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 large number of 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 are required to make these services widely available to the users. In this paper, we propose a framework that is grounded on logic and the Local-As-View approach for representing instances of the service selection problem. In our approach, Web services are semantically described using LAV mappings in terms of generic concepts from an ontology, user requests correspond to conjunctive queries on the generic concepts and, in addition, the user may specify a set of preferences that are used to rank the possible solutions to the given request. The LAV formulation allows us to cast the service selection problem as a query rewriting problem that must consider the relationships among the concepts in the ontology and the ranks induced by the preferences. Then, building on related work, we devise an encoding of the resulting query rewriting problem as a logical theory whose models are in correspondence with the solutions of the user request, and in presence of preferences, whose best models are in correspondence with the best-ranked solutions. Thus, by exploiting known properties of modern SAT solvers, we provide an efficient and scalable solution to the service selection problem. The approach provides the basis to represent a large number of real-world situations and interesting user requests.

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 a large dataset of Semantic Web Services become available, users require techniques to effectively select the services that meet their requirements. In order to achieve this goal, the services in the dataset must be tagged with their functional and non-functional properties, and the user preferences and requirements must be formally described as well. In this paper, we extend an approach traditionally used in the area of data integration

to solve the problem of selecting the best services that meet a user request, a problem that we call in this paper the Service Selection Problem (SSP).[1]

As in other approaches, we use domain ontologies for describing the services in the dataset, yet we differ in how the services are described. In this paper, we use the recent approach of Ambite et al. [3] that describes services as views on the generic concepts of the ontology following the Local-As-View (LAV) approach that is widely used in integration systems [21], instead of the traditional Global-As-View (GAV) approach where the generic concepts are expressed in terms of the services. The adoption of the LAV approach instead of GAV is not accidental. LAV descriptions are tailored towards systems with constantly changing datasets and a relatively stable set of generic concepts, while GAV descriptions are tailored towards systems with a constantly changing set of generic concepts but a relatively stable dataset of services.

As it is shown below, LAV descriptions of services correspond to mappings that define services as conjunctive queries involving the generic concepts in the ontology. Thus, every time that a service changes or a new one becomes available, only a tiny fraction of the mappings must be updated, usually just one mapping. Likewise, user requests can be modeled as conjunctive queries over the generic concepts in a way that the SSP can be cast as the problem of rewriting a query in terms of a set of views, the so-called Query Rewriting Problem (QRP) that is well-known in the area of data integration [8, 16], query optimization and data maintenance [1, 21], and for which several scalable approaches have been proposed [4, 12, 13, 21, 23]. Furthermore, user preferences and constraints on the possible solutions for a given request may be specified with a simple yet expressive language for preferences. These preferences and constraints refine and rank the set of valid rewritings of the posed query so that the best solution to the SSP corresponds to the best-ranked rewritings of the QRP.

Our solution extends the recent approach of Arvelo et al. [4] for QRPs that is based on the efficient enumeration of models for a propositional logic theory. In our case, an input instance of the SSP is converted into an instance of a QRP with preferences and constraints that is further translated into a (weighted) logical theory for which its models are in correspondence with the solutions of the SSP, and the rank of the models induced by the propositional weights corresponds to the rank of the solutions induced by the user preferences and constraints. These translations, from SSP to QRP to logic, are performed efficiently, in (low) polynomial time, and the best models are found using off-the-shelf SAT tools. Thus, we are able to exploit the benefits of modern SAT techniques such as conflict-directed backtracking and caching and decomposition of common subproblems to perform the necessary search in the combinatorial space of solutions.

In summary, we make the following crisp contributions to the problem of service selection and composition: (1) advocate the LAV approach as it provides an scalable solution for describing the continuously changing set of available Web Services, (2) propose a simple yet powerful language for expressing preferences

[1] We assume that a discovery service previously crawled the Web and located the services, and that an annotation tool stored their descriptions in our catalog.

and constraints on the valid solutions of the SSP, (3) describe how to transform the SSP to the QRP extended with preferences and constraints, and (4) describe how to change an efficient and scalable solution to the QRP, based on propositional logic and SAT tools, to handle preferences and constraints. The rest of this paper is as follows. The next two sections describe the SSP and the language of preferences and constraints, and the proposed solution to the SSP. Then, we present preliminary experiments, related work and finish with a discussion.

2 Service Selection Problem

An SSP consists of a description IS of the integration framework and a user request R. Formally, the integration framework is a tuple $IS = \langle D, S, M \rangle$ where D is the ontology of generic concepts, S is the set of available services, and M is the collection of LAV mappings that semantically describe the services in terms of the ontology. On the other hand, a user request is a tuple $R = \langle Q, P \rangle$ that is made of a query Q expressed as a conjunctive query over the generic concepts and a set of preferences P. In the following, we describe all these elements in detail and illustrate the framework through a number of 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 set of relational and constant symbols from which logical formulas can be constructed; it corresponds to 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,[2] and each c_j is a constant symbol. The axioms describe the ontology by defining the relationships between the ontology concepts. For the present work, we only consider subsumption relationships between concepts that are expressed with rules of the form:

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

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 (repetitions allowed). All these lists may be empty except $\bar{x} \cup \bar{b}$.

Although limited in appearance, subsumption rules are quite expressive as they allow us to specify diverse relationships between concepts; e.g.,

- **Hierarchy of classes and subclasses** (or types and subtypes): classes are specified with unary predicates. A subclass relationship can be specified with a simple rule; e.g., $\mathit{penguin}(x) \sqsubseteq \mathit{bird}(x)$ tells that penguins are birds.
- **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 the descendants of Queen Elizabeth II are noble.

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

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

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

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

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

2.2 Services and 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*, where each symbol S_i represents a concrete service in the Internet that "offers" some information.

The semantic description of services is expressed with the LAV paradigm in terms of mappings that describe the services in terms of concepts in the domain ontology [26]: for each service S_i, there is a mapping that describes S_i as a *conjunctive query* on the concepts in the ontology that also distinguish input and output attributes of the service. For example, a service $S(x, y)$ that returns information about flights originating at a given US city can be described as:

$$S(\$x, y) \text{ :- } \textit{flight}(x, y), \textit{uscity}(x)\,.$$

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 enforces the following:

- the service represented by S provides information in the form of tuples (x, y),
- the service is called with x as input attribute and returns (x, y),
- each tuple (x, y) returned by the service satisfies the rhs of the view; i.e., *flight*(x, y) and *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 S.

The LAV approach is commonly used in integration systems because it permits the scalability of the system as new services become available [26]. 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 [26]. 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. 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 describing the services in terms of the concepts in D. 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 a tuple $R = \langle Q, P\rangle$ where Q is a conjunctive query in terms of concepts in the ontology that describes how these concepts must be combined to resolve a given task, and a set P of preferences. For example, the query:

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

finds all cities on which a two-leg flight from Los Angeles to Paris stop. This query can be answered using the view $S(\$x, y)$ as $I(x) :- S(\mathrm{LA}, x), S(x, \mathrm{Paris})$. Observe that this rewriting is correct yet not *necessarily* complete because there may be two-leg flights from Los Angeles to Paris that stop at non-US cities (which are not available through the service $S(\$x, y)$) or because there may be a two-leg flight from Los Angeles to Paris that stops at a US city that is unknown to $S(\$x, y)$.

The preferences are used to rank the collection of valid rewritings. Once this ranking is obtained, the solution for the request R is any best-ranked valid rewriting. In this work, we consider a simple yet expressive language for preferences in which preferences are "soft constraints" on valid rewritings.

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 is associated with a cost equal to the sum of the costs of the preferences violated by the rewriting, and that these costs induce a ranking on valid rewritings. Thus, a best-ranked valid rewriting is one that has minimum cost.

It only remains to say what type of propositional formulas φ are allowed 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 or in the

services signature. 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 think on a different symbol altogether such as P_R, $[R]$, or other. The validity of a preference is defined with respect to the propositional model $\mathcal{M}(I)$ (truth assignment for the symbols in $\mathcal{L}(IS)$) constructed from a valid rewriting $I(\bar{x})$: $\mathcal{M}(I) \vDash S$ for $S \in \tau$ iff the service S appears in $I(\bar{x})$, and $\mathcal{M}(I) \vDash R$ for $R \in \sigma$ iff the concept R appears in the unique path from a concept R' to the root in the dependency graph where R' is a concept in a service $S(\bar{y})$ used in $I(\bar{x})$. 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 rewriting $I(x) :\!- S(\text{LA}, x), S(x, \text{Paris})$ defines the model $\mathcal{M}(I) = \{S = \textbf{true}, \mathit{flight} = \textbf{true}, \mathit{uscity} = \textbf{true}\}$. Finally, a preference φ holds in an answer $I(\bar{x})$ iff $\mathcal{M}(I) \vDash \varphi$.

This simple language permits us to express interesting preferences such as:

- **Hard constraints:** a soft constraint of the type $\pi = \langle \varphi, \infty \rangle$ can be thought as a hard constraint that must be satisfied by every rewriting because if the best rewriting has infinite cost, we know that there is no valid rewritig that satisfies φ.
- **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,
- **Conditional preferences:** a user 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 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 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

Consider a 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, flight^3, train^3 and uscity^1, and the constants AA, UA, AT, UP, LA, NY, and Paris. The first predicate relates cities (x, y) if there is a direct trip either by plane or train between them. 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*, and *uscity* indicates when a given city is a US city or not. The ontology axioms capture two subsumption relations:

$$\mathit{flight}(x, y, t) \sqsubseteq \mathit{trip}(x, y)\,,$$

$$\mathit{train}(x, y, t) \sqsubseteq \mathit{trip}(x, y)\,.$$

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,
- *AA-flight*(x, y) relates cities that are connected by American flights,
- *UA-flight*(x, y) relates cities that are connected by United flights,
- *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,
- *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,
- *AT-train*(x, y) relates cities that are connected by Amtrak trains, and
- *UP-train*(x, y) relates cities that are connected by Union Pacific Railway trains.

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

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

Observe that each tuple produced by each service satisfies the semantic description given in the body of the rule; e.g., the tuples that satisfy *national-flight*(x, y) 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 *national-flight*(x, y), i.e., this service is not necessarily complete.

Consider now a user who 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 with the conjunctive query:

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

Any rewriting of the ontology predicates in terms of the services that respect the input/output constraints on the parameters correspond to a *composition of services* that implements the request. For example, the following rewriting is a valid solution to the request:

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

But, the following two rewritings are not valid solutions:

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

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 SSP, since 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, users typically have preferences about using planes or trains. For this example, we study four different scenarios for user preferences and show how to model them in the proposed framework:

P1. The user prefers to fly 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 is obtained by assigning a high reward to *train*.

P2. The user is indifferent with respect to trains or airplanes, yet she 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, she prefers to always use the same airline. This preference can be modeled with the formula $\neg \textit{AA-flight} \vee \neg \textit{UA-flight}$ 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, she prefers to use UA. This is a non-trivial preference that can be modeled with the formula:

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

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 all but the first case involve preferences that can be treated as hard constraints if they are associated with infinite cost, or soft constraints meaning 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 QRP. An instance of QRP consists of a collection of views and a query on abstract concepts. The problem consists in rewriting the query in terms of the views such that each tuple produced by the rewriting is a tuple of the solution [26]. McdSat reduces QRP to the problem of finding the models of a propositional logic theory that 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 QRP instance. 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, or just one rewriting as done when sources are complete [26]. For the former, off-the-shelf model enumeration tools such as c2d [9] and Relsat [5] can be used, while off-the-shelf SAT solvers such as Minisat [14] or Rsat [22] 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, input and output attributes, the ontology of concepts with subsumption relationships, and user 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 McdSat does not provide support for constant symbols, yet incorporating this functionality is straightforward. Basically, one only has to track the unification of variables with constants using new propositional symbols, and to propagate such unifications transitively using implications in order to avoiding the unification of different constant symbols. This modification involves the addition of a small number of propositional symbols and clauses to the CNF generated by McdSat.

Input and Output Attributes The general principle for properly handling input and output attributes is that every input attribute of a service must unify either with a constant symbol or with an output attribute of another service, while avoiding the cycles in the dependencies among the services that 'produce' (output) and 'consume' (input) attributes.

This principle can be enforced by adding propositional symbols to the theory of the form $In(z, R)$ and $Out(z, R)$, where z is a variable and R is a service, and symbols $Prec(R, S)$ for each pair of services R and S. The intended interpretation for these symbols is that $In(z, R)$ holds iff attribute z is an input attribute of R, that $Out(z, R)$ holds iff attribute z is an output attribute of R, and that $Prec(R, S)$ holds iff the service R produces an attribute that is consumed by S. Accordingly, the theory is extended with clauses that enforce this interpretation plus rules of the form $Prec(R, S) \wedge Prec(S, T) \Rightarrow Prec(R, T)$ that propagate the precedence relation via transitivity, and $\neg Prec(S, S)$ that prunes rewritings containing cycles.

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

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

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

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

Then, the system must be able to infer that the query can be rewritten as $I(x, y) :\!- S_1(y), S_2(x, c)$ 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 system must also infer that $Q(x, y)$ cannot be rewritten as $I(x, y) :\!- S_1(y), S_3(x, z)$ because $R(a, y)$ unifies with $T(z, y)$ with binding $\{z = c\}$, $P(d, x, z)$ unifies with $N(z, x)$ with binding $\{z = d\}$, and these two bindings are non-unifiable since constants denote unique objects.

We incorporate the subsumption 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 $G(D)^*$, and when this covering becomes active, the bindings associated with it become active as well.

Preferences To incorporate preferences, we use the concepts of literal-ranking function and best-ranked models for propositional logic. A literal ranking function r is a function that assign ranks (weights) to 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; i.e., $r(\omega) = \sum_{\omega \models \ell} r(\ell)$ [11]. 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 [15] can be used to find a best ranked model.

For SSPs, we accommodate the preferences by using a suitably defined literal-ranking function r^* and by computing best-ranked models. 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 symbol 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 corresponds to the sum of the costs associated with the preferences violated by the model, and thus a best-ranked model corresponds 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. 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.

The input instance 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 the Finder that returns a best model. The model is given to the Decoder that reconstructs the solution to the input instance.

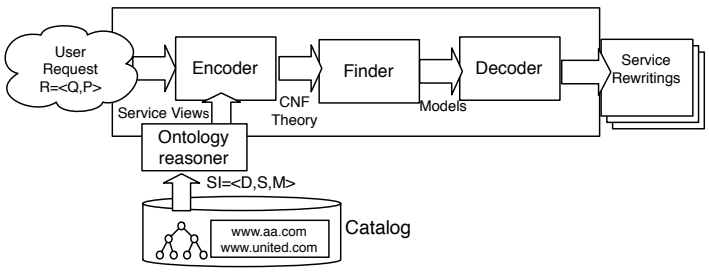

Fig. 1. System Architecture

4 Preliminary Experiments

We have developed a system prototype that implements the above ideas except for the support to distinguish input and output attributes of services (all attributes are assumed to be output), and with partial support for handling preferences; a complete implementation is ongoing work. With this prototype, we conducted experiments on two type of domains: airline domains of the type seen before and random domains. The Finder module is built using c2d (`http://reasoning.cs.ucla.edu/c2d`) that compiles (transforms) the CNF formula for the propositional theory into deterministic and decomposable negation normal form (d-DNNF) from which all models or just the best models can be efficiently enumerated in linear time [10]. The compilation process from CNF into d-DNNF is intractable in the worst case, yet this is not always the case as the experiments below show.

The objective of the experiments is to test several features of the approach and to see the scalability of the approach. The main benefit of the approach is that one can compile the logical theory for a problem instance and then calculate all the rewritings, or the best ones, any number of times, and the cost/rewards associated with the preferences can also be changed without the need to recompile the theory. Therefore, the time complexity of our approach is basically the time to compile the CNF theory into d-DNNF since calculating the CNF from the SSP and decoding the models is negligible. Thus, we only report the time to compile the CNF into d-DNNF.

4.1 Airline Experiments

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 the view is assumed to return flights between two cities served by the airline AL_i. For the query, we consider a request to find trips between Paris and New York with a number of stops. The query returns the stops and has the form:

$$Q(x_1, \ldots, x_n) \ :\!-\ \mathit{flight}(\mathrm{Paris}, x_1, t),\ \mathit{flight}(x_1, x_2, t),\ \ldots,\ \mathit{flight}(x_n, \mathrm{NY}, t)\,.$$

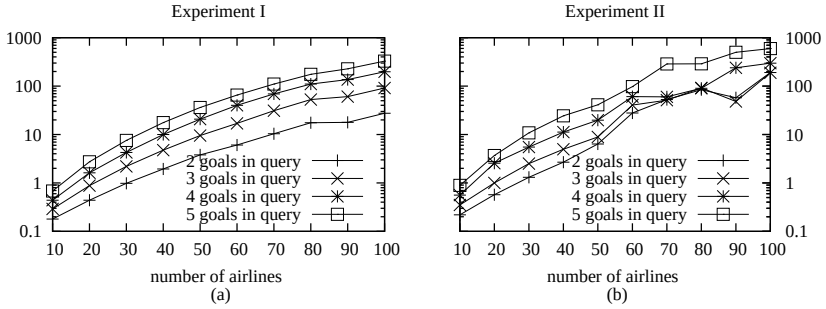

Fig. 2. Compilation times for experiments I and II for different number of goals and different number of views. Experiment II involves $n(n-1)/2$ preferences of a problem with n views. The plots are in logarithmic scale, and the time is in seconds.

Observe that the existentially quantified variable, t, is the same for each flight meaning that it can only be unified with the same constant; i.e., each leg of the flight is served by the same airline. We solved several instances for this type of query with a number of stops from 2 to 5 and a number of services from 10 to 100. Fig. 2(a) shows the results of the compilations: the vertical axis refers to the time in seconds in logarithmic scale and the horizontal axis to the number of views in the benchmark. 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 from which the best model can be computed in 0.29 seconds, and the enumeration of all models can be done in 0.47 seconds.

In the second experiment, we test our system with user preferences. The query is the same except that the existentially quantified variables are all distinct for each flight meaning that any combination of airlines can fulfill the user request. As user preferences, we consider the set $\{\neg V_i \vee \neg V_j : 1 \leq i \neq j \leq n\}$ of $n(n-1)/2$ constraints each with cost $c_{i,j}$, for a problem with n services. Thus, a best model is one that violates the minimum number of preferences and this is equivalent to using the same airline for each leg of the flight. Fig.2(b) shows the result for the compilation also in logarithmic scale. As it can be seen, the compilation times are very similar for the Experiment I where there are no preferences. The largest instance is a complex problem involving 100 views, 5 subgoals in the query and 4,950 user preferences; the total number of rewritings is 100^5, yet it can be compiled in 600 seconds.

In the third experiment, we test the system with ontologies of different sizes. Ontologies corresponding to full binary trees of depth 2 to 7 were generated with the predicate $trip(x, y)$ at the root. Then, for each node in the tree, there is a view that is described by the predicate at that node. The user request is:

$$Q(x_1, \ldots, x_4) \ :- \ uscity(x_1),\ trip(x_1, x_2),\ trip(x_2, x_3),\ trip(x_3, x_4),\ trip(x_4, x_1).$$

In all cases, the compilation time was always less than 13 seconds.

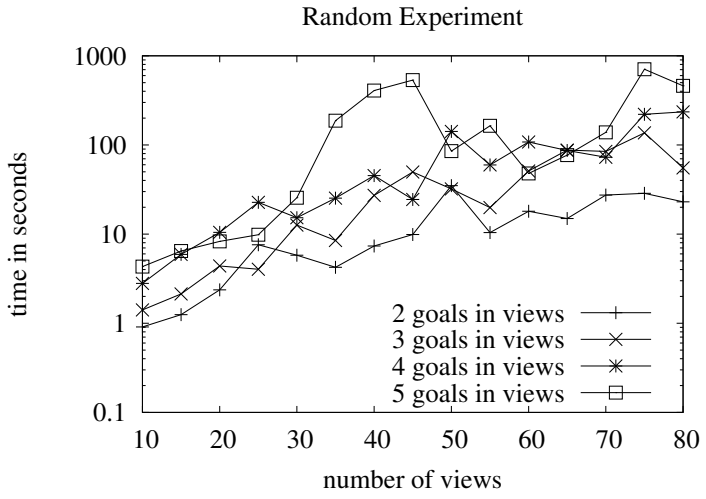

Fig. 3. Compilation times for random experiments with different number of views. The plots are in logarithmic scale and the time is in seconds.

4.2 Random Experiments

For the last experiments, we generated random unstructured instances of SSPs as follows: each user request contains 6 subgoals, 10 distinct variables and 10 distinct constant symbols, while each service view contains between 2 and 5 subgoals. The constants were randomly placed on the subgoals arguments with a 50% probability. Fig. 3 shows the compilation time for these instances. The size of the compiled theories and number of models does not increase monotonically with the number of views given the random nature of the instances. As it can be seen, these are complex instances and the approach is able to solve them in reasonable time.

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 period of time [2, 6, 17–20, 24, 25, 27].

In a series of papers, Berardi and others [6, 7] describe services and user requests in terms of deterministic finite-state machines that are encoded using Description Logics theories whose models correspond to solutions of the problem, yet there are no efficient methods to compute these models as in the case of SAT.

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 in-

teger 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 [25] 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 scale up to a 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 in the sense that every solution produced by the system is a valid solution, that every valid solution can be produced by the system, and that the best-ranked valid solution is the best solution in terms of the user preferences.

6 Discussion

We proposed a novel formalism for expressing Service Selection Problems involving an ontology of generic concepts, services described using views in terms of the concepts, following the LAV approach, and user preferences. This is a general, well-defined and scalable framework since it is based on logic, the LAV approach, and permits the modeling of real-life scenarios and preferences.

We also showed how the propositional theory used in McdSat for solving the QRP can be extended to handle SSPs. This formulation allows us to exploit the properties of modern SAT solvers to provide an efficient and scalable solution.

The preliminary experiments show that the approach can be applied to real-sized problems. We are currently working on a complete implementation of the formalism in order to offer its full expressiveness. In the future, we plan to use other off-the-shelf SAT tools such as MiniMaxSat that is able to find a best model without the need to compile the CNF into d-DNNF.

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 *ISWC*, 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, F. Cheikh, G. D. Giacomo, and F. Patrizi. Automatic Service Composition via Simulation. *Int. J. Found. Comput. Sci*, 19(2):429–451, 2008.
7. D. Berardi, G. D. Giacomo, M. Mecella, and D. Calvanese. Composing Web Services with Nondeterministic Behavior. In *ICWS*, pages 909–912, 2006.
8. H. Chen, Z. Wu, and Y. Mao. Rewriting queries using views for rdf-based relational integration. In *ICTAI*, pages 260–264, 2005.
9. A. Darwiche. The c2d compiler. http://reasoning.cs.ucla.edu/c2d/.
10. A. Darwiche. New advances in compiling cnf into decomposable negation normal form. In *ECAI*, pages 328–332, 2004.
11. A. Darwiche and P. Marquis. Compiling propositional weighted bases. *Artif. Intell.*, 157(1-2):81–113, 2004.
12. O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *PODS*, pages 109–116, 1997.
13. O. M. Duschka and M. R. Genesereth. Query planning in infomaster. In *SAC*, pages 109–111, 1997.
14. N. Een and N. Sorensson. Minisat. http://minisat.se/.
15. F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSAT: An efficient Weighted Max-SAT Solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.
16. 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.
17. M. Junghans, S. Agarwal, and R. Studer. Towards practical semantic web service discovery. In *ESWC (2)*, pages 15–29, 2010.
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. K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Proc. of 10th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT)*, pages 294–299, 2007.
23. R. Pottinger and A. Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB J.*, 10(2-3):182–198, 2001.
24. 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.
25. S. Sohrabi and S. A. McIlraith. Optimizing web service composition while enforcing regulations. In *International Semantic Web Conference*, 2009.
26. J. D. Ullman. Information integration using logical views. *Theor. Comput. Sci.*, 239(2):189–210, 2000.
27. 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.