2013

Towards optimal service composition upon QoS in agent cooperation

Chattrakul Sombattheera
Mahasarakham University, cs50@uow.edu.au

Jun Shen
University of Wollongong, jshen@uow.edu.au

Publication Details

Sombattheera, C. & Shen, J. (2013). Towards Optimal Service Composition upon QoS in Agent Cooperation. International Journal of Computational Science and Engineering, 8 (2), 119-132.

Towards optimal service composition upon QoS in agent cooperation

Abstract
It is quite common in tourism industry that a tourist would love to gain the most wonderful experience from visiting multiple places in one trip. This is a service composition problem and is difficult to manage because of several reasons. We address this problem by proposing an agent-based service composition framework to allocate to the tourist an optimal composite service. We take into account a number of factors including: 1) all the places of interest must be visited; 2) the preference on visiting places must be obeyed; 3) the total price is within the budget; 4) the time constraint must be obeyed; 5) the payoffs for service providers are worthwhile and fair. We propose a bottom-up approach to allocate the optimal service composition where intelligent agents are deployed to provide flexibility and efficiency to the system. As a result, the system is more independent and every party is better off.

Keywords
cooperation, towards, agent, composition, service, optimal, qos, upon

Disciplines
Engineering | Science and Technology Studies

Publication Details
Sombattheera, C. & Shen, J. (2013). Towards Optimal Service Composition upon QoS in Agent Cooperation. International Journal of Computational Science and Engineering, 8 (2), 119-132.

Towards Optimal Service Composition upon QoS in Agent Cooperation

Chattrakul Sombattheera*

Faculty of Informatics,
Mahasarakham University,
Khamreang, Kantarawichai, Mahasarakham , 44150 Thailand
Fax: +66 43 754 359
E-mail: chattrakul.s@msu.ac.th
*Corresponding author

Jun Shen

School of Information Systems and Technology (SISAT),
Faculty of Informatics,
University of Wollongong,
NSW 2522 Australia
Phone: +61 2 4221 3873
E-mail: jun_shen@uow.edu.au

Abstract: It is quite common in tourism industry that a tourist would love to gain the most wonderful experience from visiting multiple places in one trip. This is a service composition problem and is difficult to manage because of several reasons. We address this problem by proposing an agent-based service composition framework to allocate to the tourist an optimal composite service. We take into account a number of factors including: 1 all the places of interest must be visited 2 the preference on visiting places must be obeyed 3 the total price is within the budget 4 the time constraint must be obeyed 5 the payoffs for service providers are worthwhile and fair. We propose a bottom-up approach to allocate the optimal service composition where intelligent agents are deployed to provide flexibility and efficiency to the system. As a result, the system is more independent and every party is better off.

Keywords: service composition; preference; optimal coalition structure; multiagent systems; cooperative game.

Reference to this paper should be made as follows: Sombattheera, C. and Shen, J. (2012) 'Towards Optimal Service Composition upon QoS in Agent Cooperation', *Int. J. Computational Science and Engineering*, Vol. 1, Nos. 1/1, pp.111–111.

Biographical notes: C Sombattheera is the Leader of the Mahasarakham Intelligent Systems Laboratory at the Faculty of Informatics, Mahasarakham University, Thailand. He completed his PhD in 2010 from School of Computer Science and Software Engineering, Faculty of Informatics, University of Wollongong, NSW, Australia. His research interest includes coalition formation, multiagent systems, game theory, optimization and service composition.

Jun Shen is a senior lecturer of School of Information Systems and Technology (SISAT), leader of the Modelling Agents and Service Oriented Systems (MASOS) research group, at University of Wollongong, Australia. His research interests include business process ontology, semantic Web, Web services, knowledge Grid, mobile services. He has published more than 70 papers and is a senior member of IEEE and ACM.

1 Introduction

Since its emergence, the Web has brought us web-based applications which extends our reach to outside world more thoroughly through the channel of web services. The success of this technology has developed the need for more complicated web service application, which is composed of multiple web services. Such a technology is known as *service composition* or *composite web services*, which combines several web services in a more complex

and well structured manner to deliver a comprehensive service to customers. On top of composite web services, there is another dimension arisen from this area of research. Such a new direction is to combine services in an optimal manner to satisfy customers on a certain criteria. This new area is known as *optimal service composition*.

Among many real world domains where composite web services can be applied, tourism industry is also a very popular area in which researchers in composite web services are interested. A common scenario in tourism industry is for a web service to compose a trip based on the requirement given by the tourist. Such a requirement is usually composed of the budget, the time frame, the preferences on accommodation and food, which the tourist has. The task of the web service composition is to compose a set of service providers, e.g. hotels, airliners, etc., as per request. In this research we consider a domain where the tourist is interested in visiting places such that her satisfaction is maximal.

1.1 Motivation

It is quite common in tourism industry that a tourist would love to gain the most wonderful experience from visiting multiple places in one trip. However, most single travel packages offered by companies may not have all the interested places to the tourist. If we choose to combine these packages, the tourist may have to visit the interested places more than once. For many tourist, visiting the same place more than once is obviously unnecessary and boring. On the other hand, this may be pleasant to some tourists because they may love to visit their favorite places more often. However, visiting the same place too often can be too much too. Furthermore, combining packages is not easy to manage because companies may not like to cooperate due to several reasons. These companies, for example, are competitors in nature. Even if they decide to cooperate, there are many management issues to be solved. Most importantly, what will be the attractive payoff for these companies? Will the payoff be worthwhile for their effort and be fair among them?

We consider this is a service composition problem and is difficult to manage due to several reasons. Firstly, in order to satisfy the tourist, her preference must be precisely captured. Secondly, the preference is individually subjective to tourists. Some tourists may prefer to visit each place once, while enthusiastic ones may prefer to spend more time to the favorite places. Thirdly, composing a service is computationally complex because there are several services be offered in the market. Lastly, the composite service are likely to require cooperation among companies who are unlikely to cooperate. Hence it is a challenging problem to find a solution to satisfy the tourist and these companies.

This problem is clearly a complex one. The ultimate package to be offered to the tourist will be composed of (partial) services (each of which is taking the tourist to a particular place) from various companies such that the tourist is satisfied and every collaborative company is better off. From research perspective, it is a combinatorial optimization problem. The complexity is of class NP-hard. The question here is that how we could allocate available resources, i.e. the separate services, to satisfy the tourist, taking into account performance and economical point of view.

We extend [21] to address this problem by proposing an agent-based composite web-services framework to allocate to the tourist an optimal service composition, the one which maximally satisfies the tourist. By saying optimal service composition, we take into account a number of factors including i) all the places of interest must be visited, ii) the number of redundant places must be minimal, iii) the total price is within the budget, iv) the time constraint must be obeyed, and v) the payoffs for service providers are worthwhile and fair. We apply the concept of cooperation and optimization among agents to compose the service. This concept allows for a rapid search the the optimal composite service which maximally satisfies the tourist and proposes to companies an attractive payoff scheme. The framework also addresses weaknesses in current composite web-services technology that deploys a top-down approach where service providers are to be chosen by a service broker. We propose a bottom-up approach to allocate the optimal service composition where intelligent agents are deployed to provide flexibility and efficiency to the system. As a result, intermediate parties, such as request brokers, are less important and the system is more independent. Most importantly, every party involved are better off.

This research propose a general framework deploying intelligent agents and an algorithm to appropriately allocate available resources to service requesters. There are 3 types of agents:

- The requester agent (RA),
- The request collector agent (RCA), and
- The service provider agent (SPA).

These agents exchange information and know about each other's role, availability and capability. With the algorithm proposed, the optimal service composition will be decided by the customer.

1.2 Structure of the Paper

The remaining contents of this paper is structured as following. Section 2 surveys related work in optimal composite web services as well as the coverage on the underpinning concepts of multiagent systems used to solve the optimization problem. Section 3 describes the proposed architecture where each RCA proposes to the customer its solution. Section 4 deals with the algorithm used to allocate service providers to satisfy the customer optimally. Section 5 discusses the experiments

and presents the results. Section 6 concludes the research conducted.

2 Literature Review

Since this work proposes a new framework in composite web-services, this section reviews related research including composite web-services, cooperative game, multi-agent systems, and optimal coalition structures.

2.1 Optimal composite web services

A web service is an Internet-based application that is composed of several components distributed across the Internet. These components communicate to each other by exchanging requests and responses throughout the process cycle of the application. Requests and Responses are referred to as messages. These messages are made of XML tags. These tags are constructed based on certain languages/protocols. These languages/protocols are business- oriented and are used specifically for a particular area. While web services area has gain more and more attention from researchers, the need for more complex scenarios, where multiple layers of service providers cooperatively combine to solve the problem for requesters, has lead to another area of research in web services. This area is known as composite web services. The early work of composite web services focus merely on accomplishing the request of the requester.

However, this leaves a major flaw in composing such web services that the performance is not taken into account. Arisen from this is a new area of research, known as optimal composite web services. Optimal composite web services is the area of web services that aims to construct the best services possible from available services providers based on certain criteria. In the following, we will explore previous work in the area.

Huang, Lan and Yang [2], propose a QoS-based scheme to compose optimal services which helps service requesters select services based on single QoS-based service discovery and QoS-based optimization of service composition. A number of involved attributes include *i*) response time, which is the time required for the service providers to get back to the requesters, *ii*) reliability, which is the ability to provide requested functionality, *iii*) availability, which is the degree or frequency that the service is accessible and operational, and *iv*) price, which is the cost of service request. Multiple criteria for making decision and selecting optimal service are proposed. The scheme is efficient and works well for complicated scenarios. While their experiments look for number of tasks that can fit a request, we look for retrieving an optimal solution quickly. Furthermore, their number of tasks (98) involved is far lower than what we will do (1382958545 services, i.e the number of set partitions for 15 agents).

Cheng and Koehler [1] study how to achieve optimal pricing policies for web-enabled application services. Service providers provide a contractual service offering to deploy, host, manage, and lease what is typically packaged application software from a centrally managed facility. The application ranges from standard productivity tools to expensive applications such as Enterprise Resource Planning systems like SAP or PeopleSoft. The economic dynamics between these service providers and their potential customers are modeled to consider a two-part pricing scheme. Bulking was not considered but the service providers reimburses customers for time spent waiting for services. In addition, they required a minimal average performance guarantee. They proposed that "the optimal pricing policy need not be in the form of a fixed fee, metered price, two-part tariff, or two-part tariff plus reimbursement". They showed that there exists a unique rational expectation equilibrium. This work also differs from ours because we look into different domain and focus on achieving the optimal solution (package) quickly.

Lin, Liu, Xia and Zhang [8] tried to find the optimal capacity allocation in a clustered Web system environment so as to minimize the cost while providing the end-to-end performance guarantees. Constraints on both the average and the tail distribution of the end-to-end response times were considered. They deploy a nonlinear program to solve the problem. The results show that, under a certain condition, the solution yields a nice geometric interpretation. Also, the algorithm can yield asymptotically optimal solution for stringent service requirement. Although this work considers the problem a nonlinear function like we do, the problem domain is different. Furthermore, they are interested in end-to-end delay while we are interested in fitting maximal number of places to visit in timely fashion.

Tang and Cheng [26] study the optimal pricing and location strategy of a Web service intermediary (WSI). They model the problem as a linear city model and then extend their research on the more general model. Their analysis show that that the optimal strategy can be derived by delay cost, integration cost, and prices of the constituent Web services. Their results show that the WSI can be optimally located between the Web service providers. The also found that a penetration price can be charged when the delay cost is low. Furthermore, they also propose that multiple optimal locations for the WSI can be obtained when the proximity of Web service providers are dispersed. This work differs from our work that we consider different domain and we look forward to achieving optimal solution in timely fashion.

2.2 Cooperative game

As we have briefly discussed previous works in optimal composite web services, it is clearly shown that we need a new model for solving the problem of allocating optimal packages (service providers) to tourists as per requests. We therefore propose to deploy an idea of solving a multiagent system problem, known as *optimal*

coalition structure problem, to solve our problem in tourism domain. In the following, we will briefly explore the related work in optimal coalition structure.

Game theory is a mathematical study of decision making by multiple decision making units (agents). Game theory differs from decision theory in the sense that agents' decisions are inter-related. The ultimate objective of game theory is to find at least a stable state, known as *equilibrium*, where every agent is satisfied and does not want to change its decision. Game theory can be divided into *non-cooperative* games and *cooperative* games. In the non-cooperative games, agents cannot (by rule of the games) collaborate or communicate with each other. A well known equilibrium in this area is *Nash Equilibrium* [9] in which none of the agents can benefit from changing their strategies, given that other agents are adhere to their present strategies.

In contrast to non-cooperative game theory, cooperative game theory allows for agents to have communications that lead them to cooperation [3], from which they can benefit more individually. Agents communicate in order to negotiate with regard to whom they can cooperate and how the joint benefits will be distributed among them. When several agents make a binding agreement to cooperate, we say a *coalition* has been formed. Hence, the cooperative game theories are also known as the theories of *coalition formation* [3]. Mathematically, given set N of n agents, a coalition is a non-empty subset S of N, $S \subseteq N, S \neq \emptyset$. The set N itself is called the *grand coalition* while a coalition of one agent is called *singleton coalition*. Let $\mathbf{S}$ be the set of all coalitions, whose size of S is $2^n - 1$. Given a set of 3 agents, $N = \{n_1, n_2, n_3\}$, all the 7 coalitions are $\{n_1\}$, $\{n_2\}$, $\{n_3\}$, $\{n_1, n_2\}$, $\{n_1, n_3\}$, $\{n_2, n_3\}$ and $\{n_1, n_2, n_3\}$. As in set theory, the *cardinality*, $|S|$, of S is the size of (the number of agents in) S. Once agents have formed coalitions, they can be viewed as if they have divided themselves into a mutually exclusive and exhaustive partitions. We define a *coalition structure*, CS, as a partition of N. A CS can be described by $CS = \{S_1, S_2, \ldots, S_m\}$. The set of all CS is denoted by $\mathcal{CS}$. All CS, for example, in $\mathcal{S}$ are $\{\{n_1\}, \{n_2\}, \{n_3\}\}$, $\{\{n_1, n_2\}, \{n_3\}\}$, $\{\{n_1, n_3\}, \{n_2\}\}$, $\{\{n_2, n_3\}, \{n_1\}\}$, $\{\{n_1\}, \{n_2\}, \{n_3\}\}$.

Mathematically, a CS has to satisfy three conditions [3]:

1) $S_j \neq \emptyset, j = 1, 2, \ldots, m$,
2) $S_i \cap S_j = \emptyset$ for all $i \neq j$, and
3) $\bigcup S_j = N$.

The joint benefit of a coalition is call the *coalition value*, which is a numeric value that usually represents the utility which accrues from their cooperation. It is quite common in cooperative game theory that the coalition value is money value, e.g., dollars. Cooperative game theory assumes that there is a *characteristic function* [3], V that assigns a real number to each S, $V : 2^n \rightarrow \mathbb{R}$. We shall denote the coalition value of S with V_S. Hence, a cooperative n-person game in characteristic function form is defined by the pair $(N; V)$ [3]. The portion of V_S given to agent n_i is the *payoff*, U_i, of the agent for which the agent plays the game. The collection of payoffs to each agent is the payoff vector, $U = (U_1, U_2, \ldots, U_n)$, which specifies the payoff for each respective agent. Putting together a coalition structure and a payoff vector is the *payoff configuration* [3], $(U; CS)$, which describes a possible outcome of the game. For example,the payoff configuration $(5, 10, 5; \{n_1, n_3\}, \{n_2\})$ means agents 1 and 3 have formed a coalition and receive payoff for 5 dollars each while agent 2 remains a singleton coalition and receives 10 dollars payoff on its own.

2.3 Optimal coalition structure

Searching for optimal coalition structures has gained much attention from researchers recently. It is so important for two reasons: *i*) it indicates the optimal solution of a given system, and *ii*) it helps determining the core of the system (collective rationality). In the following, we shall discuss the overview of the problem as it is presented in the literature [11].

Given a CS, we define its value,

$$V(CS) = \sum_{S \in CS} V_S,$$

which indicates the system's utility yielded by that partitioning. An optimal coalition structure is a CS^* such that

$$CS^* = argmax_{CS \in L} V(CS)$$

The number of all coalition structures can be determined by B_n [7], *Bell Number* which is the size of the whole search space. Since the value of B_n can be very large for a small value of n, existing algorithms tend to divide the search space into small portions. There are two divisional methods. Firstly, we can categorize CSs by the number of coalitions within them [11]. We denote the set of CSs, whose number of coalitions of each CS is $1 \leq i \leq n$, by L_i. Each L_i is known as a layer. The number of CSs in L_i is known as the *Stirling Number of the Second Kind* [7]. Hence, the set of all CSs is $L = \bigcup_{i=1}^{n} L_i$. Alternatively, we can categorize CSs by the integer partition of n that describes the number of coalitions and their cardinalities. Each instance j of such a partition is known as a "pattern" [24, 22] or a "configuration" [10], G_j, which is usually written in the form $b_1 + \ldots + b_k$, where $\sum_{l=1}^{k} b_l = n$. Given a set of 4 agents, all the patterns are 4, 3+1, 2+2, 2+1+1, 1+1+1+1.

2.4 Coalition Formation in Multi-agent systems

Coalition formation in multi-agent systems is a dynamic process towards cooperation among agents, consisting of inter-related activities, i.e., deliberation and negotiation, that eventually help agents reach agreement to form coalitions. During the deliberation, agents deal with

necessary calculations, including computing coalition values, choosing potential coalition members, and computing reasonable payoffs. In negotiation, agents follow a protocol to exchange information, which is computed during their individual deliberation, among each other to convince potential coalition members to make a decision. Note that coalition formation requires simultaneous multilateral rather than bilateral negotiation [12].

Dynamic coalition formation dated back to the study of *Transfer Schemes* by Stearns [25]. This work is a mathematical analysis of the dynamic process of coalition that leads to a final payoff configuration (a payoff vector for all agents and a coalition structure of all agents) for a given game with respect to a solution concept, such as the Kernel. Agents repeatedly compute excesses of each pair of agents, balance the differences, and eventually converges to an equilibrium. Stearns [25] shows that agents can eventually converge to the Kernel with the cost of exponential time.

The real implementation of dynamic coalition formation among (computer software) agents took place between the late 80's and the early 90's. One such work was Zlotkin and Rosenschien's [27] which studied coalition formation of n agents, which were allocated parcels to deliver in a grid environment and they had to finish their tasks within limited steps. Agents are allowed to cooperate. Hence, their coalition values are the costs they can save by exchanging tasks. The payoffs to agents are calculated based on the Shapley value. This work is a good example of applying a cooperative game theory into a multi-agent system. Agents can decide to form coalitions by applying one of the proposed schemes which will suggest the most appropriate coalition for each agent itself.

Ketchpel [4] proposes a two-agent auction mechanism for coalition formation that prescribes an algorithm for agents to negotiate. Agents join auctions to win contracts from which they earn guaranteed payments. Firstly, agents broadcast their individual offers to other agents. These offers are ranked in preference orders by interested agents. Each interested agent then chooses the most attractive agent, as a potential coalition member, in order to form a coalition of 2 agents. These two agents then enter the "two agent auction" phase where an agent will act as the manager. The manager will propose to its potential coalition member a payoff. The member agent receives a fixed payment for its role in the coalition. The principle of ranking potential coalition members for negotiation seems to be a mainstream of coalition formation algorithms derived in later research [19, 18, 16, 17, 20, 15, 13, 5, 6]. Note that only a small number of coalitions are formed due to the ranking.

Similar to Ketchpel [4], Shehory and Kraus [14] explore coalition formation among cooperative agents. These agents use resources, which may include materials, energy, information, communications, etc. to fulfill their tasks. The tasks are to deliver parcels, but the coalitions can be made of more than two agents at a time. An agent will be appointed to compute the payoffs for coalition members. Alternatively, agents can negotiate about their payoffs. In this environment, strong coalitions are those whose potential coalition values are high, thus is preferred by agents. Weak coalitions are those whose potential coalition values are low. Strong coalitions are the ones which other coalitions join to form larger ones. Each coalition will have a representative, who values the potential coalition higher and takes care of negotiation with weaker coalitions. The negotiation is to distribute relevant information, i.e., payoff and resource vectors. Shehory et al. [14] also analyze the complexity of negotiation which is $2(n-1)^2$ operation for their algorithm. The payoffs to agents in each coalition are based on the "common extra" payoff, i.e., the increased value of the joint pair of coalitions. This common extra payoff will be distributed equally among agents in the new coalition. Since, agents operate in superadditive environment, they eventually form the grand coalition.

3 An Agent-based Framework

Typical web-services composition architectures deploy request brokers to locate appropriate service providers and create the complete services. Request brokers play important roles in this kind of architecture that they impose their roles as the central command unit of the systems. However, this is also a threat to the availability, performance and security of the system. Furthermore, service providers may not be able to maximize the benefit out the resources because they cannot do much on negotiation in such architectures.

In contrast to typical composite web-services architectures, we propose an architecture where the importance of centralized request brokers is minimized. We deploy intelligent agent technologies to help increase the performance of the system. Furthermore, service providers can leverage maximal benefits out of their resources. While most works in service composition are concerned with computational and networking issues, we consider a tourism domain where the quality of service is the satisfaction of a tourist. We treat the satisfaction by means of utility. The satisfaction is the preference of the tourist over the places. We allow the tourist to expressively specify the preference, and it is captured for computing the utility.

In the sections below, we propose an architecture and related underpinning components for the domain of our study.

3.1 Architecture

We model the scenario as a service-oriented computing system and deploy the multiagent systems concept to enhance the performance. The architecture is depicted in Figure 1. We firstly define optimal composite services, a multiagent system architecture, and a set of protocol in the system.

3.1.1 Representing Stakeholders with Intelligent Agents

First, we define the components of the aforementioned tourism scenario in terms of service oriented computing (SOC). The tourist will be regarded as the service requester who can submit the request for (composite) service on line. The travel agencies, who can provide travel packages, will be regarded as the service providers. In the following, we will define the stake holders in SOC in terms of an agent-based system.

A service requester will be represented by an agent running on a computer, which is connected to the Internet. We shall refer to such an agent as a requester agent (RA). For the requester, its requester agent needs to know what is the goal of the requester, e.g. to achieve its goal with minimal cost. The goal of a requester, of course, varies depending on the type and activity of the requester. For a traveling agent, the goal may be to formulate a travel package satisfying the user's specification. Such a specification, for example, may include request for minimal number of star of each hotel, traveling time from the airport, traveling time to shopping centers, specific foods, and within a given budget. For a grid agent, the goal may be to aggregate a number of hosts who have appropriate resources to execute the given tasks. The tasks may require specific programs, minimal primary storage capacity, minimal memory capacity, minimal CPU speed and node, etc.

Instead of using request broker agent, we propose a request collector agent (RCA). This type of agents acts as a blackboard to where requester agents can post their requests. The role of RCAs is not to be the manager as a request broker which brings weaknesses to the system. It does not search for service providers and choose them to compose the service providers. Instead, it receives request for services from requesters and provides details of requests to the service providers up-on requests. A requester agent can post its request to RCAs. An RCA will manage all requests, i.e. receiving new requests and make them accessible, maintaining their status whether the requests have been served, keeping tracks of the served requests. There can be multiple RCAs in a system (which we assume can be a large one). Each RCA will make its presence known to other agents in the system by broadcasting a message.

The last stakeholder is the service provider agent (SPA), which will take care of utilizing the service provider's resources. This type of agents knows at least one RCA and will keep an eye on the availability of requests for services of which they are capable. SPAs will maintain its list of relevant agents whom it might cooperate to compose a composite service. A service provider can deploy an SPA and join the system by sending out a request to participate (RTP) to existing RCAs. An RTP is a message that specifies what is the agent's capability, which is merely a set of places in its travel packages.

3.1.2 Communication Protocol

Since there are three stake holders involved in the architecture, there must be a way for each of them to know the existence of other agents before they can really compose services. As a standard means in multiagent systems, we introduce a protocol for the agents, who communicate to each other by sending messages. Here we are interested in high level communication, which can be extended by any real implementation via XML (and that is not the focus of this research).

We take into account what travel agencies do in real world tourism industry, where service providers may form loosely-coupled associations. Travel agencies in an association usually cooperate to their associative members to some extent. Then a tourist is free to negotiate with any of them and choose the best option. We follow this real world setting by allowing SPAs to register themselves as a member of an association. This association will have just one registrator, which is represented by an RCA. The role of an RCA is to run the composite algorithm to find the optimal service composition of its members and propose it to the requesting SPA, who will choose the optimal global composition.

- a message to broadcast the availability of a RCA. Any agent that wants to act as an RCA will broadcast the message which notifies other agents, including existing RCA, in the system that it wants to play the role of RCA. The message is of the form ⟨RCA_ID, RCA_AVAIL⟩, where RCA_ID is a high level identification of the agent and RCA_AVAIL is merely a plain text specifying this agent wants to be an RCA. Receiving RCAs and SPAs will update their databases accordingly.

- a message to broadcast the availability of SPA. Any agent that wants to perform the SPA role for an RCA who belongs to the same association has to make its availability and capability recognized by other agents. It will send the respective RCA the message ⟨SPA_ID, SPA_AVAIL, SPA_CAP ⟩, where SPA_ID is the identification of the agent, SPA_AVAIL is the a plain text specifying the agent wants to act as an SPA agent, and SPA_CAP is the plain text explaining the agent's capability.

- a message to acknowledge the availability of SPAs. Receiving RCAs will update their databases accordingly and send the acknowledgment message ⟨ RCA_ID, SPA_ACK ⟩ to the SPA in order to acknowledge the availability. Up-on receiving the acknowledgment message, the new SPA updates its database for existing RCAs accordingly.

- a message between RA and RCA to broadcast the request for service. A demanding requester agent will send a message ⟨RA_ID, R_SPEC⟩, where RA_ID is the identification of the RA and

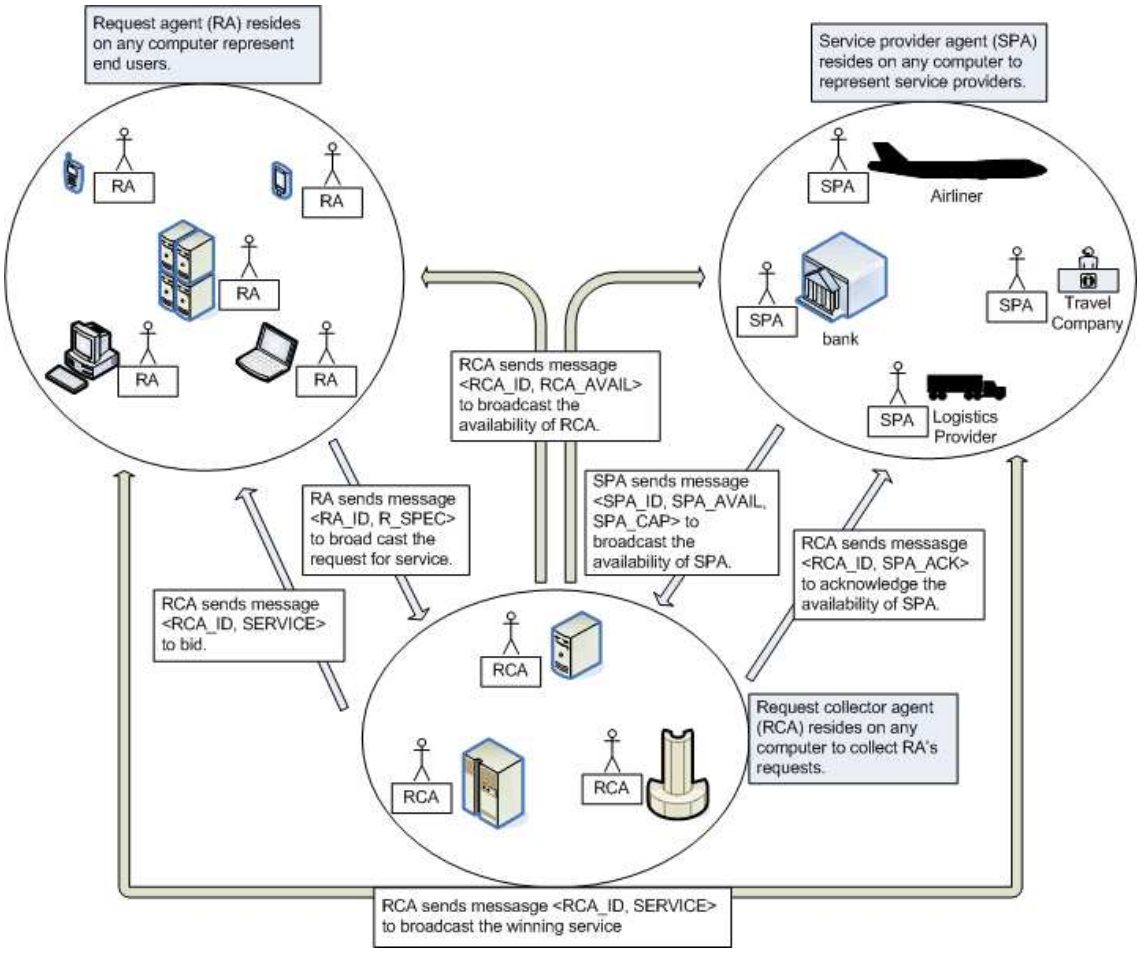

Figure 1 Architecture We deploy intelligent agents to represent stakeholders. Request agents (RA) represent end users. Request collector agents (RCA) collect requests, allocate tasks, broadcast the allocation. Service provider agents represent service providers.

R_SPEC is the request specification. Receiving RCAs will update their databases for available requests accordingly. Each RCA will execute its composition algorithm which will derive its optimal composite service.

- a message between RCA and RA to launch a bid for service. Having obtained its optimal composite service, each RCA will send the message ⟨ RCA_ID, SERVICE⟩ directly to the respective RA who will determine the winning bid.

- a message to broadcast the wining bid. SPA will send message ⟨ RCA_ID, SERVICE⟩ to all bidding RCAs in order to announce the winner. The winner is then bound to contract and provide service to SPA.

The final decision for the winning composite service is made by the tourist based on her satisfaction. This satisfaction is, of course, subjective to individual tourists and varies. For example, given the same quality of service, the composite service with lowest cost offers higher value to the tourist and satisfies the tourist more than other services.

3.2 Optimal Composite Services

Let $\mathbb{P} = \{p_i |$ where $1 \leq i \leq n\}$ be a set of n interested places, which are available to tourists. A travel package is a plan offered to tourists a number of places which they will visit. We define a package $\mathcal{P} \subseteq \mathbb{P}$ is a subset of all the places. The set of all packages is denoted by $2^{\mathbb{P}}$. Each of these package is normally offered by a company at a certain cost and will take some time to operate, i.e. there is a start time and an end time. Therefore, we consider to a package as a *service*. Let $\mathbb{C} = \{c_j |$ where $1 \leq j \leq m\}$ be the set of m companies. Let $\theta \in \mathbb{Z}^+$ be the cost of operating a service. Let (T_s, T_e) where T_s $in\mathbb{Z}^+$, $T_e \in \mathbb{Z}^+$ and $T_s < T_e$ be the *service time*, which is the pair of start and end time of the service. (In practice, we refer to time of a day in terms of hour, minute, and second. Beyond a day, we refer to time as a date which has day of month, month, and year. However, both of them can be represented by specified number of milliseconds since the standard base time known as the *epoch*, namely January 1, 1970, 00:00:00 GMT.) We denote by $\mathbb{T}$ the set of all service times. Here, a service is defined as a tuple $\mathcal{S} =< \mathcal{P}, c, \theta, T >$. The set of all services is denoted by $\mathbb{S}$. The package, service provider, cost, and service time of $\mathcal{S}$ are denoted by $\mathcal{S}.\mathcal{P}, \mathcal{S}.c, \mathcal{S}.\theta$, and $\mathcal{S}.T$, respectively. Note that a service always offer at least one place of interest to the tourist.

Normally, a tourist would like to visit as many places as possible. For an enthusiastic tourist, there might be a number of certain places that she cannot afford to miss. All these places are considered to be $\mathbb{P}$. However, the tourist is under time and budget constraints–she has some time for the whole trip and cannot afford over spending for the trip. In this work, we capture these main issues in the R_SPEC defined in the above architecture. We define R_SPEC $= \langle \mathbb{P}, \mathcal{R} \rangle$ where $\mathcal{R} = \langle \Omega, \Psi \rangle$ the request to visit places of interest with budget $\Omega \in \mathbb{Z}^+$ and time $\Psi \in \mathbb{T}$ constraints. One can always extend this definition for further details and constraints of the request specification.

We define a *composite service*

$$\mathcal{CS} = \begin{array}{c} \{\mathcal{S}_i | \, \mathcal{S}_i \in \mathbb{S} \text{ and } \bigcup \mathcal{S}_i.\mathcal{P} = \mathbb{P} \text{ and} \\ \mathcal{S}_i.\mathcal{T}.s \nleq \mathcal{S}_j.s \nleq \mathcal{S}_i.\mathcal{T}.e \text{ and} \\ \mathcal{S}_j.\mathcal{T}.s \nleq \mathcal{S}_i.s \nleq \mathcal{S}_j.\mathcal{T}.e\} \end{array}$$

is a set of services which cover all places.

Note that we are only concerned with the exhaustiveness condition of the previously defined CS, i.e. all the places must be included in the composite service, but do not have the complete mutual exclusiveness, i.e. these packages may offer the same places to the tourist $\mathcal{S}.\mathcal{P}_i \cup \mathcal{S}.\mathcal{P}_j \neq \emptyset$. On the other hand, we are concerned with the exclusiveness condition of the service times that there must be no conflicts among them, i.e. a service cannot start during the operation of another service in the same composite service. We refer to both the exhaustiveness and the exclusiveness as the *completeness* condition of the composite service. We assume that there is at least a service for each place and there is no conflict among their service times. We also assume that the budget of the tourist is at least as much as the sum of the cost of these packages. Our assumption is realistic because there are several packages (services) that will take a tourist to a place of interest. These packages usually operate as often as possible. Some may start in the morning, some may start in the afternoon and last as long as needed. However, in reality, we cannot guarantee that there exists $\mathcal{S}_i.\mathcal{P} = \mathcal{P}_k \in 2^{\mathbb{P}}$, i.e. some of the packages may not be offered as part of a service by any company due to several reasons. Given a large $\mathbb{P}$, for example, it may be too costly to offer the package of $\mathbb{P}$ itself or any package whose size is close to that of $\mathbb{P}$. Some packages containing places that are disperse in proximity can also be costly.

Given the five conditions of a composite service for attracting the tourist, the quality of service to satisfy the tourist depends on the following four conditions:

- The number of visited places must be maximized,
- The number of redundant places must be minimized,
- The total price is within the budget,
- The time constraint must be obeyed.

The last condition, the payoffs for service providers are worthwhile and fair, is more concerned with the decision of the service providers and is the the additional condition to attract cooperation among service providers.

Given a service $\mathcal{S}$, the *service frequency matrix* is

$$F(\mathcal{S}) = \begin{bmatrix} \alpha_1 \, \alpha_2 \, \alpha_3 \ldots \alpha_n \end{bmatrix}$$

where

$$\alpha_{i,1\leq i\leq m} = \begin{cases} 0 \text{ if } p_i \notin \mathcal{S}.\mathcal{P} \\ 1 \text{ if } p_i \in \mathcal{S}.\mathcal{P} \end{cases}$$

Given a composite service $\mathcal{CS}$, the *composite service frequency matrix* is

$$F(\mathcal{CS}) = \begin{bmatrix} \alpha_1,1 & \alpha_1,2 & \alpha_1,3 & \dots & \alpha_1,n \\ \alpha_2,1 & \alpha_2,2 & \alpha_2,3 & \dots & \alpha_2,n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \alpha_m,1 & \alpha_m,2 & \alpha_m,3 & \dots & \alpha_m,n \end{bmatrix}$$

The *aggregated frequency vector* of $\mathcal{CS}$ is

$$AF(\mathcal{CS}) = \begin{bmatrix} \sigma_1 \, \sigma_2 \, \sigma_3 \dots \sigma_n \end{bmatrix}$$

where

$$\sigma_j = \sum_{i=1}^{m} \alpha_{i,j}.$$

Here, we define the utility function

$$\Phi : \mathbb{P} \times \mathbb{Z}^+ \leftarrow \mathbb{Z}^+$$

which specifies the degree of satisfaction for a tourist visiting a place for a number of times.

Associated to each $\mathcal{CS}$ is the cost of composite service

$$\Theta(\mathcal{CS}) = \sum_{\mathcal{S}\in\mathcal{CS}} S.\theta$$

Let $\phi_i = \Phi(p_i, \sigma_i)$ be the utility of visiting p_i for σ_i times. We define the *satisfaction vector*

$$S(\mathcal{CS}) = \begin{bmatrix} \phi_1 \, \phi_2 \, \phi_3 \dots \phi_n, \end{bmatrix}$$

Therefore, we define the QoS function below:

$$QoS(\mathcal{CS}) = \sum_{j=1}^{n} \phi_j. \tag{1}$$

We are interested in finding, for a given request, an optimal composite service $\mathcal{CS}^*$ such that

$$\mathcal{CS}^* = arg_{max}\ QoS(SS^*). \tag{2}$$

3.2.1 Defining Utility

We refer to satisfaction in terms of utility. Since this satisfaction is subjective to the tourist's preference over places, the tourist can expressively specify the preferences (and this is also practical and realistic). In general, a tourist will be happy to visit a place once. Visiting the same place again will be unnecessary and boring. The tourist may define the utility as follows:

- The utility for the visiting a place once is 1, and
- The utility decreases by half every time a place is visited more than once.

In this case, the utility function can be defined as follow:

$$\Phi(p, \sigma) = \begin{cases} 1 & \text{if } \sigma = 1 \\ \frac{1}{2^n} & \text{if } \sigma > 1 \end{cases}$$

On the other hand, an enthusiastic tourist may prefer to visit a place more than once, while visiting other places does not excite she that much. The reason may be the place is large and it will take days for a thorough visit. De Louvre, for example, in Paris is a huge museum that no one can do a detailed visit in one single day. However, spending time more than a certain number of days immediately turns the trip into an unpleasant one. The tourist may define the utility as follows:

- For visiting the favorite place p^*, the utility increases by the number of days spending there up to 3 days.
- Visiting p^* more than 3 days negates the utility by the number of excessive days.
- The utility for the visiting $p' \neq p^*$once is 1, and
- The utility decreases by half every time p' is visited more than once.

In this case, the utility function can be defined as follow:

$$\Phi(p, \sigma) = \begin{cases} \sigma & \text{if } p = p* \text{ and } \sigma \leq 3 \\ 3 - \sigma & \text{if } p = p* \text{ and } \sigma > 3 \\ 1 & \text{if } p = p' \text{ and } \neq p^* \text{ and } \sigma = 1 \\ \frac{1}{2^n} & \text{if } p = p' \text{ and } \sigma > 1 \end{cases}$$

3.2.2 Computing Payoffs for Service Providers

Here we will apply Shapley value to distribute payoffs to service providers of a composite service. Firstly, we need to find the contribution each service provider has made in the composite service. Since we may not have all the packages offered by service providers, we cannot strictly follow the steps for computing shapley values. However, we can measure the contribution of a service provider by its importance, i.e. how much the quality of service will be decreased if the service provider is excluded from the composite service. Let $\mathcal{CS}^{-\mathcal{S}}$ be a set of all service providers in $\mathcal{CS}$ excluding $\mathcal{S}$. The contribution of of $\mathcal{S}$ is

$$CON(\mathcal{S}) = QoS(\mathcal{CS}) - QoS(\mathcal{CS}^{-\mathcal{S}})$$

We define the *accumulated contribution* of all service providers as

$$ACC(\mathcal{CS}) = \sum_{\mathcal{S}\in\mathcal{CS}} CON(\mathcal{S})$$

We define the payoff for $\mathcal{S}$ as

$$PO(\mathcal{S}) = \frac{CON(\mathcal{S})}{ACC(\mathcal{CS})} \times (\Omega - \Theta(\mathcal{CS})),$$

where $\Omega > \Theta(\mathcal{CS})$. Note that we can compute payoffs for service providers only for those who are members of profitable composite services.

4 Generating Composite Services

In this section, we will discuss about the algorithms proposed to use with the aforementioned framework. We consider composing a service in our context is similar to generating a coalition structure. Even though there are several algorithms for generating coalition structures, many of them are not applicable here due to many reasons. Firstly, since some packages may not be offered as part of any services, we cannot use these algorithms because they require all coalitions (packages). One may solve this problem by assigning value 0 to these packages, but they will uselessly occupy memory and slow down the performance of the algorithm. Secondly, there may be several services offering the same package, i.e. there may be multiple instances of the same package as part of the input. These algorithm cannot handle this setting because they require a unique coalition in the input. Here, we will apply the concept of the best-first search optimal coalition structure algorithm [23] to solve the optimal service composition problem because of its flexibility. This new algorithm will be run on each RCA.

There are two main steps in solving the problem:

- data preparation stage. In this stage, each RCA will sort services by lexicographic order in each cardinality,
- search for optimal composite service stage. In this stage, each RCA will apply the new algorithm to search for the optimal service composition.

After that, the respective RA will identify the most appropriate service composition for RA. RA will decide which one of the composite services being proposed will be the most appropriate one.

4.1 Data Preparation

We use similar data structures as in [23] to generate composite services. In this research, we consider a service is equivalent to a coalition. Services are collected, sorted by their cost in ascending order, and stored in $\mathfrak{C}$, a 2-dimensional dynamic array. The first dimension refers to the number of cardinalities, the size of (number of places in) the services, of all services. For each cardinality, the size of the respective array is the number of services of that cardinalities. We use $\mathfrak{B}$, a 2-dimension array to keep track the progress of the generation of composite services. We use $\mathfrak{CS}$, a 1-dimension array of size n to store the composite service being generated. We use $\mathfrak{R}$, a 1-dimension array of size n to store the remaining places yet to be included in the composite services. We denote by $|\mathfrak{R}|$ the number of non-zero elements of $\mathfrak{R}$, i.e number of the remaining places yet to be included in the composite services.

4.1.1 Indicating Best Candidate

The principle of the algorithm for generating composite services is to repeatedly choose the best services from the available candidates and place it in $\mathfrak{CS}$ until the completeness condition is met. Once this holds, $\mathfrak{CS}$ can be exported to $\mathcal{CS}$, i.e. a new service has been completely composed. The whole process is then repeated until no more $\mathcal{CS}$ can be generated.

The most important step in the algorithm is to identify the best candidate out of available services in order to compose $\mathfrak{CS}$. As in the algorithm on which this work is based, we need an indicative information which would direct the search towards optimality quickly. Since the quality of service in this work is subjective to the tourist's preferences over the number of times visiting places, we can apply the concept of $QoS(\mathcal{CS})$.

Let

$$QoS(\mathcal{S}) = \sum_{j=1}^{n} \alpha_j$$

be the quality of service of $\mathcal{S}$. Let $QoS(\mathfrak{CS})$ be the present quality of service of $\mathfrak{CS}$ being constructed. Let $QoS(\mathfrak{CS}^{+\mathcal{S}})$ be the projective quality of service of $\mathfrak{CS}$ if $\mathcal{S}$ is added into the present $\mathfrak{CS}$. We define the *projective contribution* of S towards $\mathfrak{CS}$ as

$$PRO(\mathcal{S}) = QoS(\mathfrak{CS}^{+\mathcal{S}}) - QoS(\mathfrak{CS}).$$

Hence, the best candidate is

$$\mathcal{S}^* = \mathcal{S}_{argmax} PRO(\mathcal{S}).$$

Note that we cannot determine $\mathcal{S}^*$ basing on $QoS(\mathcal{S})$ alone because it may result in severely decreasing $QoS(\mathfrak{CS}^{+\mathcal{S}})$.

4.1.2 Data Example

As shown in Table 1, all sorted $\mathcal{S}$s in each cardinality are placed in their respective columns from top to bottom. Apparently, we do not have any package of size 3. Furthermore, the number of $\mathcal{S}$s in each cardinality is not related to binomial coefficient (${}^nC_{|\mathcal{S}.\mathcal{P}|}$). It is arbitrarily depending on what are being offered in the system. Since each of these services can be proposed individually by service providers in the system, acquiring, sorting, and storing these data will be done on the fly.

Table 1 shows an example in our setting. There are 3 places, $\mathcal{P} = \{P_1, P_2, P_3\}$. Mathematically, these places comprises seven packages (non-empty subsets). However, in the example, there are 7 services, $\mathcal{S} = \{S_1, S_2, \ldots, S_7\}$. Assuming QoSs are given, these services are sorted in descending order based on their QoSs and placed in their respective $\mathfrak{C}$. Note that not all packages are offered in these services. Furthermore, there are 2 packages being offered twice in these services.

4.2 The Optimal Service Composition Algorithm

4.2.1 The Main Function

As shown in Algorithm 1 on page 11, the first thing to do is to set the present level l of $\mathfrak{CS}$ to 1. Then the first $\mathcal{S}^*$ is

$\mathbb{P}$	$\mathcal{S}.\mathcal{P}$	$\mathcal{S}.\theta$	$\mathfrak{C}[1](\lvert\mathcal{S}.\mathcal{P}\rvert = 1)$	$\mathfrak{C}[2](\lvert\mathcal{S}.\mathcal{P}\rvert = 2)$	$\mathfrak{C}[3](\lvert\mathcal{S}.\mathcal{P}\rvert = 3)$
p_1	$S_1.\mathcal{P} = \{p_1, p_2\}$	$S_1. = 10$	S_4 $(\bar{r} = 6)$	S_1 $(\bar{r} = 5)$	
p_2	$S_2.\mathcal{P} = \{p_2, p_3\}$	$\mathcal{S}_2.\theta = 12$	S_7 $(\bar{r} = 8)$	S_2 $(\bar{r} = 6)$	
p_3	$S_3.\mathcal{P} = \{p_1\}$	$\mathcal{S}_3.\theta = 9$	S_3 $(\bar{r} = 9)$	S_6 $(\bar{r} = 7)$	
	$S_4.\mathcal{P} = \{p_2\}$	$\mathcal{S}_4.\theta = 6$		S_5 $(\bar{r} = 9)$	
	$S_5.\mathcal{P} = \{p_1, p_3\}$	$\mathcal{S}_5.\theta = 18$			
	$S_6.\mathcal{P} = \{p_1, p_2\}$	$\mathcal{S}_6.\theta = 14$			
	$S_7.\mathcal{P} = \{p_3\}$	$\mathcal{S}_7.\theta = 8$			

Table 1 Example of data Services in each cardinality are sorted by their costs, given by the cost function

achieved by calling function *BestService*. The algorithm then goes to the main loop where the best service will be placed at the present level, i.e 1, of $\mathfrak{CS}$. The set of unvisited placed $\mathfrak{R}$ will be updated by removing the visited places $\mathcal{S}^*.\mathcal{P}$ from $\mathfrak{R}$. The variable $\mathcal{S}^*$ is then set to null. At this point, the algorithm determines if all the places are visited. If it is the case, the algorithm prints out the solution, i.e. $\mathfrak{CS}$. After that, the algorithm tries to extend the $\mathfrak{CS}$ by *i*) locating the next candidate for all cardinalities, whose value is not greater than $\mathfrak{R}$ by calling function *NextCandidate*, and *ii*) determining $\mathcal{S}^*$ for the next level of $\mathfrak{CS}$ by calling *BestService*. If $\mathfrak{S} \neq null$, i.e the best service can be identified, the level of l is set to $l + 1$, i.e it will be placed in $\mathfrak{CS}$ in the next round.

Otherwise, the algorithm tries to alter the last service of $\mathfrak{CS}$ with an alternative service. The visited places in the last service of $\mathfrak{CS}$ will be returned to $\mathfrak{R}$. The algorithm then locates the next candidate service of the respective cardinality by calling function *NextCandidate*, and identifies $\mathcal{S}^*$ by calling function *BestService*. The algorithm determines if $\mathcal{S}^*$ can be retrieved. If that is the case, the algorithm reaches the end of the loop and the new $\mathcal{S}^*$ will be placed in the present level of $\mathfrak{CS}$ in the next round.

Otherwise, the algorithm tries to shrink $\mathfrak{CS}$. It reduces the level l of $\mathfrak{CS}$ by 1 and returns the visited places in the present level of $\mathfrak{CS}$ to $\mathfrak{R}$. The position for starting search for the next candidate is assigned to u and the next candidate can be identified by calling function *NextCandidate*. The algorithm then calls function *BestService* for determining $\mathcal{S}^*$.

The algorithm will reach the end of the main loop and will go to the beginning of the main loop where it determines if the next $\mathcal{S}^*$ is empty. If that is the case, the algorithm exits the main loop and terminates. Otherwise, the algorithm enters the loop and places $\mathcal{S}^*$ into the present level of $\mathfrak{CS}$ and continues its journey through the end of the loop again.

4.2.2 The Supportive Functions

BestService: As shown in Algorithm 2 on page 12, the search for the best candidate is very simple. Firstly, the $\mathcal{S}^*$)is set to $\emptyset$ as well as its $PRO(\mathcal{S}^*)$ is set to 0. The algorithm then goes through each candidate service $\mathcal{S}$ in ascending order of the cardinality and compares $PRO(\mathcal{S})$ against $PRO(\mathcal{S}^*)$. Only if

Algorithm 1 Construct composite service by adding the best service chosen from available candidates into $\mathfrak{CS}$

```
1: set the present level to l
2: set S* to BestService(l)
3: while is there still S* do
4:     fill CS[l] with S*
5:     update R with S*
6:     reset S*
7:     if R is empty then          ▷ a solution has been created
8:         print "***** "+CS;
9:     end if                      ▷ attempt to extend layer
10:    if the present level l < n then
11:        for each cardinality of size 1 to |R| do
12:            locate the beginning position to search for the next candidate
13:        end for    ▷ Is there any valid service at the next layer?
14:        set S* to BestService(l + 1)
15:    end if
16:    if S* is not null then ▷ Extend to the next level
17:        set the next level l + 1;
18:    else   ▷ cannot extend then attempt for altering
19:        update R with CS[l]
20:        set u as the beginning position to search for next candidate
21:        set B[|CS[l]|][l] to NextCandidate(|CS[l]|, u)
22:        reset CS[l]
23:        set S* to BestService(l)        ▷ cannot alter
24:        if S* is not empty then ▷ attempt to shrink
25:            if the present level l > 1 then
26:                set l to l − 1
27:                update R with CS[l]
28:                set u as the beginning position to search for next candidate
29:                set B[|CS[l]|][l] to NextCandidate(|CS[l]|, u)
30:                reset CS[l]
31:                set S* to BestService(l);
32:            end if
33:        end if
34:    end if
35: end while
```

$PRO(\mathcal{S}) > PRO(\mathcal{S}^*)$ then $\mathcal{S}^*$ is set to $\mathcal{S}$. This way, even multiple candidate services have exactly the same $PRO(\mathcal{S})$, only the first $\mathcal{S}$ remains $\mathcal{S}^*$.

Algorithm 2 BestCandidate Function

```
1: function BestService(l)
2:     S* ← ∅
3:     PRO(S*) ← 0
4:     for |S| = 1 to |ℜ| do ▷ for each valid cardinality
5:         if 𝔅[l][c] > 0 then ▷ if there is a candidate coalition
6:             compute PRO(S)
7:             if PRO(S) > PRO(S*) then
8:                 S* ← S ▷ set the candidate as the new best service
9:             end if
10:        end if
11:    end for
12:    return S*
13: end function
```

NextCandidate: As shown in Algorithm 3 on page 12, at any layer l, the algorithm needs to prepare the candidate services in each valid cardinality. A valid cardinality is one whose value is not greater than the number of remaining places, i.e. $|\mathcal{S}.\mathcal{P}| < |\Re|$. For each of these cardinalities, we just need only the next available coalition, i.e., the one whose members are all in $\Re$, as the candidate of its cardinality for the next layer of $\mathcal{CS}$. The search for the candidate will be done towards the last service in each cardinality.

Algorithm 3 NextCandidate Function

```
1: function NextCandidate(c,p)
2:     for j = p to |𝔈| do ▷ starting from position p towards ⁿC_c
3:         if 𝔈[c][j] ⊆ ℜ then ▷ if the service at c, p is in ℜ
4:             return j ▷ return its position
5:         end if
6:     end for
7:     return 0
8: end function
```

5 Experiments

Since we are the only algorithm to solve this problem, which is quite unique to the area of web services, we run our algorithm with the proposed heuristic against the exhaustive search.

5.1 Setting

We conduct experiments on $n \in [12 \ldots 15]$ places. Due to the large search space, we set a terminate time for running the exhaustive search for optimal results.

num-agent	exhaustive	converge	terminate
12	30852	6859	7347
13	289326	6973	7629
14	2974952	7792	7954
15	29674021	7850	9342

Table 3 Raw figure results
The table shows the raw execution times achieved from the exhaustive search and our algorithm in milliseconds.

The terminate time of exhaustive search is projected. The number of services in each cardinality i in each of a variation of n is derived from ${}^nC_i$. The numbers of services in each case of n are shown in Table 2. Although the number of services in each cardinality can be arbitrary in reality, we rather set up the number of services in each cardinality this way for the sake of experiment. The numbers of ${}^nC_i$ across all cardinality form the bell shape of normal distribution with an extremely high mean. Furthermore, the distribution of the costs of the services in each cardinality is of normal distribution as well. We conducted our experiment for the extreme cases, i.e. the customer prefer no redundancy on the places to be visited. Therefore, the algorithm will use the utility function that gives $-\infty$ to any package that contains redundant places with those already placed in SS. In other words, the experiment is similar to that of [23].

In each setting (a variant of n), we run our algorithm with the generated data against the exhaustive search, which guarantees optimal results. We run the experiments a number of times for each n and compare the results achieved from our algorithm against that of the optimal cases. The experiments are conducted on an AMD Turion 64 X2 2GHz machine with 896MB of RAM.

5.2 Results

The raw figures obtained from our experiments are presented in Table 3. The equivalent results are also depicted in Figure 2, where the x axis is the number of agents, ranging from 12 to 15 agents, and the y axis is the log(10) of execution time in milliseconds. Note that we can carry out the experiments with exhaustive search for merely 14 places, which take a lot of time to finish. We project the figure for the execution time of 15 places based on the previous cases. Although we can carry out the search with our algorithm for up to 26 places, we do not do so because the difference of the results, i.e convergence and termination times of our algorithm and the exhaustive case will be too much. Note that whereas the execution time of the exhaustive search bursts exponentially, our algorithm yields pretty consistent execution time throughout all the cases, i.e. lower than 10 seconds (10^4 ms).

	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9	i=10	i=11	i=12	i=13	i=14	i=15
n=12	12	66	220	495	792	924	792	495	220	66	12	1			
n=13	13	78	286	715	1287	1716	1716	1287	715	286	78	13	1		
n=14	14	91	364	1001	2002	3003	3432	3003	2002	1001	364	91	14	1	
n=15	15	105	455	1365	3003	5005	6435	6435	5005	3003	1365	455	105	15	1

Table 2 Number of Services In each case of $8 \leq n \leq 10$, the table lists the number of services for each $1 \leq i \leq n$.

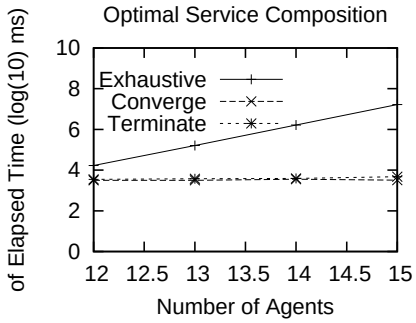

Figure 2 Experiment Results The results of our experiments are shown here. The times required for exhaustive search grow exponentially while the convergence and termination times of our algorithm are quite consistent, i.e. around 4K and 5K ms respectively.

5.3 Discussion

We have shown here how we can optimally allocate available service providers to perform tasks for requesters in global scale. Note that a number of available agents may not be allocated with any task. This means a number of these agents may not benefit from their existing resources. This is due to the fact that the allocation is focused on the benefit of the requesters. It is left open to further research to explore for the balance of being allocated with tasks and accruing benefit to service providers.

The messages being sent over the Internet in our protocols may be at risks if they are not encrypted. However, encrypting messages is considered a lower level of implementation. Some parts, if not all, of the messages can be encrypted given any latest security technology. We leave it open in our research here because the focus of this research is to optimally allocate tasks to agents.

6 Conclusion

A common problem in tourism industry is that a tourist, who is under a budget and time constraints, would prefer to visit places as many as possible in one trip. Since it is also quite common that multiple service providers offer redundant services to tourists, i.e. their traveling packages include the same places and it is boring to the tourist, it is very useful if a service composition can optimally plan for the tourist what packages to take. Since traveling agencies, as service providers, are competitors and are unlikely to cooperate, this issue is not so easy to manage in real world. Furthermore this is a hard problem from computer science perspective.

We address this problem by proposing an agent-based composite web-services framework to allocate to the tourist an optimal service composition, one which maximally satisfies the tourist. We take into account a number of factors including *i*) the number of places be visited must be maximal, *ii*) the number of redundant places must be minimal, *iii*) the total price is within the budget, and *iv*) the time constraint must be obeyed. Since current composite web-services technology deploys a top-down approach, where service providers are to be chosen by a service broker, which is inefficient in many settings, we propose a bottom-up approach to allocate the optimal service composition. We deploy a best first search algorithm that is used to solve the optimal coalition structure to solve this problem. Where other work in optimal service composition considers various aspects which are different from us, we seeks to find optimal packages for the tourist in timely fashion. The utility proposed can be substituted with other utility functions which can be more complex and appropriate to different domains. The results show that our approach can yield optimal results in less than 10 seconds for 15 places bundled in 1,382,958,545 packages, which would take around 21896 seconds for the exhaustive search.

References

[1] Hsing Cheng and Gary Koehler. Optimal pricing policies of web-enabled application services. *Decision Support Systems*, 35(3):259–272, June 2003.

[2] Angus Huang, Ci-Wei Lan, and Stephen Yang. An optimal qos-based web service selection scheme. *Information Science*, 179(19):3309–3322, September 2009.

[3] James Kahan and Amnon Rapoport. *Theories of Coalition Formation*. Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1984.

[4] Steven Ketchpel. Forming coalitions in the face of uncertain rewards. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 94)*, volume 1, pages 414–419, Seattle, WA, USA, 1994. AAAI Press.

[5] Sarit Kraus, Onn Shehory, and Gilad Taase. Coalition formation with uncertain heterogeneous information. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 03)*, pages 1–8, Melbourne, Australia, 2003. ACM Press.

[6] Sarit Kraus, Onn Shehory, and Gilad Taase. The advantages of compromising in coalition formation with incomplete information. In *Proceedings of the 3rd International Joint Conference on Autonomous Agent and Multi Agent Systems (AAMAS 04)*, pages 588–595, Washington DC, USA, 2004. IEEE Computer Society.

[7] Donald Kreher and Douglas Stinson. *Combinatorial Algorithms Generation, Enumeration and Search*. CRC Press, FA, USA, 1999.

[8] Wuqin Lin, Zhen Liu, Cathy H. Xia, and Li Zhang. Optimal capacity allocation for web systems with end-to-end delay guarantees. *Performance Evaluation*, 62(1):400–416, October 2005.

[9] John Nash. *Non-Cooperative Game*. PhD thesis, Department of Mathematics, Princeton University, Princeton, USA, May 1950.

[10] Timothy Norman, Alun Preece, Stuart Chalmers, Nicholas Jennings, Michael Luck, Viet Dang, Thuc Nguyen, Vikas Deora, Jianhua Shao, Alex Gray, and Nick Fiddian. Agent-based formation of virtual organisations. *Knowledge-Based Systems*, 17(2-4):103–111, 2004.

[11] Tuomas Sandholm, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohm. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1-2):209–238, 1999.

[12] Tuomas Sandholm and Nir Vulkan. Bargaining with deadlines. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI 99)*, pages 44–51, Orlando, FA, USA, 1999. AAAI Press.

[13] Onn Shehory, Gal Kaminka, and Eran Shoham. Multi-agent coalition re-formation and league ranking. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 04)*, pages 1348–1349, New York, USA, 2004. IEEE Computer Society.

[14] Onn Shehory and Sarit Kraus. Coalition formation among autonomous agents: Strategies and complexity. In *From Reaction to Cognition, Selected Papers from the 5th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW 93)*, volume 957/1995, pages 55–72. Springer Berlin / Heidelberg, 1995.

[15] Onn Shehory and Sarit Kraus. Task allocation via coalition formation among autonomous agents. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 655–661, Acapulco, Mexico, August 1995. Morgan Kaufman.

[16] Onn Shehory and Sarit Kraus. Formation of overlapping coalitions for precedence-ordered task-execution among autonomous agents. In *Proceedings of the 2nd International Conference on Multiagent Systems (ICMAS 96)*, pages 330–337, Kyoto, Japan, December 1996. AAAI Press.

[17] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.

[18] Onn Shehory and Sarit Kraus. Feasible formation of coalitions among autonomous agents in non-super-additive environments. *Computational Intelligence*, 15(3):218–251, August 1999.

[19] Onn Shehory, Sarit Kraus, and Osher Yadgar. Emergent cooperative goal-satisfaction in large-scale automated-agent systems. *Artificial Intelligence*, 110(1):1–55, May 1999.

[20] Onn Shehory, Katia Sycara, and Somesh Jha. Multi-agent coordination through coalition formation. In *Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages (ATAL 1997)*, number 1365 in Lecture Notes on Computer Science, pages 143–154, Providence, RI, USA, 1998. Springer-Verlag.

[21] Chattrakul Sombattheera. Optimal service composition via agent-based quality of service. In *Proceedings of the 5th Multi-Disciplinary International Workshop on Artificial Intelligence (MIWAI 2011)*, volume 7080 of *Lecture Notes in Artificial Intelligence*, pages 274–285, Heidelberg, Germany, 2011. Springer.

[22] Chattrakul Sombattheera and Aditya Ghose. A pruning-based algorithm for computing optimal coalition structures in linear production domains. In *Advances in Artificial Intelligence, Proceedings of the 19th Conference of the Canadian Society for Computational Studies of Intelligence (AI 2006)*, Lecture Notes in Computer Science, pages 13–24, Quebec, Canada, 2006. Springer–Verlag.

[23] Chattrakul Sombattheera and Aditya Ghose. A best-first anytime algorithm for computing optimal coalition structures. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 1425–1428. ACM Press, 2008.

[24] Chattrakul Sombattheera and Aditya K. Ghose. A distributed branch-and-bound algorithm for computing optimal coalition structures. In *Advances in Artificial Intelligence, Proceedings of the 4th Helenic Conference on AI*, volume 3955 of *Lecture Notes in Computer Science*, pages 334–344, Crete, Greece, 2006. Springer–Verlag.

[25] Richard Stearns. Convergent transfer schemes for n-person games. *Transactions of the American Mathematical Society*, 134(3):449–459, December 1968.

[26] Qian Tang and Hsing Cheng. Optimal location and pricing of web services intermediary. *Decision Support Systems*, 40(1):129–141, July 2005.

[27] Gilad Zlotkin and Jeffrey Rosenschein. Coalition, cryptography, and stability: Mechanisms for coalition formation in task oriented domains. In *Working Notes of the AAAI Spring Syposium on Software Agents*, pages 87–94, Stanford, CA, USA, 1994. AAAI Press.