Cost-Sensitive Probabilistic Contingent Planning for Web Service Composition

George Markou, Ioannis Refanidis

Department of Applied Informatics, University of Macedonia, Egnatias str. 156, Thessaloniki, Greece
{gmarkou, yrefanid}@uom.gr

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

This article presents a cost sensitive probabilistic contingent planning approach for automated semantic web service composition, when the execution of each web service incurs some cost, whereas its alternative outcomes along with their probabilities of occurring are known in advance. The implemented planner, MAPPPA, produces a contingent plan by integrating alternative deterministic plans previously computed in an anytime fashion from a determinized version of the original problem. Importantly, while producing each plan the planner takes into account the information that each web service contains in relation to its execution cost and probability of alternative outcomes. The article presents new extensive evaluation results regarding the approach, based on three web service composition domains.

Keywords: Contingent planning; web service; Fully Observable Probabilistic (FOP) problem.

1. Introduction

Semantic web services are web services with a formal description of their capabilities, utilizing an ontology language, e.g., OWL.[1] Given a composite user requirement, an atomic web service with the desired functionalities is not usually available; in such cases, a combination of several atomic web services may be able to satisfy that user requirement. Composite web service descriptions can be defined in semantic description languages such as OWL-S[2] and WSMO.[3]

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 a partially ordered 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 effectively utilized to tackle the automated WSC problem.

Automating the process of composing semantic web services can lead to great reductions in the time and cost required to develop enterprise web-based applications. However, since web service descriptions refer to real world web services, their outcomes cannot be anticipated in all cases. As such, a non-deterministic planning approach is required in order to automate the process of web service composition. This is a difficult task, as it has been shown that solving the composition problem of non-deterministic web services with complete information is EXP-hard.[4]

In this article, we consider semantic web services at their functionality level, that is, web services are defined in regards to preconditions and effects over ontological concepts.[5] 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.

The proposed algorithm, **MAPPPA**, first presented in our previous work,[6] 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. Merging prioritizes the alternative plans according to a custom metric that combines cost and probability to achieve the goal. This metric is updated each time a plan is selected for merging, by taking into account potential common web services between the newly merged plan and the remaining ones, under the additional assumption that re-executing the same web service within a short period of time will result in the same output.

This work has been inspired by the winner of the 2004 International Probabilistic Planning Competition (IPPC-04)[7] FF-Replan,[8] as well as by subsequent planning approaches that were based on it.[9,10] It is specifically targeted for web service composition problems and differs from related work in that it takes the probabilities and the cost of the original non-deterministic actions into consideration while generating a contingent plan. The probability of such a plan being successfully executed monotonically increases as more time is allowed for planning and more plans are generated and added to it.

This article advances our work in Ref. 6 by positioning it more accurately within the literature, as well as by presenting new evaluation results, concerning both existing and new WSC problems. Several improvements are also present in all sections of the article.

The rest of the article is structured as follows. First, we formulate the problem and position our work within the literature, both regarding contingency planning and web service composition. Next, we present the MAPPPA framework, which comprises determinizing the WSC problem, generating alternative deterministic plans and merging them in a contingent one using a custom metric for prioritization. The evaluation section presents results from three planning domains, each having two variants. Finally, we conclude the article and pose directions for future work.

2. Background

Given a set of semantic web service descriptions in OWL-S format available in a registry, an automatic translation of the original WSC problem to an AI planning one is possible. Various methods have been proposed for converting OWL-S descriptions to PDDL that do not differ significantly to each other; we adopt an approach similar to those in Klusch,

Gerber and Schmidt,[11] and Kim and Kim.[12] Our translation module is based on existing source code[13] with the necessary extensions to generate of PPDDL[14] files instead of PDDL ones.

We view WSC as "planning for service chaining",[15] 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.[15, 5] Concepts are defined within relevant ontologies (in our case, in OWL) as classes, with possibly complex relationships between them, e.g., each concept can be a supertype of or be subsumed by another concept. In this work though, we only consider exact matches,[16] and plug-in or subsume 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 requires that the OWL-S web services' inputs and preconditions (*hasInput* and *hasPrecondition* in OWL-S, respectively) be transformed to a PPDDL action's preconditions, and the add effects of the action be comprised of its outputs and effects (*hasOutput* and *hasEffect* in OWL-S, respectively).[10] 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 service's response time.[2]

The current web service repository used to evaluate our approach[17] comprises all the OWL-S descriptions obtained from the latest version of the OWL-S Service Retrieval Test Collection.[18] However, the descriptions in this collection do not contain any QoS (Quality of Service) information in relation to the services' reliability, that is, 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.

2.1. *Problem formulation*

In this section we introduce the definitions and notation in relation to probabilistic planning that we will use in the rest of this article:

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 $A_D(s) = \{\alpha \in A : \gamma(s, \alpha) \neq \emptyset\}$.

Definition 2. A planning problem is a triple $P = (s_0, s_g, D)$ where $s_0 \in S$ is the initial state and $s_g \in S$ the goal. 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 problem to be FOP.

Definition 3. If we define $S_\pi \subseteq S$ as the set of states to which any 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 $s_0 \in S_\pi$ 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.[19]

Definition 4. A policy π is closed with respect to a state s iff $S_\pi(s) \subseteq S_\pi$. If the goal can be achieved 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.[20]

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 $i \neq j$.[21] In general, though, 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).[20] 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, there are problems in which strong solutions do not exist, but strong cyclic ones do.[22]

3. Related Work

In this section we present related work, both in relation to WSC and the AI planning part.

3.1. *Web service composition approaches*

Concerning WSC, there is a 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.[5] Those closer to the MAPPPA framework are the approaches in da Costa *et al.*[23] and Kuzu and Cicekli.[24]

OWLS-Xplan presented in Ref. 11 is one of the first approaches incorporating a translation between OWL-S and PDDL. The result of this translation is given as input to a hybrid planner that combines guided local search with graph planning and a simple form

of HTN decomposition. OWLS-Xplan does not take non-determinism into account. On the other hand, although in Ref. 24 a translation very similar to that of Ref. 11 and Ref. 12 is provided, that approach acknowledges the domain's non-determinism and incorporates (a modified version of) an existing PDDL planner, in specific Simplanner[25], to tackle it through interleaving planning and execution.

The WSC approach closest to our goals and problem formulation is the one presented in Ref. 23. The web services used are described in OWL-S and their IOPEs reference OWL concepts. More importantly, the planning algorithm used generates contingent plans; specifically, all the different possible (deterministic) paths are outputted, 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. 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 Ref. 23 does not make use of any QoS based cost models, while also not taking into account a service's probability of failing. Thus, all web services are considered equal during the search phase and the plans do not differentiate between two services that produce the same results.

Deng *et al.* present a QoS-based WSC approach generating the top-*k* solutions of WSC problems through the combination of backtrack search and depth-first search.[26] The original problem is split into mutually independent smaller ones and transformed into a graph search one. Then, in a manner similar to that in Ref. 23, web services are identified as either useful or irrelevant and the best *k* solutions in terms of their response time are returned. During the generation of a solution, only the web services with the highest QoS are selected in each case, instead of the entire set of candidates. Moreover, during the merging process, if a solution subgraph can generate a superset of concepts in comparison to other ones, then all the rest are discarded instead of being merged with it. Thus, some correct compositions are not generated in exchange for improved efficiency.

Rodríguez-Mier *et al.* implemented an optimal and complete backward search approach that generates multiple optimal solutions to WSC problems without taking into consideration their non-functional properties.[27] First, an extended service dependency graph representing a suboptimal solution is created dynamically. Each layer of this graph contains all the web services that can be executed using the outputs of the ones in the previous layer. This solution is then improved by removing services that are not used and combining those having equivalent outputs. A backward search based on A*[28] generates all optimal solutions that have a different number of services and runpath, using their distance to the initial state as the heuristic. Finally, the solutions found are optimized so that the parallel execution of each is maximized and the number of services in each one is

minimized. The approach does not support alternative flow control structures and requires a pre-computed table mapping inputs to the web services that cannot be executed if they are not present. So, each time the description and/or functionalities of a service are modified, the dependency graph has to be partially recomputed, hindering the approach's efficiency.

A multi-objective optimization approach is presented in Wagner *et al.*, which outputs multiple alternative solutions to WSC problems through the use of Hierarchical Workflow Graphs.[29] The problem is first encoded as a genome and then solved through a genetic algorithm, with multiple QoS elements being considered; namely the cost of web services, the required time to output a response, and their reliability. As in Ref. 26, though, the set of alternative, and in the case of Ref. 29 also non-dominated, solutions is only generated so that a single one can be chosen, and the set as a whole is not used to tackle non-determinism. Importantly, based on its evaluation, the approach could not efficiently compute optimal solutions even for simple, small, problems. Instead, an approximation of optimal solutions was generated.[29]

Finally, an approach dedicated to non-deterministic WSC utilizing AI planning was implemented in Ref. 5 that treats the application of a web service as a belief update operation. Two special cases of WSC under uncertainty were identified 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.[30]

3.2. *Contingency planning and determinizations of non-deterministic domains*

FF-Replan utilizes the FF[31] planner 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). Ref. 8 introduced two methods of creating a "determinized" version of a non-deterministic problem. 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 Definition 1, let $A = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$, then for an action $\alpha_i \in A$ with k different probabilistic outcomes, the all-outcomes determinization is the set of deterministic actions $Det\alpha_i = \{\alpha_{i1}, \alpha_{i2}, \dots, \alpha_{ik}\}$ along with a state-transition function $\bar{\gamma}$ such that for every state s, $\gamma(s, \alpha_i) = \bar{\gamma}(s, \alpha_{i1}) \cup \dots \cup \bar{\gamma}(s, \alpha_{ik})$;[9] the single-outcome determinization for action α_i simply chooses one of the actions in $Det\alpha_i$ to replace α_i.

FF-Replan uses the all-outcomes determinization process, as do NDP[9] and the planning approach in Ref. 10. 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 Ref. 10

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.

RFF is a planner that makes use of the single-outcome determinization; it does so in order to generate offline policies that provide some guarantees against failing.[32] RFF generates multiple plans from the initial state(s) to the goal and creates a policy with a low probability of causing replanning during execution through the use of Monte-Carlo simulation. The probability that a partial policy will fail is computed and in case it is sufficiently low, the current one is returned; else, the policy is expanded further.

If the execution results in a state for which the policy does not specify any actions, RFF tries to reach neighbor states of the failure state for which an action has already been specified, instead of generating a new plan to the goal. The intuition behind this choice is that a neighbor state will probably be closer to reach than the goal, thus requiring less planning time. Similarly to FF-Replan, RFF may require replanning; it is also not guaranteed to generate optimal solutions, and does not handle dead end states.

Both the all-outcomes and the single-outcome determinizations, however, ignore the probabilities attached to the probabilistic outcomes and allow the planners that utilize them to be overly optimistic. That is, planners can 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 Yoon *et al.*;[33] 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. Thus, 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 Jiménez, Coles and Smith;[34] 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 Ref. 35 for anytime contingency planning; an initial deterministic seed plan is generated that is 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.

In Dearden *et al.*, a seed plan is constructed initially though the use of a deterministic planner, assuming that each action is executed as dictated by its expected behavior.[36] 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 the MAPPPA approach and the one in Ref. 36 attempt to generate a contingent plan with the maximum possible expected utility; however, the methodologies used to generate the respective solutions are very different, as are the sources of uncertainty, which in Ref. 36 regard continuous quantities, e.g., time.

4. Alternative Plan Generation and Merging

MAPPPA is a cost-sensitive probabilistic anytime contingent planning framework for automated web service composition, consisting 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 considered possible through the assumption that 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 is allowed to execute each of the actions at most once in each of its branches; thus, the solutions do not contain infinite branches or cycles.

Thus, the output (deterministic) solutions are weak ones; each one can only succeed in the case that all the actions in it have the desired effect as their actual outcome. If there are n actions in branch $Plan_i = [a_{1j_1}, a_{2j_2}, \dots, a_{nj_n}]$, the desired outcome j_k of each having a probability of $prob_{\alpha_{kj_k}}$ of being the actual one, then each branch has a probability $Prob_{branch} = prob_{\alpha_{1j_1}} \times prob_{\alpha_{2j_2}} \times \dots \times prob_{\alpha_{nj_n}}$ of being executed successfully.

A motivational real world example of such a problem containing web services is the following: Assume a person wants to purchase an item (the problem's goal) from the web using his credit card. We consider the state in which he has available a credit card that has not yet been charged 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 ships it to the user's address, the user can 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 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.

The algorithm does not guarantee to converge to an optimal contingent plan, although it is straightforward to see that the probability of the contingent solution being successfully executed monotonically increases as each new branch is generated and added to it. In the case that all the decision tree branches achieve the goals then the result is a strong plan. In the general case, however, the decision tree is a weak plan.

The planning process can be based on any search algorithm that returns multiple deterministic plans to the goal. We used an implementation of the A^* algorithm 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, or a limit on the number of possible solutions has been reached.

The aversion metric can be integrated into the planning process both as the past path-cost function of A^* during planning, and 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 demonstrated in Section 5. When no more plans can be generated, the outputted ones are integrated into a decision tree plan, based on the algorithms shown in Fig. 1 and Fig. 2.

In the decision tree's generation function $GenerateDT$ (Fig. 1), 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).

Function $GenerateDT$
Input A set of plans P
Output A decision tree containing all the plans

1. $Sort(P, aversion\_metric)$
2. $p = P \rightarrow first\_plan()$
3. **While** ($p \neq$ null)

4. $root = new\ DTNode()$
5. $a = p \rightarrow first\_action()$
6. $root \rightarrow addAction(a)$
7. $n = root$
8. **While** $(a \neq null)$
9. **If** $non\_deterministic(a)$
10. $n.addLeftBranch = GenerateDT\left(Sort\left(ValidPlans(P, n) \rightarrow get\_branch()\right)\right)$
11. **Endif**
12. $a = a \rightarrow next\_action()$
13. **If** $(a \neq null)$
14. $n_2 = new\ DTNode()$
15. $n.addRightBranch = n_2$
16. $n = n_2$
17. $n \rightarrow addAction(a)$
18. **Endif**
19. **EndWhile**
20. $p = p \rightarrow next\_plan()$
21. **EndWhile**
22. **Return**$(root)$

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

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 in case of re-execution. 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 been updated. 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) that 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 resulting contingent plan will be strong. Therefore, for problems that only have strong cyclic solutions, the proposed algorithm is not suitable.

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** ($a \in B$)
4. **If**($result(a) \in p \equiv result(a) \in B$)
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

4.1. *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 achieved with a probability $prob_{\alpha_i} = 0.8$, $\forall i \in \{1,3,5\}$, and with a probability of $1 - prob_{\alpha_i} = 0.2$, it fails and no effect is achieved (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 $prob_{\alpha_{41}} = 0.9$, $\text{prob}_{\alpha_{42}} = 0.1$ 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 the example in Fig. 3, we use as an aversion metric, $g = \sum cost_{a_{ij}} + \sum \frac{1}{prob_{\alpha_{ij}}+1}$, where a_{ij} is a plan's action and j is a specific outcome with probability of occurrence $prob_{\alpha_{ij}}$ and incurred cost $cost_{a_{ij}}$ As all aversion metrics, g increases as more actions with high cost and low probability of occurrence are inserted into the plan.

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, whereas a_2 and a_7 remain as-is, since they were already deterministic. The planner can generate all the solutions to the problems, which in order of increasing aversion metric value, are denoted with $Plan_1$ through $Plan_6$. $Plan_1 = [a_{11}]$, is shown in Fig. 3a, consisting of a single

action that leads directly to the goal state G. $Plan_2 = [a_{2,}a_{31}]$ 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 metric; $g_{Plan_1} = 4 + \frac{1}{0.8+1} = 4.55$, whereas $g_{Plan_2} = \left(2 + \frac{1}{1+1}\right) + \left(3 + \frac{1}{0.8+1}\right) = 6.05$. The rest of the plans along with their aversion metrics are $Plan_3 = [a_{41}, a_{51}, a_{61}, a_7]$ (Fig. 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. Having sorted the plans by their aversion factors, $Plan_1$ is inserted first 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 their 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 $g_{Plan_4} = 7.05$ and $g_{Plan_6} = 20.83$ 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.

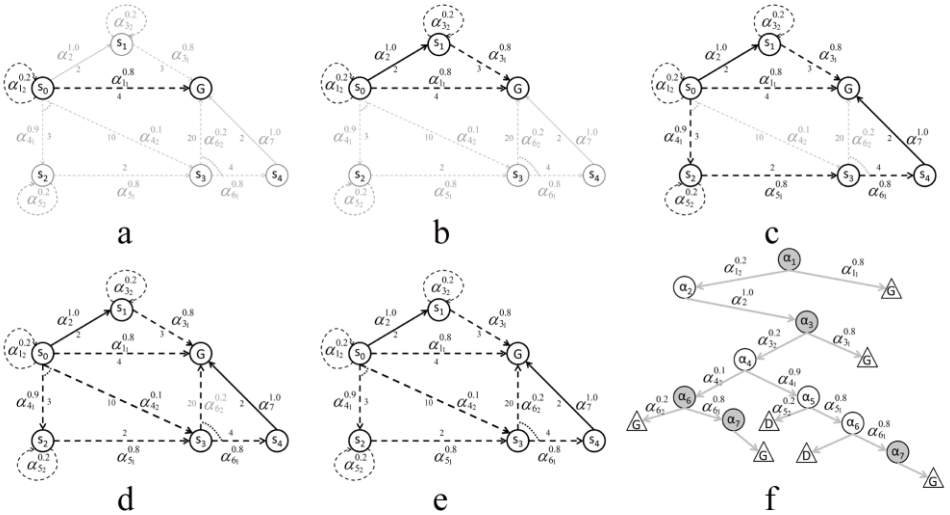

Fig. 3. (a-e) Solution steps. (f) Solution plan

5. Evaluation

In order to test the efficiency of **MAPPPA** we empirically evaluated the system using three domains; the first is the one presented in Section 4.1, namely Dom_1. The second domain, Dom_2, is a modified version of the evaluation domain from Ref. 13; 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.

In addition to the aforementioned domains taken from the literature, we created and present here for the first time a third, more complex domain, namely Dom_3, which describes a situation in which a U.S. traveler who is on a business trip is required to visit two locations at difference cities of a European country. He has to travel to both locations within the same day by car, knowing only their respective addresses; his goal is to obtain driving directions that include both the locations, as well as a visit to an ATM so that he has some cash for the trip. Moreover, he desires to know the local time of sunrise and sunset for the day he will be traveling so that he can adjust his trip accordingly for as long as there is sunlight. Finally, he would like to know whether there are any traffic related incidents along his route in order to avoid delays to his appointments, as well as the total distance he has to cover in kilometers.

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[a]. All the original web services are part of version 4.0 of the OWL-S Service Retrieval Test Collection.

All domains have two versions, one with uniform costs for the web services (Dom_x^{uni}), and one with variant costs (Dom_x^{var}). The services' costs in each one start from 1 and are taken from an exponential probability density function, that is, $(x) = e^{-(x-1)}$, for $x \geq 1$. The three domains are in increasing order of difficulty, with Dom_1 containing 7 web services, Dom_2 containing 23 web services and Dom_3 containing 44 web services. The experiments were run on a PC using a Dual-Core Intel i5 processor running at 1.6GHz, allowing at most 8 GB memory.

We examined different setup options for the evaluation; A* was the employed 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}).[37] Due to the complex way in which the aversion metric is defined, h_{max} is not admissible

[a] The created planning domains and problems, as well the web services used are available at http://ai.uom.gr/gmarkou/Files/IJAITMappaDomains/

in all of our settings. We also used two simplistic heuristics, $h = 0$ and h_{sim}. h_{sim} was calculated in a slightly different manner than in Ref. 6; here, h_{sim} is equal to the percentage of achieved goals so far, i.e., $h_{sim_i} = \frac{predicates \in s_g - predicates \in s_i}{predicates \in s_g}$, $0 \leq h_{sim_i} \leq 1$.

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

(i) $g_a = \sum\left(cost_{a_{ij}}\right) + \sum \frac{1}{\mathrm{prob}_{\alpha_{ij}}+1}$,

(ii) $\mathrm{g_b} = \sum \frac{cost_{a_{ij}}}{prob_{a_{ij}}+1}$,

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

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

The experiments' results for Dom_1, Dom_2 and Dom_3 are shown in Tables 1, 2, 3, respectively. 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 two versions of each specific domain lie in their definition of costs. The tables show the required time in milliseconds, the number of unique solutions generated and the amount of states that were expanded during the process. The best results in terms of producing the most solutions, expanding the least amount of states and requiring the least amount of time are shown in bold.

Compared to our previous results from Ref. 6, besides the variation in the computation of h_{sim} and the insertion of the new third domain, we made various optimizations in the implementation of the search algorithm. Importantly, in our previous work we tried to generate all possible solutions for the problems at hand. Since in the case of Dom_3 the number of possible solutions is extremely high and it was inefficient to compute the entire set, we modified the algorithm so as to generate a subset of them. So, MAPPPA now generates a fixed total number of solutions backwards from the goals and then prunes those solutions that are subsumed by others. The solutions returned in each run of the planner are not the same, as the plans that are generated backwards from the goals are selected stochastically. For this reason, the reported time and number of solutions in each case represent the median of 100 runs.

In the simplest domain, Dom_1, all combinations output the same number of unique solutions. Multiple combinations of path cost functions and heuristic provide similarly good results; $h = 0$ along with g_c or g_d, and h_{sim} in conjunction with g_a or g_c generate the best results in Dom_1^{uni}, and the combination of g_b with h_{sim} is the best in Dom_1^{var}.

Additionally, when $g = |a|$ is used, the number of states expanded is always the least in both versions of Dom_1. In the case of Dom_1^{var}, only the use of $g = |a|$ and the combination of g_d/h_{add} achieve this result, whereas in Dom_1^{uni}, other path cost function - heuristic combinations also expand the least amount of states, but not in all cases.

Table 1. Evaluation results – Dom_1.

	Dom_1^{uni}			Dom_1^{var}		
Path-cost/Heuristic	Time	Expanded	Solutions	Time	Expanded	Solutions
$g = \|a\| \,/\, h = 0$	1.0	**9**	**6**	3.2	**9**	**6**
$g = \|a\| \,/ h_{add}$	6.2	**9**	**6**	9.0	**9**	**6**
$g = \|a\| \,/ h_{max}$	9.4	**9**	**6**	15.4	**9**	**6**
$g = \|a\| \,/ h_{sim}$	3.2	**9**	**6**	6.4	**9**	**6**
$g_a/h = 0$	1.2	10	**6**	3.0	10	**6**
$g_b/h = 0$	3.2	**9**	**6**	6.2	10	**6**
$g_c/h = 0$	0.6	10	**6**	3.2	10	**6**
$g_d/h = 0$	**0.4**	**9**	**6**	3.1	10	**6**
g_a/h_{add}	25.0	10	**6**	30.4	10	**6**
g_b/h_{add}	28.2	**9**	**6**	23.8	10	**6**
g_c/h_{add}	30.1	10	**6**	30.0	10	**6**
g_d/h_{add}	29.6	**9**	**6**	25.4	**9**	**6**
g_a/h_{max}	29.2	10	**6**	25.8	10	**6**
g_b/h_{max}	27.2	**9**	**6**	24.8	10	**6**
g_c/h_{max}	28.0	10	**6**	22.2	10	**6**
g_d/h_{max}	22.1	**9**	**6**	28.2	10	**6**
g_a/h_{sim}	**0.4**	10	**6**	6.2	10	**6**
g_b/h_{sim}	3.8	**9**	**6**	0.6	10	**6**
g_c/h_{sim}	0.8	10	**6**	3.8	10	**6**
g_d/h_{sim}	3.0	**9**	**6**	3.6	10	**6**

Similar results are encountered in both Dom_2^{uni} and Dom_2^{var}. In both versions of the domain, if $g = |a|$ is used in combination with any heuristic, the number of states expanded is always the least (83). In contrast to Dom_1, though, using $g = |a|$ in combination with an appropriate heuristic does not necessarily lead to the best overall results. Although its use also results in the least amount of time required, with $g = |a| \,/\, h = 0$ being the best combination overall in Dom_2^{uni} and $g = |a| \,/ h_{sim}$ in Dom_2^{var}, other path cost – heuristic combinations lead to more solutions being generated. In specific, in Dom_2^{uni}, the implementation using $g_c/h = 0$ outputs 2.5 more solutions on average than $g = |a| \,/\, h = 0$, an increase of 13.66%. In Dom_2^{var}, using g_a/h_{add} produces on average 1.5 more solutions than $g = |a| \,/ h_{sim}$, an increase of 8.47%. Their use, however, requires 29.31% and 165.7% more time, accordingly.

It is also noteworthy that although using $h = 0$ and h_{sim} produces a larger set of unique solutions in Dom_2^{uni}, this result does not also hold in Dom_2^{var}. Using h_{add} and h_{max} consistently leads to similar results in terms of the produced solutions, but is also consistently inefficient in terms of time and expanded nodes required.

Table 2. Evaluation results – Dom_2.

	Dom_2^{uni}			Dom_2^{var}		
Path-cost/Heuristic	Time	Expanded	Solutions	Time	Expanded	Solutions
$g = \|a\| \ / \ h = 0$	**161.7**	**83**	18.3	177.8	**83**	17.7
$g = \|a\| \ / h_{add}$	187.2	**83**	17.9	198.1	**83**	17.9
$g = \|a\| \ / h_{max}$	199.6	**83**	18.1	193.3	**83**	18.1
$g = \|a\| \ / h_{sim}$	181.0	**83**	19.4	**144.9**	**83**	17.7
$g_a / h = 0$	207.4	106	20.1	193.3	104	15.5
$g_b / h = 0$	206.2	102	19.8	187.2	93	16.5
$g_c / h = 0$	209.1	107	**20.8**	206.0	110	15.9
$g_d / h = 0$	210.5	104	19.9	189.3	97	15.3
g_a / h_{add}	375.4	101	19.1	385.0	98	**19.2**
g_b / h_{add}	381.6	100	17.6	349.6	99	17.8
g_c / h_{add}	342.2	101	18.2	390.4	112	18.6
g_d / h_{add}	413.6	101	18.4	379.2	102	17.0
g_a / h_{max}	416.8	104	16.4	377.4	101	19.0
g_b / h_{max}	413.6	103	18.1	374.4	101	18.6
g_c / h_{max}	381.4	104	18.8	405.6	119	18.0
g_d / h_{max}	385.2	103	18.4	366.8	101	18.6
g_a / h_{sim}	204.5	106	19.2	206.0	104	16.2
g_b / h_{sim}	209.0	102	19.7	195.2	93	15.9
g_c / h_{sim}	184.2	107	20.1	204.3	110	16.2
g_d / h_{sim}	218.6	104	19.7	182.4	97	16.5

Finally, the experimental results concerning Dom_3 are consistent with the ones related to the computationally easier problems. In specific, in both versions of the domain using $g = |a| \ / \ h = 0$ expands the least amount of states (1186) and requires the least amount of time. However, other path cost function and heuristic combinations manage to either generate more solutions on average, or expand the same amount of states. In Dom_3^{uni}, g_a / h_{sim} has a very similar performance to $g = |a| \ / \ h = 0$, generating 6.46% more solutions, while requiring 7.34% more time. The use of $g_c / h = 0$ results in the 1186 states being expanded, while generating 1.3 more solutions, while g_c / h_{sim} also expands 1186 states, but outputs 1.6 less solutions on average than $g = |a| \ / \ h = 0$. Finally, using $g_c / h = 0$ requires 4.75% more time, but produces 6.46% more solutions.

In both version of Dom_3, the use of h_{add} results in the generation of the most unique solutions (on average). In Dom_3^{uni}, g_a / h_{add} outputs 22.8 solutions, an increase of 13.43% over $g = |a| \ / \ h = 0$; in Dom_3^{var}, g_c / h_{add} manages to outperform $g = |a| \ / \ h = 0$ by generating 11.73% more solutions. However, as in Dom_1 and Dom_2, using h_{add} or h_{max} results in a considerable increase in time.

In general, the experimental evaluation showed that our approach is promising. The planner is able to produce multiple unique solutions to the problems at hand, even when the problem is computationally hard and its search space is large. We experimented with problems containing almost double the number of web services than in previous work, and

even in this case, the planner managed to produce more than 20 different plans in under 2.3 seconds.

Table 3. Evaluation results – Dom_3.

	Dom_3^{uni}			Dom_3^{var}		
Path-cost/Heuristic	Time	Expanded	Solutions	Time	Expanded	Solutions
$g = \|a\| \,/\, h = 0$	**2279.3**	**1186**	20.1	**2401.2**	**1186**	21.3
$g = \|a\| \,/ h_{add}$	5488.4	**1186**	19.9	5388	**1186**	21.0
$g = \|a\| \,/ h_{max}$	5368.0	**1186**	20.0	5408.5	**1186**	21.0
$g = \|a\| \,/ h_{sim}$	2388.2	**1186**	19.5	2405.3	**1186**	18.6
$g_a/h = 0$	2422.1	1455	18.1	2452.0	1577	21.1
$g_b/h = 0$	2435.0	1455	20.7	2449.9	1512	18.2
$g_c/h = 0$	2387.7	**1186**	21.4	2479.7	1511	17.3
$g_d/h = 0$	2433.1	1455	19.7	2527.9	1660	18.0
g_a/h_{add}	62260.6	2323	**22.8**	65129.8	2419	19.0
g_b/h_{add}	62616.2	2337	18.4	65433.2	2275	17.8
g_c/h_{add}	64290.8	2238	17.4	64362.8	2370	**23.8**
g_d/h_{add}	62656.2	2335	14.8	68249.1	2455	16.4
g_a/h_{max}	40933.4	1610	16.1	43533.4	1752	16.4
g_b/h_{max}	43978.0	1692	19.2	44705.7	1713	20.1
g_c/h_{max}	39267.4	1464	19.0	44681.1	1711	19.0
g_d/h_{max}	42028.2	1598	20.9	51468.2	1924	14.3
g_a/h_{sim}	2446.7	1455	21.4	3109.0	1577	19.2
g_b/h_{sim}	2479.9	1455	18.4	2532.6	1512	15.9
g_c/h_{sim}	2367.6	**1186**	18.5	2509.6	1511	18.6
g_d/h_{sim}	2415.9	1455	20.3	2596.7	1652	19.4

Moreover, in comparison to our previous implementation, the optimizations we made resulted in a far superior performance. Table 4 presents the time required by the best combination of path cost function and heuristic for each variation of Dom_1 and Dom_2 of our previous version of the planner and the current one. In the easier domain Dom_1 the current implementation only requires a fraction of the time in comparison to the old one, whereas in Dom_2 about half the time needed previously.

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. However, in general, the use of a simple path-cost function equal to the number of actions ($g = |a|$) produces plans faster expanding the least amount of states in almost all problems. It is also clear that when using the simpler heuristics $h = 0$ and h_{sim} the planner is able to generate solutions to the problem at hand in less time than when using more complex ones.

Although h_{add} and h_{max} are more informed heuristics, it seems that, in our evaluation domains, heuristics that are easier to compute are generally more suitable. This is probably due to a combination of their fast computation and the relative

computational hardness of our evaluation domains. We expect that in other domains, results may differ. However, the usefulness of such heuristics was partly shown in the presented evaluation as well; in Dom_2^{var}, the use of h_{add} and h_{max} resulted in a more diverse set of solutions being produced than any other heuristic.

Table 4. Efficiency gains in comparison to Ref. 6 due to code optimization.

	Previous implementation	**Current implementation**
Domain	Minimum time	Minimum time
$\boldsymbol{Dom_1^{uni}}$	86	**2.1**
$\boldsymbol{Dom_1^{var}}$	88	**0.6**
$\boldsymbol{Dom_2^{uni}}$	317	**161.7**
$\boldsymbol{Dom_2^{var}}$	316	**144.9**

6. Conclusions and Future Work

In this article, we presented a contingent planning framework for the problem of semantic web service composition. **MAPPPA** is a cost-sensitive probabilistic planner that produces a contingent plan by integrating alternative deterministic solutions to a determinized version of the 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. Having made optimizations to our initial implementation we provide extensive evaluation of our approach on two domains from the literature, as well as on a new, more complex one, created by the authors for the needs of this article. The experimental results indicate that the planner's efficiency is currently significantly better.

At the time being though, our implementation of A* is not yet fully optimized; it would be 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. Such optimizations could result in an even more efficient implementation. Our main goal in the future, however, is to replace A* with an established deterministic planner to generate the alternative plans to be merged in the decision tree of 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[38] 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.

Acknowledgments

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. W3C OWL Working Group, *OWL 2 Web Ontology Language document overview (2nd ed.)* (2012), http://www.w3.org/TR/owl2-overview/.
2. D. Martin, M. Burstein, J. Hobbs, O. Lassila, et al., *OWL-S: Semantic markup for web services* (2004), http://www.w3.org/ Submission/OWL-S.
3. H. Lausen, A. Polleres and D. Roman, *Web Service Modeling Ontology (WSMO)* (2005), http://www.w3.org/Submission/WSMO.
4. W. Nam, H. Kil and D. Lee, On the computational complexity of behavioral description-based web service composition, *Theor. Comput. Sci.* **412**(48) (2011) 6736-6749.
5. 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)* **35** (2009) 49-117.
6. G. Markou and I. Refanidis, Anytime Planning for Web Service Composition via Alternative Plan Merging, *Proc. IEEE Int. Conf. on Tools with Artificial Intelligence* (2014), pp. 91-98
7. H. Younes and M. Littman, *International Probabilistic Planning Competition* (2004), http://www.cs.rutgers.edu/˜mlittman/ topics.
8. S. Yoon, A. Fern and R. Givan, FF-Replan: A baseline for probabilistic planning, *Proc. of the 17th Seventeenth Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2007
9. U. Kuter, D.S. Nau, E. Reisner and R.P. Goldman, Using classical planners to solve non-deterministic planning problems, *Proc. of the 18th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pp. 190-197, 2008.
10. J. Fu, V. Ng, F.B. Bastani and I. Yen, Simple and fast strong cyclic planning for fully-observable non-deterministic planning problems, *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence* (2011), pp. 1949-1954.
11. M. Klusch, A. Gerber and M. Schmidt, Semantic web service composition planning with OWLS-Xplan, *Proc. of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web* (2005).
12. H.S. Kim and I.C. Kim, Mapping semantic web service descriptions to planning domain knowledge, *Proc. of the 4th Int. Federation for Medical and Biological Engineering* (2006), pp. 388-391.
13. O. Hatzi, D. Vrakas, M. Nikolaidou, et al., An integrated approach to automated semantic web service composition through planning, *IEEE Trans. Services Comput.* **5**(3) (2011) 319-332.
14. H. Younes and M. Littman, *PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects*, Tech. Rep. CMU-CS-04-167, School of Computer Science (Carnegie Mellon University, 2004).
15. S. Heymans, J. Hoffmann, A. Marconi, J. Phillips and I. Weber, Semantic web services fundamentals, in *Handbook of Service Description: USDL and Its Methods*, (Springer-Verlag, 2011), pp. 135-158.
16. M. Paolucci, T. Kawamura, T. Payne and K. Sycara, Semantic matching of web services capabilities, *Proc. of the 1st Int. Semantic Web Conf.* (2002).

17. G. Markou and I. Refanidis, Composing semantic web services online and an evaluation framework, *Int. J. Adv. Internet Tech*. **6**(3-4) (2013) 114-131.
18. SemWebCentral, *OWL-S Service Retrieval Test Collection* (2010), http://semwebcentral.org/frs/?group_id=89.
19. U. Kuter, Pushing the limits of AI planning, *Proc. of the Doctoral Consortium of the 14th Int. Conf. on Automated Planning and Scheduling* (2004).
20. D. Bryce and O. Buffet, International Planning Competition Uncertainty part: benchmarks and results, *Proc. of the 6th Int. Planning Competition* (2008).
21. M. Ramírez, N. Yadav and S. Sardiña, Behavior composition as fully observable non-deterministic planning, *Proc. of the 23rd Int. Conf. on Automated Planning and Scheduling* (2013).
22. A. Cimatti, M. Pistore, M. Roveri and P. Traverso, Weak, strong and strong cyclic planning via symbolic model checking, *Artif. Intell*. **147**(1-2) (2003) 35-84.
23. L.A.G. da Costa, P.F. Pires, and M. Mattoso, Automatic composition of web services with contingency plans, *Proc. of the 2nd IEEE Int. Conf. on Web Services* (2004), pp. 454-461.
24. M. Kuzu and N. Cicekli, Dynamic planning approach to automated web service composition, *Appl. Intell*. **36** (2012) 1-28.
25. E. Onaindia, O. Sapena, L. Sebastia and E. Marzal, Simplanner: An execution-monitoring system for replanning in dynamic worlds, *Proc. of the 10th Portuguese Conf. on Artificial Intelligence* (2001), pp. 393-400.
26. S. Deng, L. Huang, W. Tan and Z. Wu, Top-K automatic service composition: A parallel method for large-scale service sets, *IEEE Trans. Autom. Sci. Eng*. **11**(3) (2014) 891-905.
27. P. Rodríguez-Mier, M. Mucientes, J.C. Vidal and M. Lama, An optimal and complete algorithm for automatic web service composition, *Int. J. Web. Serv. Res.* **9**(2) (2012) 1-20.
28. 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*. **4**(2) (1968) 100–107.
29. F. Wagner, A. Klein, B. Klopper, F. Ishikawa and S. Honiden, Multi-objective service composition with time-and input-dependent QoS, *Proc.of the 19th IEEE Int. Conf. on Web Services* (2012).
30. J. Hoffmann and R. Brafman, Conformant planning via heuristic forward search: A new approach, *Artif. Intell.***170** (2006) 507-541.
31. J. Hoffmann and B. Nebel, The FF planning system: Fast plan generation through heuristic search, *J. Artif. Intell. Res.* **14** (2001) 253–302.
32. F. Teichteil-Koenigsbuch, G. Infantes and U. Kuter, RFF: A robust, FF-based MDP planning algorithm for generating policies with low probability of failure, *Proc. of the 3rd Int. Planning Competition* (2008).
33. S. Yoon, A. Fern, R. Givan and S. Kambhampati, Probabilistic planning via determinization in hindsight, *Proc. of the 23rd Conf. on Artificial Intelligence* (2008), pp. 1010-1016.
34. S. Jiménez, A. Coles and A. Smith, Planning in probabilistic domains using a deterministic numeric planner, *Proc. of the 25th Workshop of the UK Planning and Scheduling Special Interest* Group (2006).
35. J. Foss, N. Onder and D. Smith, Preventing unrecoverable failures through precautionary planning, *Proc. of the 17th ICAPS Workshop on Moving Planning and Scheduling Systems into the Real World* (2007).
36. R. Dearden, N. Meuleau, S. Ramakrishnan, et al., Incremental contingency planning, *Proc. of the 13th ICAPS Workshop on Planning under Uncertainty* (2003).
37. B. Bonet and H. Geffner, Planning as heuristic search, *Artif. Intell*. **129**(1) (2001) 5-33, 2001.
38. K. Kritikos and D. Plexousakis, OWL-Q for semantic QoS-based web service description and discovery, *Proc. of the SMRR Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web* (2007).