POMDP PLANNING IN SERVICE COMPOSITION

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Min Chen

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

October 2011

Fargo, North Dakota

North Dakota State University
Graduate School

Title

POMDP PLANNING IN SERVICE COMPOSITION

By

MIN CHEN

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

North Dakota State University Libraries Addendum

To protect the privacy of individuals associated with the document, signatures have been removed from the digital version of this document.

ABSTRACT

Chen, Min, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, October 2011. POMDP Planning in Service Composition. Major Professor: Dr. Simone Ludwig.

Automated Web service composition is becoming an increasingly important research topic. It describes the automatic process of composing atomic services into a chain of services that provide a specific functionality that could not be achieved by atomic services alone. A service-oriented environment is dynamic in nature, meaning that services come online and go offline, services change their functionality, etc. Current classical AI planning techniques suffer from the assumption of deterministic behavior of Web services and require execution monitoring for service failures. To address this concern, Partially Observable Markov Decision Processes (POMDP) in workflow composition has been used. POMDPs provide a powerful mathematical framework for planning and decision making in noisy and/or dynamic environments. POMDPs have been widely used to model many real-world problems. This thesis develops, implements and analyzes the suitability of the POMDP approach to the Web service composition problem.

ACKNOWLEDGMENTS

There are many people I need to thank you for all the support and guidance I have been given for this thesis and my study in NDSU.

First and foremost, I need to thank you my advisor, Dr. Simone Ludwig, who I have had the pleasure of working with. Whether providing me with research ideas, making the complicated problems simple, helping me analyze results, or correcting my writing, her contribution was invaluable.

Many thanks to my previous advisor, Dr. Kendall Nygard, for the help with choosing courses and finding my research interests, as well as with my plan of study.

I need to thank you, Dr. Imad Rahal, who encouraged me to continue my graduate study in the area of Computer Science and keeps giving me advice.

DEDICATION

To my family and friends

TABLE OF CONTENTS

ABSTRACT iii

ACKNOWLEDGMENTS iv

DEDICATION v

LIST OF TABLES viii

LIST OF FIGURES ix

1. INTRODUCTION 1

    1.1. Introduction of Service Computing 1

    1.2. Motivation 4

    1.3. Contribution 4

    1.4. Outline of Thesis 5

2. AUTOMATED SERVICE COMPOSITION AND PLANNING 6

    2.1. Automatic Web Service Composition 6

    2.2. Approaches on Web Service Composition 8

        2.2.1. Workflow-based Web Service Composition 8

        2.2.2. Model-based Web Service Composition 9

        2.2.3. Theoretical-based Web Service Composition 10

        2.2.4. AI Planning Web Service Composition 10

    2.3. Discussion 12

3. POMDP AND DIFFERENT ALGORITHMS 14

    3.1. Partially Observable Markov Decision Process (POMDP) 14

        3.1.1. Problem Formulation 14

3.1.2. Solving POMDP Problems 15
3.2. Algorithms on Solving POMDP 18
3.2.1. Enumeration Algorithm............................ 19
3.2.2. Witness Algorithm 21
3.2.3. Two-Pass Algorithm 23
3.2.4. Incremental Pruning Algorithm 23
3.3. Comparison of the Four Algorithms 25
4. EXPERIMENTS AND RESULTS 27
4.1. A Case Study 27
4.2. Development of POMDP Composition Framework 29
4.3. Experiments and Results 34
4.4. Discussion 38
5. CONCLUSIONS.. 40
5.1. Summary 40
5.2. Conclusion.. 40
5.3. Future Work 41
BIBLIOGRAPHY... 42
APPENDIX A. POMDP SOLVER IN JAVA.......................... 47
APPENDIX B. POMDP INPUT FILE FORMAT 51
APPENDIX C. JAVA CODE FOR ACCESSING MYSQL 56

LIST OF TABLES

Table | Page

1 Simple example of a service composition. 4

2 Basic model for service composition. 6

3 A framework of service composition system by Rao [6]. 7

4 A POMDP agent decomposed into two parts [14]. 16

5 A t-step policy tree [27]. 18

6 Convex piecewise linear function on a continuous belief state with S = 2. 20

7 Motivating scenario with example probability values by Doshi [4]. 29

8 Variations in workflow execution as response to Web Service invocation changes. 30

9 Four scenarios of the manufacturer model by Doshi. 31

10 Framework of service composition using POMDP. 31

11 Screen shot of invocation table in MySQL database. 32

12 Screen shot of service table. 32

13 Comparison of execution time of the four algorithms. 36

14 Trends of the execution time by using the proposed four algorithms. . . . 36

15 Vector convergences by using Incremental Pruning Algorithm. 37

16 Relation between states and vectors of the optimal policies. 38

17 Speed of vector convergence with horizon=20 using Incremental Pruning. 39

LIST OF FIGURES

Figure | Page

1 Complexity comparison of the four algorithms [27]. 26

2 A sample policy which is mapping the States (S) to Actions (A). 33

3 Action comparison between the four scenarios in Figure 9. 34

4 Required iterations and vectors to generate Scenario 4 (see Figure 9). . . 38

1. INTRODUCTION

In this chapter, a brief introduction on service computing and the motivation of this research project will be given. The contribution and the overall organization structure of this thesis will be described.

1.1. Introduction of Service Computing

Services computing is a discipline combining science and technology. In particular, it is a bridge for the gap between Business Services and IT Services. The core technology suite includes Web services and service-oriented architecture (SOA), cloud computing, business consulting methodology and utilities, business process modeling, transformation and integration [1]. Services computing covers the whole life-cycle of services innovation research. It includes business componentization, services modeling, services creation, services realization, services annotation, services deployment, services discovery, services composition, services delivery, service-to-service collaboration, services monitoring, services optimization, as well as services management. The major goal of services computing is to enable services and computing technology to perform more efficiently and effectively [1].

Services are components that support the composition of distributed applications, and are offered by service providers/organizations that procure the service implementations, supply their service descriptions, and provide related technical and business support. Since services are offered by different businesses and communicate via the Internet, they provide a distributed computing infrastructure for both intra- and cross-enterprise application integration and collaboration. Descriptions of services are used to advertise the capabilities, interface, behavior, and quality of the service. Service descriptions are necessary for discovery, selection, binding, and composition of services. The service capability description presents the conceptual

purpose and expected results of the service. The service interface description publishes the service's input, output, and message types. The service behavior description describes the "expected" behavior of a service during its execution. Finally, the Quality of Service (QoS) description publishes important functional and non-functional service quality attributes, such as cost, performance metrics, security attributes, reliability, scalability, and availability. Service clients, and service workflow aggregators utilize service descriptions to achieve their objectives [2].

Leyman [3] have identified the following components within service-oriented computing:

- **Service foundations:** consists of service-oriented middleware which provides the runtime SOA infrastructure connecting heterogeneous components and systems and providing access to services over various networks such as the Internet. The middleware allows application developers to define basic service functionality describing, publishing, searching and binding of services. The overall use of the service-oriented architecture is for services to describe their capabilities in a registry, so that a user can search and discover the appropriate service based on the capabilities.

- **Service composition:** Most of the time a single service will not provide the functionality needed. Therefore, the composition of several services is necessary to produce the needed functionality. In addition, in some domains complete process flows, also referred to as workflows, are needed to provided the envisioned capability.

- **Service management and monitoring:** Service management is responsible for the installation and configuration of metrics such as Quality of Service (QoS) during service execution, in order for the system to be able to monitor the

events of the services. Service monitoring includes the collection of information produced by the services and business process, viewing the statistics of process instances including the number of instances in each of the four states (running, suspended, aborted or completed), in order to intervene in the different states of the process instances.

- **Service design and development:** The design of services is a very important component of a service-oriented architecture. A well-designed infrastructure can increase the efficiency of businesses by providing reusable, independent, automated processes as services and mechanisms to effectively use them.

Service Composition combines and reuses independently developed component services. A composite service is a collection of services combined together to achieve a desired functionality. The task of automatic service composition has been split into four phases: planning, discovery, selection, and execution [1]. The first phase involves generating a plan, i.e., all the services and the order in which they are to be composed. The plan can be generated manually, semi-automatically, or automatically. The second phase involves discovering services using the plan. In the past literature depending on the approach, often planning and discovery are combined into one component. After all services are discovered, the selection phase involves selecting the optimal solution from the available potential solutions based on non-functional properties such as the QoS properties. The last phase involves executing the services given the plan, as well as in the event one or several services are not available anymore; an alternate plan/solution has to be found. In this thesis the planning part of the service composition process is further looked at. To give an example of a service composition suppose we are looking for a service to make travel arrangements, i.e., book a flight, a hotel, and make a rental car reservation. The directory of services contains *Reserve Flight*, *Reserve Hotel*, and *Reserve Car* services. As we know service

Reserve Flight has to be executed first so that some output, e.g. arrival date and time is known and can be used as the input for the next service, which is *Reserve Hotel* in our case followed by the service *Reserve Car*, which uses the output of *Reserve Hotel* as its input. Figure 1 shows this example sequential composition as a directed acyclic graph.

Figure 1. Simple example of a service composition.

1.2. Motivation

The motivation for this research is the following. The planning approach to service composition has been heavily researched. However, a new technique to be applied to the service composition-planning problem called Partially Observable Markov Decision Process (POMDP) has been proposed and an investigation of the suitability of this approach on the composition problem has been done. Related work has been successfully applying Markov Decision Process (MDP) before. MDP is fully observable for the belief state(s). However, given the dynamic nature of service-oriented environments, i.e., services go online and offline, new services become available, and existing services change their characteristics, the environment is definitely not fully observable. Therefore, POMDP will be applied to the composition problem and will be investigated for its suitability in service-oriented environments.

1.3. Contribution

The primary contribution of this thesis is using decision-theoretic planning called POMDP to automated Web service composition based on workflow composition. The advantage of using decision-theoretic planning is that it models the uncertainty which exists in the process and produces a plan which can optimally

balance the expected risks and rewards [4].

It is often the case that a careful study of current techniques can give rise to new ideas. In light of this purpose, a POMDP composition framework has been developed to test the performance of using POMDP in automated Web service composition. The quality of this new approach has been tested from the investigations.

1.4. Outline of Thesis

The thesis is structured in the following manner. In Chapter 2, a description on automated Web service composition as well as the different planning approaches on Web service composition will be provided. POMDP, POMDP solving and four exact algorithms on POMDP planning will be described in Chapter 3. A case study on a supply chain scenario, results and observations will be presented in Chapter 4. Finally, conclusions and future work will be discussed in Chapter 5.

2. AUTOMATED SERVICE COMPOSITION AND PLANNING

In this chapter, what automatic service composition is and the framework of service composition will be briefly described. Additionally, an overview on the different techniques which have been applied in web service composition so far will be introduced.

2.1. Automatic Web Service Composition

A Web service can be implemented by invoking other web services from different companies or organizations. A *composite service* is a Web service implemented by invoking other Web services. A *basic service* or *atomic service* is a Web service implemented by accessing the local system [5]. Nowadays, increased number of companies and organizations implement their core business over the Internet. An atomic service as the basic unit of operation in a Service-oriented Architecture (SOA) is not able to handle complex tasks. Therefore, to logically connect several atomic services in order to fulfill the request from users is desirable. See Figure 2, Web service composition, simply, is the act of connecting several basic Web services as one composite service.

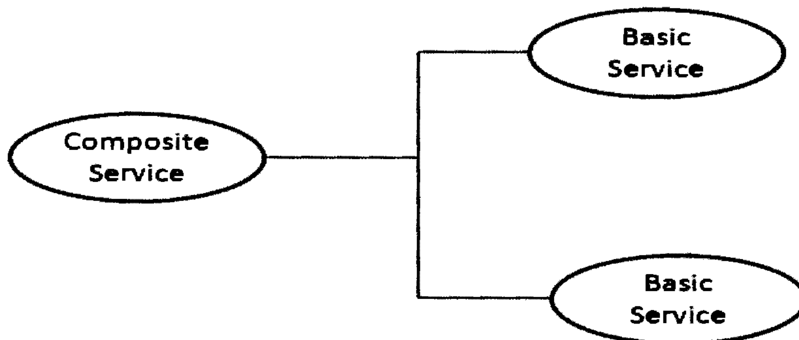

Figure 2. Basic model for service composition.

Generally, a service composition system as shown in Figure 3 have two participants, service provider and service requester. The service provider provides services to

the service requesters to use. The **Translator** translates the internal languages used by the **Process Generator** and external languages used by the servicer requesters. The **Process Generator** tries to generate plans that compose the services in the **Service Repository** in order to fulfill the requests from service requesters. The **Evaluator** evaluates the plans and decides the best one for execution in the situation that more than one plan has been found. The **Execution Engine** executes the plan chosen by the **Evaluator** and returns the result to the service providers.

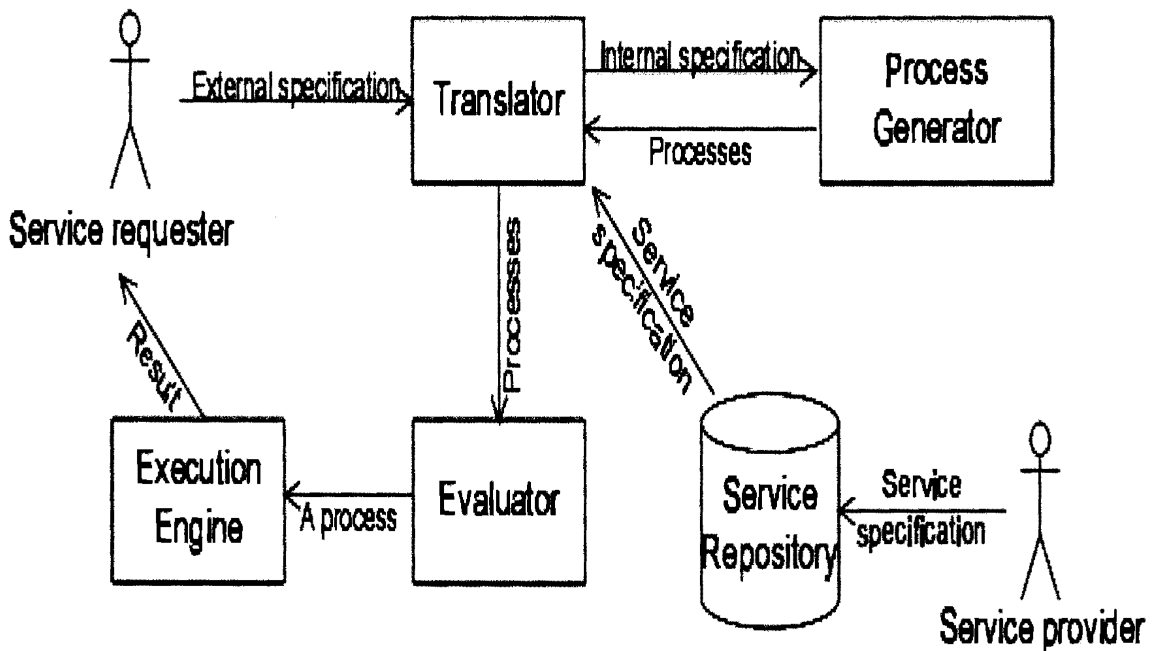

Figure 3. A framework of service composition system by Rao [6].

Specifically, automatic Web service composition involves effectively connecting and reusing atomic services to achieve the desired goal and can be split into distinct phases that constitute a complete automatic composition approach [6, 7]:

- **Presentation of Atomic Services**: this phase deals with advertisement of atomic services. Languages like UDDI [8] or DAML-S [9] can be used to advertise the atomic services at a global market places by the service providers.

- **Translation of Internal Processes**: normally, the external and internal

services languages are different. External languages are used by the users in the sense that the users can express in a relatively easy manner. Internal languages such as logical programming languages need to be more formal and precise in order to effectively describe the internals of a service. Thus, the translation between external languages and internal languages need to be developed.

- **Generation of composition process model**: based on service requester's requirements, a process generator tries to solve the requirements by composing the atomic services. This phase is the heart of the composition process since it involves generating the composition process.

- **Evaluation**: it is possible to generates more than one composite service since many services have the same or similar functionalities. This phase evaluates the composite services by their overall utilities using the utility functions.

- **Execution**: this phase involves the execution and deployment of a newly composite service after a unique composite process is selected. It can also provide a framework for monitoring an executing service.

2.2. Approaches on Web Service Composition

As the number of Web services increases dramatically, as well as the demands and updates of business environment on newer applications [5, 6, 10], it is impossible to deal with the whole process of Web service composition manually. Therefore, opportunities and needs for automated or semi-automated Web service composition technologies have increased in recent years. Generally, automated web service composition approaches can be grouped into workflow-based, model-based, theoretic-based and planning-based approaches. The four approaches will be discussed separately in the following.

2.2.1. Workflow-based Web Service Composition

Workflow can be described as movement of tasks that can be automated by invoking applications or external services through a business process. Simply, the workflow composition is to arrange activities to form a business process [11]. Workflow technology was first used by the business community. Recently, this concept has been applied to automating large-scale science [12]. A composite process is viewed as a workflow in workflow-based composition. There are two types for workflow generations in service composition: static and dynamic. The static method requires the service requester to pre-define an abstract process model before service composition planning starts. EFlow [6] which is a platform for specification, composition and management of composite series, uses a static method for workflow generation. On the other hand, the dynamic method creates the process model and selects single services automatically [6, 12]. Polymorphic Process Model (PPM) [6] combines the static and dynamic methods on service compositions.

An example of a workflow-based approach is a framework that automatically constructs a Web service composition schema from a high-level perspective [15]. The input is fed to an abstract workflow generator that attempts to create an abstract workflow that satisfies the objective by either using already generated workflows or subsets of them that are stored in a repository or by performing backtracking to find a chain of services that satisfy the objective. The identified abstract workflow is then instantiated by either finding existing services through a matchmaking algorithm that matches inputs and outputs and binds them to the workflow, or by recursively calling the abstract workflow generator if no service can be found for an activity.

2.2.2. Model-based Web Service Composition

Model-based approaches are manifold. An e-service composition tool [13, 14] implements an automated service composition using Finite State Machines (FSMs). The behavior of a service is modeled using two schemata; an external schema specifies

its exported behavior and an internal schema contains information on which a service instance executes each given action that is part of the overall service process. When the composition is synthesized, the external FSM models of the available services and target service are transformed to modal logic formulas in Deterministic Propositional Dynamic Logic (DPDL). If the resulting set of equations is satisfiable, then a FSM for the target service exists. The synthesized FSM can then be converted to a BPEL process and executed in an engine. Some recent work of the authors, replace the FSMs with transition systems and use a simulation to virtually compute all possible composition at once.

2.2.3. Theoretical-based Web Service Composition

Theoretical-based approaches use process algebraic languages, such as CCS (Calculus of Communicating Systems) [18] or pi-calculus [19] since the process specifications included in Semantic Web frameworks, such as WSMO and SWSF, are formally grounded on process algebras. Pi-calculus as introduced by Milanovic [20], can be used to describe, as well as compose Web services, and the processes can be sequences of other processes, parallel compositions, recursive executions, or choices between operations, and is therefore able to express all basic composition schemas. Information exchange is done between processes as input and output artifacts, which are exchanged on channels.

2.2.4. AI Planning Web Service Composition

AI Planning which originated from control theory and classic search methods has been applied to web service composition. Generally, a planning problem can be described as a tuple (S, s_0, G, A, Γ), where S is a set of possible states, s_0 is the initial state, G is a set of possible goal states, A is a set of actions from one state to another state, Γ is a translation relation for when a particular action is taking by given state [12, 6, 10]. Most of the planning approaches rely on a general model of

state-transition systems [21].

Lots of AI planning techniques have been applied to web service compositions: situation calculus, domain-independent heuristic, hierarchical task network planning, planning based on Markov Decision Process and planning as Model checking [21].

- **Situation calculus**: the idea is that the software agents can reason about Web services to perform automatic Web service composition by a high-level logic programming language [6, 21] to build on top of the situation calculus. The way of describing the state of world in this approach differs from other planning techniques [6, 21]. Mcllraith et. al [6] adapt and extend Golog which is a logic programming language for automatic Web service composition.

- **Domain-independent heuristic**: it builds a regression-match graph to do a backward search from the goal description [21]. The regression-match graph is an AND/OR tree. McDermott [6, 23, 24] extended Planning Domain Definition Language (PDDL) for automatic construction of Web service. McDermott introduced a new concept "value of an action" which enables one to distinguish the information transformation from state change [6, 21].

- **Hierarchical task network (HTN) planning**: it decomposes tasks recursively into smaller subtasks until it reaches primitive tasks. HTN planning can achieve good performance in realistic domains since it incorporates and exploits domain-independent control knowledge [21]. SHOP2 [22] is based on ordered task composition which is a modified version of HTN. RETSINA [23] is another planner based on HTN planning. The unexpected situations can be handled by re-planning.

- **Planning based on Markov Decision Process (MDP)**: it is introduced by Doshi [4] and Gao [24] to present a policy-based approach for dynamic Web

service composition. MDP is stochastic, sequential and fully observable. Doshi used BPEL4J API to convert a policy into a workflow. Gao used MDP based on QoS criteria to create the framework to represent Web service composition [24].

- **Planning as Model checking**: it deals with non-determinism, partial observability and extended goals [21]. This technique is used for verification of hardware and software systems to determine whether a property holds in a certain system formalized as a finite state model [6, 21]. MIPS [21] is based on binary decision diagram (BDD) is one implementation. ASTRO [25] extends existing platforms with automated Web service composition and execution monitoring functionalities [21].

2.3. Discussion

Since Web services involve sending and receiving messages, one way to describe the state is using a set of documents to represent the most accurate information of the state which is fully known. However, we faces two problems in describing the state: partially observability of state and ambiguity in state description [12]. Traditional planner like STRIPS-style planner encoded the task in terms of logical statements by their preconditions and effects [4, 10]. However, the classical planning assumes a static service environment with deterministic service outcomes.

Composite service is similar to a workflow [26] since it connects a set of basic services together with the control and data flow among the services. Similarly, a workflow has the flow of work items. On the other hand, AI planning is required to generate the plan automatically with the general assumption that each Web service can be specified by its preconditions and effects in the planning context [6]. Service composition is the problem of automatically arranging the services in a particular order to achieve one or more predetermined goal(s). The results of service composition

are usually referred to as workflows.

Workflow-based and AI planning-based approaches are interesting in the specific case of service composition. The first approach bases on the fact that the composite services are conceptually similar to workflows. On the other hand, AI planning is generating a plan which begins with an initial state by taking a series of actions to reach the goal state [7]. Hence, applying AI planning techniques to do workflow composition in dynamic service environments seems to be feasible.

3. POMDP AND DIFFERENT ALGORITHMS

In this chapter, the formulation of POMDP and the four exact algorithms: Enumeration, Witness, Two-pass and Incremental Pruning will be briefly introduced.

3.1. Partially Observable Markov Decision Process (POMDP)

If there is uncertainty in what a specific decision will do to an environment, we can call the environment *probabilistic* or *stochastic.* To make a decision when the result is not predictable in a stochastic environment is more difficult than in a deterministic environment. If the current state of the environment is unknown or incomplete in some way, we call the environment *partially observable.* By comparing the uncertainty provided by a stochastic domain, partial observability is more challenging since the uncertainty builds throughout the whole planning [27].

POMDP models an agent that makes a sequence of decisions under uncertainty to maximize its utility in the effects of given actions and given current states [28]. The POMDP model is an extension to the MDP model by adding the partial observability to it. In a partially observable environment, the agent receives observations which are insufficient to guarantee state knowledge at each step to the current state. In the following, the formulation of POMDP and how to solve the POMDP problem in general will be described.

3.1.1. Problem Formulation

POMDP can be formally defined as a tuple $(S, A, O, T, Z, R, \gamma, \Pi)$ [27, 30, 31, 32], where

- S is a finite set of possible states, A is a finite set of possible actions, O is a finite set of possible observations the agent can experience;
- $T : S \times A \to \Pi(S)$ is the state-transition function, which is mapping elements of $S \times A$ into discrete probability distribution over S. We write $T(s, a, s') =$

$p(s'|s,a)$ for the probability that the agent will make a transition from state s to state s' by taking action a.

- $Z : S \times A \rightarrow \Pi(\Omega)$ is the observation probability function, we write $Z(s, a, o) = p(o|s, a)$ for the probability of making observation $o \in O$ from state s after having taken action a.
- $R : S \times A \rightarrow \mathbb{R}$ is a reward function mapping $S \times A \rightarrow R$ that specifies a immediate reward $r^a(s) \in R$ that the agent receives from taking each action in each state.
- $\gamma \in [0, 1)$ is the discount factor to make the total reward to be finite and the problem is well-defined.
- $\Pi : S \rightarrow A$ is a mapping from S to A which specifies an action to be taken in each epoch.

3.1.2. Solving POMDP Problems

The standard approach to solve POMDP is to convert it to a continuous-space belief-state MDP [27, 29, 30]. Specifically, a POMDP can be decomposed into two parts, as shown in Figure 4. The agent is responsible for making observations and generating actions which keeps an internal belief state, b. A belief state is a discrete probability distribution over the set of environmental states. The SE is the state estimator which is responsible for updating the belief state. π is a policy which is responsible for generating actions [27, 29]. The last belief state, most recent action and observation are handled by the first component labeled SE, then the SE returns an updated belief state. The policy labeled "π" as the second component maps belief states into actions.

Given a belief state $b \in B$, where B is the set of belief states which comprises the state space. We write $b(s)$ for the probability assigned to state s when the agent's

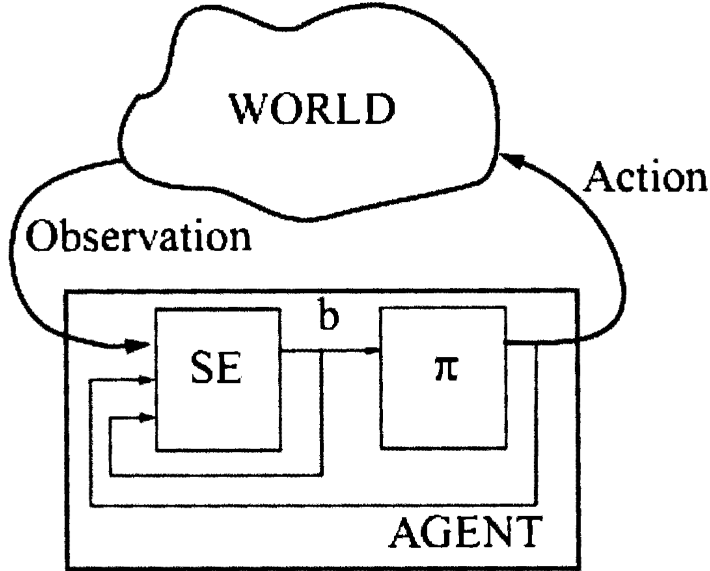

Figure 4. A POMDP agent decomposed into two parts [14].

belief stat is b. Similarly, $b'(s')$ is the probability assigned to state s' when the agent's belief state is b'. The next belief state b' is a revised estimate from b by taking action a and receiving observation o. That is [29],

$$b'(s') = \frac{1}{P^a(o|b)} Z^a(o|s') \sum_{s \in S} Pr^a(s'|s) b(s) = T(b) \tag{3.1}$$

We use $T(b)$ to represent the belief update. $V_\pi(s)$, the value of state $s \in S$, is the infinite expected total discount rewards to be received at each future time step. That is [29, 33],

$$V_\pi(s) = E(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)) \tag{3.2}$$

In infinite horizon problems, we seek to maximize the total expected rewards $V_\pi(s)$. By adding the discount factor γ to control the influence of rewards, where $0 \leq \gamma < 1$, a POMDP problem can be solved to improve a value function $V : B \rightarrow R$ for $\forall b \in B$ by iterations of a dynamic programming (DP) update [27, 29, 30]:

$$V'(b) = max_{a \in A, b \in B} \left\{ R^a(b) + \gamma \sum_{o \in O} P^a(o|b) V(T(b)) \right\} \tag{3.3}$$

Value iteration first developed for solving MDPs in order to find an optimal policy [29, 33]. As mentioned earlier, The POMDP can be formulated as a continuous-space belief-state MDP problem. Analogously, an optimal policy of a POMDP problem can be found using the value iteration approach. The Principle of Optimality by Bellman [34] shows that the stochastic dynamic programming equation given above is guaranteed to find the optimal policy (π^*) on this "belief MDP".

However, one could consider that optimal t time steps or t-horizon solution can approach the optimal infinite horizon solution if $t \to \infty$. This is the essence of value iteration. Precisely, if we write the value function of the optimal infinite horizon policy as V^*, and the value function of the optimal t-horizon policy as V_t^*. It should have the following property,

$$|V^* - V_t^*| \cong 0, \text{if } t \to \infty \tag{3.4}$$

which means that the difference between value function of the optimal infinite horizon policy and value function of the optimal t-horizon policy should tend toward 0 if t tends toward infinity. Hence, the POMDP problems which can be solved by the value iteration approach can have two stop criteria:

- ϵ-approach: we define a small number $\epsilon = 10^{-9}$, where ϵ is a lot less than zero. If $|V^* - V_t^*| \leq \epsilon$. We stop solving the problem by the assumption that the solution approximates to the optimal solution.

- horizon-approach: another way is to define the horizon number of solving the POMDP problems with certain time steps. If it reaches defined time step we have set, we will stop solving the problem whenever $|V^* - V_t^*|$ approximates to

0 or not.

3.2. Algorithms on Solving POMDP

POMDP solving or POMDP planning aims to compute an optimal or approximate policy given a POMDP model [35]. POMDP solving faces large computational challenges. Normally, there are two approaches for solving POMDP which are value iteration algorithm and policy iteration algorithm. In this section, it will only focus on the four different exact planning techniques on POMDP planning which use value iteration as the basic framework. They are Monahan's Enumeration, Sondik's Two pass, Littman's Witness and Zhang's incremental pruning algorithms [27]. Before the discussion, we need to define,

- **Policy tree** (p): generally, at a large enough time t, the decision of a t-step policy can be summarized as a t-step policy tree show in Figure 5. The top node determines the first action and then the arc is followed by a node on the next level based on the observation.

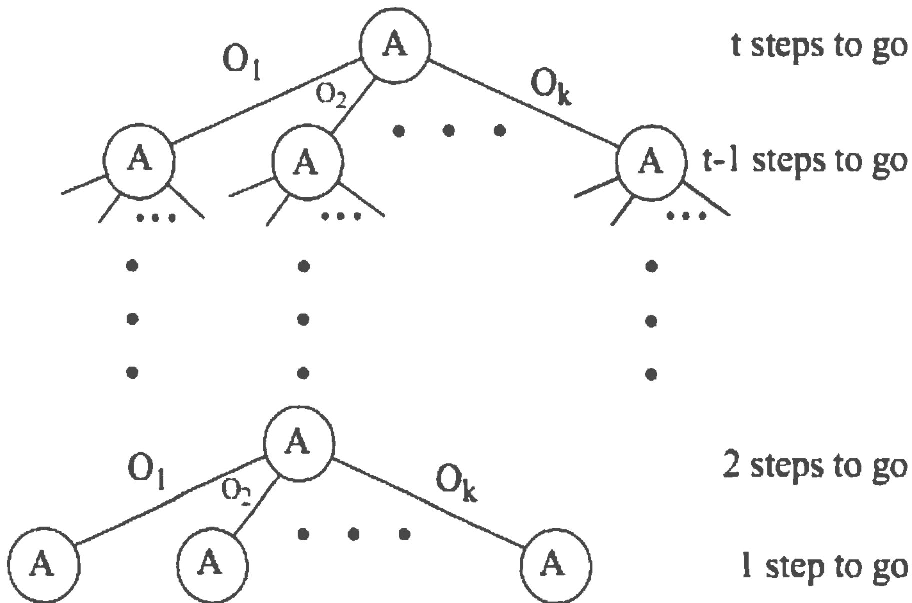

Figure 5. A t-step policy tree [27].

- **Parsimonious representations** (Γ): POMDP solving is complicated since the belief space for POMDP is continuous and cannot simply keep track of the value of belief update. However, Sondik and Monahan proved that the optimal value function has the properties of piecewise linearity and convexity [27, 30]. Therefore, we can represent the value function for a finite planning horizon h as a set of vectors $\Gamma = \{\alpha^0, \alpha^1, ..., \alpha^k\}$ called parsimonious representations. Also, the main task of POMDP solving is to minimize the size of parsimonious representations.

- **Region** ($\Re$): given a set of vectors Γ, each $\alpha \in \Gamma$ has a region of information states, $\Re(\alpha, \Gamma)$ where it dominates. That is,

  $\Re(\alpha, \Gamma) = \{b | b \cdot \alpha > b \cdot \hat{\alpha}, \forall \hat{\alpha} \in \Gamma - \{\alpha\}, b \in B\}$

- **Cross-sum** ($\bigoplus$): for two sets of vectors A and B, we define $A \bigoplus B = \{a + b | a \in A, b \in B\}$

Then, from the properties of piecewise linearity and convexity, we can compute the value of each belief state b with inner "dot product" by updating the Equation 3.3:

$$V^{\alpha}(b) = max_{\alpha \in \Gamma} \sum_{s \in S} b(s) \cdot \alpha(s) \quad (3.5)$$

The main task of POMDP solving is to minimize the size of parsimonious representations in order to reduce the computational time. Hence, Enumeration algorithms, Witness algorithms, Two-pass algorithms and Incremental Pruning algorithm will be discussed in the following.

3.2.1. Enumeration Algorithm

The Monahan's enumeration algorithm is to generate all the possible α vectors by ignoring information state. Then it uses Linear Programming (LP) to trim away

useless vectors. The Enumeration algorithm is conceptually the simplest of all the exact algorithms.

This algorithm allows DP update to be computed exactly with finite state and action space. As shown in Figure 6, it provides a simple linear programming scheme to prune the useless vectors [27]. The five vectors α_1, α_2, α_3, α_4 and α_5 shown on a belief space with two beliefs b_0 and b_1 tell us that, α_1, α_2 and α_5 in solid lines are useful vectors, α_3 and α_4 in dashed line are useless vectors. By the properties of piecewise linearity and convexity, we are only interested in the lines with color. In this way, we can trim away α_3 and α_4 to eliminate the useless vectors in order to reduce the computational time.

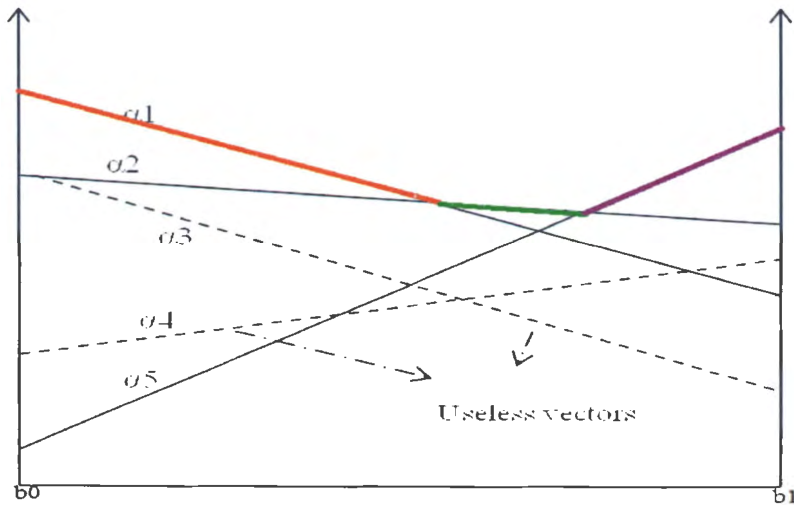

Figure 6. Convex piecewise linear function on a continuous belief state with S = 2.

Specifically, the Enumeration algorithm generates all possible vectors. We write the set of all possible vectors as $\bar{\Gamma}_n^{a,z}$. Using the operator cross-sum ($\bigoplus$), we can define,

$$\bar{\Gamma}_n^a = \bigoplus \bar{\Gamma}_n^{a,z} \tag{3.6}$$

to compute all possible combinations of vectors in $\bar{\Gamma}_n^a$. Then, the complete enumera-

tion of all possible vectors could be,

$$\bar{\Gamma}_n = \bigcup_a \bar{\Gamma}_n^a \tag{3.7}$$

Thus, with an operator *prune*, the Enumeration algorithm can be summarized as [27]:

$$\Gamma_n = prune(\bigcup_a \bigoplus_z \bar{\Gamma}_n^{a,z}) \tag{3.8}$$

Specifically, Algorithm 1 shows the basic steps of the Enumeration approach.

Algorithm 1 Enum(Γ_{n-1}, a)

$\Psi \leftarrow \cup_z \{\Gamma_n^{a,z}\}$
while $|\Psi| > 1$ **do**
  A $\leftarrow$ removeElement(Ψ)
  B $\leftarrow$ removeElement(Ψ)
  D $\leftarrow$ A $\bigoplus$ B
  $\Psi \leftarrow \Psi \cup \{D\}$
end while
$\Psi \leftarrow$ prune(Ψ)
return Ψ
end enumeration

3.2.2. Witness Algorithm

The witness algorithm tries to find the best value function for each of the actions separately. Witness algorithm computes Γ_n^a based on the idea of exploring a finite number of regions in the state space [27, 33].

As described in Equation 3.3, we can represent V_{n-1} and V_n using collections of policy trees as ν_{n-1} and ν_n respectively. In this algorithm, we first find a collection of policy trees that represents Q_t^a, which represents the expected reward by taking action a from belief state b. That is [36],

$$Q_t^a(b) = \sum_s b(s)R(s,a) + \gamma \sum_a Z(o|a,b)V_{t-1}(b') \tag{3.9}$$

Similar to Equation 3.3, this Q-function is piecewise linear and convex and can be represented by collections of policy trees. In order to find a set of policy trees to represent $Q_t^a(b)$, the witness algorithm tries to find witness points in each iteration. Then the basic approach of witness algorithm can been seen in Algorithm 2.

Algorithm 2 Witness(Γ_{n-1}, a)

Require: b $\leftarrow$ any information state.
  $\hat{\Gamma} \leftarrow \{\alpha(b)\}$ ♯ put a vector in belief state b in a set.
  $\Upsilon \leftarrow N(\alpha(b))$ ♯ list all the neighbors of that vector.
  while $\Upsilon \neq \Phi$ **do**
    v $\leftarrow$ removeElement(Υ) ♯ remove an element from Υ
    if v $\in \hat{\Gamma}$ **then**
      b $\leftarrow \phi$ ♯ b should be null.
    else
      b $\leftarrow$ findRegionPoint(v,$\hat{\Gamma}$) ♯ find whether v has region poing in $\hat{\Gamma}$
    end if
    if $b \neq \phi$ **then**
      $\hat{\Gamma} \leftarrow \hat{\Gamma} \cup \{\alpha(b)\}$
      $\Upsilon \leftarrow \Upsilon \cup \{v\}$
      $\Upsilon \leftarrow \Upsilon \cup N(\alpha(b))$
    end if
  end while
  $\Gamma_n^a \leftarrow \hat{\Gamma}$
  return Γ_n^a

In words, the first step is to initialize $\hat{\Gamma}$, the Witness algorithm selects any information state and constructs the maximal vector and all the neighbors of that state. The second step is discarding neighbors. The loop terminates if the Υ is empty by remove one neighbor at a time. The third step is checking. It will check whether a neighbor v is in $\hat{\Gamma}$ or not. If not, an information state exists in the region by checking $R(v, \hat{\Gamma})$. Also, it will use **findRegionPoint** to check the region information state and will update Υ. Finally, it will return a set of vectors Γ_n^a.

Simply, the witness algorithm is using linear programming to find a single point called "witness" with the fact that $V_t^* \neq V_t$ (see Equation 3.4). If a witness is found, it is used to determine a new vector by solving a linear program and repeat the process.

3.2.3. Two-Pass Algorithm

The biggest difference between Witness and Two-pass algorithm is that, it uses regions instead of searching witness points in the Two-pass algorithm. The main idea for Sondik's Two-pass algorithm is to keep track of the region of a vector in parsimonious set Γ_{n-1}, then finding all the adjacent regions. Recall the definition of cross-sum described before, if a vector $a + b$ would be useful, it is enough to check whether the region $R(a, A) \bigcap R(b, B)$ is empty or not. The Two-pass algorithm is using this fact to define a region for a vector to find the adjacent regions. Algorithm 3 shows the routine for computing Γ_n^a using the Two-pass algorithm. The algorithm explores a region using an LP for each belief state.

Algorithm 3 TwoPass(Γ_{n-1}, a)

Require: b $\leftarrow$ any information state.
  $\hat{\Gamma} \leftarrow \{\alpha(b)\}$ ♯ put a vector in belief state b in a set.
  $\Upsilon \leftarrow N(\alpha(b))$ ♯ list all the neighbors of that vector.
  while $\Upsilon \neq \Phi$ **do**
    $\alpha \leftarrow$ removeElement(Υ)
    $\hat{\Gamma} \leftarrow \hat{\Gamma}\{\alpha\}$
    $R \leftarrow \cap_z R(v^z, \Gamma_n^{a,z})$
    for each v $\in$ N(α) **do**
      L $\leftarrow$ setUpTwoPassLP(α, v, R)
      if feasibleLP(L) and v $\in \hat{\Gamma}$ **then**
        $\Upsilon \leftarrow \Upsilon \cup \{v\}$
      end if
    end for
  end while
  return Γ_n^a
  end TwoPass

3.2.4. Incremental Pruning Algorithm

The Incremental Pruning algorithm was proposed by Zhang and Liu [27]. The Incremental Pruning algorithm can solve the problems that cannot be solved within a reasonable time in the Witness algorithm. The Incremental Pruning uses the DP update to break down the value function V' defined in Equation 3.1 as a combination

of simpler value functions:

$$V'(b) = max_{a \in A} V^a(b) \tag{3.10}$$

$$V^a(b) = \sum_o V_o^a(b) \tag{3.11}$$

$$V_o^a(b) = \frac{R^a(b)}{|Z|} + \gamma P(o|b,a) V(b_z^a) \tag{3.12}$$

The equations above are piecewise linear and convex. *purge* is an operator that takes a set of vectors and reduces it to its unique minimum form. In order to remove dominated vectors without affecting the value of any belief state, there are two ways to test for dominated vectors. One simple way is to remove any vector that is pointwise dominated by another vector. That is,

a vector $\alpha_1(s)$ and another vector $\alpha_2(s)$, if $\alpha_1(s) < \alpha_2(s)$ for $\forall s \in S$, we can say $\alpha_1(s)$ is pointwise dominated by $\alpha_2(s)$.

However, using this way we can not find out all dominated vectors though it is fast. Another way is the linear programming method which can detect all dominated vectors. Given a vector β and a set of vectors A, where $\beta \notin A$. If we add β to A to determine whether it improves the value function represented by A or not. If it does not, then β is dominated by A. Incremental Pruning algorithm uses the two sets to prune dominated vectors of a set of vectors to its minimum size.

The Incremental Pruning algorithm uses the notation cross-sum ($\bigoplus$) to enumerate all the possible combination vectors on initializing Γ. We use an operator *purge* to prune Γ. If we have three sets of vectors A, B and C, *prune* and $\bigoplus$ have the property:

$$purge(A \bigoplus B \bigoplus C) = purge(A \bigoplus purge(B \bigoplus C)) \tag{3.13}$$

The Incremental Pruning algorithm exploits this property and then Γ^a can be

computed as:

$$\Gamma = purge(\Gamma^{a,o_1} \bigoplus purge(\Gamma^{a,o_2} ... \bigoplus purge(\Gamma^{a,o_{k-1}} \bigoplus \Gamma^{a,o_k}))) \tag{3.14}$$

In this way, the Incremental Pruning algorithm reduces the number of solving LP problem in a recursive process. It breaks down the problem to prune dominated vectors recursively to improve the computational times.

As shown in Algorithm 4, it first enumerates all the lists of vectors into a set of vector lists Ψ. Then it removes a list of vectors (A) and a list of vectors (B) from the set Ψ, then uses the operator cross-sum to list all combinations of List A and List B. Later, it will prune the dominated vectors to get a new list of vectors (D) and put it back into Ψ. Recursively, it keeps checking until Ψ is empty.

Algorithm 4 incrementalPruning(Γ_{n-1}, a)

$\Psi \leftarrow \cup_z\{\Gamma_n^{a,z}\}$
while $|\Psi| > 1$ **do**
  A $\leftarrow$ removeElement(Ψ)
  B $\leftarrow$ removeElement(Ψ)
  D $\leftarrow$ purge(A $\bigoplus$ B)
  $\Psi \leftarrow \Psi \cup \{D\}$
end while
return Ψ
end incremental pruning

3.3. Comparison of the Four Algorithms

The applicability of POMDP is limited in two perspectives. One is the data intensive, and another is that the solution of the POMDP model of most realistic problems require large computational time [31].

Let $|\Gamma_n^a| = Q$, $|Z| = Z$ and $|\Gamma_n^{a,z}| = M$, where $Q \geq M$. From the Table 1, all four algorithms have the same complexity of total LPs which is the total number of linear programming required for solving POMDPs. The total constraints required for solving the Two-pass algorithm is a lot larger than the Witness algorithm and the

Incremental Pruning algorithm. So if it has more than 3 or 4 observations ($|Z|$), the Two-pass algorithm is impractical. Also, the Witness algorithm and the Incremental Pruning algorithms have the same complexity in the worst case of total constraints. However, the Incremental Pruning algorithm is better than the Witness algorithm in the best case.

Algorithm	Total LPs	Worst case of total constraints	Best case
Enumeration	$O(ZMQ)$	-	-
Two-pass	$O(ZMQ)$	$O(Z^2M^2Q)$	$O(Z^2M^2Q)$
Witness	$O(ZMQ)$	$O(ZMQ^2)$	$O(ZMQ^2)$
Incremental Pruning	$O(ZMQ)$	$O(ZMQ^2)$	$O(ZQ^2)$

Table 1. Complexity comparison of the four algorithms [27].

4. EXPERIMENTS AND RESULTS

In this chapter, a case study from existed work and the framework on modeling the Web service composition into a POMDP problem will be introduced. Also, the experiments and results by measuring the proposed six different cases will be discussed.

4.1. A Case Study

In order to illustrate the workflow composition of web services, Doshi's [4] manufacturer models instead of using MDP has been adopted, POMDP has been applied to solve the same problem by adding observations to it.

A **manufacturer** can receive some orders and deliver some merchandise to a **retailer**. If sufficient stock exists in his **inventory**, he may satisfy the order immediately. In order to produce the goods needed for the order, the **manufacturer** can require parts from the **preferred supplier**, search for a new **supplier** or buy the parts from the **Spot Market**. In this way, the manufacturer will have the least cost if he/she satisfies the order from the inventory. It will increase costs if he tries to search from the **preferred supplier**, other new **supplier** and the **Spot Market** to produce the goods.

Figure 7 shows Doshi's motivating scenario graphically. As shown in the figure, the **manufacturer** initially attempts to fetch the order from his own **inventory**. If there is insufficient order, he will prefer to order parts from **preferred supplier** to produce the goods for the order. If he still cannot satisfy the order, he will search for new suppliers. If he still cannot find the parts, he will finally buy them from the **Spot Market**.

Specifically, from Figure 8, Doshi's model has 6 states:

- Inventory Availability

- Preferred Supplier Availability
- New Supplier Availability
- Spot market Availability
- Order Assembled
- Order Shipped.

Six actions are service invocations, which are:

- Check Inventory Status
- Check Preferred Supplier status
- Check New Supplier Status
- Check Spot Market Status
- Assemble Order
- Ship Order.

Two observations which denote the invocations' failure or success:

- Failure
- Success

The transition probabilities of each action on some states are also calculated. For example (see Figure 7),
T(Inventory Availability |Check Inventory Status, Preferred Supplier Availability) = 0.7.

If we view the workflow composition as a goal-driven problem, AI planning algorithms seem to be the suitable candidates for automatic workflow composition.

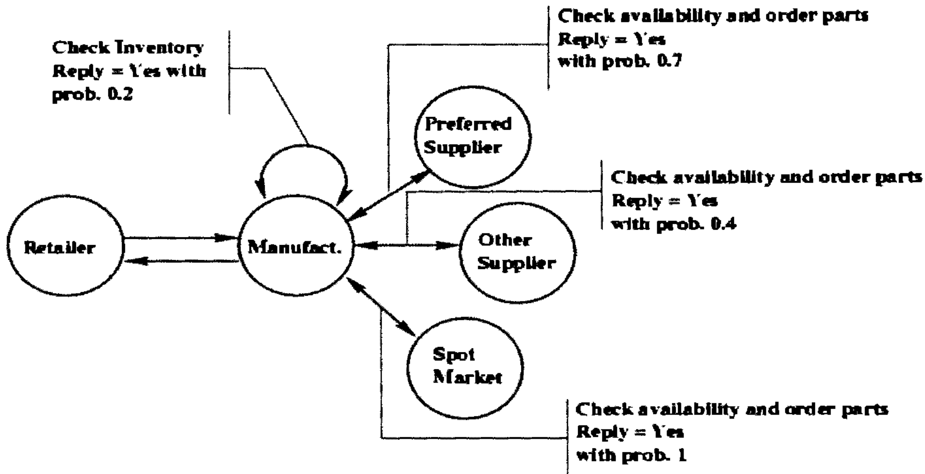

Figure 7. Motivating scenario with example probability values by Doshi [4].

However, the time, cost, factors in service level agreements [4] of the workflow composition need to be accomplished. By these aspects, the problem cannot simply be defined as a goal-driven problem. Instead, it should be addressed by decision-theoretic planning techniques. Hence, the problem can be addressed by using POMDP.

After the manufacturer problem has been modeled into the POMDP problem, the problem can be solved using the Equation 3.2 which is guaranteed to find the optimal policy for the POMDP. Then the optimal policy π^* which is a mapping from states to actions will be found.

From Figure 8, we can see that the model will have four different scenarios in Figure 9, which means that if we apply POMDP to solve this model, the optimal policy or sub-optimal policy π^* should generate 4 different types of workflows (see Figure 9) for service invocations with different transition probabilities and observation probabilities.

4.2. Development of POMDP Composition Framework

In order to apply the POMDP technique to Web service composition, a framework of the experimental process which has four phases: service invocation, extract the POMDP data, POMDP algorithms suite, and service composition (see

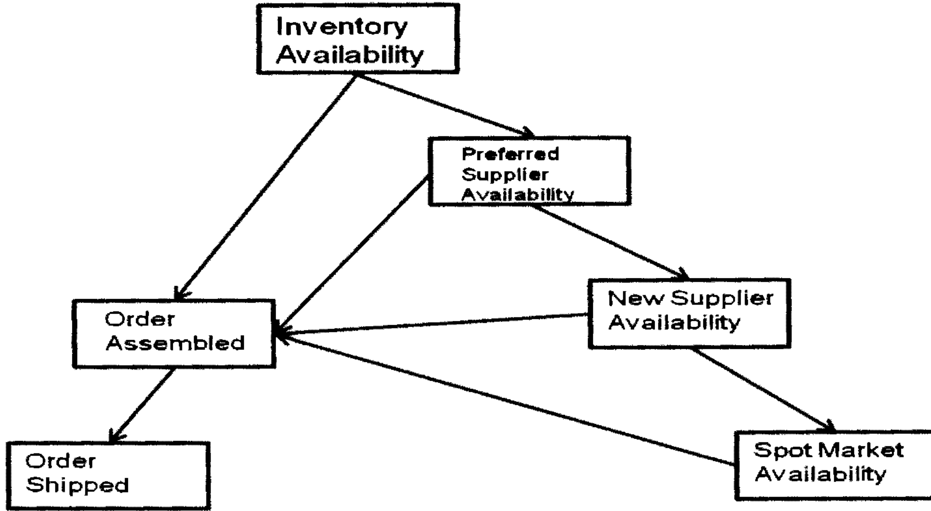

Figure 8. Variations in workflow execution as response to Web Service invocation changes.

Figure 10) has been developed.

- **Service invocation**: in this phase, the Web service invocations executions have been modeled and their execution time, observations and response time have been stored into a database. For example, the invocation will have the format as shown in Figure 11.

  srv_id is the service ID which represents each distinct service, for example, if we have 6 services, then the service ID will be a set of integers $\{0, 1, 2, 3, 4, 5\}$. The names of the services will show in Figure 12, Check_avail = Check Inventory Status, Check_Supp_Avail = Check New Supplier Status, Check_Spot_Market_Avail = Check Spot market Status, Assemble_Order = Assemble Order, Ship_order = Ship Order. *id* is the row number of the table in the query. The status is either 0.0 or 1.0 to denote the service invocation is available or not. *exec_time* is the execution time for the particular service invocation. Observation is either 0.0 or 1.0 to denote the invocation is failure or not. *resp_time* is the response time for each invocation to make a decision and transit to the next state. *wf_id* are a set of integers $\{1, 2, 3, 4\}$ which denote the

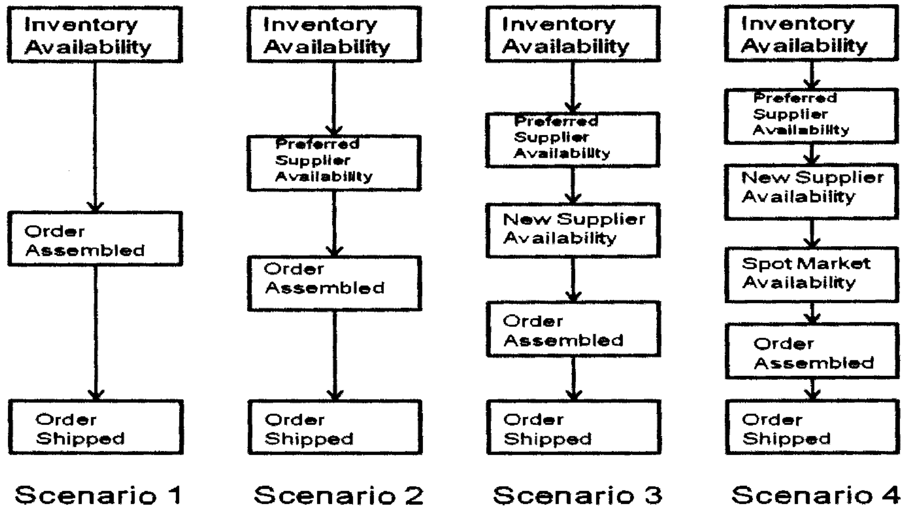

Figure 9. Four scenarios of the manufacturer model by Doshi.

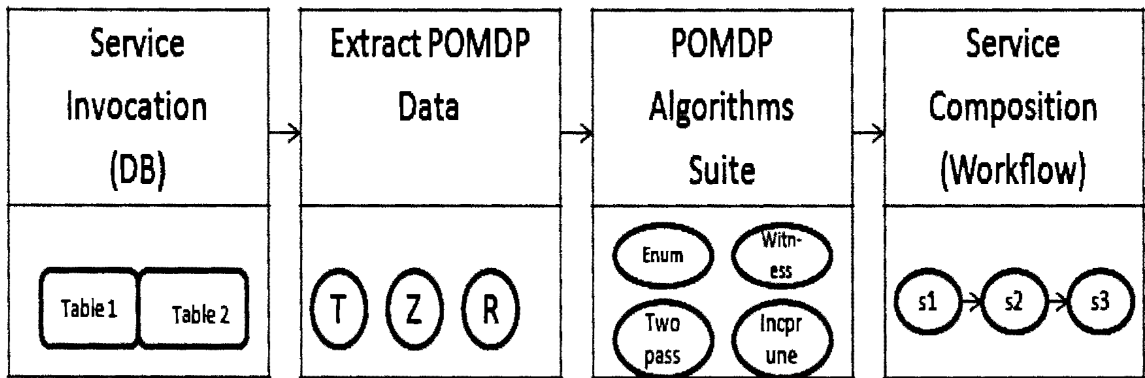

Figure 10. Framework of service composition using POMDP.

four scenarios of workflows for service composition. In this way, 10238 different invocations artificially have been generated and stored into a MySQL database.

- **Extract POMDP data**: it extracts the data from the MySQL query into a Java Programming (Appendix C) in order to calculate the transition probabilities, observation probabilities and rewards in this phase. This phases extracts the data to get a widely used POMDP problem file which has originally been used by Cassandra [27]. In a Cassandra's file format, a POMDP problem will have variables (Appendix B): discount (γ), rewards (R), states (S), actions

srv_id	id	status	exec_time	observation	resp_time	wf_id
1	622	1.0	171003.0	1.0	3.0	1
4	623	1.0	171009.0	1.0	6.0	1
5	624	1.0	171012.0	1.0	3.0	1

Figure 11. Screen shot of invocation table in MySQL database.

id	name
1	Check_avail
2	Check_Supp_Avail
3	Check_Spot_Market_Avail
4	Assemble_Order
5	Ship_order

Figure 12. Screen shot of service table.

(A), observations (O), transition probabilities (T), observation probabilities (Z). Some might have the initial belief states (s_0). For example, $\gamma = 0.95$ and $s_0 = 0$ in this thesis.

In this phase, the main routine is to calculate the transition probabilities, observation probabilities and rewards for the state-action pairs. By Bayes' Theorem, we can calculate the transition probabilities (Equation 4.1) and observation probabilities (Equation 4.2) respectively,

$$T = P(s'|s,a) = \frac{P(s,s',a)}{P(s,a)} \tag{4.1}$$

where s is the current state, s' is the next state and a is an action takes from current state to next state.

$$O = P(s|o,a) = \frac{P(s,o,a)}{P(o,a)} \tag{4.2}$$

Similarly, s is the current state, o is the observation of current state, and a is an action taking by the current state based on the observation.

The rewards are calculated by the response time for a state takes a particular

action, that is,

$$Reward(s,a) = C(s,a) \tag{4.3}$$

- **POMDP algorithms suite**: in this phase it uses the POMDP file generated in the previous phase as the input file to test four exact POMDP algorithms: Enumeration, Two-pass, Witness and Incremental Pruning (see Appendix A). In this phase, it uses the proposed four algorithms in order to get the four scenarios as mention before. After the implementations, the four exact algorithms can return an optimal policy. One sample optimal policy for a 3 states = $\{0,1,2\}$ and 3 actions = $\{0,1,2\}$ looks like Table 2.

Number of Iteration	S	A
1	0	0
2	0	1
3	1	1
4	1	1
5	2	2

Table 2. A sample policy which is mapping the States (S) to Actions (A).

- **Service composition**: the results return to the user and can be used by the workflow engine to compose and execute the workflow in this phase. We can construct a workflow using the optimal policy which have got from the previous phase to represent the service composition result graphically. Algorithm 5 [4] shows how to construct a workflow by an optimal policy.

Overall, as shown in Figure 10, firstly, it generated the data of the service invocations and stored them in a database. Secondly, it calculated the transition probabilities, observations probabilities, rewards and other variables which are required for a POMDP problem from the data. Specifically, it used the Bayesian conditional probability formula to calculate the transition and observation probabilities. Thirdly,

Algorithm 5 Translating a policy into a workflow

Require: π^* and s_0
  $s \Leftarrow s_0$
  while $s \neq$goal state **do**
    $a \Leftarrow \pi^*$
    Execute invocation a
    Get response of a and next state s'
    $s \Leftarrow s'$
  end while
  return a workflow

it tested the proposed algorithms using the required POMDP data file with different scenarios. Finally, an optimal policy from solving the POMDP problem is generated. Since the optimal policy is described as state-action pairs, it reconstruct the Web service workflow with different invocations (actions).

4.3. Experiments and Results

Doshi's model as a POMDP problem has been implemented and comparison of the required actions of the four different scenarios are shown in Figure 9.

As shown in Table 3, the number of actions of the optimal policies required in the four different scenarios tells us that if a service fails, the state remains unchanged. In this situation, the policy prescribes an optimal action depending on the current state and the observations. It is capable of optimally recovering from Web service failure while classic planning needs to be monitored for unexpected interaction between the plan and environments.

Scenario	Number of actions
1	3
2	9
3	23
4	48

Table 3. Action comparison between the four scenarios in Figure 9.

In order to show the effectiveness of automated Web service composition

based on workflow composition by using the POMDP planning, the four exact algorithms Enumeration, Two-pass, Witness and Incremental Pruning Algorithms were compared. Six cases of the Web service composition which are fully connected similar with Scenario 4: 1) 3 states and 3 actions with 2 observations, 2) 6 states and 6 actions with 2 observations, 3) 9 states and 9 actions with 2 observations, 4) 12 states and 12 actions with 2 observations 5) 15 states and 15 actions with 2 observations and 6) 18 states and 18 actions with 2 observations, were produced for the simulations.

1. *Result Comparison of Execution Time of the Four Algorithms*

In this investigation, the execution times of the four algorithms for the six cases mentioned above have been tested. Recall the complexity of the four algorithms in Chapter 3, the six cases when the workflow is fully connected like the Scenario 4 have been tested. As shown in Figure 13, Enum = Enumeration, twopass = Two-pass, incrpune = Incremental Pruning. The Two-pass algorithm is the one that has the worst execution time. The Enumeration algorithm is slightly better than the Two-pass Algorithm. Witness and Incremental Pruning algorithms almost have the same execution time. The figure shows that when the number of states and actions increases, the Witness and Incremental Pruning algorithms use much less execution time than the Enumeration and Two-pass algorithm.

2. *Investigation of Trends on Execution Time of the Four Algorithms*

In order to explore the trends of execution time of the four algorithms in the six cases, as shown in Figure 14, we can see that the four algorithms have exponential execution time of the number of states and actions. At the beginning, there are not too many differences between different algorithms with small size problem. But when the number of states increases, the Witness and Incremental Pruning algorithms performed a lot better than the Enumeration and the Two-pass algorithm.

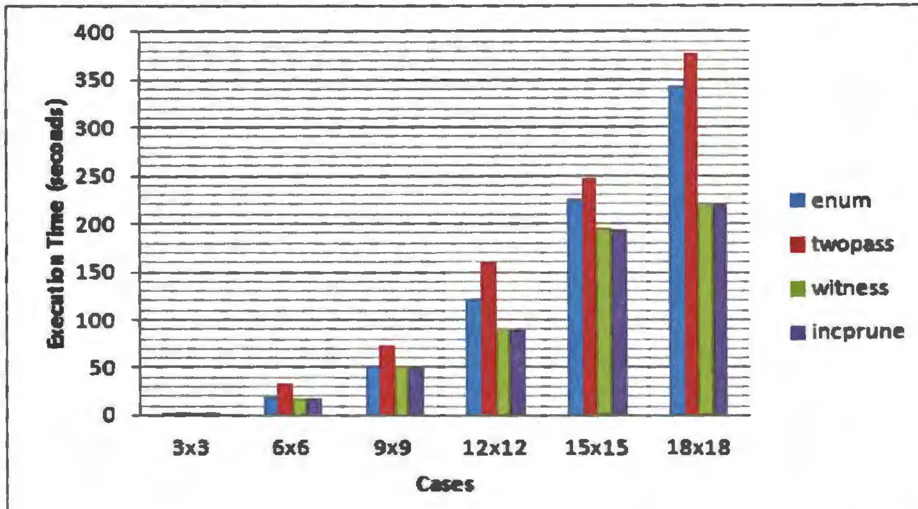

Figure 13. Comparison of execution time of the four algorithms.

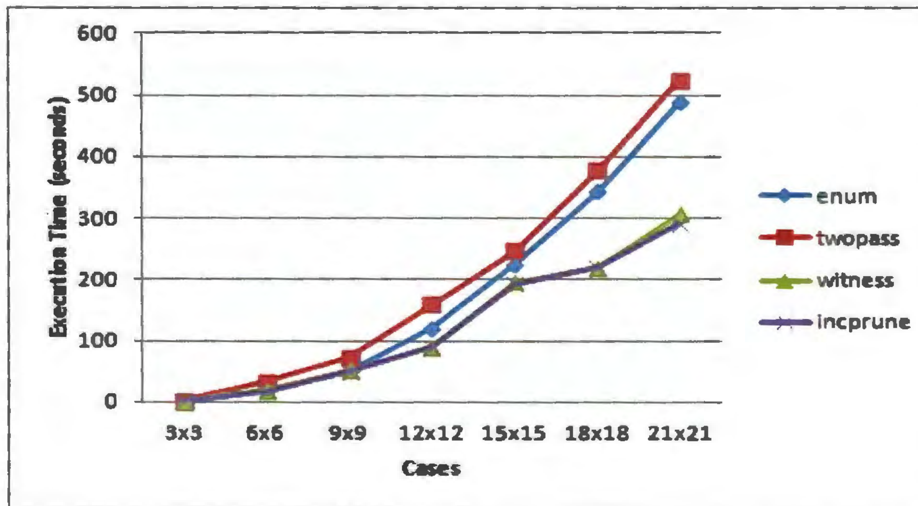

Figure 14. Trends of the execution time by using the proposed four algorithms.

3. *Investigation of Convergence of Vectors*

The convergence of vectors with the same stop criterion ($\epsilon - approach$) has been tested. Here the vectors are the actions of the optimal policies. The results of the four cases: 9×9, 12×12, 15×15, and 18×18 have been compared. In Figure 15, the number of vectors of each iteration increases quickly at first until it reaches its peak. Then it will drop and converge to a constant number. Also, as shown in the figure, after 10 iterations, the number of vectors is already starting to converge but have not met the stop criterion.

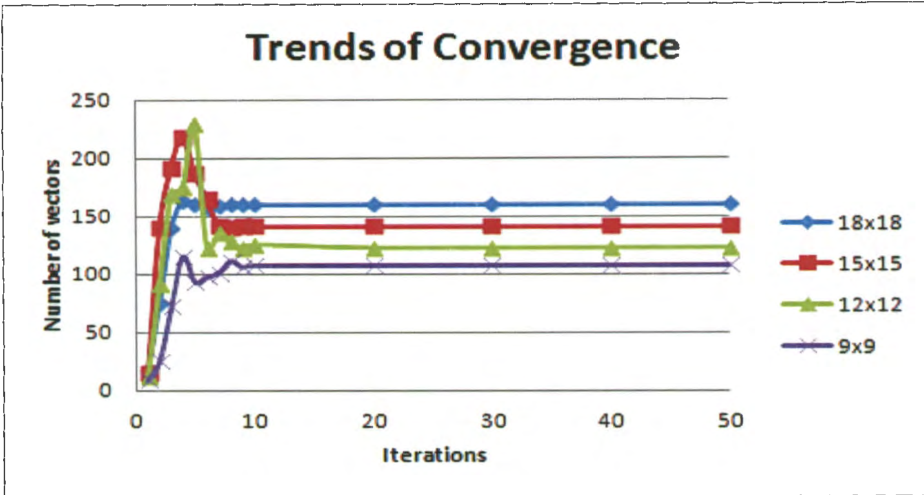

Figure 15. Vector convergences by using Incremental Pruning Algorithm.

4. *Investigation of Relation Between Number of States and Number of Vectors*

The required iterations and vectors for the six cases have been recorded. From Table 4, the required iterations of each case are around 558 using the ϵ-approach stop criterion. The number of vectors required to draw a full workflow (Scenario 4 in Figure 9) of the service invocations are shown in Table 4 too.

In order to see the relation between states and required vectors of an optimal policy, see Figure 16, as the number of states increases, the number of vectors of an optimal policy increases linearly to draw a fully connected workflow. A linear equation can be described as,

$$y = 9x + c; \tag{4.4}$$

where y is the number of vectors, x is the number of states and c is some constant.

5. *Investigation on the Speed of Vector Convergence*

In order to test the speed of convergence to near-optimality, as shown in Figure 17, the speed of the vectors converges with the horizon = 20 by using the horizon-approach stop criterion will not change too much when sizes of states and actions are small. However, as the number of states and actions increases, the total execution

$S \times A$	Number of Iterations	Number of vectors
3×3	607	12
6×6	558	87
9×9	558	107
12×12	558	123
15×15	557	140
18×18	559	167

Table 4. Required iterations and vectors to generate Scenario 4 (see Figure 9).

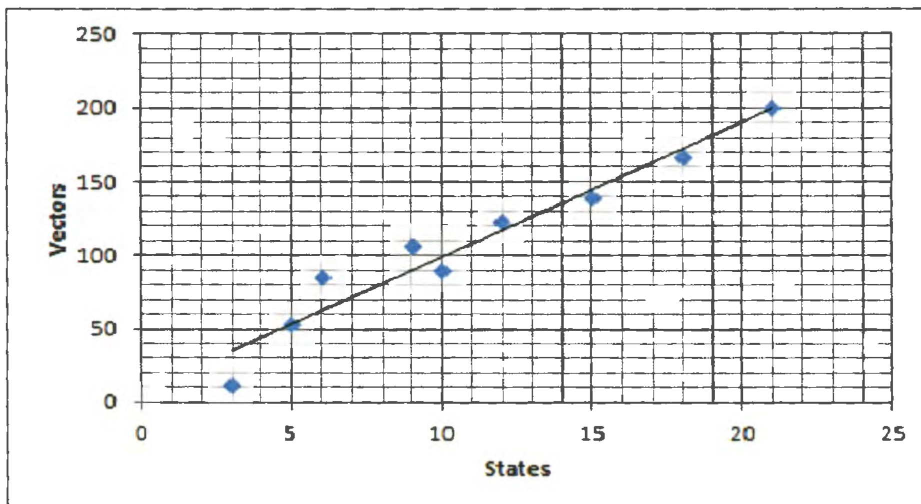

Figure 16. Relation between states and vectors of the optimal policies.

time for vector convergence increases a lot faster than before.

4.4. Discussion

The results above have shown that the POMDP planning can be used in automated Web service composition. By using a policy for workflow generation, it will always prescribe an optimal Web service to execute at that state since the policy is a mapping from state to action. In this way, if a Web service fails, the state of workflow does not change. Thus, the policy will decide to take the same action or a different one depending on which action is the best one. Therefore, it can handle the Web service failure by using the policy-based approach in Web service composition. Comparing to MDP, POMDP is partially observable while MDP is fully observable. POMDP can solve the uncertainty about the action outcome and uncertainty about

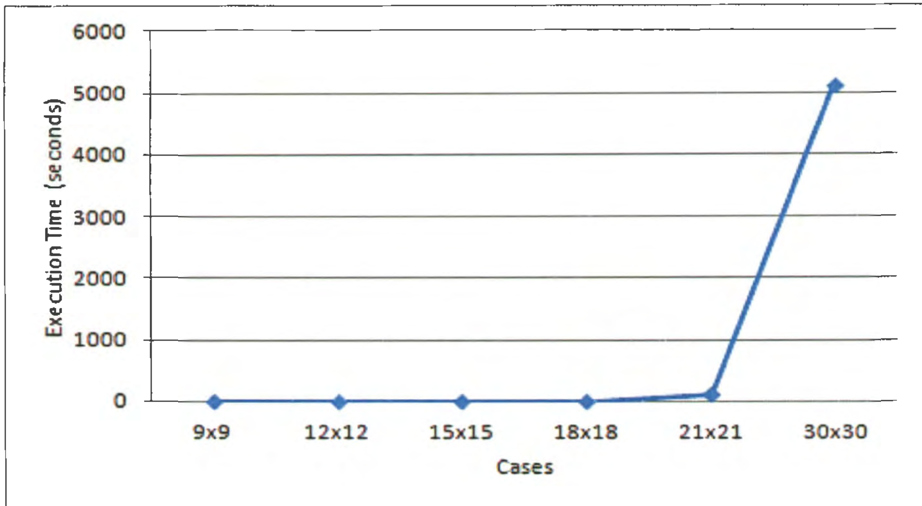

Figure 17. Speed of vector convergence with horizon=20 using Incremental Pruning.

the world state due to partially information which are more suitable for dynamic service environments. However, to apply POMDP on Web service composition is harder than using MDP. Finding an optimal policy for a POMDP is PSPACE-Complete while finding an optimal policy for a MDP is P-Complete.

However, the execution time of POMDP will increase exponentially by the number of states and actions. Only six cases have been tested with the largest number of invocations equal to 18 to get the fully connected workflow (Scenario 4). From Figure 13, we can see that the execution time is about 4 minutes for the fastest algorithm to compute an optimal policy. Even though it can guarantee to find an optimal solution, as the invocation number increases, the execution time will become extremely large. As we know, there are thousands of Web services and our approach can only solve the problem of small numbers of invocations. However, as it has been discovered in Figure 15, after 10 iterations, the vectors already started to convergence. In Figure 17, with the horizon equals to 20, it still get the approximate policy. By this way, we can reduce the execution time in order to solve relative large size composition problems.

5. CONCLUSIONS

5.1. Summary

In the first part of the thesis, the recent developments in automated Web service composition especially on workflow-based and AI planning based approaches have been introduced in Chapter 2. Because of the partial observability and ambiguity of state descriptions of Web services, as well as the composite services are workflow-like, to apply the AI planning technique to workflow composition seemed interesting. Thus, POMDP planning which can plan under uncertainty and insufficient information has been introduced in Chapter 3. Also, in order to test the performance of applying POMDP in workflow composition, experiments and analysis have been done in Chapter 4.

5.2. Conclusion

In this work, a case study on Doshi's manufacturer model by applying the POMDP technique to workflow composition has been done. Six cases with different states and actions to test the performance of POMDP service composition have been recorded. The results obtained tell us that POMDP regardless of the computational time can guarantee to compute an optimal policy in a dynamic service environment. Small size problems can be solved in a few seconds. Also, a relatively large sized problem can be solved by using the horizon-approach stop criterion to get a near optimal policy. However, while the size of the composition problems increases, the challenge of computational time increases. As we know, hundreds of services exist in the Web service environment, Web service composition using POMDP cannot be computed in a reasonable time. Also, as new services become available, the recomposition are required. Recomposition of services is time consuming and the cost increases. An efficient way to add new services into the current workflow without recomposing it needs to be considered.

5.3. Future Work

The analysis shows us that the service composition using POMDP challenges the computational time complexity of large size problems. Since there are hundreds of Web services, it is impractical to solve the POMDP in such situations. Hence, in order to reduce the computational time complexity of solving POMDP Web service compositions in future, I plan to:

- adopt hierarchical concepts into the workflow composition. Composability of service composition has the ability to form new composite services by combining the functionalities of existing services, thus, the existing service themselves can be composite. Solving a hierarchical POMDP problem of Web service composition can be decomposed into small POMDP problems in the domain based on actions.

- In order to solve the large size problem, we propose to use approximate heuristic algorithms instead of using exact algorithms. Although the approximate algorithms cannot guarantee to compute the optimal solution, the near-optimal solution with the efficiency can still work since some services have the same functionalities.

BIBLIOGRAPHY

[1] IEEE Computer Society. *Overview.* Date retrieved: Oct. 2011, from `http://tab.computer.org/tcsc/scope.htm`.

[2] M.P. Papazoglou, D. Georgakopoulos. *Service-Oriented Computing.* Communications of the ACM, 46(10):25-65, 2003.

[3] M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann. *Service-Oriented Computing: State of the Art and Research Challenges.* Computer, pp. 38-45, November, 2007.

[4] P. Doshi, R. Goodwin, R. Akkiraju, K. Verma. *Dynamic Workflow Composition: Using Markov Decision Processes.* International Journal of Web Services Research, Vol. 2, No. 1. (2005), pp. 1-17.

[5] G. Alonso. *Web services: concepts, architectures and applications.* Springern, first edition 2003.

[6] J. Rao and X. Su. *A Survey of Automated Web Service. Composition Methods.* In LNCS, Vol. 3387/2005 (2005), pp. 43-54.

[7] G. Baryannis, O. Danylevych, D. Karastoyanova, K. Kritikos, P. Leitner, F. Rosenberg, and B. Wetzstein. *Service Composition.* in Proc. S-CUBE Book, 2010, pp.55-84.

[8] *UDDI.* Online Community for the Universal Description, Discovery, and Integration. Date retrieved: 14 Oct. 2011, from `http://uddi.xml.org/`.

[9] *DAML-S 0.7 Draft Release.* DAML.org. Date retrieved: 14 Oct. 2011, from `http://www.daml.org/services/daml-s/0.7/`.

[10] A. Kim, M. Kang, E. Ioup, C. Meadows, and J. Sample. *A Framework for Automatic Web Service Composition.* Naval Research Laboratory 2009.

[11] A. DiCaterino, K. Larsen, M. Tang, and W. Wang. *An Introduction to Workflow Management Systems.* 01 Nov 1997.

[12] A. Barker and J.V. Hemert. *Scientific Workflow: A Survey and Research Directions.* Parallel Processing and Applied Mathematics In Parallel Processing and Applied Mathematics , Vol. 4967 (2008), pp. 746-753.

[13] D. Berardi, D. Calvanese, G.D. Giacomo, M. Lenzerini, and M. Mecella. *Automatic service composition based on behavioral descriptions.* Int. J. Cooperative Inf. Syst., 14(4):333-376, 2005.

[14] H. Schuster, D. Georgakopoulos, A. Cichocki, Andrzej and D. Baker. *Modeling and Composing Service-Based nd Reference Process-Based Multi-enterprise Processes.* Proceedings of the 12th International Conference on Advanced Information Systems Engineering, 2000.

[15] D. Ganesarajah, E. Lupu. *Workflow-based composition of web-services: a business model or a programming paradigm?.* 6th International Enterprise Distributed Object Computing, Lausanne, Switzerland, 2002.

[16] F. Ranno, S.K. Shrivastava, S.M. Wheater. *A language for specifying the composition of reliable distributed applications.* Distributed Computing Systems, 1998. Proceedings. 18th International Conference on 26-29 May 1998.

[17] J. Cardoso, A. Sheth. *Semantic Web Services.* Processes and Applications. Springer, 2006.

[18] R. Milner. *Communication and concurrency.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[19] R. Milner. *Communicating and mobile systems: the pi-calculus.* Cambridge University Press, fifth edition, 2004.

[20] N. Milanovic and M. Malek. *Current solutions for web service composition.* IEEE Internet Computing, 8(6):51-59, 2004.

[21] E. Sirin *Automated Composition of Web Services using AI Planning Techniques.* Date retrieved: Oct. 2011, from `http://www.cs.umd.edu/Grad/scholarlypapers/papers/aiplanning.pdf`.

[22] E. Sirin, B. Parsia, D. Wu, J. Hendler and D. Nau. *HTN planning for Web Service composition using SHOP2.* Web Semantics: Science, Services and Agents on the World Wide Web In International Semantic Web Conference 2003, Vol. 1, No. 4. (October 2004), pp. 377-396.

[23] M. Paolucci and O. Shehory and K.P. Sycara and D. Kalp and A. Pannu. *A Planning Component for RETSINA Agents.* 6th International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL), 2000.

[24] A. Gao, D. Yang, S. Tang, M. Zhang, *Web Service Composition Using Markov Decision Processes.* 2005.

[25] A. Marconi, M. Pistore, P. Traverso. *Automated Composition of Web Services: the ASTRO Approach.* IEEE Data Eng. Bull., 23-26, 2008

[26] F. Casati and M. Sayal and M. Shan. *Developing E-Services for Composing E-Services.* pages 171-186, vol 2068, 2001.

[27] A.R. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes.* Ph.D. Thesis. Brown University, Department of Computer Science, Providence, RI, 1998.

[28] L.P. Kaelbling, A.R. Cassandra , and J.A. Kurien. *Acting Under Uncertainty: Discrete Bayesian Models for Mobile-Robot.* Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 1996.

[29] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. *Planning and acting in partially observable stochastic domains.* Artificial Intelligence, Volume 101, pp. 99-134, 1998.

[30] Z. Feng and S. Zilberstein. *Region-Based Incremental Pruning for POMDPs.* Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI), 146-153, Banff, Canada. 2004.

[31] C.C. White. (1991). *A survey of solution techniques for the partially observed Markov decision process.* Annals of Operations Research 32 (1): 215-230.

[32] E.J. Sondik. *The Optimal Control of Partially Observable Markov Processes Over the Infinite Horizon: Discounted Costs.* Operations Research Vol. 26, No. 2 (Mar. - Apr., 1978), pp. 282-304.

[33] A.R. Cassandra, L.P. Kaelbling, and M.L. Littman. *Acting optimally in partially observable stochastic domains.* In Proceedings of the Twelfth National Conference on Artificial Intelligence, (AAAI) Seattle, WA, 1994.

[34] R.Bellman *Dynamic Programming.* Dover Publications, 1957.

[35] J. Baxter and P.Bartlett. *Reinforcement learning in POMDP's via direct gradient ascent.* In In Proc. 17th International Conf. on Machine Learning, pages 41C4.

[36] M.L. Littman *The Witness Algorithm: Solving Partially Observable Markov Decision Processes.* Brown University Providence RI 1994 1-48.

[37] A.R. Cassandra, M.L. Littman and N.L. Zhang. *Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes.* Uncertainty in Artificial Intelligence (UAI), 1997.

APPENDIX A. POMDP SOLVER IN JAVA

The following classes are the main class of sloving POMDP in Java.
The main class on solving POMDP:

```
public interface pomdp_solveIN {
    public void initPomdpSolve( PomdpSolveParams param );
    public void cleanUpPomdpSolve( PomdpSolveParams param );
    public void solvePomdp( PomdpSolveParams param )throws IOException;

    /* For now our default policy is just all zeroes.  */
    public AlphaList getDefaultInitialPolicy( );

    /* Some algorithms will solve one iteration of POMDP value
         iteration by breaking the problem into a separate one
         for each action. This routine will implement the basic
         structure needed and call the appropriate routines
         depending on the specific algorithm being used. Current
         algorithms that do it this way: TwoPass, Witness
         and IncrementalPruning */
    public AlphaList improveByQ( AlphaList[][] projection,
                                 PomdpSolveParams param );
    /* This does a single DP step of value iteration for a POMDP.
       It takes in the previous value function and parameters for
       solving and returns the next or improved solution.  */
    public AlphaList improveV( AlphaList prev_alpha_list,
                               PomdpSolveParams param );
}
```

Main class for defining POMDP problem:

```
public interface pomdpIN {

    /* Often we would like to do some max or min procedure and
       require initialization to the most extreme value.
       Since the extreme value depends on whether or not we
       are using rewards or costs, we have
       encapsulated this in this routine.  */
    public double worstPossibleValue();

    /* Often we would like to do some max or min procedure and
         require initialization to the most extreme value. Since
         the extreme value depends on whether or not we are using
         rewards or costs, we have encapsulated this in this
         routine.  */
    public double bestPossibleValue();

    /* Often we would like to do some max or min procedure and
       require coparing a new value to a current value. Since
       the test for which is better depends on whether rewards
       or costs are being used, we have encapsulated this in
       this routine. We also want to account
       for the precision of the current run (i.e.,
       gDoubleEqualityPrecision.)  */
```

```
    public int isBetterValue( double new_value,
                              double current,
                              double epsilon );

    /* Does the necessary things to read in and set-up a POMDP
       file.  Also precomputes which observations are possible
       and which are not.  */
    public void initializePomdp( String filename,
                                 double obs_possible_epsilon );

    /* Deallocates the POMDP read in by initializePomdp().*/
    public void cleanUpPomdp(  );
}
```

Class for cross-sum calculations:

```
 public interface cross_sumIN {
/* Takes the cross sum of two sets of vectors and returns the
  resulting set. If either A or B is null, then NULL is returned.
  If either list is empty, then an empty list is returned. The
  save_obs_sources argument deterines whther we do the
  bookkeeping required to develop a policy graph or not.  */
    public AlphaList crossSum( AlphaList A,
                               AlphaList B,
                               int save_obs_sources );
}
```

Class for Incremental Pruning algorithm:

```
public interface inc_pruneIN {

    public void initIncPrune( );
    public void cleanUpIncPrune(  );

    /* The main incremental pruning algorithm routine for
       finding the Q-function represention for value
       iteration with POMDPs.  */
    public AlphaList improveIncPrune( AlphaList[] projection,
                                      PomdpSolveParams param );
}
```

Class for solving linear programmings by using a java wrapper of lp-solve 5.5:

```
public class Lpsolve{

public static SolverResults solveLPProblem(ArrayList<double[]>
                                           Constraints, double[]
                                           ObjF, double[] RHS,
                                           char[] sense) {
    //The Solution object which will be returned
    SolverResults Sol = new SolverResults();
    if(Constraints.isEmpty())
        return null;
    try {
        int NbColumns = Constraints.get(0).length;
        int NbRows = Constraints.size();
        int ret = 0;
```

```
    //Create Lpsolve linear problem
    LpSolve LpProb = LpSolve.makeLp(0, NbColumns);
    if(LpProb.getLp() == 0) {
        //Linear problem could not be constructed.
        Sol.Status = SolverReturnStatus.ModelCreationFailure;
        ret = 1;
    }
if (ret == 0) {//Linear problem constructed OK to proceed.
    //Set AddRowMode to true
    LpProb.setAddRowmode(true);
    //Create column index Array (used to add constraints)
    int[] ColNo = new int[NbColumns];
    for (int i = 0; i < NbColumns;) {
        ColNo[i++] = i;
    }
//Add constraints
    for (int j = 0; j < NbRows; j++) {
        if(sense[j] == 'L')
            LpProb.addConstraintex(NbColumns, Constraints.get(j),
                                   ColNo, LpSolve.LE, RHS[j]);
        else if(sense[j] == 'E')
            LpProb.addConstraintex(NbColumns, Constraints.get(j),
                                   ColNo, LpSolve.EQ, RHS[j]);
        else
            LpProb.addConstraintex(NbColumns, Constraints.get(j),
                                   ColNo, LpSolve.GE, RHS[j]);
    }

    //Set Add Row Mode back to false
    LpProb.setAddRowmode(false);

    //Set Objective Function
    LpProb.setObjFnex(NbColumns, ObjF, ColNo);

    //Set Direction  Maximize
    LpProb.setMaxim();
    //Solve the Linear Problem
    ret = LpProb.solve();
    if (ret == LpSolve.OPTIMAL) {
        ret = 0;
    }
    else {
        ret = 1;
        Sol.Status = SolverReturnStatus.OptimalSolutionNotfound;
    }
    //Store Optimisation values
    if (ret == 0) {
        double[] VariableResult = new double[NbColumns];
        double[] ConstraintResult = new double[NbRows];
        double Objective = 0;
        double[] DualResult = new double[NbColumns + NbRows + 1];
        double[] Weights = new double[NbRows];
        //Get Primal Solution (variables values)
        LpProb.getVariables(VariableResult);
```

```
            //Get Objective Value
            Objective = LpProb.getObjective();
            //Get Dual Solution (weights)
            LpProb.getConstraints(ConstraintResult);
            LpProb.getDualSolution(DualResult);
            System.arraycopy(DualResult, NbColumns + 1, Weights, 0, NbRows);
            Sol.ConstraintResult = ConstraintResult;
            Sol.DualResult = DualResult;
            Sol.Objective = Objective;
            Sol.VariableResult = VariableResult;
            Sol.Weights = Weights;
            Sol.Status = SolverReturnStatus.OptimalSolutionFound;
    }
    }
    }
    catch (LpSolveException e) {
        e.printStackTrace();
        Sol.Status = SolverReturnStatus.UnknownError;
    }
    return Sol;
  }
}
```

APPENDIX B. POMDP INPUT FILE FORMAT

Description of the file format adopted from Anthony R. Cassandra [27]. The following 5 lines must appear at the beginning of the file. They may appear in any order as long as they preceed all specifications of transition probabilities, observation probabilities and rewards.

discount: a float value between 0 and 1.

values: $[reward, cost]$

states: $[integers, \langle list - of - states \rangle]$

actions: $[integers, \langle list - of - actions \rangle]$

observations: $[Integers, \langle list - of - observations \rangle]$

The definition of states, actions and observations can be either a number indicating how many there are, or it can be a list of strings, one for each entry. For example,

actions: 2

actions: Turn-left Turn-right

After the preamble, there is the optional specification of the starting state. (Note that this is ignored for some exact solution algorithms.) There are a number of different formats for the starting state. You can either:

- enumerate the probabilities for each state,
- specify a single starting state,
- give a uniform distribution over states, or
- give a uniform distribution over a subset of states.

After the initial five lines and optional starting state, the speciifications of transition probabilities, observation probabilities and rewards appear. To specify an entire transition matrix for a particular action: Where each row corresponds

T: action
f f ... f
f f ... f
.. ..
f f ... f

to one of the start states and each column specifies one of the ending states. Each entry must be separated from the next with one or more white-space characters. The state numbers go from left to right for the ending states and top to bottom for the starting states. The new-lines are just for formatting convenience and do not affect final matrix results. The only restriction is that there must be SxS values specified where 'S' is the number of states.

To specify an entire observation probability matrix for an action:

O: action
f f ... f
f f ... f
.. ..
f f ... f

The format is similiar to the transition matrices except the number of entries must be SxO where 'S' is the number of states and 'O' is the number of observations.

To specify individual rewards:

R: action : start-state : end-state : observation f

For any of the entries, an asterick for either ⟨*state*⟩, ⟨*action*⟩, ⟨*observation*⟩ indicates a wildcard that will be expanded to all existing entities. Following is an example POMDP file by our simulation,

```
discount: 0.95
values: reward
```

```
states: 6
actions: 6
observations: 2
start: 1.0 0.0 0.0 0.0 0.0 0.0

T: 0
0.0     0.2     0.0     0.0     0.8     0.0
0.0     0.33    0.33    0.0     0.34    0.0
0.0     0.0     0.33    0.33    0.34    0.0
0.0     0.0     0.0     0.5     0.5     0.0
0.0     0.0     0.0     0.0     0.5     0.5
0.2     0.2     0.2     0.2     0.2     0.0

T: 1
0.33    0.33    0.00    0.00    0.34    0.0
0.0     0.0     0.30    0.00    0.70    0.0
0.0     0.0     0.33    0.33    0.34    0.0
0.0     0.0     0.0     0.5     0.5     0.0
0.0     0.0     0.0     0.0     0.5     0.5
0.2     0.2     0.2     0.2     0.2     0.0

T: 2
0.33    0.33    0.00    0.00    0.34    0.0
0.0     0.33    0.33    0.00    0.34    0.0
0.0     0.0     0.0     0.6     0.4     0.0
0.0     0.0     0.0     0.5     0.5     0.0
0.0     0.0     0.0     0.0     0.5     0.5
0.2     0.2     0.2     0.2     0.2     0.0

T: 3
0.33    0.33    0.00    0.00    0.34    0.0
0.0     0.33    0.33    0.00    0.34    0.0
0.0     0.0     0.33    0.33    0.34    0.0
0.0     0.0     0.0     0.9     0.1     0.0
0.0     0.0     0.0     0.0     0.5     0.5
0.2     0.2     0.2     0.2     0.2     0.0

T: 4
0.33    0.33    0.00    0.00    0.34    0.0
0.0     0.33    0.33    0.00    0.34    0.0
0.0     0.0     0.33    0.33    0.34    0.0
0.0     0.0     0.0     0.0     0.5     0.5
0.0     0.0     0.0     0.0     0.2     0.8
0.2     0.2     0.2     0.2     0.2     0.0

T: 5
0.33    0.33    0.00    0.00    0.34    0.0
0.0     0.33    0.33    0.00    0.34    0.0
0.0     0.0     0.33    0.33    0.34    0.0
0.0     0.0     0.0     0.0     0.5     0.5
0.0     0.0     0.0     0.0     0.5     0.5
0.0     0.0     0.0     0.0     0.0     1

O: 0
```

```
1.0 0.0
1.0 0.0
0.5 0.5
0.4 0.6
0.5 0.5
0.5454545454545454 0.4545454545454545453

O: 1
0.6666666666666666 0.3333333333333333
0.5 0.5
0.5833333333333334 0.4166666666666667
0.4166666666666667 0.5833333333333334
0.5 0.5
0.5 0.5

O: 2
0.0 1.0
1.0 0.0
0.36363636363636365 0.6363636363636364
0.0 1.0
0.7 0.3
0.2857142857142857 0.7142857142857143

O: 3
0.5 0.5
0.625 0.375
1.0 0.0
1.0 0.0
0.7777777777777778 0.2222222222222222
0.0 1.0

O: 4
0.38461538461538464 0.6153846153846154
0.5555555555555556 0.4444444444444444
1.0 0.0
0.36363636363636365 0.6363636363636364
0.5333333333333333 0.4666666666666667
0.5333333333333333 0.4666666666666667

O: 5
0.5 0.5
0.2 0.8
1.0 0.0
0.6666666666666666 0.3333333333333333
0.4166666666666667 0.5833333333333334
0.4 0.6

R: 0 : 0 : * : * -100
R: 0 : 1 : * : * 20
R: 0 : 2 : * : * 9
R: 0 : 3 : * : * 3
R: 0 : 4 : * : * 30
R: 0 : 5 : * : * 9
```

```
R: 1 : 0 : * : * 4
R: 1 : 1 : * : * 3
R: 1 : 2 : * : * 26
R: 1 : 3 : * : * 8
R: 1 : 4 : * : * 30
R: 1 : 5 : * : * 2

R: 2 : 0 : * : * 15
R: 2 : 1 : * : * 7
R: 2 : 2 : * : * 5
R: 2 : 3 : * : * 19
R: 2 : 4 : * : * 25
R: 2 : 5 : * : * 10

R: 3 : 0 : * : * 3
R: 3 : 1 : * : * 2
R: 3 : 2 : * : * 5
R: 3 : 3 : * : * 8
R: 3 : 4 : * : * 16
R: 3 : 5 : * : * 6

R: 4 : 0 : * : * 45
R: 4 : 1 : * : * 3
R: 4 : 2 : * : * 6
R: 4 : 3 : * : * 2
R: 4 : 4 : * : * 5
R: 4 : 5 : * : * 6

R: 5 : 0 : * : * 100
R: 5 : 1 : * : * 9
R: 5 : 2 : * : * 7
R: 5 : 3 : * : * 4
R: 5 : 4 : * : * 7
R: 5 : 5 : * : * 77
```

APPENDIX C. JAVA CODE FOR ACCESSING MYSQL

Java class for accessing MySQL database and calculating the transition probabilities, observations probabilities and rewards.

```
/*If you need to seperate the states from the action names
  you need to make simple changes to the code,
  the changes are gonna be straightforward*/

/*Two function that are to be used are :
  1: assess/_State_prob which asseses the probabilities of
  type P(St1=Assembel_order | st0 = check_avail)
  2: assess_Obs_Prob which assesses the probabilities of type
  P(St1=Assembel_Order | Observation=yes , action = check_avail)
*/

public class assess_prob
{
      private String ConnectionString = "jdbc:mysql://localhost:
                                          3306/wfcomp_db?"+
                                          "user=root&password=123";
     //function to assess the probabilities of P(st1=b|st0=a)
    public double assess_State_prob(String st0,String st1,
                                    String action)
    {
          try
        {
           DB_Query db = new DB_Query(ConnectionString);
           double count =0;
           ResultSet rs = db.Execute_Query("select count(*) as
           cnt from invocation as inv1,invocation as inv2,
           service as srv1, service as srv2 where
           srv1.id=inv1.srv_id and srv2.id=inv2.srv_id and s
           rv1.name='"+st0+"' and srv2.name='"+st1+"' and
           srv2.name='"+action+"' and
           inv2.exec_time=inv1.exec_time+inv2.resp_time
           and inv1.id =inv2.id -1");

           if(rs.next())
              count = Integer.parseInt(rs.getString("cnt"));

          double total = 0;
          total = assess_single(st1,action);
          if(total!=0)
            return (double)(count/total);
          else
              return 0;
        }
        catch(SQLException ex)
        {
            System.out.println(ex.getMessage());
            return 0;
        }
    }
```

```
/*function to assess single probabilities of the
 form P(a=0), (the # of rows having a=0)*/
public double assess_single(String value, String action)
{
      try
    {
         // The Workflow instance is generated
        DB_Query db = new DB_Query(ConnectionString);
        ResultSet rs = db.Execute_Query("select count(*)
                as cnt from invocation,service where
                service.id=srv_id and
                service.name='"+value+"' and
                service.name='"+action+"'");

        int count =0;
        if(rs.next())
           count = Integer.parseInt(rs.getString("cnt"));
        return count;
    }
    catch(SQLException ex)
    {
         System.out.println(ex.getMessage());
          return 0;
    }
}
/*function to assess the probabilities of type
  P(st=st1 | obs = obs1, action = a)*/
public double assess_Obs_prob(String state,String action,String obs)
{
      try
    {
        DB_Query db = new DB_Query(ConnectionString);
        double count =0;
        ResultSet rs = db.Execute_Query("select count(*)
              as cnt from invocation as inv1,invocation as
              inv2,service as srv1, service as srv2 where
              srv1.id=inv1.srv_id and srv2.id=inv2.srv_id and
                srv1.name='"+action+"' and
                inv1.observation='"+obs+"' and
                srv2.name='"+state+"' and
                inv2.exec_time=inv1.exec_time+inv2.resp_time
                and inv1.id =inv2.id-1");

        if(rs.next())
           count = Integer.parseInt(rs.getString("cnt"));

       // calculate the divider
       double total = 0;
       rs = db.Execute_Query("select count(*) as cnt from
             invocation ,service as srv1 where srv1.id= srv_id
             and srv1.name='"+action+"' and observation='"+obs+"'");

       if(rs.next())
           total = Integer.parseInt(rs.getString("cnt"));
```

```
            if(total!=0)
              return (double)(count/total);
            else
                return 0;
        }
        catch(SQLException ex)
        {
            System.out.println(ex.getMessage());
            return 0;
        }
    }

    /*function to assess the COST
      R(st1=state1 | st0=state0, action = a)*/
    public double assess_Reward(String state0,String state1,String action)
    {
        try
        {
           DB_Query db = new DB_Query(ConnectionString);
           double cost =10000; //used as for infinity

           ResultSet rs = db.Execute_Query("select avg(inv2.resp_time)
                as cost from invocation as inv1,invocation as inv2,
                service as srv1, service as srv2 where
                srv1.id=inv1.srv_id and srv2.id=inv2.srv_id and
                srv1.name='"+state0+"' and srv2.name='"+state1+"'
                and srv2.name='"+action+"' and
                inv2.exec_time=inv1.exec_time+inv2.resp_time
                and inv1.id =inv2.id-1");

           if(rs.next()){
               if(rs.getString("cost") !=null)
                   cost = Double.parseDouble(rs.getString("cost"));

           }
              return cost;
        }
        catch(SQLException ex)
        {
            System.out.println(ex.getMessage());
            //for error
            return -1;
        }
    }
}
```