition in diverse and large-scale service networks," Services Computing, IEEE Transactions on, vol. 1, no. 1, pp. 15–32, 2008.
[20] A. M. Omer, "A framework for Automatic Web Service Composition based on service dependency analysis," Ph.D. dissertation, TU Dresden, 2011.
[21] F. Paganelli, T. Ambra, and D. Parlanti, "A qos-aware service composition approach based on semantic annotations and integer programming," International Journal of Web Information Systems, vol. 8, no. 3, pp. 296–321, 2012.
[22] P. Rodriguez Mier, M. Mucientes, and M. Lama, "A hybrid local-global optimization strategy for qos-aware service composition," in IEEE International Conferrence on Web Services (ICWS), 2015, pp. 735–738.
[23] P. Rodriguez Mier, C. Pedrinaci, M. Lama, and M. Mucientes, "An Integrated Semantic Web Service Discovery and Composition Framework," Services Computing, IEEE Transactions on, vol. PP, no. 99, 2015.
[24] T. Weise, M. Blake, and S. Bleul, "Semantic web service composition: The web service challenge perspective," in Web Services Foundations, A. Bouguettaya, Q. Z. Sheng, and F. Daniel, Eds. Springer New York, 2014, pp. 161–187.
[25] J. J.-W. Yoo, S. Kumara, D. Lee, and S.-C. Oh, "A web service composition framework using integer programming with non-functional objectives and constraints," algorithms, vol. 1, p. 7, 2008.
[26] X. Zheng and Y. Yan, "An efficient syntactic web service composition algorithm based on the planning graph model," in IEEE International Conference on Web Services(ICWS'08), 2008, pp. 691–699.

APPENDIX

ALGORITHM FOR SOLVING MAXIMAL TSC PROBLEM

The updated lines of Algorithm 1 are shown by $\triangleright$.

Algorithm 2 Maximal TSC.

$H \leftarrow \emptyset$
$\lambda(i) \leftarrow 0\ \forall i \in S \cup D$ and $p(i) \leftarrow -1\ \forall i \in D$ $\triangleright$
$nd(j) \leftarrow |in(j)|\ \forall j \in S$
$\lambda(s_0) \leftarrow +\infty$ and $L \leftarrow \{s_0\}$ $\triangleright$
for all $i \in out(s_0)$ **do**
    $\lambda(i) \leftarrow +\infty$ and $p(i) \leftarrow 0$ $\triangleright$
    insert$(\lambda(i), i, H)$
end for
while $End \notin L$ and $H \neq \emptyset$ **do**
    $i \leftarrow$ root(H) and $L \leftarrow L \cup \{i\}$
    for all $j \in S \setminus L$ such that $i \in in(j)$ **do**
        $nd(j) \leftarrow nd(j) - 1$
        if $nd(j) == 0$ **then**
            $L \leftarrow L \cup \{j\}$ and $\lambda(j) \leftarrow \min_{k \in in(j)} \lambda(k)$ $\triangleright$
            $v = \min\{\lambda(j), t(j)\}$ $\triangleright$
            for all $k \in out(j) \setminus L$ **do**
                if $\lambda(k) == 0$ **then** $\triangleright$
                    $\lambda(k) \leftarrow v$, $p(k) \leftarrow j$ and insert(v, k, H)
                else if $\lambda(k) < v$ **then** $\triangleright$
                    $\lambda(k) \leftarrow v$, $p(k) \leftarrow j$, increase(v, k, H) $\triangleright$
                end if
            end for
        end if
    end for
end while
Return $\lambda(End)$

Virginie Gabrel got her computational science PhD in 1994 at Université Paris-Dauphine, while her research has been done and funded by the DGA (Direction Générale de l'Armement). In 2005, she got the HDR at Université Paris-Dauphine. From 1996 to 2002, she has been associate professor at Paris 13 university. Since 2002, she is associate professor at Université Paris-Dauphine. From 2012 to 2014, she was assistant director of LAMSADE laboratory. She published around 40 papers in journals and conferences. Her research activities concern the modelization in Operations Research, the resolution of huge integer programs (with decomposition techniques), the industrial applications (satellite mission planning, network optimization, service composition) and the robustness in mathematical programming.

Maude Manouvrier is currently working as an Assistant Professor at the LAMSADE Lab. of Université Paris-Dauphine. She has received a Ph.D. in Computer Science in 2000. From 1998 to 2003, she has been working in an international CNRS - CONICIT cooperation with the Universidad Central de Venezuela (UCV), and from 2008 to 2012, in an international CNRS - FONACIT cooperation with the Universidad Simon Bolivar (USB - Caracas - Venezuela). She was reviewer or member of program committee of several International Conferences and Journals and was, in 2012 and 2013, Program Chair of the International Symposium on Advances in Transaction Processing (in conj. with the International Conference on Mobile Web Information Systems). She published 23 papers in international journals and conferences. Her research interests are focused on Spatio-Temporal and Multimedia Databases, Access Methods and Indexing Structures, Content-Based Image Retrieval and Web Services Composition.

Kamil Moreau is a student at Université Paris-Dauphine, currently in the 3rd year of a bachelor in Applied Mathematics and Computing Science, also in the 3rd year of the Pluridisciplinar Cycle of further studies (CyPES) at Paris Sciences et Lettres. He passed his baccalauréat with distinction in 2013. He has studied graphs since 2014 and worked over service composition problem during a research internship supervised by Virginie Gabrel in 2015.

Cécile Murat received her Ph.D. in Computer Science in 1997, from Université Paris-Dauphine. She is, from 1998, Associate Professor, in the LAMSADE Lab. She received, in 1998, for her Ph.D. studies the Award Nathalie Demassieux in Sciences ("Prix de la Chancellerie des Universités de Paris"). In 2015, she got the HDR from Université Paris-Dauphine. Her research activities, in operation research, aim to solve decision problems. The main focus of her research is to create and to develop combinatorial optimization tools, each for integrate more concepts related to the random uncertainty and others, to solve efficiently practical difficult problems. She is the author of one book and 31 articles in international journals and conferences.