Interaction Graphs for Planning Problem Decomposition

Mark Iwen*
EECS, University of Wisconsin, Milwaukee WI 53211
iwen2724@csd.uwm.edu

Amol Dattatraya Mali†
EECS, University of Wisconsin, Milwaukee WI 53211, Phone: 1-414-229-6762
mali@miller.cs.uwm.edu

Categories & Subject Descriptors: I. Computing Methodologies, I.2 Artificial Intelligence

General Terms: Algorithms, Design

Keywords

Distributed planning, Problem decomposition

1. INTRODUCTION

A brief survey of distributed planning techniques is provided in [Mali & Kambhampati 2002]. An important and perhaps the first step in solving a distributed planning problem is to decompose the given problem. Currently there are no general and automatic techniques to effectively decompose planning problems into subproblems with limited interactions. In this paper we introduce interaction graphs which can be used to achieve highly effective problem decomposition in many domains. An interaction graph allows splitting of the initial state of a planning problem besides goal and it allows splits into entirely independent subproblems under certain conditions. This eliminates the cost of resolving conflicts among plans of individual agents. In the remainder of the text IG will stand for "interaction graph".

As in most classical planners, we use STRIPS representation of actions. Predicates like *on(A,B)* can be considered as propositions by rewriting them as *onAB*. An action (e.g. move(A,B,C)) is a ground instance of an operator (e.g. move(x,y,z)). Capitalized letters in arguments of an action are specific objects and lowercase letters are variables. We denote an action by o_i. A planning problem is specified as $< I, G, O >$, where I denotes the completely specified initial state, G denotes the goal and O denotes the set of all actions in the domain. We assume that I contains only true propositions. O_i denotes the set of actions that agent i can use to synthesize its individual plan. Clearly, $O_i \subseteq O$. I_i and G_i denote, respectively, the initial world state and the conjunctive goal of the planning problem of agent i.

AAMAS'02, July 15-19, 2002, Bologna, Italy.

2. INTERACTION GRAPHS

We denote a graph by $< V, E >$ where V is the set of vertices in the graph and E is the set of edges. An IG is an undirected, simple bipartite graph. Decomposition based on this graph is obtained by detecting if it is disconnected and finding its connected components if it is disconnected. An undirected graph is connected if there is a path to reach every vertex from every other vertex.

An IG for the problem $< I, G, O >$ is constructed in the following manner: Vertices are created for each proposition true in the initial state and each proposition needed true in the goal. Two vertices v_i and v_j are connected only if the corresponding propositions have a common primary object such that the proposition in v_i belongs to the initial state and the proposition in v_j belongs to the goal. Vehicles, cities, airports, and other locations in transportation logistics are secondary objects for example. We assume that the planning domain contains both primary objects and secondary objects and that each proposition from the goal contains at least one primary object. In general, we assume that whether an object is primary or secondary is specified. However, in many domains such as blocks world, transportation logistics, rocket, meet-pass etc. objects have limited types (blocks, table, plane, truck, city, airport location, rocket etc.) and all predicates describing relations between objects are unary or binary. We have implemented a heuristic which correctly finds all secondary objects in such domains using the restricted nature of relationships in such domains. The heuristic will be discussed in the next section. We assume that the types of objects like block, package, truck, location etc. are given under a separate category in the problem specification and that they are not a part of the initial state. So, vertices for $package(P_1), package(P_2), Truck(T_1), Truck(T_2), Airport(Heathrow)$ etc. will not appear in an IG for a logistics problem. The reason for ignoring these from the IG is to keep the number of edges low. Propositions from I which contain only secondary objects are not represented in the IG. Two examples of IGs are given in Fig. 1 and Fig. 2. In figure 1 blocks are primary objects and the table is the secondary object. In figure 2 the packages are primary objects while the plane and locations are secondary. Note that the size of an IG depends only on I and G. It is not affected by O at all.

There is a polynomial time algorithm (pg.274,[Deo 1999]) for testing if a graph is disconnected and finding all of its connected components if it is disconnected. We have implemented this to find the components of an IG. Different planning problems corresponding to different components of

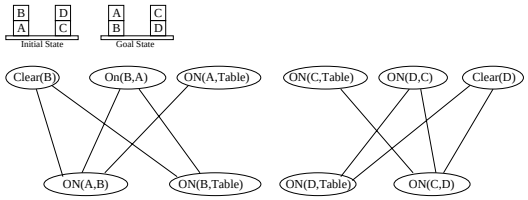

Figure 1: A disconnected IG with 2 components.

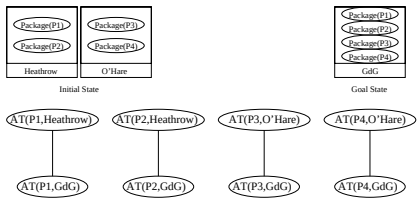

Figure 2: A disconnected IG with 4 components.

the IG (when it is disconnected) can be allocated to different agents. For plans of individual agents to be free of conflicts, whether there are shared resources must be considered. We consider a plane to be one kind of resource, a truck to be another kind of resource etc. We have the following guideline on the decomposition of a problem into p independent subproblems:

Guideline If an IG of a planning problem $< I, G, O >$ has p connected components and there are at least p resources (secondary objects) of each kind, then the planning problem can be split into p independent subproblems $< I_1, G_1, O_1 >$, $< I_2, G_2, O_2 >$, $< I_3, G_3, O_3 >$, ... $< I_p, G_p, O_p >$, such that $I = (I_1 \wedge I_2 \wedge ... \wedge I_p)$, $(G_1 \wedge G_2 \wedge G_3 \wedge ... \wedge G_p) = G$ and $O_1 \subset O, O_2 \subset O, ..., O_p \subset O$, if any one object of each resource type is enough for an agent i to generate its individual plan, irrespective of the state of the resource object in I_i.

Let us consider the example in Fig. 1. The IG yields the following problems $< I_1, G_1, O_1 >$ and $< I_2, G_2, O_2 >$. $I_1 = (clear(B) \wedge on(B, A) \wedge on(A, Table))$ and $G_1 = (on(A, B) \wedge on(B, Table))$. $I_2 = (clear(D) \wedge on(D, C) \wedge on(C, Table))$ and $G_2 = (on(C, D) \wedge on(D, Table))$. O_1 is set of all ground instances of $move(x, y, z)$, such that $x \neq Table, x \neq y, y \neq z, x \neq z, x, y, z \in \{A, B, Table\}$. O_2 is set of all ground instances of $move(x, y, z)$, such that $x \neq Table, x \neq y, y \neq z, x \neq z, x, y, z \in \{C, D, Table\}$. Note that an action involving primary objects of both agents like $move(A, B, D)$ need not be considered here.

The example in Fig. 2 is also instructive. There are four components in the IG but only one plane. Thus, via the aforementioned guideline, the subproblems corresponding to the different components aren't independent. If London had four planes they would be. Note that the notion of constructing an IG can be useful even if all agents' subproblems aren't totally independent or solvable. One can still construct an IG and form planning subproblems based on its components. These problems can be given to various agents and their plans (partial or otherwise) can be completed and merged. At the time of merging plans, a conflict resolver can introduce previously excluded actions to find a globally correct plan. This avoids loss of completeness. Algorithms for merging plans are reported in [Foulser et al 1992].

Secondary Object Detection:

We want to identify objects which are unlikely to cause conflicts between agents cooperating on a planning problem $< I, G, O >$. Furthermore, we want to do so quickly without having to consider the planning problem's action set. The general idea is that objects which may be related to many other objects at the same time are unlikely to cause conflicts between agents. For example, in the logistics domain many packages, trucks, etc. may be at a single location at once. Thus, it should cause few conflicts if all agents working on a distributed logistics problem have access to all locations. We have a generalized heuristic to automatically detect secondary objects, but here we will only consider binary predicates since they are most common.

We will say an object type (block, rocket, etc.) $T1$ is secondary with respect to another object type $T2$ in a binary predicate type P if there is a single object O_{T1} of type T_1 and two different objects Q_{T2}, R_{T2} of type $T2$ such that either both $P(O_{T1}, Q_{T2})$ and $P(O_{T1}, R_{T2})$ or both $P(Q_{T2}, O_{T1})$ and $P(R_{T2}, O_{T1})$ are true in the same state (at the same time). For example, in the example in Fig. 1, the presence of both $on(A, table)$ and $on(C, table)$ in the intial state reveals that $table$ is secondary with respect to $block$ in the $on(block, table)$ predicate type. As another example, we know that in the meet-pass domain no two trains can ever occupy the the same track at the same time. Furthermore, a train may not sit on more than one track at the same time. Therefore, the $train$ and $track$ object types will never be secondary with respect to one another in the meet-pass domain's $at(train, track)$ predicate type.

3. CONCLUSION

Most of the research in distributed planning so far is about important issues of cooperation, communication, coordination and negotiation among agents. Problem decomposition is one of the key problems in distributed planning. Current distributed planners do not use general and effective automatic problem decomposition techniques. In this paper we showed how interaction graphs can be used to decompose problems effectively. Interaction Graphs can be constructed in low order polynomial time. We showed how distributed planners can use decompositions based on interaction graphs. Experimental results which show that distributed planners using IG-based decomposition are significantly faster than centralized planners are reported in [Iwen & Mali 2002].

Acknowledgement: We thank Subbarao Kambhampati, Dana Nau, Ichiro Suzuki, Dan Weld and Linlin Zhang for useful comments on distributed planning. Mark Iwen was supported by NSF grant IIS-0119630 to Amol Mali.

[**Deo 1999**] Narsingh Deo, Graph theory with applications to engineering and computer science, Prentice Hall, 1999.

[**Foulser et al 1992**] David E. Foulser, Ming Li and Qiang Yang, Theory and algorithms for plan merging, Artificial Intelligence journal, Volume 57, Number 2-3, 1992, pp. 143-182.

[**Iwen & Mali 2002**] Mark Iwen and Amol Mali, Automatic problem decomposition for distributed planning, Proceedings of International Conference on Artificial Intelligence (IC-AI), Las Vegas, June 2002.

[**Mali & Kambhampati 2002**] Amol Mali and Subbarao Kambhampati, Distributed planning, To appear in the Encyclopaedia of Distributed Computing, 2002.