AI Planning Search Time and Space Reduction Technique for the Problem of Web Service Composition

Hossein Rahmani

Sharif University of Technology, Azadi Ave., Tehran, Iran,
P.O.Box: 11365-9517, office No: 313
H_Rahmani@ce.sharif.edu

Abstract

One of challenging problems in the domain of web services is the web service composition (i.e the task of composing web services to create new services, capable of fulfilling a need that no single service can satisfy). In real application, there will be a lot of relevant web services for a simple goal, so the search space for such a problem is huge. In this research statement, we show how by extracting some prior knowledge and using simple heuristics, search space and search time can be reduced significantly.

Proposed System

Due to large amount of the web services, adoption of the best composed services in a reasonable response time is the main goal of the proposed system. The proposed system is based on the two simple ideas decreasing both search space and search time. The first idea is *detect the useful runtime knowledge which can be processed offline* and the second idea is *use of offline extracted knowledge for directing the search result in reducing both search space and search time.*

Offline Extractable Knowledge

After inspecting several search methods which are used in the web service composition problem, we discovered that most parts of their response time are wasted in acquiring some kind of information which is extractable in offline. Use of past experience in most of them means learning, but in our system we do not mean learning**,** we just propose to keep the knowledge in easy retrievable structure like matrixes so each process is done just once even if appears several times in one problem or become visible in different problems. As we discussed about Graph based methods, during graph building phase, they spend large amount of time finding relevant services for a given goal. Since we want to avoid this time wasting state, our first data structure is ***L**iteral **S**ervice **D**ependency **M**atrix* (we called it LSDM in short) which encodes the static relationship among the literals and services. In the row "i" and column "j" of the LSDM we put the web service name which takes literal "i" as input and generate the literal "j" as output. If we imagine Fig (1) as set of available web services, then Table (1) contains its corresponding LSDM.

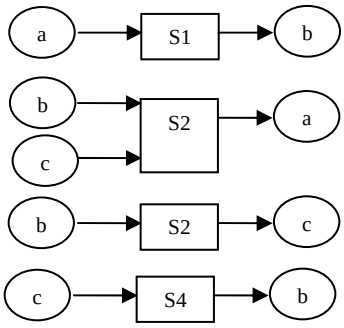

Fig 1. Simple set of web service.

Table 1. LSDM for web services shown in fig (1).

	a	b	c
a	#	S1	S3
b	S2	#	
c	S2	S4	#

The other offline extractable knowledge which extremely effects our search direction is ***G**raph **P**lan **L**iteral **M**atrix* which we called it *GPLM* in short. We use GraphPlan as distance measure between the individual literals. In the row of the "i" and column "j" of the GPLM we put distance between literal "i" and literal "j".

In order to calculate the distance between literal "i" and literal "j' we build the GraphPlan with the first layer containing literal "i" and build the graph till the literal "j" appears. Table (2) shows the distance between each distinct literal of fig (1).

Table 2.GPLM for web services shown in fig (1).

	a	b	c
a	0	1	2
b	2	0	1
c	2	1	0

Search Direction Strategy

In this part of proposed system, first we introduce a simple heuristic for directing the search process. Second, in order to reduce both search space and time, we exploit the offline extracted knowledge in our heuristic search. In our system the main search algorithm is *backward search*. In each step of backward search, there might be more than one relevant service. Since a backward search may try lots of services that can't be reached from the initial state, deciding to select which relevant service for satisfying the goal can significantly reduce the search space. As inputs of relevant chosen service will be added to the goal list, one simple idea can be about how much criticality one service will add to the problem. The Concept of criticality in the problem is how far service's inputs from the init state are. We propose simple general *min-max-min* heuristic search algorithm for directing the backward search and reducing backtracking count. As it shown in fig (2), imagine there are three relevant services (i.e. S1, S2 and S3) for a given goal G. web service S1 has two inputs (inp_11 and inp_12), web service S2 has two inputs (inp_21 and inp_22) and web service S3 has two inputs (inp_31 and inp_32). There are three literals in init state (i.e. init1, init2 and init3). The result of our proposed heuristic search is order of relevant services for satisfying the given goal "G". After finding relevant services of one given goal, we calculate the distance of each input literal of services to the literals appeared in init state (i.e. calculating distance between inp_11 to all the init1, init2 and init3 literals). We can use any kind of distance measure. Our chosen distance measure will be described later. For each input we find the minimum distance of it to the init state literals. For every relevant service, we will find the most critical input literal. In our heuristic search algorithm, we apply maximum to the input minimum distance (i.e. take maximum between *inp_11* minimum distance and *inp_12* minimum distance). Then we order relevant services based on their most critical input literal (i.e. the one which has maximum minimum distance). In simpler words, we first apply the relevant web service which it's most critical input literal has minimum value. The algorithm is general since we can use any kind of distance measure. One of the best candidates for distance measure could be GraphPlan. Most kind of graph building algorithm wastes large amount time to build a graph in runtime. Due to large number of web services in the web service composition problem, building GraphPlan in runtime will be take so much time and is not feasible for real application. By use of GPLM, we use benefits of GraphPlan and not wasting any time for building the graph in runtime. Since GraphPlan based distance between each individual literal is determined in offline and saved in the GPLM, heuristic distance calculation does not make response time longer and has no cost.

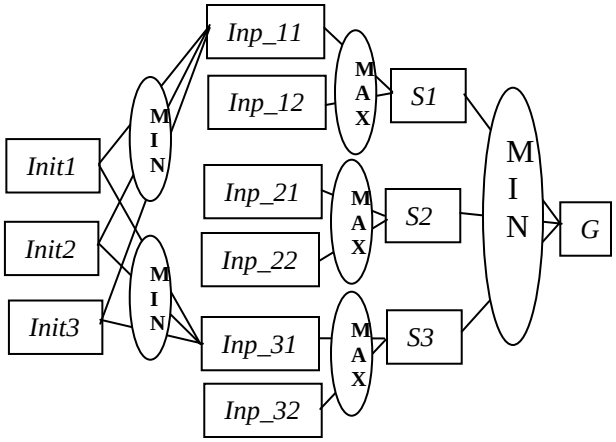

Fig 2. Min-Max-Min heuristic search algorithm.

Discussion

We applied AI Planning concepts to web service composition problem. Web service composition problem is different from the usual AI Planning problems since it is kind of real and interactive problem. The user of system provides problem description (i.e. init and goal literals) and waiting for a system's response. Response time in interactive system is so important. Experiments show that in the problems which do not have solution; "heuristic graph plan" is working so much better than the other methods. By looking to fig (3) we can find out that, although it seems the execution time difference between our proposed framework and other algorithms is small, but our proposed framework execution time is increasing linearly in the size of the problem. The other methods execution time increase near exponentially to the size of the problem.

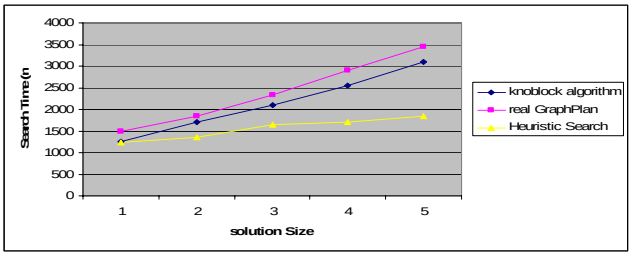

Fig 3. Search time of Algorithms based on solution size.