AUTOMATED COMPOSITION OF WEB SERVICE WORKFLOW
A Novel QoS-Enabled Multi-criteria Cost Search Algorithm

Jaina Sangtani and Gursel Serpen
Electrical Engineering and Computer Science Department, University of Toledo, MS 308, Toledo, Ohio, U.S.A.

Keywords: Service Oriented Architecture, Automated Workflow Composition, Heuristic-based Search, Multi-criteria Cost Search, Uniform Cost Search Algorithm.

Abstract: The introduction of software technology has dramatically increased the efficiency of completing tasks. Code reusability provides efficiency within the software engineering discipline. With the tumultuous increase in acceptance of service oriented architecture, and thus, a rise in the number of web services, skilled software developers spend a lot of time composing web service workflows, rather than creating innovative and efficient services. Hence, we put forward a technique of code reusability that utilizes heuristic based search methods to automate service workflow composition by weighting quality of service criteria by relevance and importance to the users. We implement a novel and heuristic-based graph creation and search algorithm where the heuristic function value is calculated through the uniform cost search based on each of the quality of service criteria specified by the user. Application of the proposed automated workflow composition algorithm is illustrated with success on an industry-grade service-oriented architecture problem.

1 INTRODUCTION

With the advent of the computer and the internet, enterprise employees have access to several applications from various providers, to perform their job functions. A single business process involves numerous individuals, applications, and frequently it extends beyond the company's boundaries into partner and customer companies. Information communication between traditional application boundaries does not give insight into the entire business process.

New methods of software development utilize loosely-coupled code to develop integrated and information-centric applications. Thus, application boundaries are no longer restrictions to the business process, as information can flow between these applications, and the entire business process is transparent to those involved. One such method of development with loosely-coupled code is Service Oriented Architecture (SOA) (Newcomer, 2005)(Rao, 2004).

The Sirena Innovation Report quotes several experts in the SOA field asserting that SOA adoption is tremendously growing. Wide acceptance of the SOA notion only began in 2005, and today "every sizable software vendor has stated its future roadmap is going to be SOA related" (Schmelzer, 2005). Agarwal et al. (2005) state "… *web services have received much interest in industry due to their potential in facilitating business to business or enterprise application integration*." Rao and Su (2004) claim that "… *Nowadays, an increasing amount of companies and organizations only implement their core business and outsource other application services over the Internet.*" Thus, the ability to efficiently and effectively select and integrate inter-organizational and heterogeneous services on the Web at runtime is an important step towards the development of the Web service applications. The number of services available over the Web has been increasing dramatically during the recent years, and one can expect to have a huge Web service repository to be searched (Rao et al., 2004).

With an enormous rise in the adoption of SOA and an equitable rise in the number of services created and stored in web service repositories, it has become difficult to effectively and efficiently select and integrate services manually. In addition, the selection and integration of services creates a massive service search space, which is difficult to explore manually, in order to find the correct integration of services to deliver the appropriate result to the users.

Sangtani J. and Serpen G. (2010).
AUTOMATED COMPOSITION OF WEB SERVICE WORKFLOW - A Novel QoS-Enabled Multi-Criteria Cost Search Algorithm.
In *Proceedings of the Fifth International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 122-131

Manual workflow composition is a challenging task for larger scale problems since its manageability for an error-free implementation and consistency becomes almost impossible to achieve as it is the typical case for any user-driven process (Kim, 2004). In order to compose the best solution, a developer must consider each quality of service (QoS) criterion and its importance to the user, and track all of these criteria through the solution process. Simple automation of the process through scripting is not possible, as it requires intelligence to develop a weighted cost value from heterogeneous QoS criteria values, and discover the optimal solution utilizing weighted costs.

The current manual method of SOA composition involves extremely skilled experts, such as software engineers, developers, and architects to create the architecture and integrate services for even the simplest applications or workflows (Hafner, 2009). Expert skill is also required to create innovative and competitive services. "*If a competitor introduces a new service, the service provider must offer a similar or better service within days or weeks, to avoid losing customers*" (Agarwal, 2005). With the pressures of the business and the demand for software developers to create new services efficiently and quickly, there is need for a more efficient method to create the architecture and compose workflows with services. The automated creation of workflows in SOA will allow skilled experts more time to concentrate on the innovative aspects of creating a variety of performance-enhanced and extremely reusable services, rather than the manual, prolonged, and error-prone tasks of composing the workflow.

The need for automated composition of service workflows emerges mainly for the following three reasons: 1) the immense size of the search space associated with the composition problem; 2) the human errors associated with composing service workflows manually; and 3) the drive to reduce the workload of software professionals so that they concentrate on more creative aspects of software development. Hence, in this paper, we present a new algorithm with a novel heuristic for automated workflow composition from a database of services.

In section 2, we discuss related works in the literature, and explain the integration of heuristic search and workflow composition. We describe the proposed algorithm for this in detail in section 3; in section 4 we portray the implementation and testing of the algorithm; and finally, we conclude the paper with section 5.

2 LITERATURE SURVEY

Services are developed to be reused by integrating them with other services to create workflows or applications. There has been a considerable amount of work in this area. Currently, in the industry, service workflows and applications are composed manually by software experts – developers and architects. The literature has a number of propositions to automate this process with logical workflow composition, semantic workflow composition, abstract process model, and artificial intelligence (AI) planning. Some of these works have been implemented, but at the present time there appears to be very little evidence of testing. A brief discussion of each of the prominent approaches ensues next.

2.1 Manual Workflow Composition

As described in Hafner (2009), the manual service workflow architecture has five layers – applications layer, web services composition or publication and discovery layer, service description layer, XML messaging layer, and transport layer. Each of these layers needs to be manually composed and created. Today's frameworks automate portions of this to ease the pain for developers. For example, the .NET framework automates the service description layer, XML messaging layer, and the transport layer. However, there are still tedious manual tasks that the service developer is expected to complete, including the composition of the services into workflows, and their integration with applications.

2.2 Logic-based Service Composition

The rapid development of service-based system approach uses alpha-logic and alpha-calculus in composing automated service workflows. A developer creates the service workflow using alpha-logic; the workflow is first converted into alpha-calculus, and then into executable format.

Savarimuthu claims that the alpha-logic notation is simpler than programming languages (2005). However, comprehension and creation of workflows with the alpha-logic notation requires high level mathematical knowledge and awareness of services. The service workflow composition is not automatic. A software expert's time and efforts are required to develop and implement service workflows with this method. The logic-based workflow composition method is a step ahead of the manual method, but it

requires advanced mathematical background on the part of the software professional.

2.3 Semantics-based Dynamic Service Composition

The semantics-based dynamic service composition approach creates an automated workflow for users through the semantics of a user request (Fujii, 2005)(Qui, 2007). The model used by Fujii, presents the user with a simple textbox through which they must enter their final outcome in plain English for a software development request. The model analyses the narrative in the user's query, and converts it into machine readable format in order to find relevant services, and compose a service workflow.

This method focuses on the translation of the user query to attain services requested by the user, but it disregards optimal search criteria and integration of services. In addition, this method has exponential time complexity, and it adds additional steps with the conversion from English narrative to machine readable format and the dynamic workflow composition.

Qui, et. Al (2007) implement a similar model with a context-aware architecture. However, this model also disregards optimal search criteria and integration of services. Additionally, this model is only tested on one example for applicability, but not for scalability or accuracy.

2.4 Abstract Process Model

The EFlow and Polymorphic Process Model form a static service composition model, where the abstract process model is created by the developer (Rao, 2004). The abstract process model requires definition of tasks and data types. The automation only includes the selection and binding of services to the tasks described in the process model. Similar to the Semantics-Based Dynamic Composition method, the Abstract Process Model focuses on service selection, but not search and integration through the graph. Service search and integration of services still remain as manual processes.

2.5 AI Planning

Artificial Intelligence (AI) planning and associated heuristic search transcends from an initial state to a final state by selecting actions within a domain which is typically modelled by a graph. The final product of a planning algorithm is a sequence of actions that achieves a desired effect (Russell, 2003). In the case of service oriented workflow, actions will be considered to map to services. Hence, each service selection is considered as a logical step in a workflow that changes the current state (starting with the initial state) of the workflow bringing it closer to the user defined business outcome or the goal state. In Figure 1, which displays a service graph, each edge in the graph is a service, and each node is a state. The initial state is the knowledge input by the user. Each state is a combination of the previously known knowledge and the output of the current service. The objective of workflow composition using AI planning is starting from the user-defined initial state, searching for the combination of services that will deliver the goal state within the service graph through an optimal path.

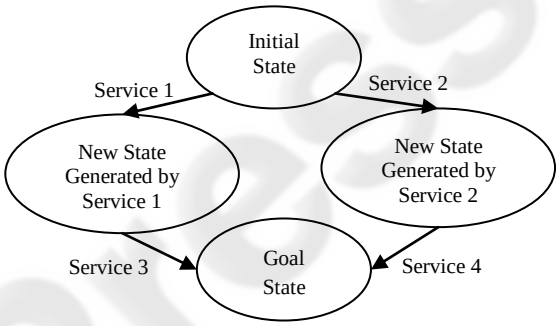


Figure 1: Service Graph.

There are several published studies that "reason about the composition of web services using goal-oriented inferencing techniques from planning" (Canny, 2003). However, the implementation in this area is limited at best. Cheatham and Cox (Cheatham, 2005), and Agarwal, et al. (2008) implement AI planning algorithms for web service composition. However, their work lacks consideration for QoS criteria, and due to lack of a computational complexity analysis, it is difficult to assess the scalability properties of their approach.

2.6 Problem Statement

Automation of the service composition is still an open problem and needed to bring the service oriented architecture methodology to a state of wider applicability and realization. Consequently, a new heuristic-based search algorithm is proposed in this paper to automate the web service composition. The proposed algorithm will search the graph model of a service domain to identify an ordered sequence of services as a plan for the composition of services. The algorithm is able to incorporate QoSs in service

composition with a multi-criteria cost measure. The functional and non-functional properties of services are also taken into account. Functional properties describe the input, output, and the activities conducted by the service. Non-functional properties are the costs borne as a result of execution of each service. Non-functional properties, also known as quality of service (QoS) of a service include aspects of services such as time to execute, resources required for execution, monetary cost of the service, and several other properties that might be of interest (Schuschel, 2004)(Peer, 2004).

3 SYSTEM ARCHITECTURE

The comprehensive solution for web service workflow composition in a Service Oriented Architecture (SOA) context entails the following five steps:

1. Acquisition of customer requirements and specifications
2. Service discovery, selection, and compilation
3. Creation of SOA graph
4. Formulation of workflow path
5. Software Integration and Implementation

The acquired customer requirements and specifications as well as a repository of services are sent into the novel graph creation algorithm. A workflow path is found within the graph utilizing a heuristic based graph search algorithm. The output of the algorithm is a sequence of services sent to a third-party executable software that generates an output value for the goal state specified in the user requirements.

3.1 Acquisition of Customer Requirements and Specifications

The initial state, goal state, and name, maximum value, and weight values for the set of Quality of Services (QoS) are acquired from the user. These entities are supplied as inputs into the web service workflow composition algorithm. It is conceived that these values can be input by a non-technical user through a simple form like a Graphical User Interface (GUI).

3.2 Service Discovery, Selection, and Compilation

Service discovery is mapping customer requirements to services which are semantically and ontologically defined in order to provide the appropriate results to the customer. Service selection and compilation is the obtaining these services from a variety of service repositories and libraries and installing the services within the customer's environment.

Service discovery, selection and compilation is essential to the area of web service composition. However, this topic is a separate field of research, and thus, considered outside the scope of this paper. There are various works of literature that focus on this topic. For example, Fujii discusses semantical representation of services through a model called Component Service Model with Semantics (COSMOS). COSMOS is implemented alongside Component Runtime Environment (CoRE) for service discovery and selection (Fujii, 2005). Additionally Qiu, et al, discuss the implementation of semantical web service selection with a context aware architecture (Qiu, 2007)

3.3 SOA Graph Creation

The SOA graph, which is proposed herein as a novel abstract model for the service workflow domain, is a specialized representation of the services selected based on the customer requirements and the normalization process. In the implementation of the heuristic-based search algorithm, the service names, their Web Service Description Language (WSDL) file URLs, and their associated QoS values are stored in a database within the customer environment, accessible to the search algorithm.

A state in the web service workflow composition domain is defined as a fixed-size set of attribute-value pairs. An attribute is any variables whose value is known at the end of a service execution. A state's attribute-value pairs are composed of the previous state's attribute-value pairs along with the outputs generated through the most recently executed service.

Each state of the environment is represented as a node into the graph. Services connect two nodes if the input of a service is contained within the source node state. The output of the service associated with the outgoing directed edge from the source node state composes the successive node state of the service. For each state in the environment, beginning with the initial state input by the customer, services are iterated through to find one(s) with desirable (or

target or goal) input variable-value (attribute-value) assignments.

Both the data type and the name of the input/knowledge criteria variables must match. In Figure 2, Node 1 contains the value for the knowledge criteria variable *x*. A service is found that has input *x*, and is connected to the graph at Node 1. The service has an output *y*. Thus, the knowledge criteria of the next node, Node 2, include the knowledge criteria of Node 1 and the output of the service (assignment to the *y* variable). The input of the next service requires both *x* and *y* variables, and the output is the value assignment for the *z* variable. Thus, the knowledge criteria of the next node, Node 3, are a combination of the knowledge criteria of the previous node, Node 2, and the output of the service (value assignment to the *z* variable).

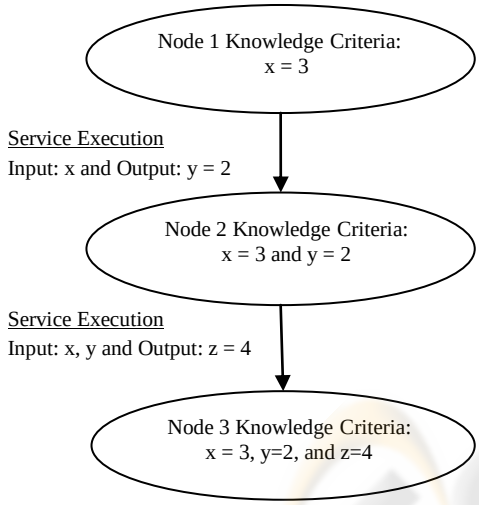


Figure 2: Knowledge Criteria.

The graph creation algorithm expects as input the initial state, the desired goal state, the name, and weight and maximum allowed value of each QoS criterion of importance to the user. All available services that match the QoS criteria described by the user are considered. Each QoS value of a service is normalized as follows:

$$N_j = Q_j / M_j, \quad (1)$$

where N_j is the normalized QoS value, Q_j is the original QoS value of the service, M_j is the maximum QoS value as specified by the user, and *j* depicts the current QoS criterion with *j*=1,2,...,*q* (*q* is the number of QoS criteria input by the user). All services with N_j values greater than 1 will not be considered for the algorithm, because their QoS values are greater than the maximum allotted by the user.

3.4 Formulation of Workflow Path

3.4.1 Heuristic Search Algorithm

The new heuristic-based search algorithm created specifically for service workflow composition is described in Figure 3. The proposed search algorithm takes into account the functional and non-functional properties of services, and selects services in the graph based on the QoSs of the service. As the QoSs that are important to the user can be difficult to anticipate, the proposed algorithm allows for multiple QoSs.

The heuristic-based search algorithm, starting from a given initial state or node *u*, selects the neighbour n_i with i=1,2,…,b_u, where b_u is the neighbour count of node *u*, with the smallest evaluation function value and adds it to the path while discarding the other neighbours. Next it expands this node n_i and calculates evaluation function values for its neighbours while retaining the neighbour with the smallest evaluation function value as part of the plan or solution path. This process continues until the goal node is reached. The evaluation function $f(n_i)$ is calculated for each neighbour node n_i through

$$f(n_i) = g(n_i) + h(n_i) \text{ for } i = 1,2,\dots,b_u. \quad (2)$$

To calculate $g(n_i)$, the weighted sum of the QoS criteria is added to $g(u)$ of the parent node *u*:

$$g(n_i) = g(u) + \sum_{j=1}^{q} \alpha_j N_j \text{ for } i=1,2,\dots,b_u, \quad (3)$$

where *j* is the current QoS criterion, and *j*=1,2,...,*q* (*q* is the number of QoS input by the user), N_j is the normalized value of the current QoS for the service that connects nodes *u* and *n*, and α_j is the weight of the current QoS as input by the user. The $g(u)$ value of the initial node is defined as 0.

To calculate $h(n_i)$, the uniform cost search algorithm is executed with node n_i as the start node once for each of the QoS criteria. The uniform cost search algorithm returns the optimal path with the least cost for each QoS criterion *j*, which is designated as $L_{i,j}$. In the process, the appropriate QoS value is stored with each node along the path of the best path. The heuristic function $h(n_i)$ is then calculated with the following formula:

$$h(n_i) = \sum_{j=1}^{q} \alpha_j L_{i,j} \text{ for } i=1,2,\dots,b_u \quad (4)$$

Multi-criteria Search Algorithm Pseudocode

Let the initial node be the start node of the path and set the current node u to the initial node.
For the current node u, expand it by generating all its neighbours n_i with i=1,2,…,b_u, where b_u is the neighbour count of node u.
For each n_i
  For each QoS criterion j with j=1,2,...,q
    Compute $N_{i,j} = Q_{i,j} / M_j$
    Compute $L_{i,j}$ with the uniform cost search
    Compute $h(n_i) = \sum_{j=1}^{q} \alpha_j L_{i,j}$
    Compute $g(n_i) = g(u) + \sum_{j=1}^{q} \alpha_j N_{i,j}$
    Compute $f(n_i) = g(n_i) + h(n_i)$
Select the n_i with the smallest $f(n_i)$ value
If the selected node n_i is not the goal, then set the selected n_i as the current node u and repeat.

$N_{i,j}$ is the normalized value of QoS criterion j for service i; $Q_{i,j}$ is the original value of QoS criterion j for service I; M_j is the maximum value of QoS criterion j as input by user; $L_{i,j}$ is the least cost value of QoS criterion j from service i to the goal node as found by the uniform cost search algorithm; and α_j is the weight of the QoS criterion j as input by user.

Figure 3: Pseudocode of Proposed Algorithm.

3.4.2 The Uniform Cost Search Algorithm Illustrated

The application of the uniform cost search algorithm to compute the least cost path with respect to a specific QoS criterion value j from a given node n to the goal node is illustrated in Figure 4. The neighbour node n is depicted by node 1 in the figure. Once node 1 is expanded, both nodes 2 and 3 are added to the fringe as neighbors of node 1. Service 1 connects node 1 and node 3, and has N_j =0.1 as the normalized value of the current QoS criterion. This value is stored as the g value of node 3. On the other hand, Service 2 connects node 1 and node 2, and has N_j=0.4, which is stored as the g value of node 2. Since, the g value of node 3 is less than that of node 2, node 3 is chosen for expansion. Node 3 has only one neighbour, node 4, which is added to the fringe. Thus, the fringe now contains node 2 and node 4. Note that nodes 3 and 4 are connected by Service 3, whose N_j value is 0.7. Accordingly, the g value of node 4 is calculated as follows: g value of node 2 (0.1) is added to the N_j value of Service 3 (0.7) to yield 0.8. The g value of node 4 is greater than that of node 2; hence, node 2 is expanded next noting that the goal node is a neighbour of node 2. Service 4, connecting node 2 to the goal node, has a N_j value of 0.3. Now the g value of goal node becomes 0.4+0.3=0.7. Next, the goal node is expanded since its g value is less than that of node 4.

Based on the g values, the path selected from node 1 to the goal node is via node 2. The least cost value, L_j, of a specific QoS criterion where j=1,2,...,q, from any particular node on the selected path to the goal node is stored within the node. Node 2 has an L_j value of 0.3, the value of Service 4, which connects it to the goal node. Node 1 has an L_j value of 0.7, the values of Service 2 and Service 4 added, which connect it to the goal node. Since, node 3 and node 4 were not on the selected path, they do not have L_j values stored. Storing the L_j value of each consecutive node ensures that when the node is selected as a neighbour node n by the heuristic search algorithm in the future, the L_j value of that node will be known, and the uniform cost algorithm will not need to be executed. For example, if node 2 is selected as the n, the uniform cost algorithm is not executed to find L_j of node 2, as it is already known that L_j of node 2 is 0.3.

3.4.3 Time and Space Complexity

The time and space complexity for the uniform cost search algorithm is given by $O(b^{1+\lfloor C^*/\epsilon \rfloor})$, where b is the branching factor, C^* is the cost of the optimal solution, and ϵ is the least cost of an action (Russell, 2009). There are two loops encircling the uniform cost search invocations. The outer loop iterates b_{ave} times, where b_{ave} is assumed to be the average branching factor of the search tree associated with the SOA graph. The inner loop executes q times, where q is the number of QoS criteria. Assuming that the maximum depth of the search tree (composed from the problem domain graph with realization and removal of loops) is m, there will be m invocations of the overall algorithm. Hence, the time complexity is given by

$$mqb \times O(b^{1+\lfloor C^*/\epsilon \rfloor}) \cong O(qmb^{2+\lfloor C^*/\epsilon \rfloor}) \quad (5)$$

where m, q, and b are finite-valued positive integers.

Only one uniform cost fringe is maintained and remains in memory at any given time. Every time the loop is restarted, the previous uniform cost fringe is discarded from memory. Hence, the worst case complexity for space is given by

$$O(m + b^{1+\lfloor C^*/\epsilon \rfloor}) \quad (6)$$

where m is the maximum depth of the search tree, which is kept in the memory for the heuristic search algorithm, and $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ is the space cost of the uniform cost search algorithm.

3.5 Software Integration and Implementation

The database of services, selected by a semantic service selection system, or by a developer, along with the algorithms for graph creation and search algorithms, and the executable OW2 Orchestra software are packaged and deployed within the user's environment for a complete solution. The user interface to the database is employed to input user requirements into the system and to extract the final output from the system is also included.

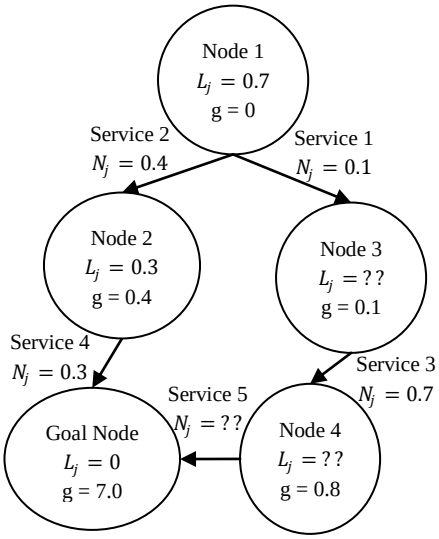


Figure 4: Uniform Cost Search Illustrated.

4 SIMULATION STUDY

The algorithm has been implemented as a web service using C# and the .Net 3.5 framework with the Visual Studio 2008 IDE from Microsoft Dreamsparks (Microsoft Corporation, 2010). The implementation includes a database for storage of services, developed with SQL Server 2008, also from Microsoft Dreamsparks (Microsoft Corporation, 2010). The name and URL of the service WSDLs are stored in a single table. A second table is created for the QoS values as related to the service.

A user interface (UI) to the web service that has fields to allow for a user to input the variables and fields of the initial state and the goal state was created. Users can add as many QoS criteria as needed by entering their names, weights, and the maximum allowable value. The UI additionally allows for new services to be entered into the database.

The SOA graph creation and heuristic-based search algorithms were tested on a 64-bit Windows 7 OS computer with an Intel cpu (core 2 duo 2.8 GHz processor), and 4 GB memory. In testing, complexity of the problems was managed through the following attributes:

a. Number of branches in the workflow (path)
b. Number of branches in the workflow (conditional)
c. Similarities of services (the more similar, the more complex)
d. Number of services required to complete the workflow

Scalability of the algorithm implementation was empirically assessed by changing the following attributes:

a. Total number of services available
b. Number of input and output parameters per service

Testing was accomplished on the following problem domains that exhibit varying degrees of complexity:

1. UK National Health Service (booking appointments, checkup, and prescription tracking) (Srivatava, 2010)
2. PostFinance (ordering services, payments, and transaction management) (Srivatava, 2010)
3. Harrods™ (online shopping, finding products, and transaction) (Srivatava, 2010)
4. Vanco (trouble ticketing workflow) (Agarwal, 2005) (Srivatava, 2010)
5. Credit Card Request (client verification, accepting/rejecting client, account processing) (SAP, NetWeaver, 2010)
6. Parts maintenance (reporting issue, approval process, analyze and fix defect, confirm work) (SAP, NetWeaver, 2010)
7. Getting a standardized permit (application processing, approval, invoice execution, creating and sending permit) (SAP, NetWeaver, 2010)
8. Finding Energy Product (request, providing data, determining, and offering a quote) (SAP, NetWeaver, 2010)
9. Customer Quote Request (requirements, design, providing a quote) (Jennings, 1996)
10. Travel Itinerary (creating itinerary for hotel and flight, billing) (Srivastava, 2003)

These problems involve complex branching, conditional and recursive branching, and a large number of services and inputs. Some of these are composition problems in the industry, whereas others such as examples 4, 9, and 10 are benchmark

problems used in the literature (Agarwal, 2005) (Jennings, 1996) (Srivastava, 2003).

The proposed algorithm has been applied to all ten problems listed above with success, however, due to space limitations, a representative but non-trival problem will be elaborated upon below. Optimality and complexity results for all the problems will be briefly presented. Accordingly, the workflow composition by the proposed algorithm for the Customer Quote Request problem (example 9) is described. This problem is the most complex in terms of branching and it deals with the most number of services as well.

4.1 Customer Quote Request Problem

The customer quote request problem requires a network to be designed based on customer requirements. The goal state of the problem is a customer quote returned. The customer submits requirements. If it is a veterinary customer, and they do not need anything to be changed, the process is terminated. If they do need changes, the requirements are identified. If it is a portfolio item that the company has, the service ID and the price of the service are simply provided as a quote to the customer. If it is not a portfolio item, a legal review is conducted, requirements are analyzed, a survey maybe conducted, a network designed, and a quote provided. (Jennings, 1996)

For the example problem, the maximum QoS values and weights are shown in Table 1. As inputs into the algorithm, QoS names, maximum values, weights, a customer requirement, customer details, and the goal state of a received customer quote are entered.

Table 1: Maximum QoS for Customer Quote Request Problem.

QoS Criterion	Maximum Value	Weight
Duration (minutes)	320	1
Volume	35	2
Price (per costing)	35	3
Penalty	30	4

There are several services in the database for this example. Table 2 displays those services in the database, and their QoS values for duration, volume, price, and penalty. Some services have variations with different QoS values. For example, there are 3 "Provide Quote" services with different values for duration, volume, price, and penalty. Some services in the table provide logic for multiple services. For example, the "Capture Customer Details - Capture Customer Requirements" 1 and 2 services in Table 2 provide the logic for "Capture Customer Details", "Is Vet Customer", Is Customer Okay" and "Capture Customer Requirements".

Figure 5 displays a part of the graph created from the services above, while noting that due to space limitations, the entire knowledge criteria for a given node (state) could not be shown. The "Yes" and "No" in some of the states in the graph is the conditional branching. The algorithm deals with the conditional branching by analyzing the value of the output of requests, along with the data types and names. For example, for the output of the "Is Customer Okay" service, the "isCustomerOkay" output variable is analyzed for its data type and name, but also for its value, "Yes" or "No" to pick the correct next service. "Identify Service Requirements Profile" will expect the "isCustomerOkay" variable to be assigned a value of "Yes" for its input.

If, for instance, all the conditions have a "Yes" output during the execution, the workflow output by the algorithm is a sequence of the following services:

1. Customer Details 1
2. Is Vet Customer
3. Is Customer Ok
4. Customer Requirements 2
5. Identify Service Requirements Profile
6. Is Portfolio Item
7. ID Service
8. Provide Quote 1

Branches are found for Customer Details, Customer Requirements, and Provide quote. Customer Details and Requirements 1 and 2, are immediately not considered for having price values (149.99 and 299.99, respectively) greater than the maximum allowed value (35).

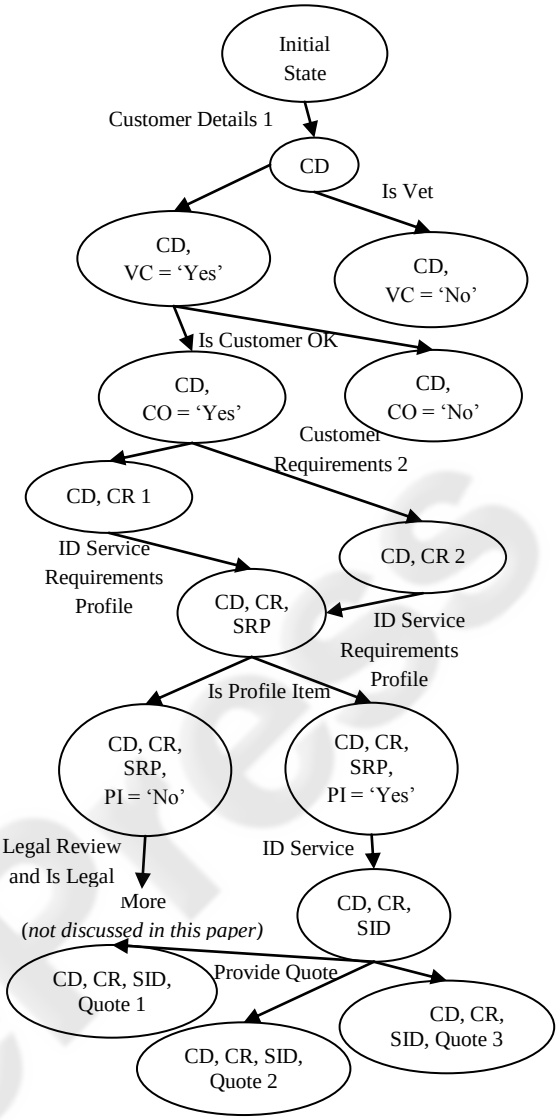
Both the "Customer Requirements" and the "Provide Quote" service branches join back to the same states in the graph. Hence, only their *g* values are considered for selection, as all their *h* values are the same. Additionally, the uniform cost search algorithm is run only twice per QoS criterion – at the initial state and at the customer requirements state. For example, the uniform cost search algorithm is run from the node "initial state" for each QoS criterion. For most nodes, there are no branches in this graph, and those nodes are selected for each QoS criterion. Examples of these nodes are "Customer Details 1", "Is Vet Customer", "Is Customer OK", etc. Thus, the least cost of each QoS criterion from these nodes to the goal node is stored within these nodes by the uniform cost search

Table 2: Services and QoS.

Service Name	Duration	Volume	Price	Penalty
Capture Customer Details - Capture Customer Requirements 1	0	5	149.99	50.0
Capture Customer Details - Capture Customer Requirements 2	1	0	299.99	1.0
Capture Customer Details 1	2	2	0.00	2.0
Capture Customer Requirements 1	25	13	5.00	2.0
Capture Customer Requirements 2	0	2	5.95	0.1
Is Vet Customer	0	0	0.00	0.0
Is Customer Okay	0	0	0.00	0.0
ID Service Requirements Profile	30	0	0.00	0.0
Is Portfolio Item	0	0	0.00	0.0
ID Service	5	0	0.00	0.0
Provide Quote 1	1	0	5.00	0.1
Provide Quote 2	12	15	1.00	0.0
Provide Quote 3	12	24	9.95	0.0
Legal Review and Is Legal	0	0	0.00	0.0
Analyse Requirements and Is Survey Required	0	0	0.00	0.0
Survey CPE 1	12	0	15.00	0.0
Survey CPE 2	1	0	59.50	0.0
Survey CPE 3	12	13	0.00	0.0
Design Network	30	15	10.00	0.0
Request Further Info 1	1	0	5.00	0.1
Request Further Info 2	12	15	1.00	0.0
Request Further Info 3	12	24	9.95	0.0

algorithm. The heuristic based search will also select these nodes because of no branching. Before these nodes are selected, their *h* values need to be computed. Computing the *h* value does not require the uniform cost search algorithm to be run. In fact, it can be computed as a weighted sum of the stored least cost values for each QoS criterion. Thus, the uniform cost search algorithm is run only one other time, when branching occurs, at the "Customer Requirements" nodes.

In the final workflow path, Customer Requirements 2 and Provide Quote 1 are both selected for smaller *g* values and all the *h* values at both branches are the same. In this example, after branching, the states coincide later in the tree. Hence, the *h* values have lesser necessity. This may not always be the case, however.



CD = Customer Details, VC = Is Vet Customer?, CO = Is Customer OK?, CR = Customer Requirements, SRP = Service Requirements Profile, PI = Is Profile Item?, SID = Service ID

Figure 5: Partial Graph of the Customer Quote Request Problem.

4.2 Discussion

The time for running the algorithm for the Customer Quote Request workflow with the "Yes" path is 0.3 minute, whereas running the algorithm on the "No" path for "Is Portfolio Item", and considering a path via the "Customer Details and Requirements" service is 0.97 minute. The "No" path has double the depth (m) of the "Yes" path. Also, the number of steps in the uniform cost search algorithm (C^*/ϵ)

will be a greater value. However, the rest of the parameters (*b* and *q*) maintain the same values.

Table 3 displays of the results for all ten problems. In all cases, optimal solutions were computed by the proposed algorithm with very low time and space costs.

Table 3: Complexity and Optimality Results for All Problems.

Problem	Time (minutes)	Space (in KB)	Optimal Solution
UK National Health	0.36	11,283	Yes
PostFinance	0.84	31, 748	Yes
Harrods	0.25	8,360	Yes
Vanco	0.73	31,732	Yes
Credit Card Request	0.46	9,293	Yes
Parts Maintenance	0.59	23,940	Yes
Standardized Permit	0.89	37,462	Yes
Energy Product	0.12	5,000	Yes
Customer Quote Req	0.97	41, 248	Yes
Travel Itinerary	0.93	45, 888	Yes

5 CONCLUSIONS

We have presented a novel heuristic-based search algorithm for automated composition of a web service workflow subject to multiple QoS criteria. The algorithm has been successfully tested on a number of complex real-life problems. Simulation study and results indicate that the proposed heuristic-based search algorithm can successfully address challenging web service composition problems within reasonable computational cost bounds.

REFERENCES

Newcomer, E; Lornow, G (2005). *Understanding SOA with Web Services.* Addison Wesley.

Rao, J., & Su, X. (2004). A Survey of Automated Web Service Composition Methods. *Proceedings of the 1st international workshop on semantic web services and web process composition*, (pp. 43-54).

Schmelzer, R. (2005). The Explosive Growth of Service-Oriented Architecture Adoption. *Information Technology for European Advancement.* Sirena Innovation Report.

Agarwal, V., Dasgupta, K., Karnik, N., Kumar, A., Kundu, A., Mittal, S., et al. (2005). A Service Creation Environment Based on End to End Composition of Web Services. *Proc. of the Fourteenth World Wide Web Conference.* Chiba: ACM.

Kim, J., Spraragen, M., & Gil, Y. (2004). An Intelligent Assistant for Interactive Workflow Composition. *Proceedings of the International Conference on Intelligent User Interfaces.* Madeira: ACM

Hafner, M., & Breu, R. (2009). *Security Engineering for Service-Oriented Architectures.* Springer Berlin Heidelberg. ISBN 978-3-540-79539-1

Savarimuthu, BTR. (2005). Agent-based integration of web services with workflow management systems. *Information discussion paper series*, *05*, Retrieved October 1, 2008.

Fujii K., Suda T. (2005). Semantics-based dynamic service composition. *IEEE journal on selected areas of communication*, *23*, Retrieved April 27, 2008.

Qiu, L., Change, L., Lin, F., Shi, Z. (2007). Context optimization of AI planning for semantic web service composition. *Service Oriented Computing and Application*, (pp. 117-128).

Canny, J. F., Edwards, D. D., Malik, J. M., & Thrun, S. (2003). *Artificial Intelligence: A Modern Approach.* Upper Saddle River: Pearson Education, Inc.

Cheatham, M., & Cox, M. T. (2005). AI Planning in Portal-based Workflow Management Systems. *Proc International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, (pp. 47-52).

Agarwal, V., Chafle, G., Mittal, S., Biplav, S. (2008). Understanding approaches for web service composition and execution. *Proc. of the 1st Bangalore annual Compute Conference.*Bangalore: COMPUTE.

Schuschel, H., & Weske, M. (2004). Automated planning in a service-oriented architecture. *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. *13*, 75-80.

Peer, J.; Vokovic, M. (2004). Towards automatic web service composition using AI planning techniques. *Proc. of the European Conf. on Web Services (ECOWS'04)*. Springer-Verlag.

Russell, J. S., & Norvig, P. (2003). *Artificial Intelligence: Modern Approach.* Prentice Hall.

Microsoft Corporation. (2010). Retrieved January 2010, from Microsoft DreamSpark: https://www.dreamspark.com/default.aspx.

Jennings, N. R., Faratin, P., Johnson, M. J., O'Brien, P., & Wiegand, M. E. (1996). Using intelligent agents to manage business processes. *Proc. of the First Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology*, (pp. 345-360).

Srivastava, B., & Koehler, J. (2003). Web Service Composition - Current Solutions and Open Problems. *ICAPS2003 Workshop on Planning for Web Services.* Trento.

Srivastava, S. (2010) Sun's SOA & Solaris TM Enterprise System. *Sun Microsystems, Inc.*

SAP NetWeaver (2010). *Enterprise Use Cases.* Retrieved January 2010, from SAP Community Network: http://www.sdn.sap.com/irj/sdn/index?rid=/webcontent/uuid/a0cac44a-5820-2b10-96ae-c7494c38dec0.