Optimal and Automatic Transactional Web Service Composition with Dependency Graph and 0-1 Linear Programming

Virginie Gabrel, Maude Manouvrier, and Cécile Murat

PSL Université Paris-Dauphine, LAMSADE UMR CNRS 7243
75775 Paris Cedex 16, France
{gabrel,manouvrier,murat}@lamsade.dauphine.fr

Abstract. In this article, we propose a model based on 0-1 linear programming for automatically determining a transactional composite web service (CWS) from a service dependency graph that optimizes a QoS measure. The QoS measure used in this model can be either a classical weighted sum of QoS criteria or a minmax-type criterion (e.g. response time). The transactional properties are a set of rules that ensures a reliable execution of the resulting CWS. The proposed 0-1 linear program is solved using a standard solver (CPLEX). Our experiments show that this new exact model surpasses two main related approaches: an approximate one based on transactional requirements and an exact one, based on 0-1 linear programming (LP), but not dealing with transactional properties. In a large majority of the test sets used for our experiments, our model finds a better solution more rapidly than both related approaches and is able to guarantee its optimality. Moreover, our model is able to find the optimal solutions of big size test sets, as the ones proposed by the Web Service Challenge 2009.

Keywords: Reliable web service composition, Service dependency graph, Integer Linear Programming model, QoS optimization.

1 Introduction

As explained in surveys [1,2], the management of large number of services in the global Internet creates many open problems, in particular in service composition, which consists in selecting/identifying several existing services and combining them into a composite one to produce value-added process.

Many approaches on QoS-aware web service (WS) composition exist, where QoS represents the quality of the service (e.g. price or response time) – see for example survey [3]. As explained in [4], the inter-operation of distributed software-systems is always affected by failures, dynamic changes, availability of resources, and others. And, as argued by [5], to make service-oriented applications more reliable, web services must be examined from a transactional perspective. The execution of a composite WS is reliable if, in case of a component WS failure, the negative impacts are negligible for the user [6]. A service

X. Franch et al. (Eds.): ICSOC 2014, LNCS 8831, pp. 108–122, 2014.

that does not provide a transactional property might be as useless as a service not providing the desired functional results. If the composition considers only functional and QoS requirements, then it is possible that during the execution the whole system becomes inconsistent in presence of failures. WS composition only based on transactional properties ensures a reliable execution, but does not guarantee an optimal QoS composite WS and WS composition only based on QoS does not guarantee a reliable execution of the resulting composite WS. Thus, QoS-aware and transactional-aware should be integrated [7].

In this article, we consider the problem qualified by [8] as the most useful and practical one, which consists in composing services by matching their parameters (input and output attributes) so that the resulting composite service can produce a set of output parameters given a set of input ones. For such problem, the service repository is generally modeled by a Service Dependency Graph (SDG) as defined in [9] and used by example in [7,10,11,12]. The contribution of this article is a new 0-1 linear programming-based optimal approach for QoS and transactional-aware service composition. Experiments show that our model outperforms the two main related approaches: an approximate one [7], based on transactional requirements and an exact one [12], based on 0-1 linear programming, but not dealing with transactional properties.

The rest of the article is structured as follows. Related work is presented in Section 2. Section 3 presents the context and the definitions. Section 4 presents our LP model. Constraints dedicated to transactional properties are separated in a specific Section 5. Section 6 gives experimental results and compare our model to the most related work. Finally, Section 7 concludes.

2 Related Work

Automatic QoS-aware service composition is subject of numerous studies (see for example survey [3]). Two approaches must be distinguished. In the first one, a predefined workflow is supposed to be known. This workflow describes a set of "abstract" tasks to be performed. Moreover, associated to each task, a set of WS with similar functionalities (but different QoS and transactional properties) is also known. The composition problem is then to select one WS per task in order to respect QoS [13,14] and transactional requirements [6,15,16]. In the second approach, the existence of a predefined workflow is not assumed. Available WS are described with a service dependency graph (see for example [7,8,10,11,12,17]) and, the composition problem is to find a sub-graph connecting inputs' query to outputs' query. In this article, we focus on such problem.

Considering QoS-aware composition based on SDG, several methodologies have been proposed: game theory, AI planning (with AND/OR graph and A$*$ algorithm), 0-1 LP (solving with a branch and bound), Petri-Net ... – see a systematic review in [18]. Concerning 0-1 linear programming approach, a model is proposed in [12,19], where a composite WS is decomposed into stages (a stage contains one WS or several WS executed in parallel) and the problem is to select one or several WS per stages. Thus the number of variables and constraints can

be huge since it is proportional to the number of WS 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 WS). The size of the model does not allow to solve big size test sets: none experimental results are given in [19], while, in [12], computational experiments are performed, for 20 WS and 200 data, taking 200 seconds to find the solution and, for 100 WS and 800 data, taking 900 seconds.

Some approaches extend QoS-aware composition to transactionnal and QoS-based approaches (see for example survey [20]). However, to the best of our knowledge, the approach of [7] is the only one proposing a WS composition algorithm based on service dependency graph integrating both transactional and QoS requirements. In this approach, the dependency graph is represented by a colored Petri Net, where places correspond to the data and transitions to the WS. The proposed algorithm is a greedy-like algorithm locally optimizing the QoS. In order to limit the execution time, the authors proposed to identify the WS which are potentially useful to answer the user query. This identification consists in selecting the transactional paths in the dependency graph, that allow to obtain an output data needed by the user from the inputs of the query. The greedy-algorithm then consists in selecting the solution from a smaller dependency graph, only containing the WS which are potentially useful or relevant to answer the user query.

3 Context and Background Definitions

In this article, the service repository (i.e. the set of available services) is represented by a directed graph $G = (X, U)$. The set of vertices X can be partitioned in two sets: S the set of vertices representing WS and, D the set of vertices representing data. In the following, for all $i \in S$, let us denote by $s(i)$ the WS represented by vertex i and, for all $i \in D$, $d(i)$ denotes the data represented by vertex i. The set of directed edges U represents two kinds of dependency: (1) an edge from $i \in S$ to $j \in D$ represents the fact that WS $s(i)$ produces data $d(j)$ ($d(j)$ is an output of $s(i)$), (2) an edge from $i \in D$ to $j \in S$ represents the fact that WS $s(j)$ needs data $d(i)$ to be executed ($d(i)$ is one intput of $s(j)$). Thus, in this graph representation, there does not exist any directed edge of the form: (i, j) with $i \in S$ and $j \in S$ or, $i \in D$ and $j \in D$. Such graph, generally called a Service Dependency Graph, is used in [7,10,11,12]. An example of SDG is presented in Fig. 1.

The user query is defined by a set I of input data (with $I \subset D$) corresponding to the information that the user provides, and a set O of output data representing the information the user needs (with $O \subset D$). Such query is also used in [7,10,11,12] for example.

A composite WS (CWS) satisfying the user query, characterized by I and O, can be represented by a connected sub-graph if and only if: (a) each $o \in O$ is covered by the sub-graph, (b) in this sub-graph, the only vertices without any predecessor belong to I, (c) if a vertex $i \in S$ is covered by the sub-graph, then

all arcs (j, i), with $j \in D$, belong to the sub-graph (indeed each WS $s(i)$ can be executed if and only if all its input data are available) and, (d) this sub-graph does not contain any directed cycle.

For example, given the graph of Fig. 1 and the query described by $I = \{1, 2\}$ and $O = \{7, 8\}$, we can propose different CWS. The CWS $\{s(16), s(18), s(15)\}$ is represented by the following sub-graph: $\{(1, 16), (16, 3), (3, 18), (18, 6), (6, 15), (15, 7), (15, 8)\}$. Let us remark that CWS $\{s(11), s(13), s(15)\}$ is not feasible since it contains the following conflicting situation: to be executed, $s(11)$ needs $d(6)$ as input, and $d(6)$ is obtained by executing $s(13)$ which input $d(4)$ is produced by $s(11)$. In terms of graph, the associated sub-graph $\{(2, 11), (11, 4), (4, 13), (13, 6), (6, 11), (6, 15), (15, 7), (15, 8)\}$ satisfies the aforementioned properties (a), (b) and (c) but does not verify property (d): $(11, 4)$, $(4, 13)$, $(13, 6)$, $(6, 11)$ is a directed cycle.

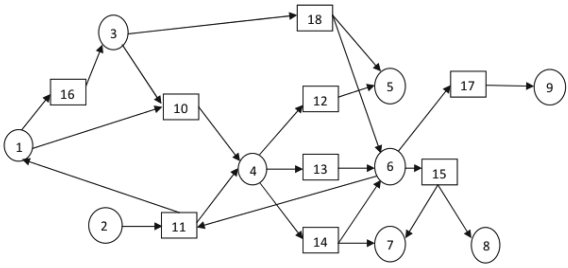

Fig. 1. A service dependency graph with $D = \{1, \ldots, 9\}$ and $S = \{10, \ldots, 18\}$

Given a user query, our problem consists in finding a reliable CWS that optimizes its overall QoS.

4 Linear Programming Model for QoS-Aware Composition

In this article, we model the QoS-aware service composition problem by a 0-1 linear programming model. Decision variables are defined in Subsection 4.1. Subsection 4.2 presents the constraints modeling the I/O of each web service, Subsection 4.3 those implied by the user query, Subsection 4.4 defines the constraints linking decision variables and Subsection 4.5 the constraints eliminating directed cycle. Subsection 4.6 recalls the entire resulting model.

4.1 Decision Variables

We have to introduce three kinds of decision variables:

1. w_i, associated with each $i \in S$: $w_i = 1$ if i is covered by the sub-graph (meaning that $s(i)$ belongs to the composite WS) and 0 otherwise,
2. x_{ij} associated with each directed edge (i, j) of U: $\forall (i, j) \in U$, $x_{ij} = 1$ if the directed edge (i, j) belongs to the sub-graph and 0 otherwise,
3. $t_i \geq 0$ associated with each vertex $i \in X$: t_i represents the topological order of vertex i in the sub-graph.

The objective function is to minimize the following function: $\sum_{i \in S} q_i w_i$, where q_i is the QoS score of WS i. In this article, as in [7,8,12], we use an aggregate QoS score corresponding to a weighted sum of QoS criteria, computed such that the lower the score q_i, the better the service i.

4.2 Constraints Modeling the Input/Output of Each Service

With the previously introduced variables, two constraints, (C_1) and (C_2), modeling the I/O of each service, can be written in a linear form. In the following, for each vertex j, let us denote by $\Gamma^-(j) = \{i \in X : (i, j) \in U\}$ its set of predecessors and, $\Gamma^+(j) = \{i \in X : (j, i) \in U\}$ its set of successors.

For all $j \in S$, a WS $s(j)$ is described by its input and output data: $s(j)$ can be executed if and only if all its input data are available, and $s(j)$ produces a set of output data. In terms of graph, these relations can be described by imposing that an arc (j, k) is in the solution (meaning that output data $d(k)$ is computed by $s(j)$) if and only if all arcs (i, j), with $i \in \Gamma^-(j)$, belong to the solution (meaning that input data for $s(j)$ are available):

$$\forall j \in S,\ \forall k \in \Gamma^+(j),\ \sum_{i \in \Gamma^-(j)} x_{ij} \geq |\Gamma^-(j)| x_{jk}\ (C_1)$$

With such a constraint, if x_{jk} equals to 1, all directed edges entering in vertex j must belong to the solution (ensuring that all input data for $s(j)$ are available). Otherwise, if x_{jk} equals to 0, the constraint is relaxed and plays no role.

Considering the graph of Fig. 1, $s(11)$ produces data $d(1)$ and $d(4)$ and needs data $d(2)$ and $d(6)$ to be executed. Therefore, two constraints must be written for describing inputs and outputs of $s(11)$: (i) considering output $d(1)$, $x_{2,11} + x_{6,11} \geq 2x_{11,1}$ and, (ii) considering output $d(4)$, $x_{2,11} + x_{6,11} \geq 2x_{11,4}$. These constraints imply that $d(1)$ or $d(4)$ can be computed by $s(11)$ if and only if $d(2)$ and $d(6)$ are available as inputs for $s(11)$.

For any data $d(j)$ not provided by the user, $j \in D \setminus I$, $d(j)$ is available when at least one WS computes it. In the associated graph, the set of WS computing $d(j)$ exactly corresponds to $\Gamma^-(j)$ inducing the following constraint:

$$\forall j \in D \setminus I,\ \forall k \in \Gamma^+(j),\ \sum_{i \in \Gamma^-(j)} x_{ij} \geq x_{jk}\ (C_2)$$

This constraint imposes that $d(j)$ can be used by $s(k)$ (inducing that variable x_{jk} is equal to 1) if and only if $d(j)$ has been computed by at least one WS $s(i)$ (leading to $\sum_{i \in \Gamma^-(j)} x_{ij} \geq 1$). When x_{jk} is equal to 0, the constraint plays no role.

Considering the graph of Fig. 1, $d(6)$ is an output of three WS $s(13)$, $s(14)$ and $s(18)$, and is an input of three WS $s(11)$, $s(15)$ and $s(17)$. Thus, three

constraints must be written: (i) As input of $s(11)$: $x_{13,6} + x_{14,6} + x_{18,6} \geq x_{6,11}$, (ii) as input of $s(15)$: $x_{13,6} + x_{14,6} + x_{18,6} \geq x_{6,15}$ and, (iii) as input of $s(17)$: $x_{13,6} + x_{14,6} + x_{18,6} \geq x_{6,17}$. Each constraint imposes that $d(6)$ is available if and only if it has been computed by $s(13)$, $s(14)$ and/or $s(18)$.

4.3 Constraints Implied by the User Query

Each data $j \in O$, needed by the user, must be computed by at least one WS:

$$\forall j \in O,\ \textstyle\sum_{i \in \Gamma^-(j)} x_{ij} \geq 1 \ (C_3)$$

Given the graph of Fig. 1, we consider a user query with $O = \{7, 8\}$. $d(7)$ can be computed by $s(14)$ or $s(15)$, and $d(8)$ by $s(15)$ only. Then, we have: $x_{14,7} + x_{15,7} \geq 1$ and $x_{15,8} \geq 1$.

Any data $d(j)$ provided by the user is available, meaning that $\forall j \in I, \forall k \in \Gamma^+(j)$, arc (j, k) can belong to the resulting sub-graph ($x_{jk} \leq 1$). Therefore, input data provided by the user do not introduce any specific constraints in the model.

4.4 Constraints Linking Decision Variables

Given a particular query, constraints (C_1) to (C_3) are sufficient to represent dependency between data and WS. These constraints only concern variables x. Thus, we have to introduce the following additional constraints for linking variables x and w:

$$\forall j \in S, \textstyle\sum_{k \in \Gamma^+(j)} x_{jk} \leq |\Gamma^+(j)| w_j \ (C_4)$$

This constraint imposes that w_j equals to 1 if at least one directed edge with initial vertex j belongs to the solution. On the contrary, when no edge of the form (j, k) belongs to the solution, constraint plays no role since it becomes $|\Gamma^+(j)| w_j \geq 0$. In this case, w_j can be equal to 1 even if the x values corresponds to a sub-graph which does not cover the vertex j. However, the value of such a solution is strictly greater than the value of the solution with the same x values and $w_j = 0$. Recalling that the objective function is to minimize $\sum_{i \in S} q_i w_i$ ($q_i > 0\ \forall i$) such a solution cannot be an optimal one.

4.5 Constraints for Eliminating Directed Cycle

The last family of constraints are introduced for eliminating directed cycle in the solution. These constraints are classical (initially proposed in [21]) and are written as follows:

$$\begin{array}{lll} t_j - t_i \geq 1 - |X|(1 - x_{ij}) & \forall (i,j) \in U & (C_5) \\ t_i = 0 & \forall i \in I & (C_6) \\ 0 \leq t_i \leq |X| - 1 & \forall i = 1, \ldots, |X| & \end{array}$$

For all $i \in X$, variable t_i represents the topological order of vertex i in the sub-graph. If x_{ij} equals to 1, arc (i, j) belongs to the sub-graph and constraint (C_5) becomes: $t_j - t_i \geq 1$. This constraint imposes that WS $s(i)$ is executed (or data $d(i)$ is obtained) before producing data $d(j)$ (or executing service $s(j)$). If x_{ij} equals to 0, constraint (C_5) becomes: $t_j - t_i \geq (1 - |X|)$. Since t_i belongs to $[0, |X| - 1]$, constraint (C_5) plays no role even if t_i equals to $(|X| - 1)$.

In graph of Fig. 1, we have previously noticed that CWS $\{s(11), s(13), s(15)\}$ is not a feasible solution. This CWS is described by variables $x_{11,4} = x_{4,13} = x_{13,6} = x_{6,11} = 1$ and constraints (C_5) lead to an unfeasibility: (i) $t_4 - t_{11} \geq 1$, (ii) $t_{13} - t_4 \geq 1$, (iii) $t_6 - t_{13} \geq 1$ and, (iv) $t_{11} - t_6 \geq 1 \Longrightarrow 0 \geq 4$.

4.6 Resulting Model

The composition problem based on service dependency graph, $G = (X, U)$, can be represented by the following 0-1 linear program:

$$\begin{cases} \min \sum_{i \in S} q_i w_i & & \\ s.t. \ \sum_{i \in \Gamma^-(j)} x_{ij} \geq |\Gamma^-(j)| x_{jk} & \forall j \in S, \ \forall k \in \Gamma^+(j) & (C_1) \\ \sum_{i \in \Gamma^-(j)} x_{ij} \geq x_{jk} & \forall j \in D \setminus I, \ \forall k \in \Gamma^+(j) & (C_2) \\ \sum_{i \in \Gamma^-(j)} x_{ij} \geq 1 & \forall j \in O & (C_3) \\ \sum_{k \in \Gamma^+(j)} x_{jk} \leq |\Gamma^+(j)| w_j & \forall j \in S & (C_4) \\ t_j - t_i \geq 1 - |X|(1 - x_{ij}) & \forall (i, j) \in U & (C_5) \\ t_i = 0 & \forall i \in I & (C_6) \\ 0 \leq t_i \leq |X| - 1 & \forall i = 1, \ldots, |X| & \\ w_i \in \{0, 1\} & \forall i \in S & \\ x_{ij} \in \{0, 1\} & \forall (i, j) \in U & \end{cases}$$

This model contains $|S| + |U| + |X|$ variables and $2|U| + |S| + |O|$ constraints.

In this aforementioned model, the QoS is only considered in the objective function. This model can easily be extended for taking into account QoS constraints. For example, let us consider a budget constraint of the form: the cost of the composite WS must be lower than C. This QoS constraint can be easily written as follows: $\sum_{i \in S} c_i w_i \leq C$ with c_i being the cost of WS $s(i)$.

This model does not deal with transactional properties. The next section presents how these properties can be included in our model.

5 Extending Our Model with Transactional Properties

To allow a QoS and transactional WS composition, constraints dealing with WS transactional properties should be added to the model. Subsection 5.1 defines the transactional properties and Subsection 5.2 presents the induced constraints.

5.1 Definitions and Context

In this article, we use the more common transactional properties for a WS (see for example survey [20]): *pivot*, *retriable* and *compensatable*. A WS is *pivot* (P)

Table 1. Transactional rules of [15]

Transactional property of a WS	Sequential incompatibility	Parallel incompatibility
P	$P \cup C$ (rule 1)	$P \cup C \cup PR$ (rule 2)
PR	$P \cup C$ (rule 3)	$P \cup C$ (rule 4)
C	$\emptyset$	$P \cup PR$ (rule 5)

if once it successfully completes, its effects remain for ever and cannot be semantically undone. If it fails, then it has no effect at all. For example, a service delivering a non refundable and non exchangeable plane ticket is pivot. A WS is *compensatable* (C) if it exists another WS, or compensation policies, which can semantically undo its execution. For example, a service allowing to reserve a room in a hotel with extra fees in case of cancellation is compensatable. A WS is *retriable* if it guarantees a successful termination after a finite number of invocations. This property is always combined with the two previous one, defining *pivot retriable* (PR) or *compensatable retriable* (CR) WS. For example, a payment service may be (pivot or compensatable) retriable in order to guarantee that the payment succeed. The authors of [15] propose transactional rules defining the possible combinations of component WS to obtain a reliable (i.e. a transactional) composite WS. These rules are summarized in Table 1 where the second and the third columns represent the transactional properties of WS which are incompatible in a composition with a WS of transactional property of column 1. For more details about these rules, reader must refer to [15].

Considering the graph $G = (X, U)$ associated to the composition problem, the transactional property of each WS induces a partition of the subset of vertices $S \subset X$ as follows: $S = P \cup C \cup PR \cup CR$ (each subset is denoted by the corresponding transactional property). In the graph of Fig. 1, we set the following partition: $P = \{12, 16, 17\}$, $C = \{10, 13\}$, $PR = \{11, 18\}$, $CR = \{14, 15\}$.

Considering a particular query with given inputs and expected outputs, introducing transactional rules implies that some paths, belonging to the sub-graph, going from an input to an expected output are eliminated (for example paths containing two vertices belonging to P – see line 1 in Table 1). Thus, in terms of our 0-1 linear program, some additional constraints must be introduced in order to eliminate the solutions which do not respect transactional rules. These additional constraints of linear form are presented in the following subsection.

5.2 Constraints Induced by Transactional Requirements

For each vertex $i \in S$ (corresponding to WS $s(i)$), we need to identify the sets of vertices $j \in S$ which can be executed after or in parallel with i. Using a Depth First Search (for DFS algorithm see for example [22]) on $G = (X, U)$, the following sets A_i and L_i can be easily computed. A_i is set of vertices $j \in S$ belonging to the DFS tree with root i. Indeed, if j belongs to the DFS tree rooted by i, then it exists a path in G from i to j meaning that WS $s(j)$ may

be executed after WS $s(i)$. L_i is the set of services $j \in S$ that can be executed in parallel of i. L_i contains any vertex $j \in S$ such that it does not exist any path from i to j neither from j to i. We have: $L_i = \{j \notin A_i : i \notin A_j\}$. For example, in graph of Fig. 1, $A_{12} = \emptyset$, $L_{12} = \{13, 14, 15, 17, 18\}$ and, $A_{16} = \{10, 11, 12, 13, 14, 15, 17, 18\}$ and $L_{16} = \emptyset$.

Based on Table 1, we must formulate the following constraints (C_7) to (C_{12}).

A transactional composite WS contains at most one pivot:

$$\sum_{i \in P} w_i \leq 1 \ (C_7)$$

For all $i \in P$, if WS $s(i)$ is executed, compensatable WS cannot be executed afterwards (see line 1 of Table 1). The set of compensatable WS that can be executed after WS $s(i)$ is represented by $A_i \cap C$ and the following constraints can be written:

$$\forall i \in P,\ t_i - t_j \geq -|X|(1 - w_i)\ \ \forall j \in A_i \cap C \ (C_8)$$

Indeed, if the vertex $i \in P$ belongs to the sub-graph, then $w_i = 1$ and the associated constraint becomes $t_i - t_j \geq 0$. Consequently, the order of vertex j is necessarily lower or equal to the order of vertex i implying that WS $s(j)$ cannot be executed after WS $s(i)$ ($s(j)$ may be executed before $s(i)$). If the vertex $i \in P$ is not cover by the sub-graph, then $w_i = 0$ and the associated constraint becomes $t_i - t_j \geq -|X|$. This constraint plays no role. For example, in graph of Fig. 1, vertex 16 represents a pivot WS. Vertex 10 represents a compensatable WS which belongs to A_{16} (since there exists a path from 16 to 10 in G). Thus, we have, for example, the following constraint: $t_{16} - t_{10} \geq -18(1 - w_{16})$ inducing that WS $s(16)$ cannot be executed before WS $s(10)$.

For all $i \in P$, if WS $s(i)$ is executed, compensatable WS and pivot retriable WS cannot be executed in parallel with $s(i)$. The set of vertices representing compensatable WS and pivot retriable WS executing in parallel of WS $s(i)$ is $(C \cup PR) \cap L_i$. Thus the following constraint must be respected:

$$\forall i \in P, \sum_{j \in \{(C \cup PR) \cap L_i\}} w_j \leq |(C \cup PR) \cap L_i|(1 - w_i) \ (C_9)$$

If the vertex $i \in P$ belongs to the sub-graph, then $w_i = 1$ and the associated constraint becomes $\sum_{j \in \{(C \cup PR) \cap L_i\}} w_j \leq 0$. Consequently, vertices belonging to $(C \cup PR) \cap L_i$ cannot be covered by the sub-graph. Otherwise, when $w_i = 0$, the associated constraint plays no role. In graph of Fig. 1, vertex 12 represents a pivot WS. The set of vertices representing WS that can be executed in parallel with WS $s(12)$ is $L_{12} = \{13, 14, 15, 17, 18\}$. Then we have, for example, the following constraint: $w_{13} + w_{18} \leq 2(1 - w_{12})$.

For all $i \in PR$, if WS $s(i)$ is executed, pivot WS and compensatable WS cannot be executed afterwards. The set of pivot or compensatable WS that can be executed after WS $s(i)$ is represented by $A_i \cap (P \cup C)$ and the following constraints can be written:

$$\forall i \in PR, t_i - t_j \geq -|X|(1 - w_i)\ \ \forall j \in A_i \cap (P \cup C) \ (C_{10})$$

In graph of Fig. 1, vertex 11 represents a pivot retriable WS. Vertex 12 represents pivot WS which belongs to A_{11} (since it exists a path from 11 to 12 in G). Thus, we have the following constraint: $t_{11} - t_{12} \geq -18(1 - w_{11})$.

For all $i \in PR$, if WS $s(i)$ is executed, compensatable WS and pivot WS cannot be executed in parallel. The set of vertices representing pivot or compensatable WS executing in parallel of WS $s(i)$ is $(C \cup P) \cap L_i$. Thus the following constraints must be respected:

$$\forall i \in PR, \sum_{j \in \{L_i \cap (P \cup C)\}} w_j \leq |L_i \cap (P \cup C)|(1 - w_i) \ (C_{11})$$

Finally, for all $i \in C$, only rule 5 must be respected: WS executed in parallel of WS $s(i)$ cannot be a pivot or a pivot retriable one. Thus, the following constraint must be respected:

$$\forall i \in C, \sum_{j \in \{L_i \cap (PR \cup P)\}} w_j \leq |L_i \cap (PR \cup P)|(1 - w_i) \ (C_{12})$$

The number of constraints induced by transactional properties is: $1 + 2 \mid P \mid +2 \mid PR \mid + \mid C \mid$, i.e. $O(\mid S \mid)$ constraints.

In the following section, we compare our model to the two main related ones: the linear-programming model of [12] and the approximate approach of [7].

6 Experimental Results

The objectives of our experiments are: without transactional requirements, (i) to compare our model with another recent model based on 0-1 linear programming proposed in [12], and (ii) to test our model on the well-known WS composition benchmark of WS-Challenge 2009 [23], and, with transactional properties, (iii) to measure the difficulty inducing by transactional requirements, and (iv) to compare the optimal solution given by our model to the feasible solution obtained with the approximate algorithm proposed in [7].

6.1 Software Configuration and Test Set Description

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, Java 7 and CPLEX solver 12.4.

We have two test sets: (a) the WS repositories and the queries of [7] and (b) the one of WS-Challenge 2009 [23]. In the first test set, there are 10 WS repositories, where the number of WS varies from 100 to 500 (see the first three lines of Table 2), and two repositories containing 1000 WS (not presented in Table 2). The number of data is 20 or 100, representing either sparce (with a small number of data and many dependencies between WS) or non-sparce dependency graphs (with many data and few dependencies between WS). Each WS has between 1 and 5 inputs and between 1 and 3 outputs, randomly generated from an ontology containing 20 generated elements. A transactional property is randomly

associated with each WS. On each WS repository, 10 user queries are randomly generated by varying the number of inputs and the number of outputs between 1 and 3 and by randomly generating the QoS score of each WS. The second test set corresponds to the 5 data sets of WS-Challenge[1] 2009 containing 500, 4000, 8000 and 15000 WS (described by their response time and throughoutput QoS values) with respectively 1500, 10000, 15000 or 25000 data.

Each 0-1 linear programming problem is solved with CPLEX solver which uses a branch and bound algorithm to search the optimal solution. We limit the computation time to 3600 seconds for the first test set and to 300 (limit time given by WS-Challenge) for the second one. Thus, two situations can occur: either CPLEX solves the problem in time and, if it exists, the optimal solution is found and otherwise the absence of solution is proved, or CPLEX is disrupted by time out. In this last case, either a solution is proposed but its solution status is unknown (the algorithm cannot prove that this solution is the optimal one because it has not enough time to explore all the feasible solution set), or no solution is founded in time (even if a solution exists).

6.2 Experiments without Transactional Requirements

Table 2. Description of the first test set and LP-based model comparison

1	WS repository	R1	R2	R3	R4	R5	R7	R8	R9	R10	R11
2	Nb of WS	100	200	300	400	500	100	200	300	400	500
3	Nb of data	20	20	20	20	20	100	100	100	100	100
4	Nb of var. in [12]	1840	2920	3730	4840	5760	4280	5240	6190	7090	8130
5	Nb of var. in P_{new}	906	1643	2228	2977	5760	988	1688	2331	2979	3679
6	Nb of const. in [12]	6344	11033	14840	19583	24209	8690	13261	17338	21550	25979
7	Nb of const. in P_{new}	1404	2554	3481	4646	24209	1397	2506	3509	4535	5625
8	Nb of const. in P_{newT}	4609	16490	26064	47206	61241	5064	13165	27953	39422	64483
9	Ratio r	2.8	2.5	2.3	1.1	4	4	2.3	2	2.3	2.14
10	P_{new} comp. time (s)	1.1	12.1	2.3	6.7	21.2	0.06	0.2	0.3	1	9.2
11	P_{newT} comp. time (s)	2	55.6	15.3	80	140.4	0.1	0.6	1.2	7.5	220.7

We first compare our 0-1 linear model, denoted P_{new}, to the one published in [12]. Results are presented in Lines 4 to 7 and in Line 9 of Table 2. In this table, the ratio r (line 9) is equal to the average value (over 10 queries for a given WS repository) of the computational time taken by CPLEX for solving the model of [12] over the one taken for solving P_{new}.

Let us recall that in P_{new}, the variables represent WS execution order, input/output WS and data availability. The 0-1 linear program sizes are reasonable since the numbers of variables and constraints only depend on the dependency graph size (number of vertices and edges). In [12], the model is based on a decomposition in stages (we choose to set a number of stages equal to 10 for all

[1] Data sets available at `http://www.it-weise.de/documents/files/wsc05-09.zip`

experiments) and program sizes are much greater since the number of variables and constraints depends on the dependency graph size times the number of stages (number of constraints and variables are presented in lines 4 to 7 of Table 2).

Consequently, an optimal solution is founded in 3 times faster in average with P_{new}. Moreover, considering big size test sets with 1000 WS and 100 data, CPLEX solves P_{new} at optimality for all the 10 queries (in 266s in average) while it is not the case for the model presented in [12]. More precisely, for 2 queries, model of [12] finds the optimal solution but without proving their optimality status in 3600s, and for one query, it computes a feasible solution with a greater objective function value. For the 7 queries in which both models can find the optimal solution, the problem is solving 7.4 times faster with P_{new}.

These experimental results prove that our 0-1 linear model is the most efficient one. It can find the optimal solution of all the considered 100 queries excepted 2 of them.

Table 3. Experiments of our model on the WS-Challenge 2009 (WSC) test sets

WSC test set	1 (500WS)	2 (4000 WS)	3 (8000 WS)	4 (8000 WS)	5 (15000 WS)
To find optimal sol.	0.35s	3.46s	4.23s	22s	27s
To prove optimality	6.65s	5.8s	6.7s	> 300s	> 300s

We also applied our model on the test sets of WS-Challenge 2009 [23], slightly adapting it by modifying the objective function in order to optimize the response time. The transformations are: (1) adding a fictitious vertex f and the fictitious arcs $(i, f), \forall i \in O$, (2) replacing the objective function by $min\ t_f$, (3) modifying constraints (C_5) that way: $t_j - t_i \geq d_i - T(1 - x_{ij}), \forall (i, j) \in U$, with d_i the response time of each WS i $(i \in 1, \ldots, |X|)$ and T an upper bound, and (4) deleting variables w_j and their corresponding constraints C_4 (since they are no more necessary). Our model finds the optimal solution for all the 5 test sets in less than 5 minutes (timeout fixed by the WS-Challenge) – see Table 3. Optimality of the solution can not be proved in 300s for the last two sets. However, the optimal solution finds for the biggest set is better than the solution proposed in WS-Challenge 2009. In terms of quality of the solutions, our model is therefore comparable to the recent related approaches [8,11,17] using the WS-Challenge data for their experiments. We have to notice that, in this article, we do not clean the dependency graph by filtering all services relevant for the query and by discarding the rest as done in the aforementioned approaches. Therefore, our model always finds the optimal solution while other approach [17] can not always find it for the biggest data set without any cleaning process. Moreover, as shown in Section 5.2 and in the next section for experiments, our model can be extended to take into account transactional properties.

6.3 Experiments with Additional Transactional Requirements

We introduce transactional properties into our model, denoted then P_{newT}. Firstly, we analyse the consequences of adding such properties into our model.

Computation times taken to solve the problem at optimality are presented in Lines 10-11 of Table 2. When introducing transactional properties in P_{new}, the number of constraints is multiplied by 6.5 (see Line 8 of Table 2) in average while the computation times are multiplied by 7 in average. The difficulty inducing by transactional requirements is important.

Table 4. Comparison between our model and the approximate approach of [7]

1	WS repository	R1	R2	R3	R4	R5	R7	R8	R9	R10	R11
2	Comp. time (s) of P_{newT}	2	55.6	15.3	80	140.4	0.1	0.6	1.2	7.5	220.7
3	(# Queries solved at optimality)	(9)	(8)	(9)	(9)	(9)	(10)	(10)	(10)	(10)	(10)
4	# Queries with no solution found	1	0	1	0	1	0	0	0	0	0
5	Comp. time (s) of [7]	2	24.9	86.4	287	786.4	0.16	1.45	4.9	12.1	25.5
6	(# Queries with solution found)	(8)	(7)	(10)	(8)	(9)	(7)	(8)	(5)	(6)	(5)
7	Approximate ratio	1.68	1.86	3.53	3	3.3	1	1	1	2.1	1

Secondly, we compare our model with the approximate approach of [7]. When we compute solutions with this approximate algorithm, two possible results can be provided: either a solution is proposed (its solution status is unknown), or no solution is proposed (even if a solution exists). Comparison between our model and the approximate approach is presented in Table 4. Line 2 contains the average computation time (in seconds) taken by CPLEX to solve queries at optimality (the number of these solved queries - at most 10 - is given in parenthesis Line 3). Among queries unsolved at optimality in 3600s, P_{newT} may find a feasible solution or may not find any solution (see Line 4). Lines 5 and 6 show the results obtained with the approximate algorithm. Line 5 corresponds to the average computation time to compute a feasible solution (the number of queries for which the approximate algorithm is able to find a solution is given in parenthesis Line 6). Line 7 presents the average approximate ratio (approximate solution value / optimal solution value). Our experimental results show that, in a large majority of queries, our approach based on the CPLEX branch and bound algorithm computes more rapidly an optimal solution than the approximate algorithm. In only one case over more than 100, the approximate algorithm finds a better solution. The queries on non-sparce SDG ($R7$ to $R11$) are easier to solve at optimality. For $R7$ and $R8$ data sets, computation times are very small: all optimal solutions are found by P_{newT} in less than 1 second. Queries on sparce SDG with 20 data ($R1$ to $R5$) are much more difficult to solve, the computation times are important even if, in average, P_{newT} computes the optimal solution more rapidly (for $R5$, 9 queries are solved at optimality with an average computation time of 140s while the approximate algorithms needs 786s to find a feasible solution with a value equals to 3.3 times the optimal value in average).

Computation times vary a lot with the query. For example, for $R2$ with an average computation times of 55.6, the "harder" query needs 412s to determine the optimal solution while 5 queries take less than 3s and 2 queries around 10s ;

the 2 remaining queries are not solved at optimality: for one query, P_{newT} finds a better solution than one computed by the approximate algorithm (the value is 15% better), and for the other query, P_{newT} finds a feasible solution while the approximate algorithm cannot provide any solution. When no solution can be found by both algorithms, we cannot conclude that no solution exists: either the approximate algorithm cannot find a feasible solution (it often occurs for queries of $R7$ to $R11$), or CPLEX hasn't enough time to find a feasible solution. If we don't impose time limit, P_{newT} should be able to find an optimal solution.

Finally we have experimented our model on two WS repositories containing 1000 WS (data sets $R6$ and $R12$). $R6$ contains 20 data and P_{newT} takes 409s in average to compute the optimal solution of 8 queries. For the two remaining queries, P_{newT} only finds a feasible solution in 3600s for one, and cannot find any solution for the other, because of timeout. $R12$ contains 100 data and P_{newT} takes 525s in average to compute the optimal solution of only 3 queries. It cannot find any solution for 4 queries and a feasible solution for the 3 remaining ones, because of timeout. With this problem size, it becomes hard to solve at optimality with CPLEX.

7 Conclusion

In this article, we present a 0-1 linear program for automatically determining a transactional composite WS optimizing QoS from a service dependency graph. With our model, the QoS and transactional-aware composition problem can be solved at optimality. As far as we know, it is the first time. With consequent experimental results, we show that our model dominates an already recent published one [12], also based on linear programming for solving the QoS-aware composition problem without transactional requirement. Our model also finds all the optimal solutions for the well-known service composition benchmark of WS-Challenge 2009. Then, we compare our approach, with the only related one including transactional requirements [7], which is an approximate approach. Experimental results show that, when an optimal solution exists, our model can find it generally faster than the related work. However, for big size test sets, a standard solver like CPLEX is too long to find optimal solution. Specific resolution methods should be proposed to solve such 0-1 linear programming model. This topic will be the focus of our future research.

References

1. Issarny, V., Georgantas, N., Hachem, S., Zarras, A., et al.: Service-oriented middleware for the Future Internet: state of the art and research directions. J. of Internet Services and App. 2(1), 23–45 (2011)
2. Dustdar, S., Pichler, R., Savenkov, V., Truong, H.L.: Quality-aware Service-oriented Data Integration: Requirements, State of the Art and Open Challenges. SIGMOD Rec. 41(1), 11–19 (2012)
3. Strunk, A.: QoS-Aware Service Composition: A Survey. In: IEEE ECOWS, pp. 67–74 (2010)

4. Liu, A., Li, Q., Huang, L., Xiao, M.: FACTS: A Framework for Fault-Tolerant Composition of Transactional Web Service. IEEE Trans. on Serv. Comp. 3(1), 46–59 (2010)
5. Badr, Y., Benslimane, D., Maamar, Z., Liu, L.: Guest Editorial: Special Section on Transactional Web Services. IEEE Trans. on Serv. Comp. 3(1), 30–31 (2010)
6. Gabrel, V., Manouvrier, M., Megdiche, I., Murat, C.: A new 0-1 linear program for QoS and transactional-aware web service composition. In: IEEE ISCC, pp. 845–850 (2012)
7. Cardinale, Y., Haddad, J.E., Manouvrier, M., Rukoz, M.: CPN-TWS: a coloured petri-net approach for transactional-QoS driven Web Service composition. Int. J. of Web and Grid Services (IJWGS) 7(1), 91–115 (2011)
8. Yan, Y., Chen, M., Yang, Y.: Anytime QoS Optimization over the PlanGraph for Web Service Composition. In: ACM SAC, pp. 1968–1975 (2012)
9. Liang, Q., Su, S.: AND/OR Graph and Search Algorithm for Discovering Composite Web Services. Int. J. Web Service Res. (IJWSR) 2(4), 48–67 (2005)
10. Gu, Z., Li, J., Xu, B.: Automatic Service Composition Based on Enhanced Service Dependency Graph. In: IEEE ICWS, pp. 246–253 (2008)
11. Jiang, W., Zhang, C., Huang, Z., Chen, M., Hu, S., Liu, Z.: QSynth: A Tool for QoS-aware Automatic Service Composition. In: IEEE ICWS, pp. 42–49 (2010)
12. Paganelli, F., Ambra, T., Parlanti, D.: A QoS-aware service composition approach based on semantic annotations and integer programming. Int. J. of Web Info. Sys. (IJWIS) 8(3), 296–321 (2012)
13. Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. IEEE Trans. on Soft. Eng. 30(5), 311–327 (2004)
14. Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for Web services selection with end-to-end QoS constraints. ACM Trans. on the Web 1, 1–26 (2007)
15. Haddad, J.E., Manouvrier, M., Rukoz, M.: TQoS: Transactional and QoS-aware selection algorithm for automatic Web service composition. IEEE Trans. on Serv. Comp. 3(1), 73–85 (2010)
16. Syu, Y., FanJiang, Y.Y., Kuo, J.Y., Ma, S.P.: Towards a Genetic Algorithm Approach to Automating Workflow Composition for Web Services with Transactional and QoS-Awareness. In: IEEE SERVICES, pp. 295–302 (2011)
17. Rodriguez-Mier, P., Mucientes, M., Lama, M.: A dynamic qoS-aware semantic web service composition algorithm. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) Service Oriented Computing. LNCS, vol. 7636, pp. 623–630. Springer, Heidelberg (2012)
18. Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I.: Software Architecture Optimization Methods: A Systematic Literature Review. IEEE Trans. on Soft. Eng. 39(5), 658–683 (2013)
19. Yoo, J.J.W., Kumara, S., Lee, D., Oh, S.C.: A Web Service Composition Framework Using Integer Programming with Non-functional Objectives and Constraints. In: IEEE CEC/EEE, pp. 347–350 (2008)
20. Cardinale, Y., Haddad, J.E., Manouvrier, M., Rukoz, M.: Transactional-aware Web Service Composition: A Survey. In: Handbook of Research on Non-Functional Prop. for Service-oriented Sys.: Future Directions, pp. 116–142. IGI Global (2011)
21. Miller, C.E., Tucker, A.W., Zemlin, R.A.: Integer Programming Formulation of Traveling Salesman Problems. J. of ACM 7(4), 326–329 (1960)
22. Aho, A.V., Hopcroft, J.E., Ullman, J.: Data Structures and Algorithms, 1st edn. Addison-Wesley Longman Pub. Co., Inc. (1983)
23. Kona, S., Bansal, A., Blake, M.B., Bleul, S., Weise, T.: WSC-2009: A Quality of Service-Oriented Web Services Challenge. In: IEEE CEC, pp. 487–490 (2009)