2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology

A Graph-based Multi-agent Planning Algorithm with QoS Guarantees

Jian Feng Zhang, Xuan Thang Nguyen and Ryszard Kowalczyk
Centre for Information Technology Research
Swinburne University of Technology
Melbourne VIC 3122, Australia
{jfzhang, xnguyen, rkowalczyk}@ict.swin.edu.au

Abstract

Many existing planning approaches assume the existence of a centralized planner that has complete information of its planning problem. However, with the increasing popularity of distributed paradigm today, a planning problem may span across the boundaries of different organizations. Consequently, such a problem is difficult to be managed by any single organization. In this paper, we propose a new graph based approach for distributed planning without a centralized planner. Our approach employs Distributed Constraint Satisfaction (DisCSP) and Graph planning techniques. It encompasses both of functional and non-functional plannings.

1. Introduction

Planning is an important activity in many real life applications such as resource allocation or travel arrangement [6]. The planning activity can be carried out centrally or distributedly by a number of participants [2]. In the last ten years, distributed environments and peer-to-peer networking have been emerging as a popular paradigm. Consequently, planning in distributed environments has become one of major interests of AI and MAS community.

Despite that many active research have been carried out on distributed planning [3] [5] [7], they have the following two limitations. Firstly, most of existing work assume that the planning are carried out by a set of agents who have complete knowledge of the environment. Secondly, they often consider the functional and non-functional requirements separately. The assumption of complete information does not hold for dynamic and open environment such as the Internet where unrelated organizations (i.e. agents) can collaborate to do business and form plans; nevertheless hardly any single organization can possess the knowledge of every other organization.The second limitation is rooted from the difference in encoding of functional and non-functional constraints. In non functional constraints, the variables are QoS parameters [10] whereas they are activities or services [11] in functional constraints. Combining these two is not very natural and hence functional planning and QoS planning are often treated as different but related problems.

Previously we have addressed the first limitation by proposing a new method for distributed multi-agent planning where the agents have local knowledge of the environment and incomplete information of other agents [11]. In this paper, we will show how QoS constraints can be effectively incorporated into the method and hence address the second limitation. With this incorporation, we solve both QoS and functional constraints in parallel at the same time and hence achieve more efficiency as compared to dealing with them sequentially.

The paper is organized as follows. Section 2 describes related work, followed by a description of our Dis-graph Planning with QoS guarantee in section 3. And we present the conclusion in Section 4.

2 Related work

Existing work on functional planning and non-functional planning can be found at [8, 1, 4, 10]. Most of those work focuses on centralized planning techniques. Recent distributed planning approaches [5, 9, 2] are in general influenced by and extended from those centralized techniques. Many centralized techniques and popular planners for functional planning, such as [1] [8], employ *Planning graph* that is a special graph based representation for a plan.

Current state of the art of distributed planning for functional composition can be found in [5, 9, 2, 3, 7]. In those work, distributed planning is in fact a process that combines individual planning and coordination [2]. Individual agents are acquainted with their own goals and then produce their plans for their goals, except that before or after their individual planning, they consider other agents' plans to avoid potential conflicts.

0-7695-3027-3/07 $25.00 © 2007 IEEE
DOI 10.1109/IAT.2007.87

It is important to note that the above approaches are limited to problems that can be decomposed in the way they assume the existence of some problem distributor agent. This agent knows or has frequently updated information about other agents, including their capabilities and surrounding environments, in order to carry out the decomposition task. Those approaches cannot scale up well for larger environments where an agent may only have partial, obsolete, or even imprecise information about other agents and those agents' local environments. Consequently, an effective distribution of a plan based on agents' capabilities is difficult.

To address the above limitation, we have previously explored DisCSP as an alternative for centralized CSP in solving planning problems in a distributed large scale network where decomposition of a problem is difficult and impractical [11]. However the work is limited to functional planning only. Similarly, we have carried out work [10] on QoS guarantees using DisCSP techniques, however those works cannot be extended easily into functional composition.

3 Approach

We use graph planning for our approach. Generally, graph based planning consists of two interleaved phases – extending planning graph, and searching for valid plans. However, instead of using a centralized planner, these two phases are carried out distributedly in our approach. In the first phase, agents collaborate to build and expand a planning graph. In the second phase, a DisCSP algorithm is used by the agents to search for a valid plan. DisCSP has been widely viewed as a powerful paradigm for solving combinatorial problems arising in distributed, multi-agent environments.

3.1 Distributed Graph Planning

In this section we present a brief review of Distributed Graph Planning (Dis-graph Planning). Detailed description can be found in[11].

Dis-graph planning is a distributed extension of centralized Graph-based planning paradigm [4]. It enables agents to generate a plan collectively in a distributed manner. For distributed planning, we devise 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 are distributed among agents. As shown in Fig 1, each agent keeps a number of *block*s which consist of consecutive levels of states and services, and the blocks link together to form a complete graph.

A dis-graph is compiled to a DisCSP to find a feasible plan. The DisCSP algorithm we use is extended ABT with

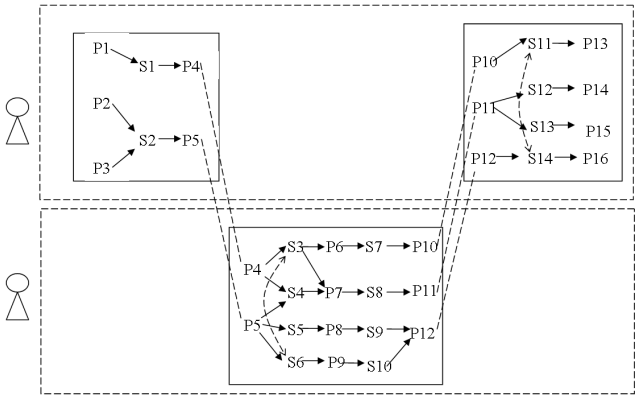

Figure 1. Distributed Planning Graph

multi-variables[11]. Basically, the way of compiling Dis-Graph to DisCSP is similar to that of compiling planning graph to CSP as shown in [4], except that shared constraints are involved to represent the links between the blocks, and a number of dummy values (e.g TRUEOP and NULL) are introduced to facilitate the shared constraints [11]. In principle, agents perform the following activities for the disgraph-to-DisCSP mapping task:

1. Provide each state 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 a variable for each state.

3. Generate constraints to represent the links between the blocks, and the dependency and exclusion relation between states and services within blocks.

3.2 QoS Constraints

For disgraph-to-DisCSP mapping w.r.t non functional planning, we focus in this paper on three popular QoS parameters, namely time, cost and availability, and describe how to encode them in CSP for the purpose of distributed graph planning. The privacy protection is an important concern of distributed graph planning. Different constraints require different level of information sharing. Here we focus on the question: how far the individual agent's information can be kept private and what is necessarily to be shared for the constraints.

Time Constraint

When making plans for real world problems, the starting time, end time and duration of executing the services often need to be included into consideration. For example, given that the plan will start to execute at hour 0, the goal states might be required to be achieved before hour 72 (within 3 days), and the service "upload cargo" are executed only

during working hours.In the meanwhile, an service can only be executed after all its precondition states are achieved.

While the temporal constraint might be very simple or complicated varying with the domains and the application situations, here we present a CSP encoding that enables agents to obtain the time points at which the states are achieved and to specify the time points when the services are able to execute. Based on this encoding various constraints can be developed and evaluated, which are beyond the scope of this paper.

In order to present temporal constraints, we do the following steps:

1. For each state *st*, introduce a state time variable *st_time* to represent the time point when the state becomes true.

2. For each service *ser*, introduce an service start time variable *ser_start* to present the time point when the service starts to execute, and an service end time variable *ser_end* to present the time point when the service finishes execution.

3. For each service $ser \neq NOOP$, and suppose *ser* has execution duration value *ser-dur*, generate constraint $st = ser \rightarrow ser\_end = ser\_start + ser\_dur$

4. For each state *st*, if service *ser* has *st* as its effect, generate the following constraint:

- If *ser* = NOOP and *ser*'s precondition is *st'*, $st = ser \rightarrow st\_time = st'\_time$

  else $st{=}ser \rightarrow st\_time \geq ser\_end$

- For all precondition states $st_1..st_n$ of *ser*, generate a constraint $st = ser \rightarrow ser\_start \geq st_1\_time \wedge ser\_start \geq st_2\_time \wedge .... \wedge ser\_start \geq st_n\_time$

5. Initial state time: for each state *st* in the initial state set, $st\_time = 0$

6. Goal state time: for state *st* in the goal state set, generate constraints as customer requirement, e.g. $st\_time \leq 100$ and $st\_time \geq 50$.

It is straightforward to extend the goal time constraints to the constraints on the start time and end time of any service.

It is obvious that the calculation of the goal states time is a Critical Path Problem, so is the calculation of internal states time. Since the agents do not possess the complete planning graph, they are not able to identify the critical path during planning. The agent has to share with its neighbors the time of the states on the borders of planning graph blocks.

E.g, in Fig 1, the time of P4 and P5 are necessarily to be shared between Agent1 and Agent2, and the time of P6 and P7 do not need to be shared.

Cost Constraint

It is often required to make sure the total cost of the services in the plan does not exceed the budget. Similar constraints include those of total response time, total memory usage, etc.

To encode cost constraints in CSP, the basic idea is to introduce a service cost variable for each service in the planning graph to indicate the cost incurred by the service. A variable *total_cost_agt* is introduced for each agent to indicate the sum of the service cost variables of the services in the agent's planning graph blocks. Agents exchange their *total_cost_agt* so that the sum of the costs from all the agents is able to be calculated and evaluated against the budget constraint.

While the encoding of cost constraint is similar to the time constraint, except that the "sum" operator is used instead of relational operators, the allocation of the constraints is much different. In ABT, the higher priority agent sends values to lower priority, and the lower one can not send value to the higher one. Hence we assign the lowest priority agent as the evaluator of global constraints.

We can implement it in two ways. In the first way, we introduce an additional agent working as a representative of the customer. It has the priority lower than all the planner agents and evaluates the values that are related to global constraints. In the second way, we can assign the planner with lowest priority to evaluate the global constraints. The advantage of the first way is that the agents have clearly defined roles. On one hand, the planner agents propose plans without revealing their service information. On the other hand, the customer representative agent checks the possible plans without reveal the amount it expects to pay, which could be critical business secrets in many situations. The advantage of the second way is that it does not need additional agent, and has the same CSP structure as the planning without global constraints.

Note that it is straightforward to switch between the two ways, we take the second way as example.Each agent *agt* perform the following tasks:

1. Introduce an service cost variable *ser_cost* for each service *ser*. The possible values of the service cost variable are 0 and the cost of *ser*.

2. Introduce a local total cost variable $total\_cost_{ser}$.

3. Generate a constraint indicating that if service *ser* is assigned to a state variable *state*, the cost of *act* is assigned to the corresponding service cost variable *ser_cost*, otherwise 0 is assigned to *ser_cost*.

4. Generate a constraint indicating that the value of $total\_cost_{agt}$ is the sum of the values of the service cost variables, i.e $total\_cost_{agt}{=}\sum^{all\ ser} ser\_cost$.

5. If *agt* is the lowest priority agent, generate a constraint indicating that the sum of the $total\_cost_{agt}$ of all the agents is no higher than the amount specified by the customer, i.e $\sum^{all\ agt} total\_cost_{agt} \leq given\ amount$.

The evaluation of the cost does not care about the "path" or "structure" of the planning graph. So agents can avoid revealing the cost of individual services, and only the variable of the sum value is necessarily shared.

Availability Constraint

In this paper we define the availability of a single service as the probability that the service is available for execution during the period when the plan is expected to be executed. We use a value $\in [0..1]$ to indicate the availability. The value can be obtained, e.g. by calculating the percentage of the time the service is available in a specific history period. The calculation of the availability of the plan can be of various forms. We present a simple calculation, where the availabilities of services are deemed as independent events, and the overall availability of a plan is the product of the availabilities of individual services.

Similar to the calculation of cost constraint, the calculation of availability does not care about the structure of planning graph. So the agents can avoid revealing the availability of individual services by calculating the overall availabilities of their planning graph blocks locally, and sharing the overall availabilities with the global constraint evaluator agent.

Each agent does the following tasks:

1. Introduce an service availability variable *ser_avi* for each service *ser*. The possible values of the service availability variable are 1 and the availability value of *ser*.

2. Introduce a local overall availability variable $overall\_avi_{agt}$.

3. Generate a constraint indicating that if service *ser* is assigned to a state variable *st*, the availability value of *ser* is assigned to the corresponding service availability variable *ser_avi*, otherwise 1 is assigned to *ser_avi*.

4. Generate a constraint indicating that the value of $overall\_avi_{agt}$ is the product of the service availability variables, i.e $overall\_avi_{agt} = \prod^{all\ ser} ser\_avi$.

5. If *agt* is the lowest priority agent, generate a constraint indicating that the product of the $overall\_avi_{agt}$ of all the agents is no less than the availability required by the customer, i.e

$\prod^{all\ agt} overall\_avi_{agt} \leq$ *required availability* .

4 Conclusions

In this paper we have presented an approach to distributed multi-agent planning where the agents have separate sets of services and intend to fulfill a task collaboratively. Our approach combines techniques from Distributed Constraint Satisfaction and Graph planning fields. It has following features:

-It is a decentralized planning algorithm, solving problems without collecting the knowledge from individual agents.

-It limits the knowledge shared during planning and allows agents to keep their privacy as much as possible.

-It removes the reliance on problem decomposition.

-It considers functional constraints and QoS constraints in the same time.

Our approach combines techniques from Distributed Constraint Satisfaction and Graph planning fields. It shows the Planning Graph + CSP paradigm may succeed in distributed environment as it did in centralized centralized planning environment.

In future work, extensive experiments will be carried out to study the scalability and performance of the approach.

References

[1] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, 1995.

[2] M. de Weerdt, A. ter Mors, and C. Witteveen. Multi-agent planning: An introduction to planning and coordination. In *Handouts of the European Agent Summer School*, pages 1–32, 2005.

[3] K. S. Decker and V. R. Lesser. Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(2):319–346, 1992.

[4] M. B. Do and S. Kambhampati. Solving planning-graph by compiling it into CSP. In *Artificial Intelligence Planning Systems*, pages 82–91, 2000.

[5] E. H. Durfee. Distributed problem solving and planning. In *Multiagent systems: a modern approach to distributed artificial intelligence*, pages 121–164. MIT Press, Cambridge, MA, USA, 1999.

[6] E. H. Durfee and V. R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, - 1991.

[7] E. Ephrati and J. S. Rosenschein. Multi–agent planning as the process of merging distributed sub–plans. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 115–129, 1993.

[8] A. Gerevini and I. Serina. Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, 2000.

[9] A. D. Mali and S. Kambhampati. Distributed planning. In *The Encyclopedia of Distributed Computing, Kluwer Academic Publishers*, 2003.

[10] X. T. Nguyen and R. Kowalczyk. An agent based qos conflict mediation framework for web services compositions. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-06)*, pages 522–528, USA, 2006. IEEE Computer Society.

[11] J. F. Zhang, X. T. Nguyen, and R. Kowalczyk. Graph based multi-agent replanning algorithm. In *Proceedings of The Sixth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2007)*, pages 793–800, 2007.