Agent-based Dis-graph Planning Algorithm for Web Service Composition

Jian Feng Zhang and Ryszard Kowalczyk
Faculty of Information and
Communication Technologies
Swinburne University of Technology
Melbourne VIC 3122, Australia
Email:{jfzhang, rkowalczyk}@ict.swin.edu.au

Abstract

In agent-based planning for service composition, a single agent may not have access to all services required to fulfill a given goal. Cooperation among multiple agents, each of whom has a limited set of services, can increase the chance of generating a valid/effective composition. However, agents may be reluctant to reveal their services to other agents during cooperation due to the concerns to privacy, security, and difficulty of a heavy communication overhead. In this paper we present a distributed algorithm that allows agents to cooperate and form a composition in such a situation. This algorithm can be viewed as a distributed extension of the reputed centralized planning paradigm, i.e. Graph planning + CSP. We adapt classical planning graph to distributed planning graph, represent the graph in distributed CSP (DisCSP) instead of CSP, and solve the DisCSP with distributed search algorithm. All agents have equal authority and no centralized coordinator is required in the whole process.

1 Introduction

Web Services (WS) and service oriented architecture (SOA) have received a wide interest, as their underlying infrastructure enables loosely-coupled integration and composition of complex services. Automated support to the composition of Web Services is a vital factor for realizing the potential value of WS and SOA. Some efforts have been published in recent years where existing AI planning techniques are generally adopted to compose services. For example, WS-GEN automates Web services composition using MBP, which is an AI planner based on model checking [8]. Wu et al. [11] utilizes an HTN planner SHOP2 [7]. OWLS-XPlan is based on XPlan, which is a hybrid planner combining HTN planning and graph-plan based Fast-Forward planning [6].

However, the approaches assume a single planner has an access to all the services required to create a workable/effective composition for a given goal. We argue that in the Web service environment this is not always the case. A single planner may not have an access to all the required services, and therefore multiple planners need to cooperate and contribute their services to generate a composition.

In this paper we present an approach to distributed composition of Web service, where multiple agents, each of whom has access to a limited set of services, contribute their services to create a composition collaboratively. We combine the techniques from planning graph and CSP to practically solve the automatic composition of Web service in distributed environment in a decentralized fashion.

The remainder of this paper is organized as follows. Section 2 describes the motivation and background of our work. Section 3 gives a detailed description of the algorithm. It begins with an overview of the system architecture and procedure of planning, then gives the description of constructing dis-graph, compiling dis-graph to DisCSP (Distributed CSP) and solving the DisCSP. Section 4 presents a brief comparison to related work. We present the conclusion in Section 5.

2 Motivation and Background

In this section we give a brief description about the motivation and background of our algorithm. Our algorithm is a distributed extension of a classical planning paradigm, planning graph + CSP [4]. Planning graph [1] is one of the most efficient domain independent heuristics in AI planning. CSP (Constraint Satisfaction Problem) has been adopted to graph planning since the first algorithm of this line, GRAPHPLAN, was developed. Distributed CSP (DisCSP) [12] shows the opportunity of shifting graph planning to distributed environment, to meet the requirements of Web service applications.
2.1 Motivation

In this paper we argue that in some Web service scenarios the knowledge necessary for making plans is distributed by nature, and it is impossible or ineffective to gather all information together during planning, which necessitates distributed planning approach.

There can be several reasons why we can not gather necessary information, ranging from security concerns, concern of business secrets, limited access to available services, to the absence of information retrieving/sharing facility.

Business secrets and security Existing work often assumes agents are willing to share their knowledge. However, in real world an agent most likely refuses to expose its data to the outside world.

Limited access to available services In real world data are valuable assets. An agent may have access to some database but not to others, may have more knowledge about services in one business/geographical area but less about those in other areas.

Absence of information retrieving/sharing facility The number of services can be huge and continuously growing, but retrieving/sharing facilities are relatively inefficient. A search engine/index server may well cover only a small portion of existing services. On the other hand, even the small portion is too voluminous to be transferred through network.

Due to reasons like these, we believe in Web service environment there is a growing need for distributed planning, where agents contribute to service composition but keep their privacy at the same time.

2.2 Background

Our work builds on existing research on graph based planning and CSP. Graph based planning utilizes Planning Graph as a heuristic to constrain the search for plans. It is popular in AI planning research due to its good performance [1]. Previous work shows that search in a planning graph can be formulated as solving a constraint satisfaction problem [4]. DisCSP shows the opportunity of adapting the centralized graph based planning to distributed environment.

Graph based planning

GRAPHPLAN [1] is the first planning algorithm using a planning graph. Generally, the graph based planning consists of two interleaved phases – extend the planning graph, and search for plans. A planning graph is a directed leveled graph (Fig 1). It consists of two kinds of alternating levels, state levels and service levels. The first level consists of initial states. The second level consists of services whose preconditions are present in the first level. The third level consists of the states appearing in the first level and the states brought by the services in the second level as their effects. In this way the graph is extended by state levels and service levels alternatively. During the construction, mutex relations are also identified. Two services in the same level are mutex if their preconditions and effects are inconsistent. Two states in the same level are mutex if all the services supporting the first state are mutex with all the services supporting the second state [4].

When the graph reaches a level where all goal states are present, the algorithm searches for plans. A valid plan is a subgraph without mutex services. If no plan is found, the planning graph is extended again. The search can be carried out by a backward search as in GRAPHPLAN, or by translating the graph into a CSP and solving it, as in [4].

Solving planning graph with CSP

Constraint Satisfaction Problem (CSP) is a problem solving paradigm applicable to a wide range of problems.

Definition (CSP }): A CSP is defined as:
1) A set of n variables \( X_1, X_2, ..., X_n \)
2) A set of n domains \( D_1, D_2, ..., D_n \) for the variables,
3) A set of constraints upon the values of the variables

Solving a CSP is a process of searching for an assignment of values to variables that satisfies all the constraints.

The search in CSP is similar to the search for plans in a planning graph [4]. The (partial) planning graph in Fig 1 can be translated into a CSP (Fig 2). Each service is trans-
lated into a CSP value. Each states is translated into a CSP variable, with the services supporting the state as possible values of the variable. Mutex constraints represent the mutex relations. Activity constraints represent the dependencies of services on the preceding services. If a solution is found, the services assigned to the states constitute a plan.

**DisCSP**

DisCSP is a variation of CSP where variables and constraints are distributed among agents [12]. Distributed search algorithms such as ABT (Asynchronous Backtracking), Weak-Commitment ABT and AAS (Asynchronous Aggregate Search) are used to solve a DisCSP [12] [9]. It does not need to gather all information from agents into a centralized place during the search. This feature allows cooperation among agents concerned about privacies. In our approach, we extend ABT to search for plans in a (distributed) planning graph.

### 3 Dis-graph Planning

#### 3.1 Overview

In this section we illustrate our approach by comparing it with classical graph based planning. For distributed planning, we devise a distributed planning graph (dis-graph) based on the idea of planning graph. Similar to a planning graph, a dis-graph consists of alternating levels of states and services, except that the levels is distributed among agents. Each agent constructs and keeps a part of dis-graph, called a fragment. In the search phase, we translate the dis-graph to a DisCSP instead of a CSP and solve it.

In our approach we model a Web service composition problem as an AI planning problem, so the terms plan and composition are used interchangeably in this paper.

**Definition (Service):** A service is defined by:
1) A set of states as preconditions. The service can be executed if all preconditions are satisfied (true).
2) A set of states as effects, which become true after the service is executed.

**Definition (Composition problem):** A composition problem is defined by the following components:
1) A set of initial states.
2) A set of goal states.
3) A set of available services.

A valid solution of the problem is a composition (plan), which is a sequence of services that lead from initial states to goal states. Since some of the services are allowed to be executed in parallel, the composition is in fact a directed network of services.

A distributed planning graph (dis-graph) under construction. Agent 1 contributes 1 block to the graph, agent 2 contributes 2 blocks and agent 3 contributes 1 block.

**Definition (Block):** A block is a series of consecutive state and service levels kept by one agent.

**Definition (Fragment):** All the blocks maintained and kept by an agent is referred to as a *fragment* of the planning graph.

From the perspective of the collective behavior of the agents, dis-graph planning is carried out as algorithm 1.

**Algorithm 1 Dis-graph planning**

1: composition problem is dispensed to the agents; each agent creates a planning graph in its memory, with initial states as the first level
2: extending graph
3: publicize latest state level
4: if reach-all-goals then
5: search-for-valid-plan
6: if found valid plan then
7: return;
8: end if
9: end if
10: decide-next-state-level
11: goto 2

In line 2, the planning graph is extended. Each time an agent executes line 2, it constructs one service level and one state level in its graph fragment. Then agents publicize their latest state levels. This information is used to check goals (line 4) and decide next level (line 10). *reach-all-goals* checks whether all goal states are present. If all goal states appear, *search-for-valid-plan* (line 5) is launched, which includes compiling planning graph to DisCSP, and solving...
DisCSP. If a plan is found, algorithm terminates with success. If no plan is found, agents decide-next-state-level to prepare a state level as foundation for further extending, and go to line 2. The termination conditions are not listed completely here, which could be the reach of an acceptable plan, reach of the limits of iteration times, etc.

Reach-all-goals: After the fragments are extended with a new state level, goal states may appear in some of them. Since the latest state levels have been publicized, agents can check goal states independently, and proceed to proper operation (search for plans or decide next level) without further coordination.

Detailed description of decide-next-state-level and search-for-valid-plan are presented in following sections.

Main features of dis-graph planning are briefly described as follows:

1) Agents are of same authority. There is no centralized coordinator.

2) Planning graph is distributed. Each agent keeps a fragment and none of them has the complete vision. This feature suits scenarios where agents are concerned about privacy.

3) The states in the graph are public information, and services are kept by agents as private information.

4) The search for plans is distributed. Each agent compiles its fragment to a set of CSP variables and constraints, and participates in distributed search to look for a solution accepted by all agents.

3.2 Construction of dis-graph

Fragments of a dis-graph are extended by the agents synchronously. From the perspective of an individual agent, it constructs its graph fragment as described in Algorithm 2.

Algorithm 2 Construction of graph fragment

1: construct the first level with the initial states
2: construct the service level with services supported by previous state level
3: construct the state level with the states produced by the previous service level
4: publicize the latest state level to other agents via message, and wait for the other agents’ messages
5: after receive messages from all agents, checks whether reach-all-goals
6: if all the goals are reached in this level then
7: search-for-valid-plan
8: else
9: decide-next-state-level
10: goto 2
11: end if

In the operation decide-next-state-level within Algorithm 2, an agent decides whether to extend its fragment from its latest state level, or to adopt other agents’ latest state level as foundation for further extending. We introduce two strategies for decide-next-state-level.

Choose the largest This strategy chooses the agent who has the largest latest state level as the winner. All the other agents adopt the winner’s latest state level as the foundation for extending their fragments, as in Fig 3.

Merge all This strategy merges all the agents’ latest state level to form a larger one. All agents adopt it as the foundation for further extending.

The Choose the largest strategy has less communication overhead than Merge all, but Merge all strategy has more chances to reach plans, or to reach a plan earlier.

3.3 Search for valid plans

If all the goal states are present, the search for plans is launched. From the viewpoint of a collective behavior of the agents, search is carried out by compiling dis-graph to DisCSP and solving it. Form each agent’s perspective, it compiles its dis-graph fragment to a set of CSP variables and constraints, and participates in distributed search for a DisCSP solution.

Figure 4. Compile Distributed Planning Graph

Compiling planning graph to CSP

All agents compile their planning graph fragments to CSP independently. We illustrate the compilation with the example in Fig 4, which shows a part of a dis-graph constructed with Choose the largest strategy. Two of agent1’s blocks and one of agent2’s blocks are shown in the figure. In principle, each agent does the following tasks: 1) Provide each state in its fragment with a unique CSP variable number, and each service with a unique CSP value number. The mapping from states/services to numbers is maintained by...
the agent locally. 2) Generate variables, including private variables and shared variables as described below. 3) Generate constraints, including private constraints and shared constraints as described below.

**Shared and private variables:** States on the borders of the blocks, e.g. P4, concern two agents, and the corresponding variables are shared variables. States inside the blocks, e.g. P6, concern one agent, and the corresponding variables are private variables.

A shared variable’s domain in one agent is different from the domain in another agent. For example, agent2 can assign S9, S10 or null to P12, and agent1 can assign null or some service belonging to agent2, without knowing the name to it. In order to indicate the some service without knowing the name, we introduce a dummy service TRUEOP. Hence P12 has domain \{null, S9, S10\} in agent2, and domain \{null, TRUEOP\} in agent1.

**Shared and private constraints:** The constraints between the states within a block concern one agent, and they are the agent’s private constraints. If we view each block as a small planning graph, private constraints are generated as the same as in classical graph planning.

 Besides those, we need constraints on shared variables. For example, agent1 and agent2’s assignments to P12 must comply with the rule: "agent1 assign TRUEOP to P12 ⇒ agent2 assign a service (S9 or S10) to P12; agent1 assign null to P12 ⇒ agent2 can assign any value (null, S9 or S10) to P12." These rules are shared constraints.

It is notable that the shared constraints are directional. From left to right in Fig 4, the succeeding block depends on the preceding block, so the shared constraints are put by the succeeding block to the preceding one. For example, the constraint upon P12 is the requirement from agent1 to agent2, and the constraint upon P4 is from agent2 to agent1.

In the example, agent1 generates the following variables and constraints:

Variables(private): none
Variables(shared): P1 : \{TRUEOP, null\}, P2 : \{TRUEOP, null\}, P3 : \{TRUEOP, null\}, …

Constraints(private):
(activity) P5 = S2 ⇒ P2 \neq null \land P3 \neq null,
P4 = S1 ⇒ P1 \neq null, …
(Mutex) P13 = S11 ⇒ P16 \neq S14

Agent2 compiles its block in Fig 4 in the same way.

Furthermore, both of the agents generate shared constraints as follows:

\[ P4(\text{agent}2) = \text{TRUEOP} ⇒ P4(\text{agent}1) \neq \text{null}, \]
\[ P5(\text{agent}2) = \text{TRUEOP} ⇒ P5(\text{agent}1) \neq \text{null}, \]
Now each agent has 1) a set of private variables, 2) a set of shared variables, and 3) a set of shared constraints involving shared variables.

With these variables and constraints, agents participate in distributed search for assignments of values to the variables that satisfy both private and shared constraints.

**Solving CSP**

From an agent’s viewpoint, it has two overlapped goals: 1) an assignment to shared and private variables satisfying private constraints, and 2) an assignment to shared variables satisfying shared constraints.

For the first goal, the agent carries out local CSP search with existing algorithm/implementation, e.g. JSolver [2]. For the second goal, the agent participate in a distributed search, e.g. ABT (Asynchronous Backtracking) [12]. Local search and distributed search interact by exchanging assignments to shared variables.

Regarding distributed search, we describe how ABT can be extended to solve our DisCSP. A detailed description of ABT can be found in [12]. To apply ABT, we made two extensions:

1) ABT assumes one agent has exactly one variable, but in our case one agent has a number of (shared) variables. We relax the assumption to allow one agent to have more than one variable.

2) In ABT there is only one constraint between two agents. To meet this assumption we bundle shared constraints between two agents as a "compounded constraint". However, in ABT, a constraint is directed from higher priority agent to lower priority agent, but in our case there are constraints in both directions between two agents. For example, the constraint on P12 is the requirement from agent1 to agent2, and the constraint on P4 is the requirement from agent2 to agent1. To bundle these constraints into one "compounded constraint", we remove the directions of requirements by specifying an order as

\[ \text{null} < \text{TRUEOP} < \text{services} \]

and transforming the constraints to ≤ and ≥ relationships.

For example, the constraints on P12 is transformed to

\[ P12 \text{ in agent1} \leq P12 \text{ in agent2} \]

Similarly, the constraint on P4 is transformed to

\[ P4 \text{ in agent1} \geq P4 \text{ in agent2} \]

To set up the order, we simply reserve the CSP value number 0 for null and 1 for TRUEOP as public numbers, and allow agents assign 2 and above to their services. Then we bundle the constraints and direct them arbitrarily.

Agents cooperate to look for assignments to shared variables following ABT algorithm, except for a difference from regular ABT that when an agent checks shared variables after receiving an ok?, or instantiates shared variables, it invokes local search to evaluate their consistency with private variables and constraints.

**4 Related works**

There has been considerable work addressing distributed planning and multi-agent planning. For example, Generalized Partial Global Planning (GPGP) [3] uses Partial Global
Plan (PGP) mechanism to ensure consistency between the plans of multiple agents. Agents exchange parts of their plans. An agent stores its knowledge of other agents’ plans in its PGP, with which it can find possible positive and negative effects. In work of [5], a goal is divided into subgoals. Agents solve their local goals, and merge them into a global goal. Since the subgoals are solved in parallel, it consumes less planning time than centralized planning.

Our approach is different from most existing work in two facets. First, most existing work assumes a problem can be decomposed, solved independently, and synthesized. Agents coordinate to resolve conflicts between sub-solutions. In our approach problem can not be decomposed and agents cooperate to make a single plan from scratch. Second, our approach assumes the agents require to keep some knowledge as privacy during cooperation, which is seldom addressed in existing work.

The work in [10] addressed the problem similar to ours, where self-interested agents need to achieve a goal. That approach uses auction mechanism. The original problem is put up at auction. If an agent can achieve a part of the problem, it bids to take over the problem, and put the unachievable part up to the auction as a subproblem. This approach also relies on knowledge of how to decompose a problem, which is difficult to obtain in real world. Our approach does not rely on it.

5 Conclusions

In this paper we have proposed an approach to distributed composition of Web services by multiple agents that have separate sets of services available to them. The agents are willing to contribute their services to make a single plan collectively but do not want other agents to know about their services. We present an algorithm which allows this kind of cooperation, as a distributed extension of a reputed centralized planning paradigm, i.e. graph based planning + CSP. In our algorithm, distributed planning graph (dis-graph) is constructed and stored by multiples agents. Agents exchange the states but keep the services information private. The fragments of the dis-graph are compiled by the agents independently into DisCSP, and then agents participate in distributed search for a solution to the DisCSP. Hence there is no need of centralized coordinator through the whole process of planning.

References