Anytime Planning for Web Service Composition via Alternative Plan Merging

George Markou and Ioannis Refanidis
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
{gmarkou, yrefanid}@uom.gr

Abstract: This paper presents an approach towards automated semantic web service composition using contingent planning. The presented planner, MAPPPA, is an anytime probabilistic planner tailor-made for web service composition problems that produces a contingent plan by integrating alternative deterministic plans previously computed from a determinized version of the original planning problem. The planner can produce each plan without disregarding the important information that each web service contains in relation to its cost and probability of alternative outcomes. MAPPPA has already been integrated into our prototype web service composition platform, MADSWAN, producing promising results.

Keywords:—contingent planning; web service; Fully Observable Probabilistic (FOP) problem

1. Introduction

Semantic web services are software elements with a formal description of their capabilities. Given a composite user requirement, an atomic web service with the desired functionalities may not exist; in that case, a combination of various atomic web services may be required to achieve the goal.

AI planning is the task of coming up with a partially ordered set of actions that achieve a goal. If one considers Web Service Composition (WSC) as the task of finding and ordering a set of atomic web services so that their aggregate behavior achieves the intended goal of the desired composite web service, then the connection between WSC and AI planning becomes obvious and existing AI planning techniques can be efficiently utilized to tackle the problem of WSC.

Being able to compose semantic web services automatically can lead to great reductions in the time and money required to produce enterprise applications. An additional burden of the WSC process, though, lies in the fact that, since web service descriptions refer to real world web services, the outcomes of their execution cannot be anticipated in advance. As such, a non-deterministic approach is required; however, this is a difficult task. In [1], it is shown that solving the composition problem of non-deterministic web services with complete information is EXP-hard.

We consider semantic web services at their functionality level [2], in which web services are defined in regards to preconditions and effects over ontological concepts. Web services are also allowed to have multiple alternative outcomes, each with a probability of occurring attached to it, thus the problem is formulated as a fully observable probabilistic (FOP) one.

MAPPPA (an anagram of **A**nytime **P**robabilistic **P**lanning via **A**lternative **P**lan **M**erging) is based on an anytime contingent planning framework. Initially, the web service domain is modeled as a probabilistic planning one. Then, a determinized version of it is produced, while retaining the information the original one contains in regard to the probabilities and cost of the web services/actions. This problem is solved repeatedly by a deterministic planner until either a time limit set by the user is reached, or no more plans can be generated. Finally, the deterministic plans are merged in a decision tree, using the initial non-deterministic actions as its decision nodes. A heuristic based on the expected cost of each plan is used, in an attempt to minimize the size of the decision tree.

This work is inspired by the winner of the 2004 International Probabilistic Planning Competition (IPPC-04) [3] FF-Replan [4], as well as subsequent planning approaches that were based on it, such as [5] and [6]. Unlike these approaches, the one presented here takes the probabilities of the original non-deterministic actions into consideration while generating the contingent plan. The probability of the contingent plan being successfully executed monotonically increases as more time is allowed for the planner and more plans are generated and added to it. Finally, this approach is specifically targeted for web service composition problems.

The rest of the paper is structured as follows. First, we formulate the problem and shortly review related work, both regarding contingency planning and web service composition. Next, we illustrate our approach, i.e., how we generate an initial plan and merge it with subsequent plan branches. The evaluation section presents results on two planning domains, each having two variants. Finally, we conclude the paper, present shortcomings of our current approach and pose directions for future work.

2. Background

Given a set of semantic web service descriptions in OWL-S [7] format available in a registry, an automatic translation of the original WSC problem to an AI planning one is possible. Various works have proposed such conversion schemas from OWL-S to PDDL that do not differ significantly to each other; we adopt an approach similar to [8, 9]. Our translation module is based on the source code of [10], with the necessary extensions to accommodate the generation of PPDDL [11] files instead of PDDL ones.

We view WSC as "planning for service chaining" [12], that is, we only take into account semantic web services at their functionality level; this is described in the service profile part of OWL-S. In essence, all technical details regarding the actual data structures or WSDL schemata used in the web services are ignored, and each atomic web service is considered to be executed instantly, with an one shot functionality.

The semantic web services are characterized by their name, Inputs/Outputs and Preconditions/Effects (IOPEs); the web services' inputs and outputs comprise a set of typed objects, the typing of which is in regard to their membership in ontological concepts [12, 2]. Concepts are defined within relevant ontologies (in our case, in OWL [13]) as classes, with possibly complex relationships between them, e.g., each concept can be a supertype of or be subsumed by, that is, be a subtype of, another concept. In this work though, we only consider exact matches [14], and plug-in or subsumes relationships between ontological concepts are not taken into account. We also assume that web services have neither delete effects, nor negated preconditions.

The basic translation process that we follow [10] requires that the OWL-S web services' inputs and preconditions (*hasInput* and *hasPrecondition* in OWL-S, respectively) be transformed to a (P)PDDL action's preconditions, and the add effects of the action be comprised of its outputs and effects (*hasOutput* and *hasEffect* in OWL-S, respectively). The OWL-S profile of a web service allows the definition of non-functional properties of it; the standard supports the qualitative definition of a web service, within a specified rating system of the publisher's choice. It also supports an unbounded list of service parameters that can contain any type of information, e.g., the web service's average, min or response time [7].

The current web service repository used in our approach [15] comprises all the OWL-S descriptions obtained from the latest version of the OWL-S Service Retrieval Test Collection [16]. However, the descriptions in this collection do not contain any QoS (Quality of Service) information in relation to the services' reliability, i.e., their ability to execute their stated purpose for a specified period of time, or cost (of requesting and executing the service). For this reason, for now, we assign random costs and probabilities to each effect of an action in our evaluation tests.

The set of ontological concepts that is present as IOPEs for the web services constitutes the domain's predicates. The initial state and the goal of the problem consist of a conjunction of literals, i.e., ontological concepts, and the solution of the planning process essentially provides a template for execution, having selected the necessary web services and their order of execution, without though specifying any interaction pattern between them.

A. Problem Formulation

Here we introduce the definitions and notation in relation to probabilistic planning that we will use in the rest of this paper:

Definition 1: A probabilistic planning domain is of the form $D = (S, A, \gamma, Pr, Co)$, where S is a finite set of states, A is a finite set of actions, $\gamma : S \times A \rightarrow 2^S$ is the state-transition function, $Pr : S \times A \times S \rightarrow [0,1]$ is the probability-transition function and $Co : S \times A \times S \rightarrow \mathbb{N}$ is a bounded cost-function. The set of all actions that can be applied to state s (in the specific domain D) is $A_D(s) = \{\alpha \in A : \gamma(s, \alpha) \neq 0\}$.

Definition 2: A planning problem is a triple $P = (s_0, s_g, D)$ where $s_0 \in S$ is the initial state, $s_g \in S$ is the goal one. The definition of the initial state is with respect to closed world semantics; moreover, the initial state only contains static propositions whose truth value cannot change during the planning process. We consider the planning problem to be FOP.

Definition 3: If we define $S_\pi \subseteq S$ as the set of states to which an action has been assigned under policy π, and $S_\pi(s)$ as the set of reachable states from s using π, solutions to FOP planning problems can be (total) policies $\pi : S \rightarrow A$, or partial functions from a set S_π to A, with $s_0 \subseteq S_\pi$ and S_π closed under policy π; then for $s \in S_\pi$ the solution π dictates

to apply action $\pi(s)$in state s [17]. If $a \in A_D(s)$ then $Pr(s, a, s')$ is the probability of reaching state s' if action a is applied to state s.

Definition 4: A policy π is closed with respect to a state s iff $S_\pi(s) \subseteq S_\pi$. If the goal state can be reached using π from all (π-reachable) states $s' \in S_\pi(s)$, then π is considered proper w.r.t. a state s. Iff π is both closed and proper w.r.t. the initial state s_0 then it is deemed a valid solution [18]. Valid solutions can be either strong, or strong cyclic; a policy π is acyclic if for all possible executions $\tau = s_0 s_1 s_2 \ldots$ of π from s_0, it holds that $s_i \neq s_j$, for any $i \neq j$ [19]. In general, though, according to [20], solutions to non-deterministic problems can be either weak (with a chance of success), strong (guaranteed to achieve the goal despite non-determinism), or strong cyclic (guaranteed to achieve the goal with iterative trial-and-error strategies).

Assuming that each outcome of an action a from state s has a non-zero probability, then a strong plan reaches the goal with probability 1 in at most n steps, $n \leq |S|$, since the execution path of a strong (acyclic) plan cannot pass through the same state twice. It should be noted that since strong solutions to a planning problem are a subset of the strong cyclic solutions [20], there are problems in which strong solutions do not exist, but strong cyclic ones do.

3. Related Work

In this section we present related work that in some cases provides the basis for our work, both in relation to WSC and the AI planning part.

A. Web Service Composition Approaches

Concerning WSC, there is a vast variety of approaches, with differences in regard to the compatibility to specific - standard or proprietary - input languages, the use of semantic technologies, the types of problems tackled and their general purpose [2]. The approaches closest to the one presented here are the ones in [21, 22].

OWLS-Xplan [8] is one of the first approaches incorporating a translation between OWL-S and PDDL. The result of this translation was given as input to a hybrid planner that combined guided local search with graph planning and a simple form of HTN decomposition. The system did not take non-determinism into account. On the other hand, although [22] provide a translation very similar to that of [8] and [9], their approach acknowledges the domain's non-determinism and incorporates (a modified version of) an existing PDDL planner, Simplanner [23], to tackle it through interleaving planning and execution.

The WSC approach closest to our goals and problem formulation is the one presented in [21]. The web services used are described in OWL-S and their IOPEs reference OWL concepts. More importantly, the planning algorithm used outputs contingent plans; specifically, all the different possible (deterministic) paths are generated, with a reordering of the paths so that the shortest ones are executed first.

The concept of useful web services is introduced, that is, services that generate outputs and effects that are part of the goal, or inputs and preconditions of other useful web services. For each user output, effect or activity a vector is created, containing only the useful operations that produce it. Then, all the possible paths comprising an element of each vector are produced. The use of only the useful web services and not the entire set of them allows for a restriction of the search space. However, users of the approach are required to specify a workflow through its inputs and activities that must be executed before planning, in a proprietary high level workflow description language, thereby restricting its automatic nature. Moreover, the approach in [21] does neither make use of any QoS based cost models nor take into account a service's probability of failing. In that way, all web services are considered equal during the search phase and the plans do not differentiate between two services that produce the same results.

Finally, [2] implemented an approach dedicated to non-deterministic WSC utilizing AI planning. The authors treat the application of a web service as a belief update operation and identify two special cases of WSC under uncertainty that are tractable and allow for a compilation of the original problem into conformant planning. Subsequently, an existing conformant planner is used, namely, Conformant-FF [24].

B. Contingency Planning and Determinizations of Non-deterministic Domains

Our original inspiration for the non-deterministic approach used in this paper was FF-Replan. FF-Replan utilizes the FF planner [25] to generate a single plan for a deterministic version of the original Markov Decision Process (MDP) problem, and produces a new plan each time one of its simulated executions of the current plan results in an unexpected state (from that state to the goal state).

In [4] two methods of creating a "determinized" version of a non-deterministic problem are introduced. The first was single-outcome determinization, which selects only one probabilistic effect as the outcome of each probabilistic action, without taking into account the rest of its effects or any of their probabilities. The second one, all-outcomes

determinization, creates a new action for each of the outcomes of the probabilistic effects in the original non-deterministic action.

For the non-deterministic planning domain D described in Section II.A.1, let $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, then for an action $\alpha_i \in A$ with k different probabilistic outcomes, the all-outcomes determinization is the set of deterministic actions $\text{Det}\alpha_i = \{\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{ik}\}$ along with a state-transition function $\bar{\gamma}$ such that for every state s, $\gamma(s, \alpha_i) = \bar{\gamma}(s, \alpha_{i1}) \cup \ldots \cup \bar{\gamma}(s, \alpha_{ik})$ [5]; the single-outcome determinization for action α_i simply chooses one of the actions in $\text{Det}\alpha_i$ to replace α_i.

FF-Replan uses the all-outcomes determinization process, as do NDP [5] and the planning approach in [6]. NDP is very similar to FF-Replan; in contrast to it, though, NDP is sound and/or complete on the condition that the deterministic planner used is sound and/or complete accordingly. Moreover, NDP, as well as the approach in [6] that uses NDP as its starting point, generate strong cyclic solutions. FF-Replan on the other hand, generates a single weak plan every time it is executed.

The all-outcomes determinization, however, ignores the probabilities attached to the probabilistic outcomes and allows the planning approaches that make use of it to be overly optimistic. That is, planners simply choose the most convenient action, one that achieves the desired effect, regardless of its underlying (true) likelihood.

An alternative, dynamic, determinization process is presented in [26]; this approach is a generalization of FF-Replan, which, instead of solving a single determinized problem, randomly produces various determinized problems and combines their solutions. The value of each state is approximated by sampling these (non-stationary) problems originating from it, with the selected outcome for each action varying with time. The problems are then solved in hindsight and the values of the states are combined. In that way, the determinization process is no longer static as in FF-Replan, which can be simply considered an optimistic approximation of hindsight optimization.

An attempt to take the actual outcomes' probabilities into account is made in [27]; the all-outcomes determinization is combined with a translation of these probabilities to associated cost values for the new deterministic actions' outcomes, each denoting the risk of it failing. In that way, a metric planner compliant with PDDL is able to improve the robustness of the plans by trying to minimize the sum of the negative logarithms of the success probabilities, which in turn minimizes the product of the failure probabilities.

This translation process was used in [28] for incremental (or limited) contingency planning; an initial deterministic seed plan is generated, which is then iteratively improved through an analysis and repair cycle. The original plan is analyzed to find outcomes that are relatively probable and result in dead-ends, and then precautionary actions are added in relation to the most problematic outcomes. Recoverable failures, on the other hand, are left as-is in the plan and are repaired through replanning at execution time.

Our approach for generating a contingent plan is similar to [29] which, initially, constructs a seed plan though the use of a deterministic planner, assuming that each action is executed as dictated by its expected behavior. Then, additional (deterministic) branches are generated and added to the existing plan incrementally, so as to improve its expected utility. The best place to insert a branch is computed based on the amount of utility gained in the seed plan from adding it at the specific place. Since this computation is intractable, an approximation is used instead, using Monte Carlo simulations and propagating utility distributions through a graph. This process is repeated either until the plan is sufficiently robust, or the process runs out of time. Both approaches attempt to generate a contingent plan with the maximum possible expected utility; however, we do not generate our plans in the same way as in [29], whereas the sources of uncertainty in [29] regard continuous quantities, e.g., time.

```
Function GenerateDT
Input A set of plans P
Output A decision tree containing all the plans
1.  Sort(P, aversion_metric)
2.  p = P → first_plan()
3.  While (p ≠ null)
4.      root = new DTNode()
5.      a = p → first_action()
6.      root → addAction(a)
7.      n = root
8.      While (a ≠ null)
9.          If non_deterministic(a)
10.             n.addLeftBranch =
                GenerateDT(Sort(ValidPlans(P, n) →
                get_branch()))
11.         Endif
12.         a = a → next_action()
13.         If (a ≠ null)
14.             n_2 = new DTNode()
15.             n.addRightBranch = n_2
16.             n = n_2
17.             n → addAction(a)
18.         Endif
19.     EndWhile
20.     p = p → next_plan()
21. EndWhile
22. Return(root)
```

4. Alternative Plan Generation and Merging

MAPPPA is an anytime contingent planning algorithm; in short, the planning process consists of three steps: creating a determinized version of the domain based on the original probabilistic one, generating multiple solutions to the new deterministic problem, and finally, merging these

solutions in a single decision tree, which constitutes the contingent plan.

This approach is not a generic contingency planning one; it is tailor-made for web service composition problems and specifically for ones where the web services involved do not produce irreversible results. That is, we assume that it is always possible, from any state in the problem, to return to the initial one. This is made possible through our assumption that the initial state contains static propositions and the actions in the planning domain do not have delete effects or negated preconditions.

Moreover, we assume that once an action has been executed with a specific effect as an outcome, then for the rest of the particular branch it cannot be executed again with a different outcome. This assumption also holds if the action's execution produces undesired effects, i.e., once an action fails in a particular branch, we assume that it will always fail in that branch. This is a realistic assumption in a WSC domain, as a web service that produces undesirable effects for some reason, e.g., network failure, is not probable to be available again in a very short amount of time. A consequence of this assumption is that the contingent plan can execute each of the actions at most once in each of its branches; thus, the solutions do not contain infinite branches or cycles.

For this reason, the generated (deterministic) solutions are weak ones; they can only succeed in the case that all the actions in the plan actually have the desired effect as their actual outcome. If there are n actions in branch $\text{Plan}_i = [a_{1j}, a_{2j}, \ldots, a_{nj}]$, the desired outcome j of each having a probability of $\text{prob}_{\alpha_{ij}}$ of being the actual one, then each branch has a probability $\text{Prob}_{\text{branch}} = \text{prob}_{\alpha_{1j}} \times \text{prob}_{\alpha_{2j}} \times \ldots \times \text{prob}_{\alpha_{nj}}$ of being executed successfully.

A real world example of such a problem containing web services would be the following: Assume a person wants to purchase an item (the problem's goal state) from the web using his credit card. We consider the state in which he has a credit card (that has not been charged) available and the product has not been purchased to be the problem's initial state. Using an eBay web service, the user searches for the item among various eBay sellers and finds several ones that have the item in stock. In the case that none of these sellers offers the item in the desired price or ship it to the user's address, the user can at any time return to the initial state, having an available credit card with the goal of purchasing the particular item. He can then choose an alternative action, e.g., using an Amazon web service to search for the item in the Amazon web store.

MAPPPA adopts the all-outcomes determinization process, but in order to produce meaningful plans in relation to the actual outcomes' probabilities, each deterministic action produced from the original domain is associated with an aversion factor. That is, we maintain the original probabilities and costs from the probabilistic domain and combine them into a single metric that indicates how much the planner should try to avoid using this action due to its high probability of failing or its high cost in case it succeeds.

Fig. 1. Function $GenerateDT$ - Generation of the decision tree

```
Function ValidPlans
Input A set of plans P and a branch B
Output A simplified set of plans than can be added to B
1.  For (Plan p in P)
2.      For (action a in p)
3.          If (a ∈ B)
4.              If(result(a) ∈ p ≡ result(a) ∈ B)
5.                  p = p − {a}
6.              Else
7.                  P = P − {p}; break
8.              Endif
9.          Endif
10.     Endfor
11. Endfor
12. Return P
```

Fig. 2. Function $ValidPlans$ - Insertion of valid plans in the decision tree

The algorithm is not guaranteed to converge to an optimal contingent plan; of course, its probability of the contingent solution being successfully executed monotonically increases as each new branch is generated and added to it. In the case all the decision tree branches achieve the goals, the DT is a strong plan. In the general case however, the decision tree is a weak plan.

The planning process in our approach can be based on any search algorithm that returns multiple deterministic plans to the goal. We used an implementation of a variation of the A* algorithm [30] for our evaluation. A* has been forced to continue finding solutions after the first one is found, whereas it terminates only when either a time limit has been reached, or a limit on the number of possible solutions has been set.

Our aversion metric can be integrated into the planning process both as the past path-cost function of A* during planning, as well as a sorting metric of the plans afterwards. Since there is obviously a variety of ways to combine the probability of an action to produce a specific effect with the cost of its execution, we considered several options, as is demonstrated in the Evaluation section. When no more plans can be generated, either because all the possible plans

have already been found or due to a time limit, the plans are integrated into a decision tree plan, based on the algorithms shown in Fig. 1-2.

In the decision tree's generation function ($GenerateDT$), the algorithm's input are the determinized plans, with their deterministic actions mapped back to their non-deterministic counterparts;, these plans are sorted by their ascending aversion factors (Fig. 1, line 1) and the first action of the first plan (based on its order of execution) is added to the tree. While there are still plans left in the input set, a new decision tree node is created based on the first action of the current plan (lines 4-6). Then for the rest of the actions in the current plan, if the current action considered is non-deterministic, a left branch is added to the current decision tree node, corresponding to the result of the action's alternative outcome, or its failure; the actual branch is returned as the result of a recursive call to $GenerateDT$(Fig. 1, line 10).

The recursive call's input is the result of a call to $ValidPlans$ (Fig. 2). $ValidPlans$ is called with the set of all plans and the current branch as its inputs. Then, the plans in this set that are valid candidates for insertion at this point are computed, and a new simplified set of them is returned.

If a plan in the input set of $ValidPlans$ contains at any point actions that have already been executed in its input branch, there are two possibilities: If the action was executed having the same outcome as the one dictated by the current plan, then the plan will be inserted into this branch without the particular action (Fig. 2, lines 4-5). This is the result of our assumption that if an action has been executed with a particular result it will have this result for its entire branch. Moreover, as there are no delete effects or negated preconditions, since the action is considered to have already been executed, its output effects still hold, and as such, there is no need for it to be executed again. If the plan contains an action that was previously executed in the current branch having a different outcome, then the entire plan is not contained in the plan set returned by the function for this particular branch (Fig. 2, lines 6-7).

Since the set of valid plans is not the same as the original one, as some actions have been removed from them, their cost and probability of successful execution have also changed. For this reason, the set of plans that is used as an input for the recursive call to $GenerateDT$ is sorted again by their plans' ascending aversion factors (Fig 1, line 10), so as to be inserted to the branch in the correct order.

For any action in the current plan considered, whether deterministic or non-deterministic, a right branch is created for the current node (Fig 1, lines 14-15) corresponds to the action's desired outcome in the particular plan, and the next action in the plan is added to it (Fig. 1, lines 12 and 17). This procedure is repeated for all the plans in the original plan set.

The algorithm's output is a combination of weak solutions; as such, it may produce strong contingent solutions, if all possible plans are computed. However, in general, there is no guarantee that the plans will be strong. Moreover, despite the fact that certain problems may only have strong cyclic solutions and no strong ones, at this point our algorithm does not generate such solutions.

A. Contingent Plan Example

An example of the contingent planning procedure we propose for a simple domain is shown in Fig. 3. Fig. 3a-3e present the generation of the plans. The problem consists of a (single) initial state s_0 and the goal state G (each state is represented by a circle). Actions a_2 and a_7 are deterministic (denoted by a straight line), while the rest of the actions are probabilistic (denoted by a dotted line); for each probabilistic action a_i, $i \in \{1,3,5\}$, its (single) effect is executed with a probability $prob_{\alpha_{ij}} = 0.8, \forall i \in \{1,3,5\}$, and with a probability of $1 - prob_{\alpha_{ij}} = 0.2$, it fails and the current state does not change. Actions a_4 and a_6 have two different effects, each having a different probability of being produced when executing the related action, with $prob_{\alpha_{41}} = 0.9$, $prob_{\alpha_{42}} = 0.1$ and $prob_{\alpha_{61}} = 0.8$, $prob_{\alpha_{62}} = 0.2$. The cost associated with each action is shown opposite it, e.g., $cost_{\alpha_{11}} = 4$, $cost_{\alpha_7} = 2$. For this example, we use as an aversion metric, $g = \sum cost_{a_{ij}} + \sum \frac{1}{prob_{\alpha_{ij}}+1}$.

After the all-outcomes determinization process, the actions available to the deterministic planner are a_{11}, a_{12}, a_2, a_{31}, a_{32}, a_{41}, a_{42}, a_{51}, a_{52}, a_{61}, a_{62} and a_7. Of these, a_{1_2}, a_{32}, and a_{52} are discarded as they do not produce any effect, and a_2 and a_7 remain as-is, as they were already deterministic. The first deterministic plan that is generated, $Plan_1 = [a_{11}]$, is shown in Fig. 3a, consisting only of one action that leads directly to the goal state G. The second generated plan $Plan_2 = [a_{2,}a_{31}]$, has the same probability of reaching the goal $(prob_{Plan_1} = 0.8, prob_{Plan_2} = 1 * 0.8 = 0.8)$, but its cost is greater, thus leading to a larger aversion factor; $g_{Plan_1} = 4 + \frac{1}{0.8+1} = 4.55$, whereas $g_{Plan_2} = \left(2 + \frac{1}{1+1}\right) + \left(3 + \frac{1}{0.8+1}\right) = 6.05$. The planner then generates plans $Plan_3 = [a_{41}, a_{51}, a_{61}, a_7]$ (Fig.

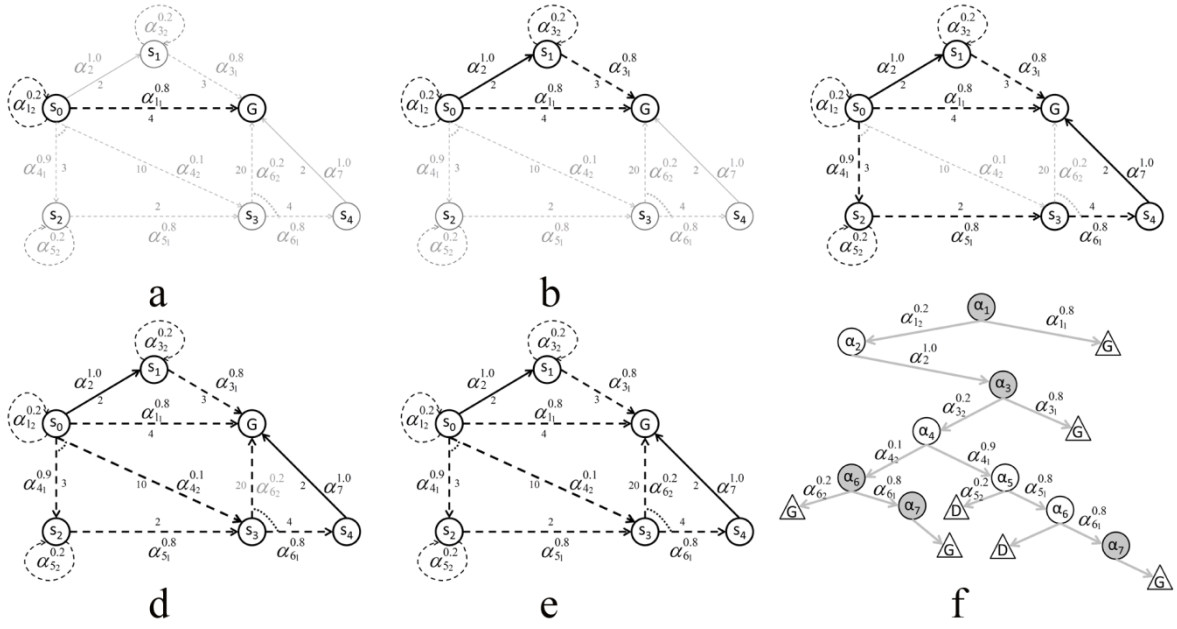

Fig. 3. (a-e) Solution steps. (f) Solution plan

3, $g_{Plan_3} = 13.13$), $Plan_4 = [a_{42}, a_{61}, a_7]$ (Fig. 3d, $g_{Plan_4} = 17.96$), $Plan_5 = [a_{41}, a_{51}, a_{62}]$ and $Plan_6 = [a_{42}, a_{62}]$ (both shown in Fig. 3e, $g_{Plan_5} = 19.91$, $g_{Plan_6} = 24.74$).

Fig. 3f presents the form of the contingent solution; Circular nodes are chance nodes that lead to alternatives over which the planner has no control, i.e., they denote non-deterministic actions, with the ones that can potentially lead to the goal being depicted in grey. Triangular nodes are end nodes; either goal ones ("*G*"), or dead-end ("*D*") ones. The edges depict the actual executed outcome of the action. The plans are sorted by their aversion factors and $Plan_1$ is inserted into the decision tree. The successful execution of a_1 is added as the first right branch in the tree. The left branch of a_1 corresponds to its failure, and requires the computation of the possible plans for insertion. Since no other plan contains a_1, all plans except $Plan_1$can be inserted; moreover they all retain the same aversion metric value. $Plan_2$ is added; since a_2 is deterministic, it leads to a_3, where the previous procedure is followed again. Following the insertion of $a_4$and its outcome a_{42}, $Plan_1$, $Plan_2$, $Plan_3$ and $Plan_5$ are rejected, as they contain outcomes $a_{11}, a_{31}, a_{41}$and a_{41}respectively, which correspond to actions that have already been executed with different results. $Plan_4$ and $Plan_6$ are valid, and since $a_{42}$is now considered to have happened, only their remaining portions need to be added, i.e., $[a_{61}, a_7]$ and $[a_{62}]$ respectively. As such, their probabilities and cost have changed to $g_{Plan_4} = 20.83$ and $g_{Plan_6} = 7.05$ and $Plan_6$ is chosen as the right branch of the tree. After the execution of the left branch at that point (a_{62}), only $Plan_4$ is valid, and as it contains no more actions, we reach a goal state.

5. Evaluation

In order to test the efficiency of **MAPPPA** we empirically evaluated the system using two domains; the first is the one presented in section IV-A, namely Dom_1. The second domain, Dom_2, is a modified version of the evaluation domain from [10]; the problem describes a user that desires to purchase a book through an electronic bookstore. He knows the book title and its author, and can provide his credit card information, and shipping address. The results of the output composite web service should be the book's purchase, as well as shipping dates and customs cost for it.

We created a probabilistic version of each domain, with costs attached to the web services that comprise it, and a subset of the web services being deterministic. Moreover, we added alternative web services achieving the same results as the existing ones with different probabilities and/or costs, as well as different preconditions, in order to facilitate the generation of alternative plans. All the original web services are part of version 4.0 of the OWL-S Service Retrieval Test Collection.

Both domains have two versions, one with uniform costs for the web services ($\mathrm{Dom}_{\mathrm{x}}^{\mathrm{uni}}$), and one with variant ones for each web service ($\mathrm{Dom}_{\mathrm{x}}^{\mathrm{var}}$). In all domains the services' costs start from 1 and are taken from an exponential

probability density function, i.e., $(x) = e^{-(x-1)}$, for $x \geq 1$. The experiments were run on a PC using a Dual-Core Intel i5 processor running at 1.6GHz, allowing at most 4GB memory.

We examined several different setup options for the evaluation; A* was the implemented search algorithm, for which we tested various combinations of the path-cost function as well as of the heuristic estimates, one variation of which corresponds to Breadth First Search (BFS).

In regard to the heuristic estimates for A*, we investigated four different ones. The first being the max heuristic (h_{max}), and the second being the additive heuristic (h_{add}) [31], with the extra assumption that the action costs are unary. Due to the complex way in which the aversion metric is defined, h_{max} is not admissible in all of our settings. We also used two simplistic heuristics, $h = 0$ and a simple one,h_{sim}, that provides the number of open goals, i.e., goals that have not yet been achieved: $h_{sim_i} = |predicates \in s_g/s_i|$.

We considered four different path-cost functions / aversion metrics for A*:

i. $g_a = \sum(cost_{a_{ij}}) + \sum \frac{1}{prob_{\alpha_{ij}}+1}$,

ii. $g_b = \sum \frac{cost_{a_{ij}}}{prob_{a_{ij}}+1}$

iii. $g_c = \prod \frac{cost_{a_{ij}}}{prob_{a_{ij}}+1}$,

iv. $g_d = \sum cost_{a_{ij}} - \prod prob_{a_{ij}}$

All the experiments' results are shown in Table I. In all cases we refer to the domain instead of the problem, as the initial states/goals of the problems do not change; the differences between the domains lie in their definition of costs. Dom_1 has a total of 6 possible solutions; Dom_2 has a total of 5712 solutions, of which only 6 are not subsumed by others. In all cases, the algorithms outputted all possible solutions (6/5712 respectively), however the resulting decision trees varied according to the selected cost function. The combination of path-cost function and heuristic that produces all the plans in the least amount of time and expands the least amount of states is shown in bold.

In regard to the heuristics, in the simpler Dom_1, in both its uniform and variant cost versions, all four heuristics, in any combination with a path-cost function, produce all the possible solutions in under 150 milliseconds. It is clear, though, that the simpler heuristics $h = 0$ and h_{sim} produce all the plans in less time than the others. Surprisingly, the efficiency of $h = 0$ and h_{sim} is also evident in the more complex Dom_2; using h_{sim} and $h = 0$ in combination with any path-cost function is significantly faster than using h_{max} or h_{add}. However, in both Dom_2^{uni} and Dom_2^{var} the use of a simple path-cost function equal to the number of actions ($g = |a|$) produces the plans faster expanding the least amount of states.

Although h_{add} and h_{max} are more informed heuristics, it seems that in our two evaluation domains heuristics that

TABLE I. EVALUATION RESULTS

	Dom_1^{uni}		Dom_1^{var}		Dom_2^{uni}		Dom_2^{var}	
Path-cost/Heuristic	Time[a]	Expanded states	Time	Expanded states	Time	Expanded states	Time	Expanded states
$g = \|a\| \ / \ h = 0$ (BFS)	123	9	95	9	338	83	320	83
$g = \|a\| \ / h_{add}$	102	9	111	9	345	83	330	83
$g = \|a\| \ / h_{max}$	110	9	115	9	332	83	359	83
$g = \|a\| \ / h_{sim}$	94	9	**88**	**9**	**317**	**83**	**316**	**83**
$g_a / h = 0$	97	10	89	10	403	106	414	104
$g_a / h = 0$	86	9	99	10	420	102	347	93
$g_c / h = 0$	92	10	88	10	404	107	386	110
$g_d / h = 0$	**91**	**9**	89	10	347	104	366	97
g_a / h_{add}	128	10	118	10	747	135	736	136
g_b / h_{add}	110	10	125	10	744	135	745	137
g_c / h_{add}	123	10	110	10	746	135	737	123
g_d / h_{add}	140	10	114	10	663	135	751	140
g_a / h_{max}	110	10	123	10	737	124	852	121
g_b / h_{max}	120	9	116	10	752	121	711	117
g_c / h_{max}	120	10	116	10	766	122	751	122
g_d / h_{max}	141	9	124	10	688	121	782	121
g_a / h_{sim}	92	10	97	10	423	116	397	113
g_b / h_{sim}	100	10	95	10	420	112	364	105
g_c / h_{sim}	93	10	91	10	394	112	419	117
g_d / h_{sim}	100	9	104	10	383	112	418	105

[a]All time measurements are in milliseconds

are easier to compute are generally more suitable. This is probably due to a combination of their fast computation and the fact that the domains are not especially computationally hard. We expect that in other domains, results may differ.

As to the use of path-cost functions, the results in Dom_1 indicate that in simple domains the choice of path-cost function is not as important as the heuristic, at least in relation to the time needed to produce all the plans. However, using $(g = |a|)$ as the path-cost function regardless of the heuristic results in the least amount of states being expanded in all four of the domains, whereas no other combination of path-cost function and heuristic achieves such result (apart from a few cases in the simplest domain, Dom_1^{uni}).

Overall, the evaluation shows that in the evaluation domains, the approach is efficient, with simple, non-time-consuming heuristics, being the method of choice. Surprisingly enough, even a blind search algorithm, such as BFS, produces good results. Moreover, at the time being, our implementation of A* is not optimized, generating all the different solutions and then dispensing those that are subsumed by others. It would be very simple to prune the paths that are subsumed by others, by checking whether all the previously added actions of the path already form another, shorter, solution. Another optimization would be to prune a path in which the planner tries to add a deterministic action $a_{xi,}$ when an action a_{xj} (that is, the same non-deterministic action with a different alternative outcome) is already in the same path. Such optimizations could result in a vastly more efficient implementation; however, we leave it for future work.

6. Conclusions and Future Work

In this paper, we presented a contingent planning framework for the problem of semantic web service composition. **MAPPPA** is an anytime probabilistic planner that produces a contingent plan by integrating alternative deterministic solutions to a determinized probabilistic problem. The determinized problem is generated without disregarding each web service's non-functional information regarding its reliability or cost of requesting and executing. The planner produced promising results in our evaluation and has already been integrated into our prototype web service composition platform, **MADSWAN**.

Our main goal in the future is to also use an existing deterministic planner to generate the alternative plans that comprise the contingent solution, so as to make use of optimizations and search techniques that are found in state-of-the-art planners. In regard to OWL-S, for the time being, we only consider exact matches between ontological concepts; we plan to extend the matches to also cover plug-in or subsumes relationships. Finally, there are OWL-S extensions that allow the efficient description of web service QoS elements; one such extension is OWL-Q [32] that supports the definition of QoS elements such as the execution time of a web service, its reliability and availability, or its cost. We plan to integrate such an extension into our approach.

ACKNOWLEDGMENT

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

REFERENCES

[1] W. Nam, H. Kil, and D. Lee 2011, "On the computational complexity of behavioral description-based web service composition," *Theor. Comput. Sci.*, vol. 412, no. 48, pp. 6736-6749, 2011.

[2] J. Hoffmann, P. Bertoli, M. Helmert, and M. Pistore, "Message-based web service composition, integrity constraints, and planning under uncertainty: A new connection," *J. Artif. Intell. Res. (JAIR)* vol. 35, pp. 49-117, 2009.

[3] H. Younes and M. Littman. (2004) *International Probabilistic Planning Competition*. [Online]. Available: http://www.cs.rutgers.edu/~mlittman/ topics

[4] S. Yoon, A. Fern, and R. Givan, "FF-Replan: A baseline for probabilistic planning," in *Proc. of the Seventeenth Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2007

[5] U. Kuter, D.S. Nau, E. Reisner, and R.P. Goldman, "Using classical planners to solve non-deterministic planning problems," in *Proc. of the Eighteenth Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pp. 190-197, 2008.

[6] J. Fu, V. Ng, F.B. Bastani, and I. Yen, "Simple and fast strong cyclic planning for fully-observable non-deterministic planning problems," in *Proc. of the Twenty-second Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 1949-1954, 2011.

[7] D. Martin, M. Burstein, J. Hobbs, O. Lassila, et al. (2004) *OWL-S: Semantic markup for web services*. [Online]. Available: http://www.w3.org/ Submission/OWL-S

[8] M. Klusch, A. Gerber, and M. Schmidt, "Semantic web service composition planning with OWLS-Xplan," in *Proc. of the First Int. AAAI Fall Symposium on Agents and the Semantic Web*, 2005.

[9] H.S. Kim, and I.C. Kim, "Mapping semantic web service descriptions to planning domain knowledge," in *Proc. of the Fourth Int. Federation for Medical and Biological Engineering (IFMBE)*, pp. 388-391, 2006.

[10] O. Hatzi, D. Vrakas, M. Nikolaidou, et al., "An integrated approach to automated semantic web service composition through planning," *IEEE Trans. Services Comput.*, vol. 5, no. 3, pp. 319-332, 2011.

[11] H. Younes and M. Littman, "PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-04-167, 2004.

[12] S. Heymans, J. Hoffmann, A. Marconi, J. Phillips, and I. Weber, *Handbook of Service Description: USDL and Its Methods*, A. Springer-Verlag, 2011, ch. 6.

[13] W3C OWL Working Group. (2012) *OWL 2 Web Ontology Language document overview (2nd ed.)*. [Online]. Available: http://www.w3.org/TR/owl2-overview/

[14] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic matching of web services capabilities," in *Proc. of the First Int. Semantic Web Conf. (ISWC)*, 2002.

[15] G. Markou and I. Refanidis, "Composing semantic web services online and an evaluation framework," *Int. J. Adv. Internet Tech*., vol. 6, no. 3-4, pp. 114-131, 2013.

[16] SemWebCentral. (2010) *OWL-S Service Retrieval Test Collection*. [Online]. Available: http://semwebcentral.org/frs/?group_id=89

[17] U. Kuter, "Pushing the limits of AI planning," in *Proc. of the Doctoral Consortium of the Fourteenth Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2004.

[18] D. Bryce, and O. Buffet, "International Planning Competition Uncertainty part: benchmarks and results," in *Proc. of the Sixth Int. Planning Competition (IPC)*, 2008.

[19] M. Ramírez, N. Yadav, and S. Sardiña, "Behavior composition as fully observable non-deterministic planning," in *Proc. of the Twenty-third Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2013.

[20] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, "Weak, strong, and strong cyclic planning via symbolic model checking," *Artif. Intell.*, vol. 147, no. 1-2, pp. 35-84, 2003.

[21] L.A.G. da Costa, P.F. Pires, and M. Mattoso, "Automatic composition of web services with contingency plans," in *Proc. of the IEEE Int. Conf. on Web Services (ICWS)*, pp. 454-461, 2004.

[22] M. Kuzu and N. Cicekli, "Dynamic planning approach to automated web service composition," *Appl. Intell.*, vol. 36, pp. 1-28, 2012.

[23] E. Onaindia, O. Sapena, L. Sebastia, and E. Marzal, "Simplanner: An execution-monitoring system for replanning in dynamic worlds," in *Proc. of the Tenth Portuguese Conf. on Artificial Intelligence (EPIA)*, pp. 393-400, 2001.

[24] J. Hoffmann and R. Brafman, "Conformant planning via heuristic forward search: A new approach," *Artif. Intell.*, vol 170, no. 6-7, pp. 507-541, 2006.

[25] J. Hoffmann, and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *J. Artif. Intell. Res. (JAIR)* vol. 14, pp. 253–302, 2001.

[26] S. Yoon, A. Fern, R. Givan, and S. Kambhampati, "Probabilistic planning via determinization in hindsight," in *Proc. of the Twenty-third Conf. on Artificial Intelligence (AAAI)*, pp. 1010-1016, 2008.

[27] S. Jiménez, A. Coles, and A. Smith, "Planning in probabilistic domains using a deterministic numeric planner, in *Proc. of the Twenty-fifth Workshop of the UK Planning and Scheduling Special Interest Group (PlanSig)*, 2006.

[28] J. Foss, N. Onder, and D. Smith, " Preventing unrecoverable failures through precautionary planning," in *Proc. of the Seventeenth ICAPS Workshop on Moving Planning and Scheduling Systems into the Real World*, 2007.

[29] R. Dearden, N. Meuleau, S. Ramakrishnan, D.E. Smith, and Washington, R. "Incremental contingency planning," in *Proc. of the Thirteenth ICAPS Workshop on Planning under Uncertainty*, 2003.

[30] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern*., vol. 4, no. 2, pp. 100–107, 1968.

[31] B. Bonet, and H. Geffner, "Planning as heuristic search," *Artif. Intell.*, vol. 129, no. 1, pp. 5-33, 2001.

[32] K. Kritikos, and D. Plexousakis, "OWL-Q for semantic QoS-based web service description and discovery," in *Proc. of the SMRR Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMRR)*,2007.