An Expressive and Efficient Solution to the Service Selection Problem

Daniel Izquierdo, María-Esther Vidal, and Blai Bonet

Departamento de Computación
Universidad Simón Bolívar
Caracas 89000, Venezuela
{idaniel,mvidal,bonet}@ldc.usb.ve

Abstract. Given the tremendous amount of potential Semantic Web Services that can be created from online sources by using existing annotation tools, expressive formalisms and efficient and scalable approaches to solve the Service Selection Problem (SSP) are required. In this paper, we propose a framework that is firmly grounded on logic. The framework adopts the Local-As-View approach [21] that semantically describes services in terms of concepts in a domain ontology. A user request is comprised of a conjunctive query on concepts from the domain ontology and a set of preferences; the query expresses the abstract concepts that must be combined to solve the query, and the preferences are used to rank the solutions. Relations in the domain ontology are considered to augment the space of available services. Using this representation, the problem of service selection is cast as a problem of query rewriting, from the area of integration systems. Then, building on related work, we devise an encoding of SSP as a logical theory whose models are in correspondence with the service rewritings of the user query, and in presence of preferences, whose best models are in correspondence with the best-ranked valid solutions. Thus, by exploiting known properties of modern SAT solvers, we provide an efficient and scalable solution to SSP. The approach provides the basis to represent a large number of real-world situations; additionally, it does not only scale up to large instances, as the experimental results show, but is also sound and complete since, being founded on logic, is amenable to formal analysis.

1 Introduction

Existing Web infrastructures support the publication and access to a tremendous amount of Web data sources, some of which can be labeled and converted into Semantic Web Services by using existing annotation tools like the one proposed by Ambite et al. [3]. Once this huge dataset of Semantic Web Services become available, users more than ever will require techniques to effectively select the services that meet their requirements. In order to achieve this goal, functional and non-functional properties of the services should be precisely described, as well as criteria to select and rank the solutions that best meet user requests. In this paper, we extend an approach traditionally used in the area of data integration to solve the Service Selection Problem. First, concepts from a domain ontology provide a precise description of the semantics of available sources. These descriptions correspond to views on concepts of the domain ontology and follow the Local-As-View approach (LAV) [21]. LAV mappings are able to scale to service datasets that constantly change, in contrast to Global-As-View definitions (GAV), where generic concepts are defined in terms of the available sources and cannot be easily modified when sources constantly change. A user request is modeled as a conjunctive query on the ontology concepts, and user preferences are associated with a cost and constraint the solutions of the query. Based on this integration framework, we define the problem of selecting the services that best meet the user request, as the problem of enumerating and ranking the service rewritings of the user query. Each rewriting is associated with a cost equal to the cost of the preferences violated by the rewriting, and the best ranked rewriting is one that has minimum cost among all valid rewritings. Knowledge encoded in the domain ontology is used to expand the space of rewritings. Thus, our proposed service selection approach is cast into the well-known Query Rewriting Problem (QRP) for LAV which is central to integration systems. The QRP has been shown to be NP-complete [25], and it consists of a conjunctive query that must be answered in terms of views in which the query and the views are described using LAV with abstract relations. This problem is important in the context of data integration [9, 17], and query optimization and data maintenance [1, 21], and several approaches have been defined that scale up to a large number of views [4, 13, 14, 21, 22].

We propose a solution that extends the recent approach of Arvelo et al. [4] which is based on the efficient enumeration of models of a propositional logic theory. Similarly, given an instance of SSP, a logical theory is constructed such that each model of the theory encodes a valid rewriting; the theory can be constructed in polynomial time from the instance of the SSP. All valid rewritings are obtained by enumerating all models of the theory in linear time by using off-the-shelf model enumeration solvers such as c2d [10] and Relsat [5]; off-the- seft SAT solvers such as Minisat [15] can be used to produce all the minimal rewritings. Thus, we exploit known properties of modern SAT solvers that implement non-chronological backtacking via clause learning, caching of common subproblems and heuristics, to provide an efficient and scalable solution to SSP.

To summarize the contributions of our work are as follows:

- Service descriptions as LAV mappings on ontology concepts and user preferences as logic constraints, to provide an expressive description framework.
- Formalization of SSP as the well-known QRP, and the extension of efficient and scalable existing solutions to exploit the properties of modern SAT solvers to construct a logical theory that encodes an instance of SSP, in a way that the theory models encode the valid rewritings and best models correspond to the optimal (best) rewritings.
- Reduction of the combinatorial search, by efficiently enumerating the solutions.

The paper contains five more sections. The next section defines our approach and illustrates the SSP with typical examples. Section 3 describes our solution and system architecture; Section 4 reports our empirical results over different benchmark problems. Sections 5 summarizes existing approaches; and the paper concludes with a final discussion in Section 6.

2 Service Selection Problem

We consider service selection problems comprised of the description IS of the integration framework and of a user request R over the IS.

Our setting is based on the assuption that services are semantically described in terms of a domain ontology. Formally, the integration framework corresponds to a tuple $IS = \langle D, S, M \rangle$ where D is the domain ontology, S describes the available services, and M is the collection of LAV mappings that define the services in S as views on the ontology D. On the other hand, the user input $R = \langle Q, P \rangle$ consists of a request for composition of services, described as a conjunctive query Q over the ontology, and a set P of preferences for the requested composition.

In the following subsections, we describe in detail all the ingredients making up a service selection problem, and illustrate the framework with several examples.

2.1 Domain Ontology

The domain ontology D is a tuple $\langle \sigma, A \rangle$ where σ is a *signature* for a logical language and A is a collection of axioms describing the ontology. A signature σ is a tuple $\langle R_1^{r_1}, \ldots, R_n^{r_n}, c_1, \ldots, c_m \rangle$ where each R_i is a relational symbol of arity r_i,[1] and each c_j is a constant symbol. The set of axioms describe the ontology by defining the relationships between the ontology relations. For the present work, we only assume subsumption relationships among concepts that are expressed with rules of the form:

$$R(\bar{x}, \bar{y}, \bar{a}) \sqsubseteq P(\bar{x}, \bar{b}) , \qquad (1)$$

where R and P are predicates in σ of appropriate arity, $\bar{x}$ and $\bar{y}$ are lists of variables (repetitions allowed), and $\bar{a}$ and $\bar{b}$ are lists of constant symbols in σ (repetitions allowed). All these lists except $\bar{x}$ can be empty.

Although limited in appearance, rules of this form are quite expressive as they allow us to specify diverse relationships among predicates. For example,

– **Hierarchy of classes and subclasses** (or types and subtypes): classes are specified with unary predicates. A subclass relantionship among two classes is directly specified with a simple rule; e.g., the rule $\mathit{penguin}(x) \sqsubseteq \mathit{bird}(x)$ declares that every penguin is a bird.

[1] We use the notation R^r to say that R is a relational symbol of arity r.

– **Subrelations via specialization:** A subrelation of R^r can be specified by constraining another relation P^s $(r < s)$. For example, the rule:

$$\mathit{descendant}(\mathit{Elizabeth\ II}, x) \ \sqsubseteq \ \mathit{noble}(x) \tag{2}$$

tells that any descendant of Queen Elizabeth II is a noble.

– **Indirect subsumption:** it is even possible to specify a subrelation via another seemengly unrelated predicate. For example, the rule:

$$\mathit{citizen\text{-}of}(x, \mathit{Montreal}) \ \sqsubseteq \ \mathit{lives\text{-}in}(x, \mathit{Canada})$$

says that when the second argument of 'CitizenOf' is fixed to the constant 'Montreal', the tuples in the relation 'CitizenOf' are contained in the set of tuples in the relation 'LivesIn' whose second component is 'Canada'.

However, we require that the *dependency graph* for the ontology to be a *forest of trees.* The dependency graph $G(D)$ is a labeled directed graph that is constructed as follows: the nodes are the relational symbols in the signature σ, and there is an edge (R, P) in the graph iff there is an rule of the form: (1) that is labeled with the *bindings* induced by the rule. For example, if $\mathit{descendant}(x, y)$ and $\mathit{noble}(z)$ are two predicates in the signature and there is the rule (2), there edge from Descendant to Noble is labeled with the bindings $\{x = \mathit{Elizabeth\ II}, y = z\}$.

2.2 Services and LAV Mappings

The available services are represented by means of another signature $S = \tau = \langle S_1^{s_1}, \ldots, S_k^{s_k} \rangle$, called the *services signature*, that consists of relational symbols S_i of arity s_i: each such symbol S_i represents a concrete service in the Semantic Web that "offers" some information.

The semantic description of services is expressed following the LAV paradigm in terms of LAV mappings. These mappings describe the services in terms of concepts in the domain ontology [25]: for each service S_i, there is a LAV mapping that describes S_i as a *conjunctive query* on the concepts in the ontology; input and output attributes of the service are also distinguished. For example, a service $S(x, y)$ that returns information about flights originating at a given US city can be described with the view:

$$S(\$x, y) \ :- \ \mathit{flight}(x, y), \mathit{uscity}(x)\ ,$$

where $flight^2$ and uscity^1 are relational symbols in the ontology; the symbol \$ denotes that x is an input attribute. The semantic interpretation of a mapping like this one enforce the following:

– the service represented by S provides information in the form of tuples (x, y),
– each tuple (x, y) returned by the service satisfies the rhs of the view; i.e., $\mathit{flight}(x, y)$ and $\mathit{uscity}(x)$, and
– the views are not necessarily *complete*; i.e., there may be other tuples (x, y) that satisfy the rhs of the view but which are not available through the service represented by S.

The LAV approach is commonly used in integration systems because it permits the scalability of the system as new services become available [25]. Under LAV, the appearance of a new service only causes the addition of a new mapping describing the service in terms of the concepts in the ontology. Under GAV, on the other hand, the ontology concepts are semantically described using views in terms of the sources of information. Thus, the extension or modification of the ontology is an easy task in GAV as it only involves the addition or local modification of few descriptions [25].

Therefore, the LAV approach is best suited for applications with a stable ontology but with changing data sources whereas the GAV approach is best suited for applications with stable data sources and a changing ontology. In any case, the chosen approach is guarantee to efficiently scale with respect to the changing characteristic but not with respect to the stable one; e.g., a LAV description can be efficiently updated as data sources are added or deleted, yet a slight revision of the ontology implies a revision of all LAV mappings.

For the Semantic Web, we assume that the ontology of concepts is the stable component. We believe that this is a reasonable assumption since, once a common language is agreed upon to describe Web resources, the only changing characteristic is the number and nature of resources which constantly pop up or disappear from the Web.

Up to here, we have described all elements in the integration framework $IS = \langle D, S, M \rangle$ where $D = \langle \sigma, A \rangle$ is an ontology of concepts, $S = \tau$ represent the available services in the Web and M is a collection of LAV mappings that describe the services in terms of the concepts in the ontology. The integration framework can be thought as the "knowledge base" (KB) in a system designed for answering requests about the selection and composition of Web services. Ideally, the KB should support the efficient processing of user requests.

2.3 User Requests

A user request is formally described by a tuple $R = \langle Q, P \rangle$ where Q is a conjunctive query in terms of concepts from the ontology that expresses how the abstract concepts must be combined to resolve a given task, and a set P of preferences. For example, the query:

$$Q(x) \;:-\; \mathit{flight}(\mathrm{LA}, x), \mathit{flight}(x, \mathrm{Paris})$$

can be used to find all cities on which a two-leg flight from Los Angeles to Paris stop. Continuing with the above example, this query can be answered using the view represented by S as $I(x) :- S(\mathrm{LA}, x), S(x, \mathrm{Paris})$. This rewriting is correct yet not necessarily complete as there may be two-leg flights from Los Angeles to Paris that stop at non-US cities, e.g., London.

The preferences are used to rank the collection of all valid rewritings of the query. Once this ranking is obtained, the solution of request R is any best-ranked valid rewriting. In this work, we consider a simple yet expressive system of preferences in which each preference is given as a 'soft constraint' on the rewriting of the query. A soft constraint is a tuple $\pi = \langle \varphi, c \rangle$ where φ is a propositional formula and c is the cost associated with φ. The idea is that each valid rewriting of the query is associated with a cost equal to the sum of the costs of the preferences violated by the rewriting, and that the best ranked rewriting is one that has minimum cost among all valid rewriting.

It only remains to say over which language the formulas φ are allowed to be and when a preference is violated by a rewriting. The set of propositions for constructing preferences is $\mathcal{L}(IS) = \{R : R \in \sigma\} \cup \{S : S \in \tau\}$ that corresponds to the relational symbols either in the ontology signature σ or in the services signature τ. Elements of $\mathcal{L}(IS)$ are propositional symbols that should not be confused with their relational interpretation in IS; indeed, if the reader is more comfortable, he may use a different symbols altogether such as P_R, $[R]$, or other.

The validity of a preference is established with respect to the propositional model $\mathcal{M}(I)$ (truth assignment for the symbols in $\mathcal{L}(IS)$) constructed from an answer $I(\bar{x})$. The model $\mathcal{M}(I)$ is defined by $S = \mathbf{true}$ iff the service S appears in $I(\bar{x})$, for $S \in \tau$, or by $R = \mathbf{true}$ iff the concept R appears in the unique path from a concept R' to the root in the dependency graph $G(D)$ where R' appears in a service $S(\bar{y})$ used in $I(\bar{x})$, for $R \in \sigma$. That is, the model makes true the service symbols used in $I(\bar{x})$, or the ontology symbols used in services in $I(\bar{x})$, or the ontology symbols that can be reached from the latter in the dependency graph $G(D)$. For example, the answer $I(x) :- S(\text{LA}, x), S(x, \text{Paris})$ defines the model $\mathcal{M}(I) = \{S = \mathbf{true}, \mathit{flight} = \mathbf{true}, \mathit{uscity} = \mathbf{true}\}$.

A preference φ holds in an answer $I(\bar{x})$ iff $\mathcal{M}(I) \vDash \varphi$. This simple system of preferences permits us to represent interesting cases such as:

- **Hard constraints:** any constraint $\pi = \langle \varphi, \infty \rangle$ is treated as a hard constraint that must be satisfied by every rewriting,
- **QoS preferences:** this type of preferences can be used to assign absolute quantities of reward/cost to single services as the one used for integrated QoS parameters. For example, if each service S_i is associated with a QoS reward of r_i, then the collection of preferences $\pi_i = \langle \neg S_i, -r_i \rangle$ selects a valid rewriting with services that have the highest combined QoS as the best rewriting,
- **Conditional preferences:** a user's preference of the type 'if service S is used, then service R should be used as well' can be modeled with the hard constraint $S \Rightarrow R$,
- **Preferences of the type at-least-one:** a user's preference of the type that at least one of the services $S_1, \ldots, S_n$ should be used in the rewriting, can be modeled with the hard constraint $S_1 \vee \cdots \vee S_n$, and
- **Preferences of the type at-most-one:** a user's preference of the type that at most one of the services $S_1, \ldots, S_n$ should be used in the rewriting, can be modeled with the collection $\{\neg S_i \vee \neg S_j : i \neq j\}$ of hard constraints.

2.4 Examples

Let us consider a simple travel-information system that contains information about flight and train trips between cities and information about which cities are in the US. The domain ontology is comprised of the predicates *trip*^{2}, *AA-flight*^{2}, *UA-flight*^{2}, *AT-train*^{2}, *UP-train*^{2}, *flight*^{3}, *train*^{3} and *uscity*^{1}, and the constants AA, UA, AT, UP, LA, NY, Paris. The first predicate relates cities (x, y) if there is a direct trip either by plane or train between them. The predicates *AA-flight* and *UA-flight* relate cities that are connected by direct flights either with American (AA) or United (UA). The predicates *AT-train* and *UP-train* relate cities that are connected by direct train trips either with Amtrak (AT) or Union Pacific Railway (UP). The flight predicate relates (x, y, t) whenever there is a direct flight from x to y operated by airline t, and similarly for *train*. Finally, *uscity* indicates when a given city is a US city or not. There are various subsumption relationships among these predicates that are captured by the following axioms:

$$\begin{aligned} \mathit{flight}(x,y,t) &\sqsubseteq \mathit{trip}(x,y)\,, \\ \mathit{train}(x,y,t) &\sqsubseteq \mathit{trip}(x,y)\,. \end{aligned}$$

For the services, assume that the available data sources on the Internet contain the following information:

– *national-flight*(x, y) relates two US cities that are connected by a direct flight,
– *one-way-flight*(x, y) relates two cities that are connected by a one-way flight,
– *one-stop*(x, y) relates two cities that are connected by a one-stop flight,
– *to-pa*(x) tells if there is a direct flight from x to Paris,
– *onestop-to-pa*(x, y) if there is a flight from x to Paris with a stop at y,
– *from-la*(x) tells if there is a flight from Los Angeles to x,
– *national-train*(x, y) relates US cities that are connected by a direct train, and
– *one-way-train*(x, y) relates US cities that are connected by a one-way train.

These services are semantically described using the concepts in the ontology by the following LAV mappings:

$$\begin{aligned} \mathit{national\text{-}flight}(x,y) &\coloneq \mathit{flight}(x,y,t),\ \mathit{uscity}(x),\ \mathit{uscity}(y)\,, \\ \mathit{AA\text{-}flight}(x,y) &\coloneq \mathit{flight}(x,y,\mathrm{AA})\,, \\ \mathit{UA\text{-}flight}(x,y) &\coloneq \mathit{flight}(x,y,\mathrm{UA})\,, \\ \mathit{one\text{-}stop}(x,z) &\coloneq \mathit{flight}(x,y,t),\ \mathit{flight}(y,z,t)\,, \\ \mathit{to\text{-}pa}(x) &\coloneq \mathit{flight}(x,\mathrm{Paris},\mathrm{AA})\,, \\ \mathit{from\text{-}la}(x) &\coloneq \mathit{flight}(\mathrm{LA},x,\mathrm{UA})\,, \\ \mathit{national\text{-}train}(x,y) &\coloneq \mathit{train}(x,y,t),\ \mathit{uscity}(x),\ \mathit{uscity}(y)\,, \\ \mathit{AT\text{-}train}(x,y) &\coloneq \mathit{train}(x,y,\mathrm{AT})\,, \\ \mathit{UP\text{-}train}(x,y) &\coloneq \mathit{train}(x,y,\mathrm{UP})\,. \end{aligned}$$

Observe that each tuple produced by each service satisfies the semantic description expressed by the body of the rule. For example, the tuples that satisfy the predicate *national-flight*(x, y) also meet the conjunctive formula:

$$\exists t(\mathit{flight}(x,y,t)\ \wedge\ \mathit{uscity}(x)\ \wedge\ \mathit{uscity}(y))\,.$$

However, there may be tuples that satisfy this formula that are not produced by the service represented by $\textit{national-flight}(x, y)$. These elements make up the integration framework IS. In the rest of this section. we give examples of possible user requests and their solutions.

Suppose that a user is interested in identifying the services able to retrieve one-stop round trips from a US city x to any city y in the world. Notice that the trip from x to y stops at a city u, that the back trip from y to v stops at a city v, and that u may not be equal to v. This request can be modeled as the following connjunctive query:

$$Q(x, u, y, v) \ :-\ \textit{uscity}(x),\ \textit{trip}(x, u),\ \textit{trip}(u, y),\ \textit{trip}(y, v),\ \textit{trip}(v, x)\,.$$

Any rewriting of the ontology predicates in terms of the services correspond to a *composition of services* that implements the user request. For example, the following rewriting is a valid solution to the request:

$$\begin{aligned} I(x, u, y, v) \ :-\ & \textit{national-flight}(x, u),\ \textit{to-pa}(u), \\ & \textit{one-way-flight}(\text{Paris}, v),\ \textit{national-flight}(v, x)\,. \end{aligned}$$

But, the following two rewritings are not valid solutions:

$$\begin{aligned} I'(x, u, y, v) \ &:-\ \textit{national-flight}(x, u),\ \textit{to-pa}(u),\ \textit{from-la}(v),\ \textit{national-flight}(v, x)\,, \\ I''(x, u, y, v) \ &:-\ \textit{one-stop}(x, y),\ \textit{one-way-flight}(y, v),\ \textit{national-flight}(v, x)\,. \end{aligned}$$

The first is not valid because it maps the query variable y into two different constants Paris and LA that denote different cities, and the second rewriting is not valid because the service $\textit{one-stop}(x, y)$ does not receive as input, or produce as output, the middle city u where the flight from x to y must stop.

As shown, one can use a system for rewriting queries in terms of views for computing solutions to the Service Selection Problem, as the valid solutions correspond to the valid rewritings of the query. However, in the presence of user preferences, the solutions must be ranked according to the preferences and the best solutions should be returned.

To illustrate the use of preferences, consider the following request:

$$Q(x, y) \ :-\ \textit{trip}(\text{LA}, x),\ \textit{trip}(x, \text{NY}),\ \textit{trip}(\text{NY}, y),\ \textit{trip}(y, \text{LA})$$

that looks for round-trips between Los Angeles and New York such that each direction is a one-stop trip. Observe that the query is posed in a way that there are no restrictions whatsoever on the use of planes or trains for any leg of the trip. However, as is typical, users have preferences about using planes or trains. For this example, we study four different scenarios for user's preferences and show how to model them in the proposed framework:

P1. The user prefers to flight rather than to travel by train. This can be modeled by assigning a high reward to the symbol *flight*. Likewise, a preference of trains over airplanes can be modeled by assigning a high reward to the symbol *train*.

P2. The user is indiferent with respect to trains or airplanes, yet s/he does not want to mix both. This preference is an at-most-one preference over the set $\{\textit{flight}, \textit{train}\}$ that corresponds to the formula $\neg\textit{flight} \vee \neg\textit{train}$ and a cost for the violation of the preference.

P3. If the user travels by airplane, s/he prefers to always use the same airline (independently of the airline). This preference can be modeled with the formula $\neg AA\text{-}\textit{flight} \vee \neg UA\text{-}\textit{flight}$ together with a cost. Additionally, the other means of air transportation should be 'disabled' since they may return flights operated by any airline; e.g., add the constraint $\neg\textit{national-flight}$ with a high cost.

P4. Finally, if the user travels by airplane, s/he prefers to use UA. This is a complex preference that can be modeled with the formula

$$(\textit{flight} \Rightarrow UA\text{-}\textit{flight}) \wedge (\neg UA\text{-}\textit{flight} \vee \neg AA\text{-}\textit{flight})\,.$$

The first part says that if a leg of the trip is done by plane, then UA must be used, while the second part says that whenever UA is used, AA should not be used. Also, the services that do not guarantee airline operators should be disabled as in the previous case.

All these preferences correspond to formulas over the propositional language $\mathcal{L}(IS)$. The formulas for the last three preferences can be treated as hard constraints if the are associated with infinite cost, or as soft constraints meaning that the user prefers (but is not limited to) solutions that do not violate the preferences.

3 Solution and System Architecture

We extend the McdSat system of Arvelo et al. [4] for Query Rewriting Problems (QRP). An instance of a QRP consists of a collection of views and a query on abstract concepts. The problem consists is in "re-writing" the query in terms of the views so that each tuple produced by the rewriting is a tuple of the solution. McdSat reduces the QRP into the problem of finding the models of a propositional logic theory. The theory constructed by McdSat satisfies the following properties: (1) there is a 1-1 correspondence between the valid rewritings of the query and the models of the logical theory, (2) given a model of the theory, one can recover the corresponding rewriting in linear time, and (3) the theory can be constructed in polynomial time from the instance of the QRP. Once the logical theory is constructed, one can be interested in finding all minimal rewritings of the query as done in data integration systems with incomplete sources [25], or just a model of the theory [**?**]. For the former, off-the-shelf model enumeration solvers such as c2d [10] and Relsat [5] can be used, while off-the-shelf SAT solvers such as Minisat [15] can be used in the latter case.

In this section, we have just enough space to explain how the logical theory constructed by McdSat can be extended to capture the features associated with SSPs that are not present in QRPs; namely, handling constant symbols and input and output attributes, the ontology of concepts with subsumption relationships, and user's preferences. The result is an extended theory whose models are in correspondence with the valid solutions of the SSP and, in the presence of preferences, whose *best models* are in correspondence with the best-ranked valid solutions of the SSP.

Constant Symbols and Input Output Attributes McdSat did not provide support for constant symbols in the language or service input and output attributes, yet incorporating these functionality is straightforward. First to consider constant symbols, basically, one only has to track the unification of variables with constants using new propositional symbols, and to propagate such unifications transitively using propositional implications to avoiding the unification of different constant symbols. Second, to respect the input and output attributes of the available services, one has to ensure that input attributes of a service unify with constant symbols in the user query or unify with output attributes returned by other services in the rewriting. These modifications involve the addition of a relatively small number of new propositional symbols and clauses to the CNF generated by McdSat.

Ontology The subsumption relationships in the ontology make the rewriting process more complex as now one need to consider unification of predicate symbols of different name and arity. Indeed, consider the following four ontology concepts, where a, b, c and d are constant symbols, and x, y and z are variables:

$$\begin{aligned} R(a,y) &\sqsubseteq T(c,y)\,, \\ P(b,y,z) &\sqsubseteq R(a,y)\,, \\ M(a,x) &\sqsubseteq N(d,x)\,, \\ P(d,x,z) &\sqsubseteq M(a,x)\,, \end{aligned}$$

and the following user request $Q(x,y)$ with the services $S1$, $S2$ and $S3$:

$$\begin{aligned} Q(x,y) &\coloneq T(z,y),\, N(z,x)\,, \\ S_1(y) &\coloneq R(a,y)\,, \\ S_2(x,z) &\coloneq N(z,x)\,, \\ S_3(x,z) &\coloneq P(d,x,z)\,. \end{aligned}$$

Then, the system must be able to infer that the query can be rewritten as $I(x,y) \coloneq S_1(y), S_2(x,c)$ since $R(a,y)$ unifies with $T(z,y)$ producing the *binding* $\{z=c\}$, and $S_2(x,z)$ unifies with $N(z,x)$ and becomes $S_2(x,c)$ once the binding is propagated. On the other hand, the query cannot be rewritten as $I(x,y) : -S_1(y), S_3(x,z)$ because $R(a,y)$ unifies with $T(z,y)$ with the binding $\{z=c\}$, $P(d,x,z)$ unifies with $N(z,x)$ with the binding $\{z=d\}$, and these bindings are non-unifiable since constants are assumed to denote unique objects.

We incorporate the subsumtion relation into McdSat by means of the dependency graph $G(D)$. Once the graph is built using the subsumption rules, its transitive closure is computed along with the bindings associated with each edge: edges generated by the transitive closure have labels that correspond to the union of the bindings along the edges that generate this edge (if the set of bindings is inconsistent, then the label is assigned the binding **false**). These labels are unique and well defined as $G(D)$ is assumed to be a forest of trees. Once the transitive closure $G(D)^*$ is computed, all edges with inconsistent labels can be dropped. The transitive closure is then used to extend the rules in the logical theory that permit the cover of relational symbols in the query with symbols in the views: a predicate P is allowed to cover a predicate R whenever there is an edge from P to R in $G(D)^*$, yet if this covering becomes active, the bindings associated with it become active as well.

Preferences In order to account for the preferences, we shall use the concepts literal-ranking functions and best-ranked models for propositonal logic. A literal ranking function r is a function that assign ranks (real numbers) to propositional literals. Given a literal-ranking function r, the rank $r(\omega)$ of a model ω is the aggregation of the ranks for each literal made true by the model: $r(\omega) = \sum_{\omega \models \ell} r(\ell)$. Thus, the models can be ordered by their rank and the best-ranked models are the models with minimum rank. Some model enumerators like c2d can be used to compute all the best-ranked models of a propositional theory. Likewise, Weighted-Max-SAT solvers such as MiniMaxSAT [16] can be used to find a best ranked model of a propositional theory.

For SSPs, we accomodate the preferences by using a suitably defined literal-ranking function r^* and by computing best-ranked models [12]. First, a new propositional variable is created for each relational symbol in the ontology and services signatures along with clauses that turn this proposition true whenever the corresponding symbols become active (true). Second, for each preference $\pi = \langle \varphi, c \rangle$, a new propositional symbol p_π is created along with the formula $p_\pi \Leftrightarrow \varphi$. Thus, p_π is true iff φ is satisfied in the model (rewriting). Finally, the literal-ranking function r^* is defined as $r^*(\neg p_\pi) = c$ for each such preference. Clearly, the rank of a model correspond to the sum of the costs associated with the preferences violated by the model, and thus a best-ranked model correspond to a rewriting of minimum regret.

3.1 System Architecture

We define an architecture for solving SSPs that is comprised of a Catalog of service descriptions, an ontology reasoner, the Encoder, the best model Finder, and the Decoder. Figure 1 depicts the overall architecture of the system. In this framework, an instance of SSP consists of an integration framework IS and a user request R made of a conjunctive query Q and a collection of preferences. The Catalog of the system is populated with the components of the integration system, i.e., the domain ontology including the subsumption rules, the services and the LAV mappings between them.

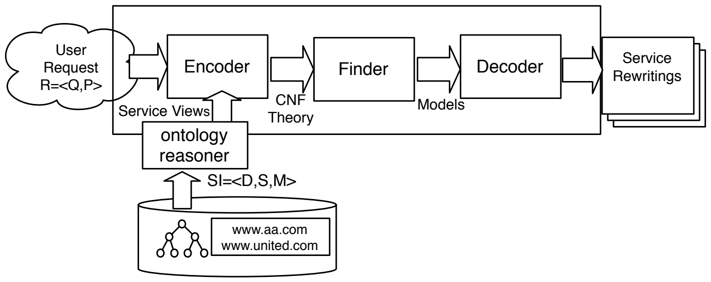

Fig. 1. System Architecture

The input instance of the SSP is then translated into a CNF theory and a literal-ranking function r^* by the Encoder module. The Encoder makes use of the transitive closure $G(D)^*$ that is calculated by the ontology reasoner together with the bindings associated with the edges. Once the theory is obtained, it is fed to a best-model finder. Once the finder return a best model, it is given to the Decoder that then re-construct the rewriting from the model.

4 Experimental Results

We conducted an empirical analysis on three benchmarks. All the experiments were run on a desktop machine with an Intel Core 2 Duo 2GHz CPU and 4Gb of memory, and the time was measured with the Unix time command. We used the compiler c2d, [2] to compile the CNF formula into d-DNNF[12]; c2d makes use of modern SAT techniques such as conflict-directed backtracking, clause learning and caching[11] to enumerate the theory models from the d-DNNF.

The objective of the experiment is to assess the performance of the proposal on varying conditions. The main benefit of our approach is that one can compile the logical theory for a problem instance and then calculate all the instantiations, or the best ones, any number of times. The cost model for finding best instantiations can be changed with no need to recompile the logical theory. Therefore, the time complexity of our approach is basically the time to encode the SSP as a CNF plus the time to find the (best) models and the time to decode the models. The times to encode and decode are negligible compared to the time to enumerate the models. Because of this, we focus on the time to enumerate the models of the benchmark problems. All the CNF theories are of low width.

The first benchmark consists of problems for air-travel queries. Service views are of the form $V_i(x,y) :- \mathit{flight}(x,y,\mathrm{AL}_i)$ where AL_i is a constant that denotes the name of an airline, and $\mathit{flight}(x,y,\mathrm{AL}_i)$ relates the cities x and y such that there is a flight between x and y served by AL_i. This service is assumed to return all flights between two cities with an specific airline. The user request has the

[2] `http://reasoning.cs.ucla.edu/c2d`

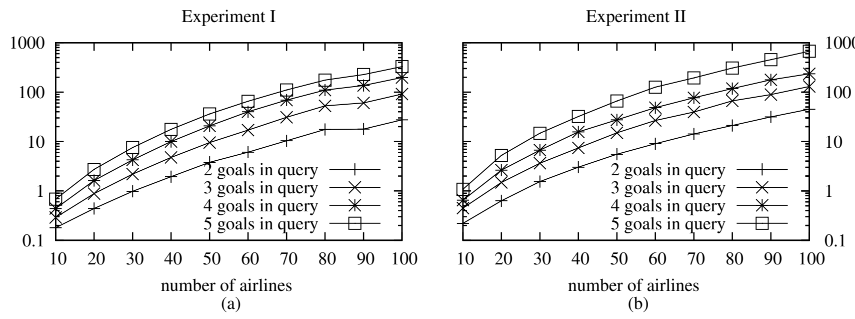

Fig. 2. Compilation times for experiments I and II for different number of goals and different number of views. The plots are in logarithmic scale, and the time is in seconds.

form

$$Q(x_1,\ldots,x_n)\ :-\ \mathit{flight}(\text{Paris},x_1,z),\ \mathit{flight}(x_1,x_2,z),\ \ldots,\ \mathit{flight}(x_n,\text{NY},z)\,.$$

The benchmark includes user requests with 2 to 5 sub-goals and sets of 10 to 100 service views. The results for the compilation are shown in panel (a) of Fig. 2. This is a plot in logarithmic scale that suggests a sub-exponential behavior. In any case, the results show good performance since realistic instances of the problem (sets of 100 airlines with 5-stop flights) can be compiled in 328 seconds. The size in disk of the d-DNNF for 100 airlines and 5-stop flights is 3.4Mb. On this d-DNNF, the best model can be computed in 0.29 seconds, and the enumeration of all models in 0.47 seconds.

In an attempt to induce an exponential growth in the compilation time, in the second benchmark we add a second service view for each airline. This modification increases the number of valid rewritings from linear to exponential since each leg of a flight can now be instantiated by two services and thus a flight with n legs may have up to 2^n rewritings. We ran the compiler for instances comprising the same number of user request sub-goals and total number of service views. The results plotted in logarithmic scale are shown in panel (b) of Fig. 2.

These tests show good performance for this type of problems, but they do not involve service views with multiple sub-goals. We therefore designed a third experiment that consists unstructured, randomly generated instances. Each instance contains three variables per domain ontology concept in the user request, ten distinct variables and ten distinct constants, six sub-goals in the user request, 2 to 5 sub-goals in the service views, and a varying number of service views. The chance that an argument of a domain ontology concept is bound to a constant is 50%. The results are depicted in panel (a) of Fig. 3. The compilation time for these instances does not grow monotonically since they are randomly generated. The same happens for the size of the theories and the number of models. For example, the d-DNNF for a problem with 45 views each with 5 sub-goals is of size 5.1Mb and has 1.26×10^8 models. The time to find the best model for this

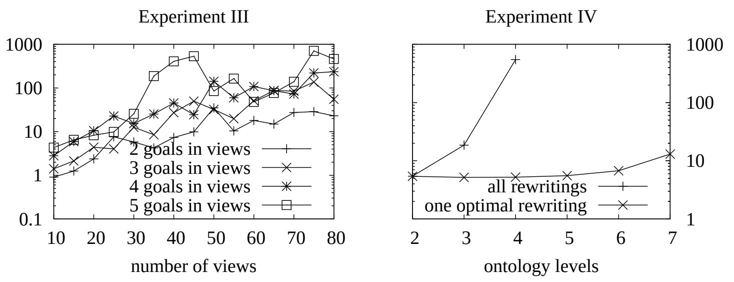

Fig. 3. Compilation times for experiment III for different number of goals and different number of views. Time for enumerating rewritings for experiment IV. The plots are in logarithmic scale, and the time is in seconds.

d-DNNF is 0.46 seconds while the time to enumerate all models is about 17 hours.

Finally, to study the effects of the ontology size, we considered the first benchmark and built ontologies where the predicates *flight*(x, y, z) and *uscity*(x) were leaves; ontologies were binary trees comprised of 2, 3, 4, 5, 6 and 7 levels. The user request has the form:

$$Q(x_1, x_2, x_3, x_4) \text{ :- } \mathit{uscity}(x_1),\ \mathit{trip}(x_1, x_2),\ \mathit{trip}(x_2, x_3),\ \mathit{trip}(x_3, x_4),\ \mathit{trip}(x_4, x_1).$$

where *trip*(x, y) was the root of the ontologies. The results are reported in panel (b) of Fig. 3. The time to enumerate all the rewritings suggests a sub-exponential behavior, while the time to find the best models does not grow monotonically. On the other hand, the size of the theories and the number of models grow exponentially. For example, the d-DNNF for this user request and the ontology with 7 levels is of size 1.7Mb and has 2.6×10^8 models. The time to enumerate all models is about 147 hours.

These are preliminary experiments, yet the results show that the proposed approach efficiently scales for problems with several goals and views. We believe that these results are encouraging and motivate us to continue this research. We plan to conduct additional experiments with other types and sizes of user requests and, when possible, compare with other approaches.

5 Related Work

The problem of selecting the services that satisfy a user request is a combinatorial optimization problem and several heuristics have been proposed to find a good solution in a reasonably short period of time [2, 7, 18–20, 23, 24, 26].

In a series of papers, Berardi and others [6–8] describe services and user requests in terms of deterministic finite-state machines that are encoded using Description Logic theories whose models correspond to solutions of the problem; properties of Description Logics reasoning methods are not exploited.

Ko et al. [18] propose a constraint-based approach that encodes the non-functional permissible values as a set of constraints whose violation needs to be minimized. Alrifai and Risse [2] develop a two-fold solution that uses a hybrid integer programming algorithm to find the decomposition of global QoS into local constraints, and then, selects the services that best meet the local constraints.

Recently, two planning-based approaches have been proposed. Kuter and Golbeck [19] extend the SHOP2 planning algorithm to select the trustworthy composition of services that implement a given OWL-S process model, while Sohrabi and McIlraith [24] propose a HTN planning-based solution where user preference metrics and domain regulations are used to guide the planner into the space of relevant compositions. Finally, Lécué [20] develops a genetic-based algorithm to identify the composition of services that best meet the quality criteria for a set of QoS parameters.

These existing solutions are able to scale up to relatively large number of abstract concepts. In addition to scalability, our approach provides a more expressive framework where services are semantically described in terms of domain ontology concepts, user preferences restrict the space of solutions, and ontology relationships augment the space of possible solutions. Finally, our approach is sound and complete.

6 Conclusions and Future Work

We have shown how the propositional theory used in McdSat for computing rewritings of queries can be extended to efficiently encode SSP; thus, by exploiting known properties of modern SAT solvers, we provide an efficient and scalable solution to SSP. In addition, we provide an expressive framework by defining LAV mappings to semantically describe services, and by expressing user requests as a conjunctive query over the domain ontology concepts and a set of preferences that restrict the SSP solutions.

The experimental results show that the approach can be applied to real-sized problems; however, if d-DNNF format is used, the approach is only possible when the compilation of the CNF theory into d-DNNF format succeeds. Thus, to overcome this limitation, in the future, we plan to study the performance and scalability of our approach in other off-the-self model enumeration and SAT solvers such as Relsat, Minisat and MiniMaxSAT. We also are interested in using other target compilation languages like Ordered Binary Decision Diagrams.

References

1. F. N. Afrati, C. Li, and J. D. Ullman. Using views to generate efficient evaluation plans for queries. *J. Comput. Syst. Sci.*, 73(5):703–724, 2007.
2. M. Alrifai and T. Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *WWW*, pages 881–890, 2009.
3. J. L. Ambite, S. Darbha, A. Goel, C. A. Knoblock, K. Lerman, R. Parundekar, and T. A. Russ. Automatically constructing semantic web services from online sources. In *International Semantic Web Conference*, pages 17–32, 2009.

4. Y. Arvelo, B. Bonet, and M.-E. Vidal. Compilation of query-rewriting problems into tractable fragments of propositional logic. In *AAAI*, 2006.
5. R. Bayardo. relsat: A Propositional Satisfiability Solver and Model Counter. http://code.google.com/p/relsat/.
6. D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic Composition of Transition-based Semantic Web Services with Messaging. In *VLDB*, 2005.
7. D. Berardi, F. Cheikh, G. D. Giacomo, and F. Patrizi. Automatic Service Composition via Simulation. *Int. J. Found. Comput. Sci*, 19(2):429–451, 2008.
8. D. Berardi, G. D. Giacomo, M. Mecella, and D. Calvanese. Composing Web Services with Nondeterministic Behavior. In *ICWS*, pages 909–912, 2006.
9. H. Chen, Z. Wu, and Y. Mao. Rewriting queries using views for rdf-based relational integration. In *ICTAI*, pages 260–264, 2005.
10. A. Darwiche. The c2d compiler. http://reasoning.cs.ucla.edu/c2d/.
11. A. Darwiche. New advances in compiling cnf into decomposable negation normal form. In *ECAI*, pages 328–332, 2004.
12. A. Darwiche and P. Marquis. Compiling propositional weighted bases. *Artif. Intell.*, 157(1-2):81–113, 2004.
13. O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *PODS*, pages 109–116, 1997.
14. O. M. Duschka and M. R. Genesereth. Query planning in infomaster. In *SAC*, pages 109–111, 1997.
15. N. Een and N. Sorensson. Minisat. http://minisat.se/.
16. F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSAT: An efficient Weighted Max-SAT Solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.
17. H. Jaudoin, J.-M. Petit, C. Rey, M. Schneider, and F. Toumani. Query rewriting using views in presence of value constraints. In *Description Logics*, 2005.
18. J. M. Ko, C. O. Kim, and I.-H. Kwon. Quality-of-Service Oriented Web Service Composition Algorithm and Planning Architecture. *Journal of Systems and Software*, 81(11):2079–2090, 2008.
19. U. Kuter and J. Golbeck. Semantic web service composition in social environments. In *International Semantic Web Conference*, pages 344–358, 2009.
20. F. Lécué. Optimizing qos-aware semantic web service composition. In *International Semantic Web Conference*, pages 375–391, 2009.
21. A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB*, pages 251–262, 1996.
22. R. Pottinger and A. Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB J.*, 10(2-3):182–198, 2001.
23. H. Rahmani, G. GhasemSani, and H. Abolhassani. Automatic Web Service Composition Considering User Non-functional Preferences. *Next Generation Web Services Practices*, 0:33–38, 2008.
24. S. Sohrabi and S. A. McIlraith. Optimizing web service composition while enforcing regulations. In *International Semantic Web Conference*, 2009.
25. J. D. Ullman. Information integration using logical views. *Theor. Comput. Sci.*, 239(2):189–210, 2000.
26. H. Wada, P. Champrasert, J. Suzuki, and K. Oba. Multiobjective Optimization of SLA-aware Service Composition. In *IEEE Congress on Services, Workshop on Methodologies for Non-functional Properties in Services Computing*, 2008.