Composing Business Processes with Partial Observable Problem Space in Web Services Environments

Yuhong Yan[1], Yong Liang[2], Han Liang[2]

[1] *NRC-IIT, Fredericton, NB, Canada,*
Yuhong.yan@nrc.gc.ca
[2] *Faculty of Computer Science, UNB, Canada*
{*Yong.liang, Han.liang*}*@unb.ca*

Abstract

Composing business processes from individual services can be viewed as a planning problem in which a planner determines the execution orders of services in a process. Most existing Web Service composition research considers connecting Web Services into a business process. We argue that most existing Web Services are informative Web Services that are not the actual business services, but give the parameters of their correspondent business services. The planning problem is not only to select the proper business services, but also to determine the parameters of the business services which affect the ordering of the business services. Furthermore, it is not possible to extract all information from informative Web Services through queries. The planner has to work with the problem space that is not fully enumerable. This paper presents a method to optimize planning results with incompletely observed problem space. Genetic Algorithms (GA) help to navigate the incompletely observed problem space. At each loop of GA, Web Service data are queried and a new sub problem space is built. The planner works with the sub problem space and calculates all feasible plans. The plans are evaluated by GA in fitness function and the best plans are kept for the next loop of GA. The fitness function of GA reflects domain-dependent user preferences. The selected final plan is an optimized feasible plan though global optimization is not guaranteed.

1. Introduction

Web Services are emerging as a powerful technique for organizations to implement business processes through selecting and combining Web Services according to the business tasks. Both industrial and academic communities have presented the languages and standards to embrace the utilization of Web Services composition, such as BPEL4WS [1], OWL-S [2] and WSCI [3].

Web Services composition can be seen as a planning problem that a sequence of services is composed into a business process to reach business goals. For some reason, Artificial Intelligence (AI) planning researchers are especially interested in this application. Classic AI planners model the states, the actions and the goals. An action is applicable in a state if its preconditions are satisfied, and it can transfer the system to another state. The planner is to determine the sequential orders of the actions in order to reach the goals. Several techniques, such as search and logic inference, can be used to build the planners [6]. For Web Service composition, Web Services are modeled as actions and the business processes as plans to connect the Web Services. Existing approaches [10][11][12] work when all relevant service descriptions are initially loaded into the reasoning engine, they can not perform the services discovery at runtime.

We argue that, in the real world, most Web Services are informative Web Services that only provide information about the actual business services. For example, we can get flight information from a Web Service for a travel plan. But this flight information Web Services is an informative Web Service that does not provide the business service itself. Therefore, this information Web Service is not part of the travel plan. In this paper, we study a wider Web Service composition problem where informative Web Services are queried to get the parameters of their correspondent services. The resulting business process can be any kind of service or activity without being limiting to Web Services. We consider this as a more realistic Web Service composition problem.

The challenge of the wider Web Service composition problem is that the problem space is not fully enumerable. It is impossible to get all the information stored in the databases of different Web Services through queries. Therefore we can't model the whole problem space and feed it into a planner. In this paper, we present a solution

to combine GA with planning so that GA helps to navigate in the large search space and build sub space by querying Web Services. At each GA loop, the planner is called to construct plans at the sub problem space. The resulting plan is considered to be optimal.

The rest of the paper is organized as follows: section 2 gives the basic knowledge on GA and AI planning; section 3 presents the wide Web Service composition problem; section 4 presents our solution to the wide Web Service composition problem; section 5 describes the implementation details and experimental results; section 6 compares our work with related work; and, section 7 draws the conclusion.

2. Background

The method of Web Service composition developed in this paper is based on AI planning and Genetic Algorithms. Some background knowledge is introduced here.

2.1 AI Planning for Web Service Composition

Many research efforts tackling Web Service composition problems via AI planning have been reported [10][11][12][13]. Classic AI planning representation has three parts: the planning domains, problems and solutions [6]. The *planning domain* is a state transition system $\Sigma = (S, A, \gamma)$, where S is the set of all possible states of the world, A is the set of actions the planner can perform in attempting to change one state to another in the world, and the translation relation $\gamma \subset S \times A \times S$ defines the precondition and effects for the execution of each action. We use $\gamma(s, a)$ to represent an action a applied to state s. The *planning problem* is a triple $P = (\Sigma, S_0, G)$ [6], where $S_0 \subset S$ denotes the initial state of the world, $G \subset S$ denotes the goal state of the world the planning system attempts to reach. A *plan* π is any sequence of actions $\pi = \langle a_1, a_2, \ldots, a_k \rangle$ is a *solution* to P if $G \subseteq \gamma(S_0, \pi)$.

[6] introduces several techniques to build AI planners, namely, search, planning-graph, propositional satisfiability, constraint satisfaction and logic inference. Checking the planners used for Web Service composition, In [11], the planner is based on model checking techniques. In [12] the planner is based on situation calculus, a first order language for states and actions in dynamic domain. In this paper, we use SHOP2 [8], Simple Hierarchical Ordered Planner, as the planner, similar as [10]. Developed by the University of Maryland, SHOP2 is a domain-independent planning system based on HTN planning. Compared with classical planning, HTN is designed to perform some set of tasks rather than achieve a set of goals. To create plans, HTN planning recursively decomposes a task into smaller and smaller subtasks by a set of Methods until primitive tasks are reached and can be performed directly.

PDDL [7] (Planning Domain Definition Language) is the language originally proposed by the AIPS-98 Competition Committee for use in defining problem domains. It is the standard language for the representation of planning domains and problems. Planning tasks defined by PDDL are separated into two parts: one is domain, which includes predicates and actions; the other is problem, which includes objects, initial states, and goals. PDDL is used to describe problem and domain in this paper.

Existing work in [10][11][12] about using AI planning for Web Services focuses on how to convert Web Service composition problem into planning problem and how to represent the user constraints and goals from knowledge representation point of view. These methods work when all relevant service descriptions are initially loaded into the reasoning engine, they do not perform the services discovery at runtime. This paper presents another point of view on Web Service composition problem.

2.2 Genetic Algorithms

Genetic Algorithms (GA) introduced in the 1970s by John Holland [5]. The concept comes from the principle of Survival of the Fittest by Charles Darwin. GA is proposed as a search algorithm and has proven to be powerful in rapidly discovering good solutions for some difficult problems, especially when the search space is large, complex and poorly understood. In GA, each possible solution is represented by a chromosome and the goal solution is to be optimized corresponds to the fitness function. GA creates a population of solutions, and then generation by generation, the population is manipulated by genetic operators such as mutation and crossover. The propagation stops when some termination criteria are satisfied and the optimized solutions are found.

GA is intrinsically parallel and inclined to determine the global optimum. Since GA can generate many offspring in a complete loop, it can explore the search space in a multi-direction way. GA has been proven to be effective at escaping local optima and discovering the global optimum through genetic operations in some problems. With crossover, there is a transfer of information between successful solutions, which means offspring can benefit from what parents have learned, and parental schemata can be mixed and combined so as to reproduce next generations with strengths of both their parents. Therefore, GA has a higher probability of finding the global optimal solution in a relatively short time compared with other classical heuristic search algorithms.

3. Distinguishing Web Services Composition from Business Process Composition

The existence of BPEL4WS and other business process modelling languages from the Web Service community make people equate Web Service composition to business process composition. We need to point out that business process is a much broader concept than a process composed by Web Services. According to [9]:

Business Process is a collection of related, structured activities – a chain of events – that produces a specific service or product for a particular customer or customers.

The activities in business processes can be of many kinds, including activities fulfilled by humans. Only those activities that use data flow can be wrapped as Web Services. If a Web Service composition problem only studies the business process composed by Web Services, we call it a **narrow Web Service composition problem** (**narrow WSC**). Most existing Web Service composition papers actually study narrow WSC problems (cf. section 2.1). When using planning for narrow WSC problems, a Web Service is modelled with pre-conditions and post-conditions. The execution of a Web Service would change the state of the world. The planner decides orders of the Web Services according to the goals. From a software engineering point of view, the outputs of a Web Service are the inputs of its following Web Services.

However, narrow WSC does not cover all the scenarios for which Web Services can be used for business process composition. For example, if we want to build a software agent that can search for Web Services that provide hotel and airline information to build a travel plan, the Web Services that the agent uses are not part of the business process. Therefore, we need to introduce the **wide Web Service composition problem** (**Wide WSC**):

The **wide Web Service composition** problem is to build a business process by ***using Web Services***.

Using Web Services means not only compose Web Services into a business process, but also use information from Web Services at the planning phase. Many existing Web Services are what we call **informative Web Services** that only provide information about a service or a product. They are not the activities in the composed business process. Examples of informative Web Service are flight schedule Web Services and stock quote Web Services. Invoking this kind of Web Services does not change the state of the world, because no real business actions are taken. Currently a large part of available Web Services are informative Web Services. Some evidences are the services listed by www.xmethods.com. Another example is that a travel plan is decided by the availability of flights and hotels and the information can be got only by informative Web Services. In contrast to informative Web Services, we call the Web Services that provide actual business services **executable Web Services**. Some examples are order processing Web Services and ticket purchasing Web Services. Narrow WSC is a special case that all business activities in the composed business process are executable Web Services. The relations of narrow WSC and wide WSC to Business Process Composition (BPC) is:

$$\text{Narrow WSC} \subset \text{Wide WCS} \subset \text{BPC} \quad (1)$$

The output of a wide WSC problem is a normal business process that includes many kinds of activities, and is not limited to Web Services. The representation of a business process can use a BPEL4WS with some work-around, or some description languages from Workflow community.

In this paper, we focus on the Wide WSC problem that is to use informative Web Services for planning a business process. We will demonstrate how to tuning the parameters of services to optimize the orders of activities.

4. Solving the Wide WSC Problem

4.1 Motivating Example

Consider a travel plan problem. Suppose John wants to travel from Fredericton to Toronto on business. Here are the activities he proposes:

- Possibly leave on Saturday and return on the following Friday. (The date is flexible depending on the price of the flight ticket and hotel.)
- Business meetings on Monday and Tuesday.
- Stay in hotel in Toronto.
- Rent a car after arriving at Toronto.

John is free on Thursday afternoon, so he plans to do the following:

- If the weather is good, watch a baseball game.
- If the weather is bad, see a music performance and find a restaurant for supper.

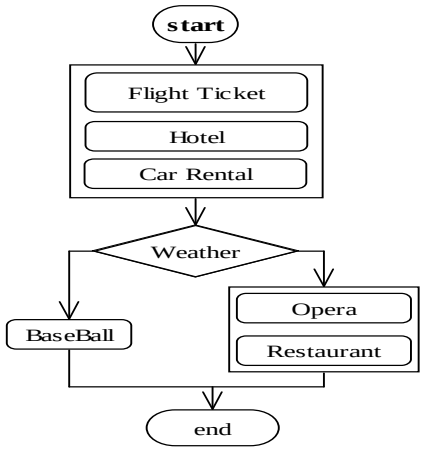

Figure 1. John's travel activity flowchart

There are some constraints, e.g., the date of booking a hotel and renting a car depends on the flight ticket, and the order of having supper and watching music performance depends on the price of the music. (If the

afternoon music ticket is expensive, John will have supper first watch the performance at night.) Suppose several Web Services provide information about flights, hotels and car rentals. The business process composer needs to choose the best service and determine the plan for the trip. Figure 1 is the flowchart of John's activities.

4.2 The Composer and the Partial Observable Problem Space

We want to make use of informative Web Services to compose a business process. The working environment of a Web Service composer is shown in figure 2. The user requirements are fed into the Web Service composer (WS Composer in Figure 2). The WS Composer queries informative Web Services ($IWS_{1...n}$ in figure 2) about the parameters of services. The business process is the output of the WS Composer.

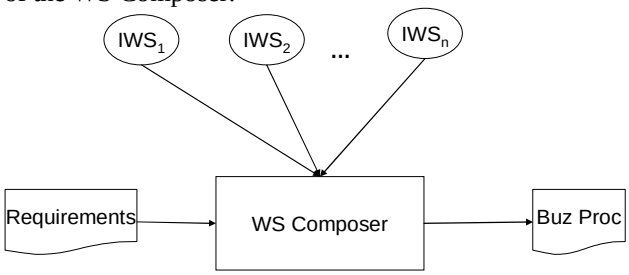

Figure 2. The working environment of Web Services composer

WS Composer needs to solve the following issues:
1) ***Orders*** of the services to reach the goal. AI planning is a proper choice for this task.
2) ***Constraints*** among the services. Although the services are independent to each other as a single service, when they were put into a sequential execution path, the execution of the services depends on the input and output of the context services.
3) ***Parameters*** of the services. Planning can also deal with parameters if parameters are modelled in the problem.

The difficulty is that in the Web Service environment, the parameter data belongs to different Web Services. This means that it is impossible to enumerate the problem space and feed it into a planner. ***The problem space is not fully observable.*** In order to get the optimized solution, we need yet another tool to navigate the problem space. This leads to one solution presented in the next section.

4.3 Combining Genetic Algorithms with Planning

We present a method to optimize planning results with an incompletely observed problem space where Genetic Algorithms (GA) is chosen to navigate.

The COMPOSER algorithm is in Figure 3. The main function is based loosely on the GA described in [14]. The Web Services are organized as clusters based on the similarity of their functions. For example all flight information Web Services are in one cluster. Figure 4 shows *n* clusters. One Web Service $ws_{k(i)}$ is selected randomly from cluster *i*. At each loop (**) in Figure 3, two individuals *x* and *y* will be chosen as parents, the selection based on the fitness of the individual. Once two parents have been selected, the GA recombines them to create two new offspring through crossover and mutation operations. Then each new individual will be evaluated by CHROMOSOME-EVALUATION. The process (**) will go on until we generate the same number of offspring as that of the parent. Those new offspring are one generation. Process (*) will repeat generation by generation until one of the termination criteria is satisfied.

```
function COMPOSER
  inputs: population            //a set of individuals
  returns: an individual

  domain ← PREPAREDOMAIN        //manually
  repeat                                          (*)
    new_population ← empty set
    loop for i from 1 to SIZE(population) do      (**)
        x ← SELECT(population)
        y ← SELECT(population)
        child ← CROSSOVER(x,y)
        child ← MUTATE(child)
        CHROMOSOME-EVALUATION(child)
        add child to new_population
    end of loop
    population ← new_population
  until some termination criteria is met
  return the best individual in population
end COMPOSER

function CHROMOSOME-EVALUATION
  inputs: chromosome
          FITNESS-FN    //a function that measures
                        //the fitness of an individual
          USER-PRE-FN //a function that represents
                        //users preference
  returns: a fitness-value

   gene ← DECODE(chromosome)
   data ← QUERY-WS          //get data from WS
   //generate problem in PDDL format
   problem ← PREPAREPROBLEMS(data)
   //generate all solutions
   solutions ← PLANNER(domain, problem)
   //Choose the best solution
   best-fitness-value ← FITNESS-FN(solutions)
end of CHROMOSOME-EVALUATION
```

Figure 3. Web Services composition algorithm

In the function CHROMOSOME-EVALUATION, the chromosome will be decoded into genes which represents the correspondent Web Services $\{ws_{k(i)} | i=1,\ldots,n\}$ (as in figure 4). These Web Services are queried to get data set $\{Data_{1\sim d(i)} | i=1,\ldots,n\}$ which are the parameters of the services. The data are used to generated the *problem* in PDDL format. The planner is called to find all the possible plans with the predefined *domain* and the generated *problem*. The best solution in this planning execution will be scored as the fitness value of that chromosome. In this model, the planner is executed every time a chromosome is evaluated. The Planner take the chromosome as input and output the fitness value, the orders and constraints of the services are handled in the planner.

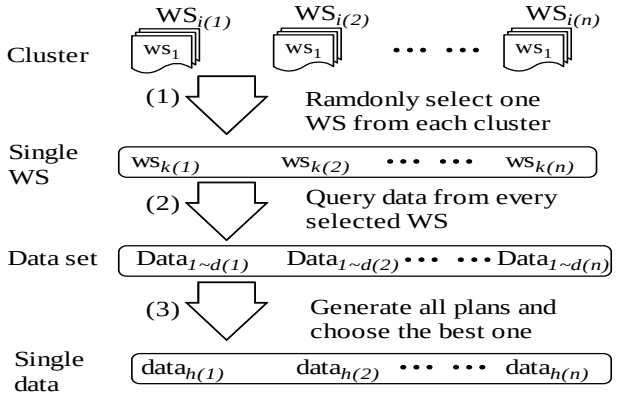

Figure 4. The process of generating an individual

5. Algorithm Implementation and Results

5.1 Implementation of the Genetic Algorithms

5.1.1 Definition of chromosome. In our model, we use a binary bit string, also called a chromosome, to encode the business process. We define *Chromosome* = [C_1, C_2, …, C_n], where n is the number of Web Services clusters, a Web Services cluster is a group of Web Services that perform a specific task, and C_i represents a specific Web Service. That means we select one Web Service from one cluster. We assume that the Web Services in one cluster perform the same task and the clusters are disjoint. Forming the clusters and the possible heterogeneous descriptions and interfaces of the Web Services are not under consideration of this paper.

According to our scenario, we have six clusters: Flight Ticket; Hotel; Baseball; Opera; Car Rental: and, Restaurant. The solution can be the combination of some or all clusters, but the chromosome should include all clusters, since we do not know which cluster will be chosen until we run it in the planner. We suppose that only one service can be selected from one Web Services cluster.

Before encoding the chromosome, we should collect all the available Web Services in the same cluster and assign each concrete Web Service a binary bit string as a gene. For example, the Sheraton Hotel Web Service might be represented by a binary bit string {0000100}. The length of gene depends on the number of the Web Services in the cluster. If the length is too short, it can not represent all the Web Services in this cluster. If the length is too long, it might have many useless genes that do not represent any Web Service, and hence could make the convergence of GA slow.

Table 1 lists one of the possible Web Service combinations. So the chromosome for this solution is: [0011001,0101011,1100111,1010101,1111001,1010111]. Other potential solutions might be the composition of selecting SouthWest Airline, Hilton hotel, etc.

5.1.2 Selection of Chromosome. An important aspect in GA is to decide which individuals should be chosen as parents. The selection is based on the fitness of the individuals. According to the "survival of the fittest" principle, a chromosome with higher fitness values has a greater chance of being selected as a parent. In our model, we use the roulette wheel selection.

Every time a new population is made, the chance that we might lose the string with the best evaluation occurs. This could make the result unstable and slow the convergence. To overcome this problem, we use *elitism* by simply copying the two best chromosomes into the succeeding generation.

5.1.3 Genetic operations. Once two parents are selected, we need to recombine them by performing genetic operation to generate new offspring. Two new individuals are created that typically share many of the characteristics of their "parents." The genetic operations we used in this paper are crossover and mutation.

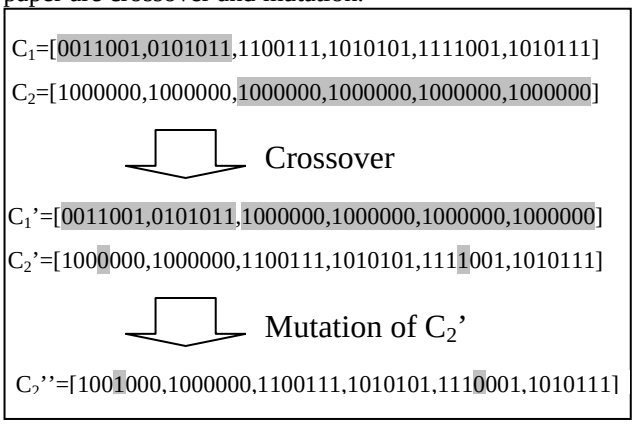

Figure 5. Crossover and mutation operations

Here, we use one-point crossover, which randomly selects an integer k between 1 and the number of the cluster n, [1, n]. Two new chromosomes are created by

swapping all bits between position k and n. Note that the crossover operator is not always performed, and the probability of the crossover depends on the crossover rate.

Mutation is performed after the crossover. The purpose of mutation is to introduce divergence into a converging population, which can avoid setting the algorithm trapped into a local optimal solution. Each bit along the chromosome might have a chance to flip from 1 to 0 or vice versa. As in the crossover operation, the probability of doing the mutation operation depends on the mutation rate, which is very small. Figure 5 shows the examples of crossover and mutation operations.

5.1.4 Definition of Fitness Function. The fitness function really depends on the specific problem we are trying to solve, but the general idea is to give a higher score to the closer solution. After every new chromosome is generated, the solution must be evaluated in order to decide whether it is a good or bad individual by using fitness function. In our model, each chromosome might have zero or more solutions to the problem because each Web Service might provide more than one piece of data.

For example, in the chromosome shown in Table 1, we will have two solutions because Air Canada Web Service provides two pieces of flight information. Typically, in a business process, low cost is one of the important user considerations. In this case, we chose Flight AC1024 instead of Flight AC08. Other criteria for selecting the solution can also be defined according to the user's preferences and common sense in practice. We do not consider those non-measurable aspects or non-functional demand in this paper.

Table 1. Potential solutions for John's trip

Gene	Web Service	candidate data
0011001	Air Canada	Flight AC08, $300
		Flight AC1024, $250
0101011	Hotels.com	$50 for each day
1100111	National car rental	$100
1010101	Japanese food	$15
1111001	Best baseball	$15
1010111	Classic opera	$10

After we choose the best solution, we can assign a fitness score to that chromosome. The score can be the inverse of the total cost of all services, that is, $1/\sum(\text{cost})$. For example, we assume the total cost of the solution in the previous chromosome has a value of 380. Then the fitness score of this chromosome is 1/380. If the planner can not find any solution for a chromosome, we consider this chromosome is unvalued and give it a very low fitness score.

5.1.5 Termination. The iterative process continues until one of the termination conditions has been reached. In this paper, the terminating flags are: (1) when a known optimal or acceptable solution level is attained; (2) when the best fitness score is kept the same for more than a pre-setting repeated time; and, (3) when a maximum number of generations have been performed.

5.1.6 Parameters. Some factors can affect the result of the GA greatly, such as population size, crossover probability, mutation probability, and the strategy for a selection method. In our (Table 1) model, we set the parameters as follows: population size = 60; crossover rate = 0.6; mutation rate = 0.001; the maximum generation value was set to 100; and the elitism value is 2.

5.2 Planning

Every chromosome must be evaluated to be able to distinguish between good and bad. The following steps describe how we use planning to help evaluating the chromosome.

5.2.1 Domain Description in PDDL. In our model, the description of the problem will change according to the chromosome, but the description of the domain will be the same, since it is in the same travel domain. Therefore, we need to define the domain only once at the beginning. Figure 6 is the snippet of the domain description.

```
(defdomain tripPlanning
(
  ;;define book ticket operator
  (:operator (!book-ticket ?s1 ?d1 ?name1 ?date1 ?flight1)
    (
      (trans-info ?s1 ?d1 ?name1 ?date1x ?time1x
       ?date1xx ?time1xx ?flight1 ?price1)
    )
    ( )
    ((by-airlineservice ?s1 ?d1 ?name1 ?date1 ?flight1))
  );;end of book ticket operator
  ;;define book hotel operator
  (:operator (!book-hotel ?name2 ?date2 ?days2)
        (
         (hotel-info ?name2 ?date2 ?price2)
        )
        ()
        ((by-hotelservice ?name2 ?date2 ?days2))
  );;end of book hotel operator

  … …
```

Figure 6. Snippet of domain description

5.2.2 Problem Description in PDDL. The problem description will change according to the chromosome because different chromosomes have different combinations of Web Services, and each Web Service will provide different data. So, to generate the problem dynamically, we need to query data from the given Web Service. (In our model, we use the artificial data set to simulate the data from different Web Services.)

Note: not all genes can be mapped to the Web Service, since some genes are useless. In this case, it will return an empty data set.

After querying all the data, we can generate the problem using the template. For each domain, we define a problem template. Figure 7 is the snippet of the problem description.

```
(defproblem problem tripPlanning (
;; Flight Data
(trans-info fredericton toronto
  aircanada117 20051201 6 20051201
  6 F11730 236.1764)
(trans-info fredericton Toronto
   aircanada117 20051201 6 20051201
  6 F11752 261.2015)

  … …
;; Hotel Data
(hotel-info sherton50 20051201
165.8853)
(hotel-info sherton50 20051201
169.6691)

  … …
;; Other Information
(stay-days 6)(total-cost 0))
((travel-planning fredericton
toronto)))
```

Figure 6. Snippet of problem description

5.2.3 Execution of Planning. After the domain and problem are generated, the planner can be executed. The solution of the planning will be scored by fitness function in the GA. The integration of the planner and GA is showed in Figure 4.

5.3 Results

To test the motivating example in section 4.1, we set up some artificial data as follows:

- Six clusters.
- 100 Web Services in each cluster.
- 100 pieces of data in each Web Service.

The goal is to find the lowest cost solution for John's trip.

In this model, there is no specific known solution, which means no goal state. So we will choose the (2) and (3) in the section **5.1.5** as terminating flags. In the experiment, the highest score chromosome in the last generation is the best solution. The best result is:

{1010000,0011110,0101011,1011111,*,0010001}

In the result chromosome, gene * represents Baseball Web Service. We can see that the choice of Baseball Web Service will not affect the final score because there is no action on the Baseball Web Services cluster in the solution list. We can see from Figure 7 that the mean fitness value is improved with the continuing iteration. The activity flowchart of the final solution is shown in Figure 8. It includes six actions: booking flight ticket; booking hotel; renting a car; booking museum ticket; booking restaurant; and, the last action of calculating the total cost.

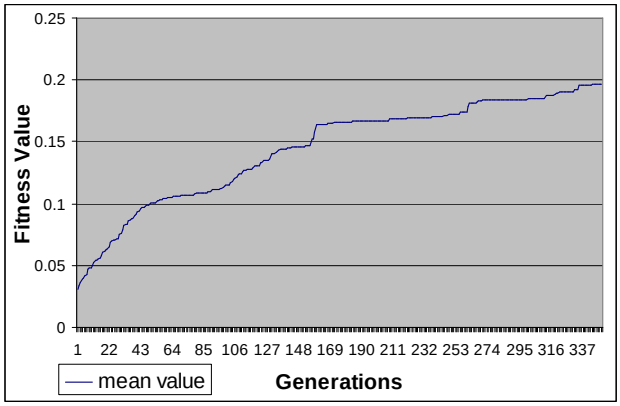

Figure 7. The evolution procedure

```
A plan with cost 6.0 was found:

(!book-ticket fredericton toronto flightws80 20051201
f8021 150.17)
(!book-hotel hotelws30 20051201 63.02)
(!rentcar carrentalws43 17.3209)
(!book-opera operaws17 13.1143 17:00 18:30)
(!book-restaurant restaurantws95 8.432)
(!set-cost 0.0 252.0572)
```

Figure 8. Final Solution

6. Comparison with Related Work

This paper uses GA and AI planning to solve the Wide WCS problem. It is related to, but different from, some existing work. First, most existing Web Service composition papers using AI planning discuss the narrow WCS problem [10][11][12][13] are based on the matching of input, output parameters and the properties of the services, beyond this, our approach focus on the dynamics of information queried from the services. The planning execution is leaded by GAs and executed multiple times. We also argue that the Wide WCS problem is a more realistic problem in the real world, where most available Web Services are informative Web Services. Second, the Wide WCS problem is also different from the distributed AI planning problem. Distributed AI planning studies planning problems in a multi-agent system where each agent has its beliefs, goals and intentions [15]. In a Web Service environment, Web Services themselves are not intelligent agents that are ready to negotiate and change their actions. Indeed, Wide WCS problem is not a distributed planning problem in this sense, because planning is done at a central point. The distributed environment just makes the problem space un-enumerable.

Some other work does not use planning for Web Service composition. Reference [16] considers Web Service compositions as parameter selections when the template of a process is determined. Though planning can deal with parameters if they are modeling in the problem, most WS composition papers using planning do not model parameters. However, our method models parameters and determines both the orders of services and parameters of services. Reference [4] uses also GA to select services. Their work uses process templates where services are selected based on QoA parameters. It is different from the problem we study in this paper.

7. Conclusions

This paper presents a method to solve the Wide WCS problem where informative Web Services are queried to get parameter information for services. We regard this as the common situation in the real world, since most of the available Web Services are informative Web Services. Since we can't get all information in all the databases of all the Web Services, the problem space is only partially observable through querying the Web Services. We present a solution to combine GA with planning techniques to get optimal plans. In each GA loop, Web Services are queried and new sub problem spaces are constructed. The planner generates plans in the sub space. The plans are evaluated by the fitness function to select better plans. This method is demonstrated by an example. The results show it is a workable solution. In the future, we will work on efficiency improvement and will automatically formulate the user preferences as constraints.

8. References

[1] Web Services Business Process Execution Language Version 2.0, Working Draft, 23 August 2005, http://xml.coverpages.org/WSBPEL-SpecDraftV181.pdf, retrieved in Dec. 2005.

[2] OWL-S: Semantic Markup for Web Services, W3C Member Submission 22 November 2004. http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/, retrieved in Dec. 2005.

[3] Web Service Choreography Interface (WSCI) 1.0, W3C Note 8 August 2002. http://www.w3.org/TR/wsci/, retrieved in Dec. 2005.

[4] Zhang, L.-J., Li, B., Chao, T., Chang, H. "On Demand Web Services-Based Business Process Composition", Proceedings of the 2003 IEEE Conference on System, Man, and Cybernetics (SMC'03), Washington DC, 5-8 Oct. 2003 pp:4057 - 4064 vol.4.

[5] Holland, J.H. "Adaptation in Natural and Artificial Systems", Cambridge, MA. The MIT Press.

[6] Ghallab, M., Nau, D., and Traverso, P., "Automated Planning: Theory and Practice", Morgan Kaufmann, 2004.

[7] The Language of the Fourth International Planning Competition. ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/, retrieved in Dec. 2005.

[8] Documentation for JSHOP 2.0. http://www.cs.umd.edu/projects/shop, retrieved in Dec. 2005.

[9] Glossary of US Government Accountability Office, http://www.gao.gov/policy/itguide/glossary.htm, retrieved in Dec. 2005.

[10] Wu, D. and Parsia, B., "Automating DAML-S Web Services Composition Using SHOP2", Proceedings of ISWC 2003, Sanibel, Island, FL, USA October 2003, pp.195-210.

[11] Pistore, M., Traverso, P., Bertoli, A., and Marconi, A., "Automated Synthesis of Composite BPEL4WS Web Services". In Proc. of ICWS 2005, Orlando, Florida, USA, July 11-15 2005, pp.293-301.

[12] McIlraith, S. and Son, T.C. "Adapting golog for composition of semantic Web Services", Proc. of the 11th international conference on World Wide Web, Honolulu, Hawaii, USA, 2002, pp. 77 – 88.

[13] Rao, J. and Su, S. "A Survey of Automated Web Service Composition Methods", Semantic Web Services and Web Process Composition, San Diego, California, USA, July 6, 2004, pp.43-54.

[14] Russell, S., and Norvig, P., "Artificial Intelligence: A Modern Approach", second edition, Prentice Hall, p116.

[15] Durfee, E., "Planning in Distributed Artificial Intelligence", Foundations of Distributed Artificial Intelligence, Edited by G. M. P. O'Hare and N. R. Jennings, John Wiley & Sons, Inc. 1996, pp.231-245.

[16] ten Teije, A., van Harmelen, F., and Wielinga, B. "Configuration of Web Services as Parametric Design". Valencia, Spain, ECAI 2004, August 23-27, pp.1097-1098.