Preference-Based Web Service Composition: A Middle Ground between Execution and Search

Shirin Sohrabi and Sheila A. McIlraith

Department of Computer Science, University of Toronto, Toronto, Canada
{shirin,sheila}@cs.toronto.edu

Abstract. Much of the research on automated Web Service Composition (WSC) relates it to an AI planning task, where the composition is primarily done offline prior to execution. Recent research on WSC has argued convincingly for the importance of optimizing quality of service, trust, and user preferences. While some of this optimization can be done offline, many interesting and useful optimizations are data-dependent, and must be done following execution of at least some information-gathering services. In this paper, we examine this class of WSC problems, attempting to balance the trade-off between offline composition and online information gathering with a view to producing high-quality compositions efficiently and without excessive data gathering. Our investigation is performed in the context of the semantic web employing an existing preference-based Hierarchical Task Network WSC system. Our experiments illustrate the potential improvement in both the quality and speed of composition generation afforded by our approach.

1 Introduction

Web Service Composition (WSC) requires a computer program to automatically select, integrate, and invoke multiple web services in order to achieve a user-defined objective. It is an example of the more general task of composing business processes or component software. Automated WSC is motivated by the need to improve the efficiency of composing and integrating services. A number of Business Process Management (BPM) systems exist to help organizations optimize business performance by discovering, managing, composing, and integrating business processes, including SAP's NetWeaver, and IBM's WebSphere and BPM Suite. With the advent of cloud computing, an increasing number of small- and medium-sized businesses are attempting to blend cloud services from multiple providers. Performing such integration and interoperation manually is costly and time consuming. Automated WSC and semantic integration address this emerging challenge [14]. For the purposes of this paper, we illustrate concepts in terms of the intuitive but over-used travel domain, however compelling examples exist in sectors such as Banking and Finance, Government, Healthcare and Life Sciences, Insurance, Retail, and Supply Chain Management. Many of these applications exploit extensive internet- or intranet-accessible data and will directly benefit from the work described here.

P.F. Patel-Schneider et al. (Eds.): ISWC 2010, Part I, LNCS 6496, pp. 713–729, 2010.

A popular approach to WSC is to characterize it as an Artificial Intelligence (AI) planning task and to solve it as such (e.g., [13,15,4]). In previous work (e.g., [15,20,19]) we have argued that for a number of WSC problems it is desirable to specify a **flexible workflow**, generic procedure, or composition template that specifies the basic steps of the composition at an abstract level, but has sufficient flexibility to support their customization for different stakeholders, scenarios, and applications. To this end, we have specified flexible workflows using Golog (e.g., [15,20]), or alternatively Hierarchical Task Networks (HTNs) [19], and developed associated machinery for WSC. We are not alone is proposing such a vision. Others have similarly used HTNs (e.g., [17]) and finite state automata (e.g., [5]) to specify composition objectives with varying flexibility.

While customization of flexible workflows can take the form of hard constraints imposed by the specific application scenario and its stakeholders, in cases where such customizing constraints are conflicting, some form of prioritization is required. Similarly, in cases where customizations are desirable but not mandatory, customizations can be specified as preferences. This observation has led us to characterize the WSC task as a preference-based planning (PBP) task where actions (services, service parameters, and/or data) are selected not only to achieve the composition objective but to produce compositions that are of high quality with respect to quality of service, trust, or other composition-, service-, or data-oriented user preferences (e.g., [20,11,19,1,10,21]).

Previous work on preference-based WSC (and indeed much of the work on WSC without preferences) has assumed that all the information required to generate the composition is on hand at the outset, and as such, composition is done offline followed by subsequent execution of the composition, perhaps in association with execution monitoring. However, this is not realistic in many settings. Consider the task of travel planning or any other multi-step purchasing process on the Web. A good part of the composition for these domains involves data gathering, followed by generation of an optimized composition with respect to that data and other criteria. Indeed many of the choice points relating to the composition require data acquired at execution time.

To address this, most current WSC systems will acquire all the information required for the composition prior to initiating composition generation. This can result in a lot of unnecessary data access. Further, it results in an enormous search space for a planner. Most state-of-the-art planners require actions to be grounded. However, unlike typical planning applications, many WSC applications are data-intensive, which results in an enormous number of ground actions and a huge search space. While this space may still be manageable for computing *a* composition, to compute *an optimal* composition, and to guarantee optimality, the entire search space must be searched, at least implicitly. This has the effect that most data-intensive WSC tasks that involve optimization of data (like picking preferred flights) will not scale using conventional PBP techniques.

Consider a flexible workflow that describes the travel domain in terms of the tasks of booking transportation and booking accommodations, with varying options for their realization. We add to this the following preferences: *If destination*

is more than 500 km away, book a flight, otherwise I prefer to rent a car; I prefer to fly with a Star Alliance carrier; I prefer to book cars with Avis, and if not Budget; I prefer to book a Hilton hotel, and if not a Sheraton. A naive PBP would access all the flight, car, hotel, etc. information prior to composition and create grounded actions (e.g., *book-car*(*Avis,Pria,Daily,$39,...*)) for each data instance, resulting in a huge set of actions. In order to guarantee optimality of a composition, one needs to guarantee that all compositions were considered, which would (naively) involve considering all combinations of flight-hotel and/or car-hotel. However, there is clearly a smarter way to do this. In particular, either flight information or car rental information (but not both) need to be considered, depending on the distance to destination. Further, the choice of airline is independent of the choice of hotel, so optimality can be guaranteed by optimizing these choices independently. These simple, intuitive observations provide motivation for the work presented here.

In this paper, we investigate the class of WSC problems that endeavour to generate high-quality compositions through optimization of service and data selection. We attempt to balance the trade-off between offline composition and online information gathering with a view to producing high-quality compositions. Our objective is to minimize data access and to make optimization as efficient as possible by exploiting the independence of ground actions within the search space. Finally we wish to ensure that our techniques will maintain the guarantees a more naive approach would afford, including guarantees regarding the soundness of our compositions and their optimality.

Our investigation is performed in the context of our existing preference-based HTN WSC system, HTNWSC-P [19]. We propose a means of analyzing a WSC problem in order to identify places where optimization can be localized while preserving global optimality. Further, building on previous work that addresses the problem of information gathering (e.g., [15,9]), we propose a middle-ground execution engine that executes information-gathering services, as needed, while only *simulating* the execution of world-altering services. In doing so, the HTN WSC engine is able to benefit from the further knowledge afforded by information-gathering while still supporting backtrack search, by not actually or not necessarily executing world-altering services. We illustrate the effectiveness of our approach through experimentation.

2 Background and Preliminaries

The setting for this work is the semantic web. We assume that both the Web services and our composition template are described in OWL-S, an ontology for describing Web services [12]. We use an OWL-S to HTN translator to translate the OWL-S process descriptions and composition template to an HTN domain description and initial task network, respectively. Customization of the composition template is specified in PDDL3, the Planning Domain Definition Language, which provides a means of specifying preferences for planning domains [6]. Web service compositions now take the form of plans, and optimized compositions

take the form of optimized PBPs. In order to compute such PBPs, we exploit our previous work [18], which uses state-of-the-art heuristic search techniques to generate optimized PBPs from HTN specifications. Note throughout this paper we distinguish between information-gathering actions – actions that collect data, and world-altering actions – actions that effect change in the world.

HTN Planning: Hierarchical Task Network (HTN) planning [7] is a popular and widely used planning paradigm that has been employed for WSC (e.g., [17,11]). Given an initial state, an initial task network (the objective to be achieved), and a domain description comprising a set of operators and *methods* – a description of how tasks can be decomposed, an HTN planner constructs a plan by repeatedly decomposing tasks into smaller and smaller subtasks until a primitive decomposition of the initial task network is found. In the travel domain, the initial task network is the single task *arrange-travel*. This task can be decomposed into arranging transportation, accommodations, local transportation, activities, tours, and entertainment. Basic definitions are taken from [7].

Definition 1 (HTN Planning Problem). *An HTN planning problem is a 3-tuple* $\mathcal{P} = (s_0, w_0, D)$ *where* s_0 *is the initial state,* w_0 *is the initial task network, and* D *is the HTN planning domain which consists of a set of operators and methods.*

An operator is a primitive action, described by its name, preconditions and effects. In the travel domain, ignoring the parameters, operators might include: *book-hotel* and *book-flight*. A *task* consists of a task symbol and a list of arguments. A task is primitive if its task symbol is an operator name and its parameters match, otherwise it is *nonprimitive*. *arrange-transportation* and *arrange-activity* are nonprimitive tasks, while *book-tour* and *book-car* are primitive.

A method, m, is a 4-tuple (*name(m), task(m),subtasks(m), constr(m)*) corresponding to the method's name, a nonprimitive task and the method's task network, comprising subtasks and constraints. Method m is relevant for a task t if there is a substitution σ such that $\sigma(t) =$*task*(m). Several methods can be relevant to a particular nonprimitive task t, leading to different decompositions of t. In our example, the method with *name by-air-trans* can be used to decompose the *task arrange-trans* into the *subtasks* of booking a flight and paying, with the constraint (*constr*) that the booking precede payment.

Definition 2 (Task Network). *A task network is a pair w=(U, C) where U is a set of task nodes and C is a set of constraints. The constraints normally considered are of type precedence constraint, before-constraint, after-constraint or between-constraint.*

Definition 3 (Plan). $\pi = o_1 o_2 \dots o_k$ *is a plan for HTN planning program* $\mathcal{P} = (s_0, w_0, D)$ *if there is a primitive decomposition,* w*, of* w_0 *of which* π *is an instance.*

Specifying User Preferences and Constraints: Customizing preferences and constraints are specified in a version of PDDL3 that we have augmented

to express preferences over how HTN tasks are parameterized and decomposed as well as preferences over service (i.e., task) properties [19,18]. This allows us to combine optimization of service selection (such as quality of service) with optimization of the composition. This augmented version of PDDL3 supports specification of temporally extended preferences via a subset of Linear Temporal Logic (LTL). *always*, *sometime*, *sometime-before* are among the supported constructs. **occ**(a) refers to the occurrence of a primitive task, while **initiate**(x) and **terminate**(x) refer to the initiation and termination of a nonprimitive task or method. To specify preferences over non-functional properties of services such as trust, reliability, and reputation, we associate a unique id with each task via the predicate *isAssociatedWith* and augment the domain with additional predicates for these properties The constructs described above are used to describe desirable properties of plans. These properties (called preferences) are then aggregated together into an objective function. Some simplified examples follow.

```
(preference p1  (sometime (initiate (book-flight AirCanada Eco Direct))))
(preference p2  (always (not (occ (pay MasterCard)))))
(preference p3  (imply (hasBookedCar ?Z) (sometime (occ (pay ?Z AE)))))
```

p1 states that at some point the user books a direct economy flight with Air Canada, **p2** states that the user never pays by Mastercard, and **p3** states that if a car is booked, at some point the user pays with their American Express (AE).

The quality of a plan is measured by the value of a PDDL3 **metric function** – an objective function over preferences that can either be maximized or minimized. The PDDL3 function `is-violated` takes as input a preference name and returns the *number of times* the corresponding preference is violated. The example metric function below stipulates that it is to be minimized. As such, the lower its value, the higher the quality of the plan. The violation of individual preferences can be weighted to reflect their relative importance. E.g.,

```
(:metric minimize (+ (* 2 (is-violated p1))   (* 1 (is-violated p2))))
```

specifies that it is twice as important to satisfy **p1** as to satisfy preference **p2**. Note that since the metric function is a weighted sum of individual preference formulae, by trying to minimize its value, it automatically deals with inconsistent preferences. Hence, an appropriate trade-off between inconsistent preferences is made so that the metric function can be optimized.

Definition 4 (Preference-based HTN Planning). *An HTN planning problem with user preferences is described as a 4-tuple* $\mathcal{P} = (s_0, w_0, D, \preceq)$ *where* $\preceq$ *is a preorder between plans. A plan* π *is a solution to* $\mathcal{P}$ *if and only if:* π *is a plan for* $\mathcal{P}' = (s_0, w_0, D)$ *and there does not exist a plan* π' *for* $\mathcal{P}'$ *such that* π' *is more preferred than* π*.*

The $\preceq$ relation can be defined in many ways (e.g., $\preceq$ can be quantitatively defined using a metric function). Note, from now on we will refer to the metric function as M, and use $M(N)$ to denote the value of the metric in a search node N (a search node contains the current state, task network, and partial plan).

3 Decoupling Data Optimization from Search

Given the HTN domain description of a WSC problem, the initial task network, and the customizing constraints and preferences, we are interested in generating a high-quality (ideally optimal) composition. Unfortunately, unlike the task of generating a composition, its optimization requires considering all alternative compositions, at least implicitly. And even in the case where the composition can be decomposed into independent subproblems, customizing preferences and constraints over the composition can introduce new inter-dependencies.

In previous work [20,19] we proposed an algorithm based on planning with heuristic search that employs a best-first, forward search strategy capable of computing an optimal composition. We elaborate on the algorithm in Section 5. Here we consider how to exploit this algorithm in data-intensive settings where the search space can be prohibitively large.

As noted earlier, data acquired via information gathering is typically encoded as parameters of the actions that act on that data. E.g., the *book-flight* action would be parameterized by the data associated with a flight, such as airline, origin, destination, fare class, etc. State-of-the-art planning algorithms require actions/operators to be grounded. As such, in data-intensive settings, there can be an enormous number of ground actions and as a consequence an enormous search space to explore. Consider a simplified version of the task of booking a flight, a hotel, a car, and booking a tour for a vacation. Assume that these four tasks can be performed in any order and are completely independent of each other. Given 20 possible flights, 10 hotels, 10 types of car, 5 tours of the city, and 4! ways in which the booking of these items can be performed, there are 20*10*10*5*4! different compositions that need to be explored (at least implicitly) to determine the optimal composition. Using the algorithm proposed in our previous work, some of these combinations will be eliminated by our exploitation of state-of-the-art heuristic search and sound pruning – a means of pruning partial plans that have no prospect of producing a plan that is superior to the current best plan. Nevertheless, the algorithm is still doing a lot of unnecessary search.

From our experience with WSC applications that involve preferences, we observe that most of the search time is spent on resolving the optimization that relates to the data that we have collected. We henceforth refer to this type of optimization as ***data optimization***. We observe that just as the subtasks afford a degree of independence in many WSC scenarios, so too do the different data choices, and that this independence allows us to perform some optimization *locally*, external to the composition process, or even arbitrarily (if they don't matter) while still guaranteeing that the choice does not eliminate the globally optimal solution. For example, in our simplified scenario we can select the best car, best flight, best hotel, and best tour independently of each other. And in doing so, we can reduce the search space to (20+10+10+5)*4!. More generally, if we are able to identify that subset of the data that is relevant to the optimization of the composition and attempt to localize its optimization then we can significantly streamline our search.

In what follows, we elaborate on the exploitation of three scenarios: (1) a data choice must be done in concert with the composition but choosing the optimal data can be localized; (2) a data choice can be optimized in isolation of the composition generation process; and (3) a data choice is irrelevant to the optimization of the composition and can be made arbitrarily. We begin by defining the notion of localized data optimization and identify conditions under which it retains the possibility of finding the optimal solution.

Definition 5 (Localized Data Optimization with respect to an Operator). *Let $\mathcal{P}'$ be an information-gathering HTN planning problem with preferences, following Definitions 4 and 8. Let N be a search node that represents a partial plan, and let O be the world-altering operator that is to be applied next in our search – the operator that extends the partial plan currently under consideration. Let $N_1 ... N_k$ be different nodes that result from different possible groundings of O from node N. Localized data optimization for O selects node N_i, $1 \leq i \leq k$ if $M(N_i) \leq M(N_j)$, $\forall 1 \leq j \leq k$, where $M(N)$ is the metric value of search node N.*

According to the above definition, the node with the least metric value is selected when localized data optimization for an operator is performed. The question is when is such a strategy sound, i.e., when can we do such a local selection without eliminating the overall best solution? For example, assume a best flight among all available flights is selected, but the selected flight arrives at night preventing the planner from booking an activity for that day. In such situations, even though the selected flight is the best flight choice among all available flights in isolation (or locally), because of the interactions among operators within and between tasks, this choice is not the best choice for the composition.

Definition 6 (Sound Localized Data Optimization with respect to an Operator). *Let $\mathcal{P}'$, N, O, $N_1 ... N_k$ be as in Definition 5. Localized data optimization with respect to O is said to be sound if there does not exist a plan extending any node N_j, $1 \leq j \leq k$ that would result in a better metric value than any plan extending the node N_i that is selected via localized data optimization. Hence, if there exists an optimal plan π from extending the partial plan in node N, π is not achievable from extending any of the nodes N_j and is only achievable from N_i.*

This definition has important implications. If localized data optimization is sound, then all nodes N_j can be pruned from the search space because we know the optimal plan cannot be reached by extending any of these nodes. Now that we know the condition under which localized data optimization is sound, we need to discuss how such an operator can be identified. Doing so involves analyzing the structure of the planning problem to identify operators that are completely independent and have no interactions with the rest of the planning problem including (1) the operators and methods in the domain, (2) the user preferences, and (3) the hard constraints, assuming for simplicity that there are no indirect effects that we have to worry about. The following is a syntactic criterion that

can be used to identify operators whose grounding choices will have no impact on the rest of the decisions made during the generation of a composition.

Definition 7 (Non-interacting Operator with respect to the Domain). *An operator O is said to be non-interacting with respect to the domain if (1) no predicate in the precondition of O or in the condition of the conditional effect statement of O appears in the effect of any other operator in the domain, and (2) there is no predicate in the effect of O that appears in the precondition (or in the condition of the conditional effect statement) of any other operators or methods*[1] *of the planning problem.*

Intuitively this definition says that nothing affects the execution or outcome of this operator. Returning to our example, if the flight booking operator changes anything that is a precondition of another operator, then the flight booking operator interacts with that operator. E.g., if the flight booking operator has the effect of depleting available monetary funds, precluding the booking of a particular hotel, or if it results in arrival at a time that impacts the booking of a tour, then it is considered to interact with other aspects of the problem.

The above condition can be easily checked as a preprocessing step by analyzing the domain definition. However, syntactically identifying how preferences play a role in data interactions is more difficult, particularly when trajectory preferences – preferences expressed in a subset of LTL – are involved. One way to identify interacting operators with respect to the preferences, is to determine whether the operator's add effects – the positive effects of an operator – appear in any preference formulae. More specifically to enforce non-interaction, we need to ensure that the add effects of the operator never appear in the "*b* part" of preference formulae, where the "*b* part" is as follows: (*sometime-after* b a) (*always* (*imply* b a)) or (*sometime* (*imply* b a)). This is because the "*b* part" is the condition that if true requires the preference formula to be true, and in particular necessitates the "*a* part" holding. Thus, if the "*b* part" refers to an add effect of a world-altering operator for which localized data optimization is performed, and the "*a* part" is hard or impossible to achieve then the choice made in the data optimization interacts with a choice that has to be made later.

Theorem 1 (Criterion for Sound Localized Data Optimization). *If an operator O is non-interacting with respect to the planning domain, user preferences, and hard constraints then performing localized data optimization on this operator is sound.*

To this point we have defined the notion of localized data optimization and identified some syntactic criteria that will ensure its soundness. Before concluding, we informally discuss two further cases. We observe that in some instances the optimization of data can be completely separated or decoupled from the dynamics of the composition problem and the optimal data choice can be determined as a separate process. For example, if a user's sole preference is to book the

[1] Precondition for a method can be specified as a before constraint.

cheapest car, then the identification of what car to book can be performed in isolation of the generation of the composition altogether. Further, some data choices have no effect at all on the quality of the composition and as such can be made arbitrarily. For example, if the user does not care what car they rent, then the choice of rental car can be made arbitrarily. In both of these cases, the search space can exclude consideration of the different data values by insertion of a single placeholder value. Execution of the information-gathering service can be delayed until after composition, and the placeholder resolved at that time.

4 Middle-Ground Execution

For many WSC problems it is impractical, and often impossible to reduce the WSC problem to a planning problem with complete initial state – i.e., for which all the information necessary to generate a composition (and in our case to optimize it) is known prior to commencement of the search for a composition. In the travel domain this would necessitate collecting data relating to all the different modes of transportation, means of accommodation, etc. The space of ground actions would be enormous and the planning and optimization task unsolvable. However, one can instead imagine gathering information as it becomes necessary to choice points in the generation and optimization of the composition, and using this to inform the search for different compositions. In this section, we investigate how to perform information gathering in this manner.

The problem of gathering information during composition has been examined in several research papers (e.g., [15,17,8]). McIlraith and Son in [15] describe a middle-ground interpreter that collects relevant information, but only simulates the effects of world-altering actions. Their interpreter works under the **Invocation and Reasonable Persistence (IRP) Assumption** that (1) assumes all information gathering actions can be executed by the middle-ground interpreter and (2) assumes that the gathered information persists for a reasonable period of time, and none of the actions in the composition cause this assumption to be violated. Kuter et al. in [8] take a similar approach but their work focuses on dealing with services that do not return a result (if any) immediately. They provide a Query Manager that allows the planner to continue search without waiting for all of the information-gathering services to return data. They also assume that the information-gathering services are executable (similar to condition 1 of IRP) but they allow the planner itself to change the gathered information during planning (a variant of condition 2 of IRP). More recently, Au et al [2] proposed an approach to relaxing the IRP assumption, however their approach does not seem amenable to generating optimized compositions.

Our translation builds on the work by Sirin et al. [17]. We encode each OWL-S atomic process as an HTN operator and each OWL-S composite process as an HTN method. Similarly, we assume that all atomic processes are either information gathering or world altering and distinguish our set of planning operators accordingly. The fidelity of our translation relies on the IRP assumption, i.e., none of the actions in the HTN or any exogenous action can violate the assumption. To improve the efficiency of the system by avoiding multiple calls to the

same source with the same parameters, we implement a caching system similar to [17]. However instead of using a monitoring system we modify the translation of information-gathering atomic processes into HTN operators (this operator has preconditions that externally call information-gathering sources and add the return response) to explicitly encode the caching for the gathered information, and to reflect the different courses of action that must be followed. We consider the following 3 cases in our translation:

1. cannot delay the call and are calling the information source for the first time, so call the information source and cache the gathered information.
2. cannot delay the call and have already called the information source once, so use the cached information.
3. can delay the call to the information source, so use a placeholder data value.

Our translation relies on the use of a SHOP2-based HTN planner; it exploits SHOP2's features to perform runtime binding of variables and to make external procedure calls to invoke services. The full translation is excluded for space.

In Section 3, we discussed circumstances where data optimization can be performed in isolation of the generation of the composition. This can occur when the data is irrelevant to the optimization of the composition. i.e., it is not mentioned in any preferences, or when the data choice does not does not interact with the dynamics of the composition. For example, consider the book-hotel service and the information-gathering service that gathers information regarding available hotels. If the user has no preference regarding the choice of hotel, then it is efficient to delay the execution of this information-gathering service and the arbitrary selection of a hotel until after the composition is generated. To implement this, we identify these data and associated services a prior and modify the translation to remove the execution of the information-gathering service and to replace occurrences of the data with placeholders. The information-gathering service is then executed following composition generation and the placeholder replaced with an appropriate choice.

Similar to [8], let X be a set of information-gathering services available during planning. Then we represent the body of information that can be obtained from services in X as $\delta(X)$. More specifically, $\delta(X)$ represents all possible bindings of the predicates that appear in the `output` or the `postcondition` of the OWL-S descriptions of the services in X. Note that we operate under the IRP assumption, and more specifically, we assume that the results returned from these sources will not change during the planning step.

Definition 8 (Information-Gathering HTN Planning Problem). *An information-gathering HTN planning problem $\mathcal{P}'$ is a 3-tuple (s'_0, w'_0, D') where s'_0 is what is known of the initial state, and w'_0 and D' are generated following our modified OWL-S to HTN translator, described above. Assuming the IRP assumption holds for our planning problem, we define a corresponding HTN planning problem $\mathcal{P} = (s_0, w_0, D)$ where s_0 is a consistent complete initial state such that $s'_0 \cup \delta(X) \subseteq s_0$, w_0 is the initial task network, D is an HTN planning domain, and where w_0 and D are generated using the original OWL-S to HTN translator described in [17].*

From Definitions 3 and 8, a plan for the information-gathering HTN planning problem is a primitive decomposition of the task network w'_0. To find such a decomposition, some information-gathering operators, as dictated by the methods and operators of the domain, have to be applied to collect the relevant information needed to successfully decompose w'_0. These operators interact with the information sources and add new information to the state of the planning problem. The following theorem establishes soundness of our approach.

Theorem 2. *Let $\mathcal{P}$ and $\mathcal{P}'$ be corresponding planning problems as defined above. π is a plan for $\mathcal{P}'$ if and only if π is a plan for $\mathcal{P}$.*

The above theorem states that if a plan can be found in the information-gathering problem $\mathcal{P}'$ the same plan can be found from the corresponding complete problem $\mathcal{P}$, and vice versa. This holds by looking at the relevant search space. The following corollary immediately follows. Recall π is an optimal plan for $\mathcal{P}$ if there is no other plan of superior quality.

Corollary 1. *Let $\mathcal{P}$ and $\mathcal{P}'$ be corresponding planning problems as described in Theorem 1. π is an optimal plan for $\mathcal{P}'$ if and only if π is an optimal plan for $\mathcal{P}$.*

5 Computing a Preferred Composition

In this section, we address the problem of computing a most preferred composition by using AI planning techniques to help guide the construction of the composition. Our algorithm performs best-first, incremental search and uses state-of-the-art heuristics developed in [18]. The search is performed in a series of *episodes*, each of which returns a plan with better quality than the previous plan. The search in each episode performs branch-and-bound pruning, that is we prune nodes from the search space if provably there does not exist a plan extending this node with a better metric value than the one found in the previous episode. In addition, we perform sound localized data optimization on some already identified non-interacting operators. The two important heuristics we use are the Optimistic Metric Function (OM) and the Lookahead Metric Function (LA). The OM function estimates optimistically the metric value resulting from the current node. LA function estimate the metric of the *best successor* to the current node. In short, it first solves the current node up to a certain depth, and then it computes a single decomposition for each of the resulting nodes and returns the best metric value among all the fully decomposed nodes.

Our algorithm is outlined in Figure 1. The algorithm takes as input an information-gathering HTN planning problem (s'_0, w'_0, D'), a metric function MetricFn, and a heuristic function HeuristicFn. The nodes are of the form $\langle s, w, partialP \rangle$, where s is a plan state, w is a task network, and $partialP$ is a partial plan. This means w remains to be decomposed in state s and state s is reached from s'_0 by performing the sequence of actions $partialP$. The algorithm keeps the elements of *frontier* sorted according to the function HeuristicFn.

```
function HTNWSC(s'_0, w'_0, D', MetricFn,HeuristicFn)
  frontier ← ⟨s'_0, w'_0, ∅⟩, bestMetric ← worst case upper bound          ▷ initialization
  while frontier is not empty do
    current ← frontier's first element      ▷ best element since frontier is always sorted
    ⟨s, w, partialP⟩ ← current      ▷ establish the current values for s, w, and partialP
    lbound ← MetricBoundFn(s)                    ▷ estimating the lower bound for s
    if lbound < bestMetric then                  ▷ pruning suboptimal partial plans
      if w = ∅ and current's metric < bestMetric then
        Output plan partialP, bestMetric ← MetricFn(s)
      succ ← successors of current
      if possible to perform sound localized data optimization then
        succ ← the best node among successors of current      ▷ pruning other nodes
      frontier ← merge succ into frontier
```

Fig. 1. A sketch of our HTN WSC algorithm

The HeuristicFn function we use is a *prioritized sequence* of our heuristics (i.e., when comparing two nodes we look at the value of their heuristics in sequence to break ties when needed). We use a variable *bestMetric* that stores the metric value of the best plan found so far. This variable is initialized to a worst case upper bound. In each iteration of the while loop, the algorithm extracts the first element from the *frontier* and initializes the *current*. Then, it estimates a lowerbound, *lbound*, using the function MetricBoundFn and prunes nodes with a *lbound* greater than or equal to *bestMetric*. If *current* corresponds to a plan (i.e., w is empty), *bestMetric* is updated, and the plan is returned.

All successors to *current* are computed using the Partial-order Forward Decomposition procedure (PFD) [7]. If computing a successor to *current* implies picking a primitive task to decompose next and it is possible to perform sound localized data optimization for the operator that accomplishes this task, then data optimization on this node will select the best successor according to MetricFn and replace *succ* with the selected node[2]. The resulting *succ* is then merged into the *frontier*. Note that *succ* will have only one element if the algorithm chose to perform localized data optimization, that is all other nodes will get pruned from the search space. The search terminates when *frontier* is empty.

Optimality and Pruning. The search space for computing the preferred composition is significantly reduced by the flexible workflow captured in the structure of the HTN, by pruning performed from incremental search, and by the localized data optimization. So, under sound pruning we can guarantee that by exhausting the search space, an optimal plan can be found. We use the OM function to estimate the lower bound. Baier et al. [3] show that the OM function provides sound pruning under certain conditions.

[2] There are some subtleties, not discussed here, that ensure all appropriate grounding choices are considered and evaluated.

% of Identified Non-Inteferences	Case 1 Time(sec)	Case 2 Time(sec)	Case 3 Time(sec)	Case 4 Time(sec)	Average STI	Average PMI
0%	128	131	136	277	1.00	50.89%
20%	80	80	88	221	1.51	50.89%
40%	41	39	50	178	2.69	50.89%
60%	29	29	40	119	3.66	50.89%
80%	23	23	33	89	4.62	50.89%
100%	17	18	30	30	7.14	44.91%

Fig. 2. Time comparison between the four cases that found the optimal plan even without localized data optimization. STI is the search time improvement between each case and the no data optimization case (i.e., 0% case). PMI is the percent metric improvement. i.e., the percent difference between the metric of the first and the last plan returned relative to the first plan.

Proposition 1. *The OM function provides sound pruning if the metric function is non-decreasing in the number of satisfied preferences, non-decreasing in plan length, and independent of other state properties. A metric is non-decreasing in plan length if one cannot make a plan better by increasing its length only.*

Theorem 3. *If the OM function used to calculate the lower bound provides sound pruning, and any localized data optimization performed is sound, then the last plan returned, if any from the algorithm, is optimal.*

The proof follows from the proof of optimality for the **HPlan-P** planner [3] using Definition 7 and Theorem 1.

6 Implementation and Evaluation

We implemented our proof-of-concept WSC engine with two modules: a preprocessor and a preference-based HTN planner. The preprocessor reads PDDL3 problems and generates an HTN planning problem. Additionally, it finds non-interacting operators, making it possible to perform sound localized data optimization on this selection. Our implementation builds on **HTNPlan-P** [18], itself a modification of the LISP version of **SHOP2** [16], that implements the algorithm and heuristics described above. We have three main objectives in our experimental evaluation: (1) to measure the search time gain as well as the quality improvement by performing localized data optimization, (2) to see if performing localized data optimization helps in finding the optimal plan, (3) to investigate if the improvement (both time and quality) depends on other dimensions of search such as the heuristics used or the difficulty of the domain.

We use the travel domain described in this paper as our benchmark. We created 8 problem sets each with 6 different instances (we have 48 instances in total). In half of the problem sets we allowed interleaving of tasks and in the other half we did not. An example of interleaving is one that allows booking an accommodation when a transportation is booked, but not paid for (i.e., the transportation task is not done yet). Furthermore, the problem sets within the

allowed (or not allowed) interleaving group differ in the difficulty of their top-level task. In the easiest case, the order of the execution of all tasks in *arrange-travel* (e.g., *arrange-trans*, *arrange-acc*, and *arrange-activity*) was known, and in the hardest case, these tasks could be carried out in any order. As explained earlier, if there are n tasks and they can be carried out in any order, then in the worst case there are $n!$ different combinations to evaluate in order to find the optimal composition. Finally in each problem set we know the number of non-interacting operators, but intentionally select the percentage of the one identified from this range [0, 20, 40, 60, 80, 100]. So in the 0% case none of the non-interacting operators are identified, hence, no localized data optimization can be performed, on the other hand in the 100% case all of the non-interacting operators are known, and localized data optimization is performed whenever possible. We used a 60 minute time out and a limit of 1 GB per process.

We ran all of the instances in two modes, one that makes use of the LA heuristic and one that does not. To compare the relative performance between the two modes, we averaged the percent metric difference of the final plan (relative to the worst plan) for all our 48 instances. This difference is 43% indicating that not surprisingly, using the LA heuristic greatly improves the quality of search. In particular, without the use of the LA heuristic, an optimal plan was not found in any of the instances. However, when the LA heuristic was used, many instances found an optimal plan. In particular, in four of the problem sets (we named them cases 1-4 in Figure 2), an optimal plan was found even without any localized data optimization The result shows (see Figure 2) that as the percentage of identified non-interacting operators increases (i.e., more localized data optimization is done), the time it took to find the optimal plan decreases. We averaged this improvement and show it in the STI column (search time improvement with respect to the 0% case). This column shows that optimal plans are found for example, 2.69 times faster than the 0% case in the 40% case, and 7.14 times faster in the 100% case. Also recall that our algorithm is incremental, performing search in a series, each one returning a better-quality plan than the last. To see how effective this approach is, we calculated the *percent metric improvement* (PMI), i.e., the percent difference between the metric of the first and the last plan returned relative to the first plan. The result shows that the incremental approach improves the quality of the plan almost by 50%.

Finally, we looked at the other four cases where without localized data optimization an optimal plan was not found. Out of these, in two, an optimal plan was found in the 100% case and this was found 3.5 times faster than the time it took to find a non-optimal plan in the 0% case. This suggests that doing localized data optimization for these harder problem sets is helpful. In the remaining two cases, an optimal plan was not found even with optimization. This is not surprising, since the search space in these sets is very large, and pruning even though helpful, is not able to exhaust the search space; in these cases interleaving was allowed and the top level tasks were unordered. However, we observed that with optimization, the quality of the final plan was improved by 10%, and the time spend on finding this better quality plan was 5 times faster.

7 Summary and Related Work

A significant number of WSC problems involve both optimization of the composition and the collection of information. Work on preference-based WSC has begun to address this problem but much of the work has ignored the critical information-gathering component, assuming that all information is given a priori. In this paper, we are motivated by the observation that even though some classes of WSC problems can be addressed without the need for any execution during the composition phase, without explicit consideration of the data, and without consideration of preferences that distinguish high-quality solutions, many interesting and useful compositions must be done hand in hand with the data collection and optimization. Specifically this is done following execution of some information-gathering services. The main contributions of this paper include: identification of a way to exploit structure in the preference specification and domain in order to generate compositions more efficiently by performing what we call *localized data optimization*, identification of a condition where performing localized data optimization is sound, development of an execution engine for preference-based WSC that interleaves online information gathering with offline search as deemed necessary, and identification of a case where we could prove the optimality of resulting compositions. To assess the effectiveness of our approach to WSC, we performed experiments to evaluate the performance of our system. We showed that our approach to data optimization has the potential to greatly improve the quality of compositions and the speed with which they are generated. While the focus of this paper was reasonably narrow, the problem it presents and the advances it makes are important first steps in addressing a broad and important problem.

While no other WSC planners can perform true preference-based planning, **SHOP2** [16] and **enquirer** [8] handle some simple user constraints. The **scup** prototype PBP planner in [11] is related but there are several differences to our work. In particular, their preferences are pre-processed into task networks and conflicting user preferences are detected and removed prior to invocation of their planner. Further, they do not consider handling regulations and are not able to specify preferences over the quality of services.

Another body of related work is the research on quality-driven WSC (e.g., [10,21,1]). This research addresses the problem of run-time service selection based on the functional (e.g., input and output matching) and non-functional (e.g.., reliability, availability, and reputation) properties of a service. This is addressed by encoding the problem as an optimization problem that can be solved using for example: Integer Programming (e.g., [21]), Mixed Integer Programming (e.g., [1]) or Genetic Algorithms (e.g., [10]). Our work differs in many ways. In particular, in our framework we are able to find a composition that is optimal with respect to the user's preferences some of which are over the entire composition, and we can do so while interleaving execution and search. Further, we are concerned with optimizing the selection of data within the services in addition to the selection of services themselves based on their quality.

Acknowledgements. We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Ontario Ministry of Innovations Early Researcher Award (ERA).

References

1. Alrifai, M., Risse, T.: Combining global optimization with local selection for efficient QoS-aware service composition. In: Proc. of the 18th Int'l World Wide Web Conference (WWW 2009), pp. 881–890 (2009)
2. Au, T.C., Nau, D.S.: Reactive query policies: A formalism for planning with volatile external information. In: Proc. of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM), pp. 243–250 (2007)
3. Baier, J.A., Bacchus, F., McIlraith, S.A.: A heuristic search approach to planning with temporally extended preferences. Artificial Intelligence 173(5-6), 593–618 (2009)
4. Bertoli, P., Kazhamiakin, R., Paolucci, M., Pistore, M., Raik, H., Wagner, M.: Continuous orchestration of Web services via planning. In: Proc. of the 19th Int'l Conference on Automated Planning and Scheduling (ICAPS), pp. 18–25 (2009)
5. Calvanese, D., Giacomo, G.D., Lenzerini, M., Mecella, M., Patrizi, F.: Automatic service composition and synthesis: the Roman Model. IEEE Data Eng. Bull. 31(3), 18–22 (2008)
6. Gerevini, A., Haslum, P., Long, D., Saetti, A., Dimopoulos, Y.: Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. Artificial Intelligence 173(5-6), 619–668 (2009)
7. Ghallab, M., Nau, D., Traverso, P.: Hierarchical Task Network Planning. In: Automated Planning: Theory and Practice. Morgan Kaufmann, San Francisco (2004)
8. Kuter, U., Sirin, E., Nau, D.S., Parsia, B., Hendler, J.A.: Information gathering during planning for Web service composition. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 335–349. Springer, Heidelberg (2004)
9. Kuter, U., Sirin, E., Parsia, B., Nau, D.S., Hendler, J.A.: Information gathering during planning for Web service composition. J. Web Sem. 3(2-3), 183–205 (2005)
10. Lécué, F.: Optimizing QoS-aware semantic Web service composition. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 375–391. Springer, Heidelberg (2009)
11. Lin, N., Kuter, U., Sirin, E.: Web service composition with user preferences. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 629–643. Springer, Heidelberg (2008)
12. Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D., Sirin, E., Srinivasan, N.: Bringing semantics to Web services with OWL-S. World Wide Web Journal 10(3), 243–277 (2007)
13. McDermott, D.V.: Estimated-regression planning for interactions with Web services. In: Proc. of the 6th Int'l Conference on Artificial Intelligence Planning and Scheduling (AIPS), pp. 204–211 (2002)
14. McDougall, P.: IBM eyes plug-and-play cloud framework, informationWeek (July 8, 2010)
15. McIlraith, S., Son, T.: Adapting Golog for composition of semantic Web services. In: Proc. of the 8th Int'l Conference on Knowledge Representation and Reasoning (KR), pp. 482–493 (2002)

16. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. Journal of Artificial Intelligence Research 20, 379–404 (2003)
17. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for Web service composition using SHOP2. J. Web Sem. 1(4), 377–396 (2005)
18. Sohrabi, S., Baier, J.A., McIlraith, S.A.: HTN planning with preferences. In: Proc. of the 21st Int'l Joint Conference on Artificial Intelligence (IJCAI), pp. 1790–1797 (2009)
19. Sohrabi, S., McIlraith, S.A.: Optimizing Web service composition while enforcing regulations. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 601–617. Springer, Heidelberg (2009)
20. Sohrabi, S., Prokoshyna, N., McIlraith, S.A.: Web service composition via generic procedures and customizing user preferences. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 597–611. Springer, Heidelberg (2006)
21. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Trans. Software Eng. 30(5), 311–327 (2004)