Graph-based Multi-agent Replanning Algorithm

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

ABSTRACT

The paper presents a new approach for multi-agent replanning based on Distributed Constraint Satisfaction (DisCSP) and Graph planning techniques. In this approach, a new distributed refinement strategy is proposed to construct a graph plan for fixing errors occurred during the plan execution. The strategy employs an "max-branching" heuristic that can reduce the final graph plan size and allow faster completion time for the graph construction. The graph plan is then compiled into a DisCSP problem and solved using a multi-variable version of the Asynchronous Backtracking Algorithm. The approach is demonstrated with experiments which show that distributed planning graph and CSP can practically solve the replanning problems in a multi-agent environment.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Plan execution, formation, and generation; I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Algorithms

Keywords

multi-agent, planning, replanning

1. INTRODUCTION

Many real life problems require planning in which a number of participants (i.e. agents) perform tasks collaboratively in order to achieve a goal [5]. With the increasing popularity of distributed network environments and peer-to-peer paradigm, distributed planning has captured the interests of AI and MAS community and been investigated in a number of work [5]. Examples of distributed planning can be found in logistics where an international transport company has to distribute transportation tasks among its subdivisions and subcontractors in different geographical areas, or a network of coordinated sensors to monitor a large area for vehicle movements [11].

Durfee[10] classifies distributed planning into three main types: Centralized Planning for Distributed Plans, Distributed Planning for Centralized Plans, and Distributed Planning for Distributed Plans. This classification is based on the distribution of the plans themselves or of the planning mechanism. Centralized Planning for Distributed Plans refers to a set of distributed plans that are formulated using a centralized coordinating agent, e.g [17]. Distributed Planning for Centralized Plan(s) refers to a distributed solving process carried out by cooperative planning agents to form a complex plan. Distributed Planning for Distributed Plans are popular in MultiAgent literature, e.g [21] [6]. For this type of planning, both the plans and the planning mechanism are distributed.

In a dynamic environment, execution errors may happen to any plan due to uncertainty and failure of individual actions. Therefore, an indispensable part of a planning system is the capability of replanning. The importance of replanning in executing a plan has been discussed in [14] and [7].

Whilst a number of approaches have been proposed for distributed planning and replanning [6] [12], most of them assume that the planning and replanning are carried out by a set of agents who have complete knowledge of the environment. This assumption 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. However, hardly any single organization can possess the knowledge of every other organization in order to solve the whole planning problem effectively.

In this paper we address this limitation of existing approaches by proposing a new method for distributed multi-agent replanning where the agents have local knowledge of the environment and incomplete information of the other agents. Major contributions of this work are an expansion based replanning strategy and a novel Distributed-graph Planning algorithm (Dis-graph Planning) to fulfill replanning operation, to practically solve multi-agent replanning problem.

The paper is organized as follows. Section 2 describes related work, followed by preliminaries in Section 3. Section 4 gives a detailed description of our approach, including an overview, a description of the expansion-based replanning strategy, a description of Dis-graph Planning algorithm. Section 5 presents a scenario and our prototype. And we present the conclusion in Section 7.

2. RELATED WORK

Replanning can be considered as a specific case of planning[19]. Distributed planning/replanning approaches [10, 15, 5] are in general influenced by and extended from existing centralized planning/replanning techniques. Many centralized planning techniques and popular planners such as[1][13] employ *Planning graphs* that is a special graph based representation for a plan. In general, a planning graph is constructed by a forward chaining search. Possible candidate action sequences of the graph are then searched backward to form a final plan [1]. In [8], adaptation of CSP algorithms has been proposed for this backward search.

Now a typical approach used for replanning is to anticipate possible situations in the planning phase and accommodate various situations occurring in execution phase. In the work of Drabble et al [9], preassembled repair plans are prepared in advance, and are invoked to deal with specific exceptions during execution. This class of approaches may work well in relatively static and predictable environment. In more dynamic and uncertain environments where it is hard to anticipate possible exceptions, *monitoring-and-replanning* may work better. The basic idea is to generate a (partially) new plan in case when one or more actions have problems during execution [14]. For example, GPG [13] is capable of replanning based on graph planning [2]. Replan [3] replans by analyzing an abstraction hierarchy of actions.

Current state of the art of distributed planning and replanning can be found in [10, 15, 5, 6, 12, 12, 21], These approaches advocate some or all of the following 6 phases [10] [5] of planning/replanning:

1. decomposing problems to subproblem;
2. allocating subproblems to agents;
3. defining constraints for individual agents to prevent them from producing conflicting plans;
4. Solving planning problem at each individual agent;
5. coordinating the individual plans;
6. executing the individual plans and integrating their results.

Under those phases, distributed planning is in fact a process that combines planning and coordination [5], i.e. coordinated individual planning. Similar to centralized planning, individual agents are acquainted with their own goals and then produce their plans to achieve their goals, except that before or after their individual planning, they consider other agents' plans to avoid potential conflicts.

As can be seen, 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 about other agents and those agents' local environments.

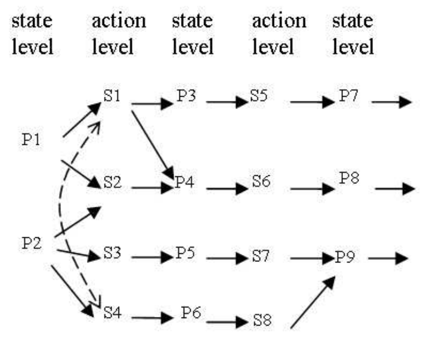

Figure 1: Planning Graph

Domains:P7:{S5, null}, P8:{S6, null}, P9:{S7, S8, null},
P3:{S1, null}, P4:{S1, S2, null}……
Constraints (mutex):
P3 = S1 => P6 $\neq$ S4 ……
Constraints (activity):
P7 = S5 => P3 $\neq$ null, P8 = S6 => P4 $\neq$ null
P9 = S7 => P5 $\neq$ null, P9 = S8 => P6 $\neq$ null
……

Figure 2: CSP translated from planning graph

Consequently, an effective distribution of a plan based on agents' capabilities is difficult.

In a relation to existing work on planning and replanning, we explore DisCSP as an alternative for centralized CSP in solving re-planning problems in a distributed large scale network where decomposition of a problem is difficult and impractical. To our knowledge, there has not been any work on using DisCSP techniques for solving the planning problem so far.

3. PRELIMINARIES

Distributed (Re-)Planning: Given:

- a set of actions $\boldsymbol{A}$
- a set of states $\boldsymbol{S}$
- a set of agents $\boldsymbol{AG}$

For each agent $Ag \in \boldsymbol{AG}$, it has access to

- a subset of actions $\boldsymbol{A'}$ where $\boldsymbol{A'} \subset \boldsymbol{A}$
- a subset of states $\boldsymbol{S'}$ where $\boldsymbol{S'} \subset \boldsymbol{S}$

A distributed (re-)planning is the construction of a sequence of actions in $\boldsymbol{A}$ such that executing the actions by $\boldsymbol{AG}$ can move the state of the real world from some initial state to a final state in which certain goals can be achieved.

Graph based planning Graph based planning utilizes *planning graph* [1] as a heuristic to constrain the search for plans. This technique is popular in AI planning research due to its good performance [1]. GRAPHPLAN [1] is the first planning algorithm using this technique. Generally, graph based planning consists of two interleaved phases – extending planning graph, and searching for valid plans. A planning graph is a directed leveled graph, as shown in Fig 1. It consists of two kinds of alternating levels, state levels and

action levels. The first level consists of initial states. The n+1 level consists of actions whose preconditions are present in the n level. The n+2 level consists of the states appearing in the n level and the states brought by the actions in the n+1 level as their effects. In this way the graph is extended by state levels and action levels alternatively. During the construction, mutex relations are also identified. Two actions in the same level are mutex if their preconditions and effects are inconsistent. Two states in the same level are mutex if all the actions supporting the first state are mutex with all the actions supporting the second state. [8]
When the graph reaches a level where all goal states are present, the algorithm starts to search for a valid plan. A valid plan is a subgraph where there are no mutex actions in each level. If no valid plan is found, the planning graph is extended again. Search for the valid plan can be carried out by backward search as in GRAPHPLAN, or by translating the graph to a CSP and solving it, as in [8].

CSP and DisCSP Compiling a planning graph into CSP brings several benefits to planning [16]. It enables planning problems to be solved with available standard CSP tools, and can exploit the latest advancement of CSP tools and techniques. Another benefit is the potential to integrate planning problem with other problems that can be solved with CSP techniques, such like scheduling, resource allocation and QoS (Quality of Service) management problems. It can be beneficial especially in the context of complex applications, e.g. Web service composition. Distributed Constraint Satisfaction (DisCSP) is a variation of CSP in which variables and constraints are distributed among multiple agents [23]. A number of distributed search algorithms, such as ABT (Asynchronous Backtracking), Weak-Commitment ABT and AAS(Asynchronous Aggregate Search), can be used to solve a DisCSP [23] [18]. Solving a DisCSP does not require all information to be gathered to a centralized solver. It allows cooperation among agents concerned about their privacy.

4. APPROACH

Our approach consists of three major parts: an expansion-based unrefinement strategy, a method to construct planning graph in a distributed manner, and a mechanism to encode the distributed planning graph into a DisCSP to find plans.

In general, replanning process fixes a plan with two basic operations, unrefinement and refinement [19]. The unrefinement operation removes some actions that may obstacle the reachability to goal. The refinement operation adds actions that improve the reachability to goal.

We use an expansion-based unrefinement strategy in our approach. It is an extension of the unrefinement strategies presented in the existing work [13] [22][19][20]. In those work unrefinement is started from actions depending on initial states or producing goals, and the plan is shrunk from the outside in. The reason for that kind of strategy is as remarked by Kambhampati [14], that previous research on replanning stem from the research on plan reuse: reuse the current plan to solve a new problem that has new initial and goal states, hence the replanning systems often assume the failure of a plan comes from altered initial or goal states. However, in our work the objective of replanning is to solve execution failure, which may happen in any action of the plan. So in our work expansion-based refinement starts from any actions that fail.

Our Dis-graph planning algorithm is built upon existing research on graph based planning and CSP. Previous work shows that search in a planning graph can be formulated as solving a constraint satisfaction problem [8]. We extend the idea of planning graph to distributed planning graph, and combine it with DisCSP to generate plans in distributed manner.

4.1 Expansion-based Replanning Strategy

In this section we present our expansion-based replanning strategy. If an action in the plan fails, the expansion-based replanning starts by considering the failed action as a replanning area. The unrefinement operation removes the action(s) in replanning area in order to allow the refinement operation to generate an equivalent sub-plan that covers the replanning area. The refinement operation is in fact a distributed planning process, which will be described in the next section. If the refinement operation does not find a equivalent subplan, the unrefinement operation expands the replanning area around the failed action, i.e. removes more actions adjacent to the failed one, to allow the refinement operation to search for an equivalent subplan for the enlarged area. In this way, the unrefinement operation explores larger and larger areas to allow the refinement operation to search for sub-plans.

Combining with the refinement (planning) operation, we can describe the expansion based strategy as follows:

1. Given the initial planning problem $\boldsymbol{Pb}$=($\boldsymbol{I}$, $\boldsymbol{G}$) where $\boldsymbol{I}$ is a set of initial states and $\boldsymbol{G}$ a set of goal states, a plan $\boldsymbol{Pl}$ is a network of actions that lead from $\boldsymbol{I}$ to $\boldsymbol{G}$.

2. If an action $\boldsymbol{a}$ in $\boldsymbol{Pl}$ fails, we define a replanning area $\boldsymbol{RA} = \{\boldsymbol{a}\}$.

3. We see $\boldsymbol{RA}$ as a small plan, and construct a planning problem $\boldsymbol{Pb'}$=($\boldsymbol{I'}$, $\boldsymbol{G'}$) where $\boldsymbol{I'}$ is the set of states used by $\boldsymbol{RA}$ as preconditions and $\boldsymbol{G'}$ is the set of states produced by $\boldsymbol{RA}$ as effects.

4. We search for a plan for $\boldsymbol{Pb'}$. If a plan $\boldsymbol{Pl'}$ is found, $\boldsymbol{Pl'}$ replaces $\boldsymbol{RA}$ in $\boldsymbol{Pl}$ and goto 5; if no plan is found, we expand $\boldsymbol{RA}$ so that it includes more actions in $\boldsymbol{Pl}$ that are adjacent to it $\boldsymbol{RA} \leftarrow \boldsymbol{RA} \cup \{a | a \textit{ is adjacent to } \boldsymbol{RA}\}$, and goto 3.

5. Resume the execution of $\boldsymbol{Pl}$

The major motivation of the expansion-based strategy is to save replanning cost. We consider two types of cost involved in replanning. The first is the computing effort consumed by the refinement operation, i.e planning. The less is the resource (time, space) consumed by refinement, the better is the performance of the replanning approach. The second is the cost incurred by the change to the current plan. The less is the change to the current plan, the less is the business cost caused by replanning. In the worst case replanning will terminate when the replanning area has been expanded as large as the original plan. However, in most cases, replanning is expected to succeed in a smaller area given that sufficient actions are available from the agents.

4.2 Distributed Graph Planning

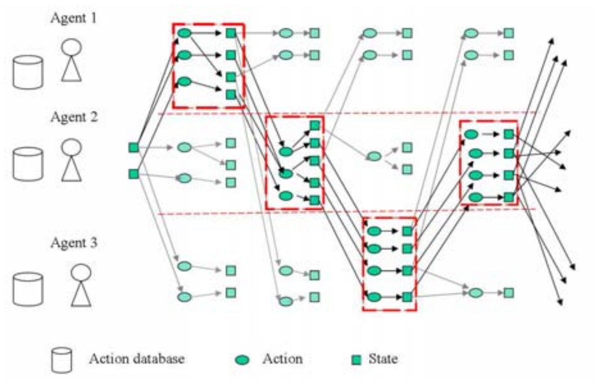

Figure 3: Constructing Distributed Planning Graph (with *Max branching* strategy)

In this section we present our Distributed Graph Planning (Dis-graph Planning) algorithm that serves for the refinement operation in replanning. Dis-graph planning enables agents to generate a plan collectively in a distributed manner. So practically, Dis-graph planning enables fully distributed planning, hence also provides the support to fully distributed replanning. Dis-graph planning is a distributed extension of centralized Graph-based planning paradigm. For distributed planning, we devise a distributed planning graph (dis-graph) based on the idea of a planning graph. Similar to a planning graph, a dis-graph consists of alternating levels of states and actions, 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 into a DisCSP (instead of a CSP) and solve it in a distributed way.

A distributed planning graph (dis-graph) under construction is shown in Fig 3, where 3 agents are involved in construction. Agent 1 contributes 1 block to the graph, agent 2 contributes 2 blocks and agent 3 contributes 1 block.

Block A block is a series of consecutive state and action levels kept by one agent.

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:  planning 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. Since each agent has access to only a subset ***A'*** of available actions ***A***, it constructs new levels with actions from the subset ***A'***. Each time an agent executes line 2, it constructs one action 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 on the next level (line 10). *reach-all-goals* checks whether all the goal states are present. If all the goal states appear, *search-for-valid-plan* (line 5) is launched, which includes compiling the planning graph to DisCSP, and solving DisCSP. If a plan is found, the algorithm terminates with success. If no plan is found, the agents *decide-next-state-level* to prepare a state level as foundation for further extending, and go to line 2.

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

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 the actions 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 acceptable for all the agents.

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 action level with actions supported by pre-
    vious state level
3:  construct the state level with the states produced by the
    previous action level
4:  publicize the latest state level to other agents via mes-
    sage, and wait for the other agents' messages
5:  after receive messages from all agents, checks whether
    reach-all-goals
6:  if 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.*

Max branching strategy This strategy chooses the agent that has the largest number of states in the 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 *Max branching* strategy has less communication overhead than *Merge all*, but *Merge all* strategy has chances to reach a plan earlier. *Max branching* strategy does not discard any possibility of finding a plan, i.e. if there is a plan that can be reached by *Merge all* strategy, it can also be reached by *Max branching* strategy (although it is probably not the first plan being reached), and vice versa. The reason resides in the fact that the states present in one state level of the planning graph will be carried to the following state level, so every certain state level accumulates all the states that appeared in previous state levels. Suppose an action *a* belonging to an agent *agt*, with the precondition *c1* and the effect *c2*. If in a certain state level *c1* is present, when constructing the next levels *agt* will select *a* in its action level and put *c2* in its next state level. Therefore:

- In the case of *Merge all* strategy, action *a* and state *c2* are included in planning graph as well as *c1*.
- In the case of *Max branching* strategy, if *agt* is the winner, action *a* and state *c2* are included in planning graph as well as *c1*. If *agt* is not the winner, *agt*'s next action level, which includes *a*, and its next state level, which includes *c2*, are discarded. However, state *c1* will still appear in the following state levels of the planning graph, so *a* and *c2* can be rediscovered later.

4.3 DisCSP to find 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. Fromm 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.

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 *Max branching* strategy. Two of agent1's blocks and one of agent2's blocks are shown in the figure. In principle, each agent performs the following tasks: 1) Provide each state in its fragment with a unique CSP variable number, and each action with a unique CSP value number. The mapping from states/actions 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 action belonging to agent2, without knowing the name* to it. In order to indicate the *some action without knowing the name*, we introduce a dummy action *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 in the same way 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 an action (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): $P1 : \{TRUEOP, null\}$,

Constraints(private):

(activity) $P5 = S2 \Rightarrow P2 \neq null \wedge P3 \neq null$,

$P4 = S1 \Rightarrow P1 \neq null, \ldots$

(Mutex) $P13 = S11 \Rightarrow P16 \neq S14$

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

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

$P4(agent2) = TRUEOP \Rightarrow P4(agent1) \neq null$,

$P5(agent2) = TRUEOP \Rightarrow P5(agent1) \neq null, \ldots$

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 CSP algorithms/implementations, e.g. JSolver [4]. For the second goal, the agent participate in a distributed search, e.g. ABT (Asynchronous Backtracking) [23]. 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 [23]. 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, and the constraint is directed from higher priority agent to lower priority agent. In our problem there are multiple constraints between two agents, say A and B, and some of the constraints are from A to B while the others are from B to A.

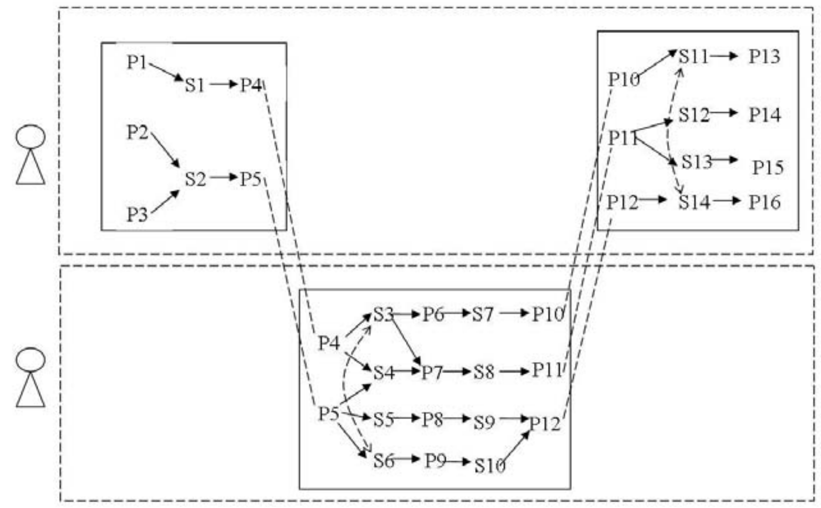

Figure 4: Compile Distributed Planning Graph

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

$null < TRUEOP < actions$

and transforming the constraints to $\leq$ and $\geq$ relationships. For example, the constraints on P12 is transformed to

P12 in agent1 $\leq$ P12 in agent2

Similarly, the constraint on P4 is transformed to

P4 in agent1 $\geq$ P4 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 actions. 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.

5. SCENARIO AND PROTOTYPE

We present a scenario in this section to illustrate how expansion-based replanning works.The task requested by the customer is to transport goods from the factory A in city A to factory B in city B. Three companies accept the request. One company performs local transportation inside city A, one company performs local transportation inside city B, and the last company performs inter-city transportation between city A and B. The map (Fig 5) shows the possible ways from factory A to factory B. The dash lines mark three areas that the three companies are responsible for respectively.

In our approach, planning from draft is treated as a special case of replanning which involves refinement but not unrefinement. Given initial state "atFactoryA" and goal state "atFactoryB", the three companies (agents) generate the initial plan as follows:

$t1 \rightarrow t2 \rightarrow t3 \rightarrow t4 \rightarrow t5 \rightarrow t6$

t1: Factory A to Warehouse A2

t2: Warehouse A2 to Airport A

t3: Airport A to Airport B

t4: Airport B to Warehouse B2

t5: Warehouse B2 to Warehouse B1

t6: Warehouse B1 to Factory B.

During the execution, t3 (from Airport A to Airport B) fails. Figure 6 shows the replanning process, in which t3,t2,t4 and t5 are removed sequentially until a subplan (t11, t12, t13 and t14) is found to fix the plan.

We implemented the algorithm for illustration and experiment within a prototype. Figure7 shows its screenshot with a sample plan generated by the system. The actions and task requests are described in a simplified subset of PDDL (Planning Domain Definition Language), and the agents are running on a multi-agent simulation system developed by our group IAMAS of Swinburne (http://ciamas.it.swin.edu.au/software/simulator-doc). The plan generated in the scenario described above is a sequence of actions, however, by the nature of graph based planning, the plans generated by our approach are partially ordered in general. In scenarios involving more actions the plans can accommodate concurrent actions.

Our observation in experiment explains that the expansion based strategy saves the cost of refinement and changes to the current plan. We observed that the number of added actions and the number of removed actions are in an approximate direct ratio:the number of added actions is no

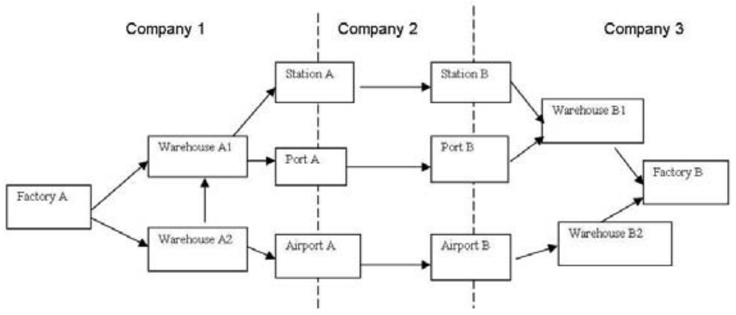

Figure 5: Map

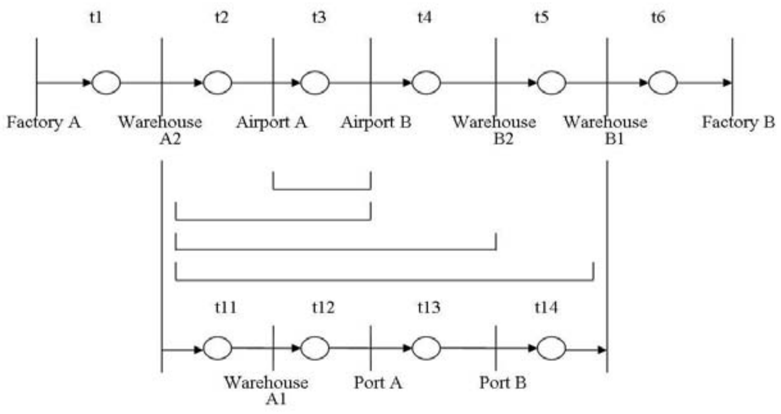

Figure 6: Replanning after t3 fails

less than, and will not be too much larger than the number of removed actions. Since the number of removed actions is decided by the size of replanning area, if the replanning succeeds in a smaller replanning area, the size of the planning problem solved by refinement operation (hence the computation resource consumed by refinement) is smaller, and the change to the current plan (i.e. the number of removed actions and added actions) is less. Due to space limits the detailed results of the experiments will be presented in a separate paper.

6. CONCLUSIONS

In this paper we have presented an approach to distributed multi-agent replanning where the agents have separate sets of actions and are willing to contribute their actions to fulfill a task cooperatively. Our approach has following features:

1. It is a decentralized replanning algorithm. It solves replanning problems without collecting the knowledge necessary for planning (i.e. actions) from individual agents. Agents are of equal authority during replanning, without centralized coordinator.

2. It limits the knowledge shared during replanning and allows agents to keep their privacy as much as possible. Specifically, individual agents do not need to reveal their actions to other agents.

3. It removes the reliance on problem decomposition. Existing multi-agent planning/replanning algorithms rely on the assumption that the given problem can be decomposed properly into sub-problems that can be solved by individual agents. This assumption is often too strong for real life problems.

4. It saves replanning efforts and limits changes to the executing plan with an expansion based strategy.

5. Since the algorithm is based on planning graph, the result plan is a partially ordered plan, i.e. it allows some actions executed concurrently.

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 we intend to explore the following issues. We will investigate different expansion strategies and study their performances in relation to the factors such as number of actions and number of participating agents. We will look

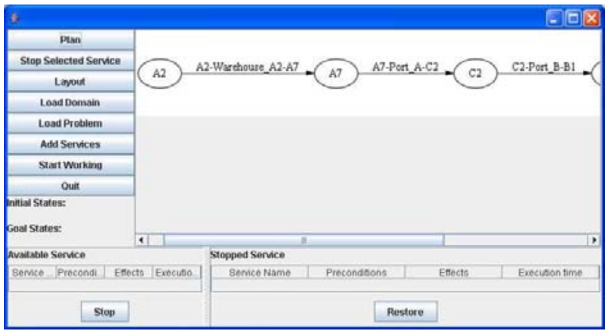

Figure 7: A screenshot of the implemented prototype

into the situations where there are multiple simultaneous failures. Extensive experiments will be carried out to study the scalability and performance of the approach.

7. 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] A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artif. Intell.*, 90(1-2):281–300, 1997.

[3] G. Boella and R. Damiano. A replanning algorithm for a reactive agent architecture. *j-LECT-NOTES-COMP-SCI*, 2443:183–192, 2002.

[4] H. Chun. Constraint programming in java with jsolver. In *Proceedings of the First International Conference and Exhibition on the Practical Application of Constraint Technologies and Logic Programming. London.*, 1999.

[5] 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.

[6] 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.

[7] M. Desjardins, E. Durfee, C. Ortiz, and M. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 4, 1999.

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

[9] B. Drabble, J. Dalton, and A. Tate. Repairing plans on the fly. In *Proc. of the NASA Workshop on Planning and Scheduling for Space*, 1997.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] W. C. a. S. Kambhampati. Replanning: A new perspective. In *Poster Program, ICAPS 2005, Monterey, California, U.S.A.*, 2005.

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

[16] D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[17] F. Pecora and A. Cesta. Planning and Scheduling Ingredients for a Multi-Agent System. In *Proceedings of UK PLANSIG02 Workshop, Delft, The Netherlands*, 2002.

[18] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *AAAI/IAAI*, pages 917–922, 2000.

[19] R. van der Krogt and M. de Weerdt. The two faces of plan repair. In *Proceedings of the Sixteenth Belgium-Netherlands Conference on Artificial Intelligence (BNAIC-04)*, pages 147–154, 2004.

[20] R. van der Krogt and M. de Weerdt. Plan repair as an extension of planning. In *Proceedings of the International Conference on Planning and Scheduling (ICAPS-05)*, pages 161–170, 2005.

[21] R. van der Krogt and M. de Weerdt. Self-interested planning agents using plan repair. In *Proceedings of the ICAPS 2005 Workshop on Multiagent Planning and Scheduling*, pages 36–44, 2005.

[22] R. P. van der Krogt and M. M. de Weerdt. The two faces of plan repair. In *Proceedings of the BNAIC (BNAIC-04)*, pages 147–154, 2004.

[23] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering*, 10(5):673–685, 1998.