Distributed Graphplan

Mark Iwen & Amol Dattatraya Mali
Electrical Engineering & Computer Science
University of Wisconsin, Milwaukee, WI 53211
iwen2724@uwm.edu, mali@miller.cs.uwm.edu, Fax: 1-414-229-2769

Abstract

(Appears as a regular paper in the Proceedings of IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Washington D.C, IEEE Computer Society, Nov. 2002, pp. 138-145. 27 out of 84 submissions were accepted as regular papers leading to 32 % acceptance rate.)

Significant advances in plan synthesis under classical assumptions have occurred in last seven years. All such efficient planners are centralized planners. One very major development among these is the Graphplan planner. Its popularity is clear from its several efficient adaptations/extensions. Since several practical planning problems are solved in a distributed manner, it is important to adapt Graphplan to distributed planning. This involves dealing with significant challenges like decomposing goal and set of actions without losing completeness. We report two sound two-agent planners DGP (distributed Graphplan) and IG-DGP (interaction graph-based DGP). Decomposition of goal and action set in DGP is carried out manually and that in IG-DGP is carried out automatically based on a new representation called interaction graphs. Our empirical evaluation shows that both these distributed planners are faster than Graphplan. IG-DGP is orders of magnitude faster than Graphplan. IG-DGP is significantly benefitted by the interaction graphs which allow decomposition of a problem into fully independent subproblems under certain conditions. IG-DGP is a hybrid planner in which a centralized planner processes a problem until it becomes separable into two independent subproblems that are passed to a distributed planner. This paper also shows that advances in centralized planning can significantly benefit distributed planners.

1 Introduction

Broadly, distributed planning is the problem of finding a course of action that helps a set of agents in a given initial configuration to collectively satisfy certain desired behavioral constraints. The motivation behind distributed planning is to get the efficiency of parallel processing, the robustness of distributed systems and the simplicity of incremental construction and debugging. Problems that are inherently distributed (because of different spatial locations or privacy or security reasons) or decomposable into subproblems with limited interactions are good candidates for distributed planning. Importance of distributed planning is clear from the DARPA initiative air battle management program [2], air campaign planning [10] and the DARPA initiative pilot's associate program [7]. A brief review of various distributed planning techniques is given in [8].

Several advances have occurred in synthesis of plans in a centralized fashion (i.e. using only one agent) under classical assumptions in last six years. These have made it possible to generate plans faster and also improve their quality (generally measured in terms of the number of steps and/or number of actions). One of them is the highly popular Graphplan planner [3] which uses a compact representation called "planning graph". Graphplan generates plans with fewest number of steps. Graphplan generated plans faster than all classical planners developed before 1995. Graphplan has been improved/extended in several ways. DPPlan [1] is a variant of Graphplan which carries out global search in the space of planning graphs, simplifying them with logical inference rules. STAN [6] is a variant of Graphplan which detects and pre-compiles pairs of actions and pairs of propositions that are always mutually exclusive, to avoid their repeated discovery. [9] report a variant of Graphplan for temporal planning. Most of the work in distributed planning is about communication, co-ordination and negotiation between planning agents. The problem of improving computational efficiency of distributed planners has not been significantly addressed.

Given this, the potential of Graphplan in distributed planning is worthy of investigation. We develop two adaptations of Graphplan for distributed planning (DGP and IG-DGP). Each of these contains two planning agents. DGP uses a manual decomposition of goal and set of actions. IG-DGP uses a new representation called an "interaction

graph" which is highly effective at splitting a problem. By using the interaction graph, the cost of resolving interactions between plans of various agents can be eliminated under certain conditions. Another novel feature of IG-DGP is the use of a centralized planning technique as a preprocessing strategy to derive independent subproblems for distributed planning. DGP and IG-DGP have the following in common: agents in these planners use Graphplan to synthesize individual plans and interactions between the plans are resolved. Our empirical evaluation of DGP, IG-DGP and Graphplan shows that though both DGP and IG-DGP are faster than Graphplan, IG-DGP performs orders of magnitude faster than Graphplan. Our work shows how recent advances in centralized planning and interaction graph-based decomposition can benefit distributed planning.

2 Background

In this section, we explain our notation and the Graphplan algorithm [3]. Like most classical planners, we use STRIPS representation. Predicates like $on(A, B)$ are 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/operator are specific objects and lowercase letters are variables. We use "o_i" symbol to denote an action. A planning problem is specified as $< I, G, O >$, where I denotes completely specified initial state, G denotes goal and O denotes the set of actions in domain. We assume that I contains only true propositions. O_i denotes the set of actions that agent i can use to synthesize its plan. Clearly, $O_i \subset O$. I_i and G_i respectively denote initial world state and conjunctive goal of planning problem of agent i. FSS denotes forward state space. The indices of steps in a k step plan range from 0 to $(k-1)$.

Graphplan: Graphplan [3] works in 2 phases. The first involves growing a **planning graph** and is called the **plangraph** construction phase. This is a forward phase, beginning with the initial state. The second phase is a solution extraction phase. This is backward search phase starting with the goal. Plangraph, or planning graph (PG), has two kinds of levels called **action levels** and **proposition levels**. The 0th proposition level is the same as the initial state. The 0th proposition level occurs before the 0th action level which in turn occurs before the 1st proposition level which precedes the 1st action level, etc. In general, the i th proposition level is immediately succeeded by the i th action level. And, the i th action level immediately precedes $(i+1)$ th proposition level. A proposition level and an action level can be considered as sets whose members are the same as the contents of these levels. Note that a proposition level is not same as a world state. A proposition level can have both a fluent and its negation.

Plangraph is a compact representation of an exponentially large number of action sequences in the search tree of an FSS planner. The i th action level in the plangraph contains all actions whose all preconditions appear in the i th proposition level. There is also a dummy action called a **no-op** or **maintenance action**, or **persistence action** in the i th action level for each proposition in the i th proposition level. The precondition and effect of this action are the proposition for which the action is created. This action is included in the plangraph because if no action changing the truth of the proposition occurs, the truth of the proposition remains same. The $(i+1)$ th proposition level is the union of the i th proposition level and the effects of the actions in the i th action level. Thus, proposition level i is a superset of proposition level $(i-1)$. Similarly, action level i is a superset of action level $(i-1)$.

There are three kinds of edges in plangraph: **(i)** edges from propositions in proposition level i to the same propositions in proposition level $(i+1)$ (for no-ops), **(ii)** edges from propositions in proposition level i to actions (whose precondition list contains these propositions) in action level i and **(iii)** edges from actions in action level i to propositions (which are effects of these actions) in proposition level $(i+1)$.

A key to the efficiency of Graphplan is the inference of binary **mutex** (mutually exclusive) relations. Two kinds of mutexes are found: (i) mutexes between actions, and (ii) mutexes between propositions. Each of these two kinds of mutexes could be static or dynamic. Static mutexes are found by examining preconditions and effects of actions. Dynamic mutexes are found by propagating static mutexes using truths of propositions in the initial state. Note that dynamic mutexes may be permanent or temporary. Two actions in action level i are mutex if (i) their effects are inconsistent (the effect of one action is the negation of some effect of another action), or (ii) one action deletes some precondition of another action, or (iii) the actions have preconditions that are mutex at proposition level i. For (iii) one needs to know the definition of mutex propositions given next. Two propositions p, q in proposition level i are mutex if (i) p is the negation of q, or (ii) all ways (actions) of achieving p are mutex with all ways (actions) of achieving q. Note that while considering all ways of achieving a proposition, no-ops are also considered.

If the plangraph has a proposition level that contains all propositions from the goal such that no two of these are mutex, Graphplan starts a backward search for a plan. For each proposition in the goal, it chooses a source of support (an action) in the immediately preceding action level. The preconditions of these actions become subgoals to be achieved. If no two preconditions of the chosen actions are mutex, it chooses sources of support for these subgoals from the immediately preceding action level and continues this

process. In case subgoals are found to be mutex, it backtracks, chooses different sources of support, and repeats this process. If all combinations of the supporting actions fail for each subgoal at each proposition level, then Graphplan grows plangraph with one more action level and one more proposition level and tries the backward solution extraction process again. Graphplan is guaranteed to report unsolvability of a problem.

3 DGP

In this section, we describe the DGP algorithm. G_1 and G_2 are obtained by manual decomposition of goal. O_1 and O_2 are obtained by manual decomposition. $(G_1 \wedge G_2) = G$ and $(O_1 \cup O_2) \subseteq O$. The DGP planning algorithm contains following steps: **(i)** Agent 1 uses Graphplan to solve the problem $< I, G_1, O_1 >$. Agent 2 uses Graphplan to solve the problem $< I, G_2, O_2 >$. **(ii)** The number of steps in their plans are made equal by filling the shorter plan with no-ops. The global plan is obtained by taking a union of sets of actions at respective steps in individual plans. So the set of actions at i th step in global plan is a union of the sets of actions at i th step in individual plans. **(iii)** The global plan is checked by progression. If it is correct and achieves goal, it is returned. **(iv)** This is conflict resolution step. Actions that do not appear in individual plans can be introduced in the individual plans in this step, only if these actions appear in individual planning graphs through which the agents successfully extracted solutions. Conflicts are resolved by a backward global search whose action selection is restricted by the actions from corresponding action levels in individual planning graphs. Note that the backward search is not carried out on planning graph which is a union of the individual planning graphs. Since the size of the union of planning graphs is at least as high as the size of the largest of these, not searching over the union of planning graphs preserves benefits of decomposition. There is only one kind of conflict that may occur between individual plans - an action o_x in plan of agent 1 being static mutex with an action o_y at same step i in plan of agent 2, including no-ops. This mutex is removed by selecting some other action o_z from the action level i from the individual planning graphs and removing either o_x or o_y. The set of subgoals that must be true after last i steps of the global plan is updated and the backward search is repeated until all subgoals to be achieved are true in I or the space of repairs is exhausted. **(v)** Backward global search in step (iv) is repeated to find a plan of length 1 higher than the previous length.

In the backward search in step (iv), static and dynamic mutexes between actions in planning graph of agent 1 are used to identify failing action selections. Static and dynamic mutexes between actions in planning graph of agent 2 are also used. These are all found in step (i) itself. Backward search in step (iv) considers only static mutexes between actions from the same action levels in the two planning graphs. Note that dynamic mutexes between actions o_x and o_y that belong to two different planning graphs are not found. Their computation requires more global reasoning. Not computing these mutexes does not hurt the soundness of DGP, because of the correct and continuous updating of subgoals to be achieved in backward search in step (iv).

4 IG-DGP

In this section, we first explain the notion of an interaction graph along with some terms from graph theory and a graph algorithm. Then we describe the IG-DGP planning algorithm.

4.1 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 in the graph. An interaction graph is an undirected, simple bipartite graph. Since the graph is simple, no two vertices have two or more edges connecting them. The graph does not have any loops. Decomposition based on this graph is obtained by detecting if it is disconnected and finding its connected components if it is disconnected. A graph is connected if there is a path to reach each of its vertices from every other vertex. A graph $< V_1, E_1 >$ is a component or subgraph of graph $< V_2, E_2 >$ if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$.

Interaction graph for a problem $< I, G, O >$ is constructed in the following manner: Vertices are created for each proposition true in initial state and each proposition needed true in goal with some exceptions mentioned later. 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 initial state and the proposition in v_j belongs to goal. Vehicles, cities, airports and other (non-airport) locations in transportation logistics are secondary objects. Some secondary objects are resources, e.g. trucks, planes and grippers. We assume that the planning domain contains both primary and secondary objects. We assume that each proposition in goal contains at least one primary object. We assume that types of objects like package, truck, location etc. are given under a separate category in problem specification and that they are not a part of initial state. So vertices for $package(P_1), package(P_2), Truck(T_1), Truck(T_2), Airport(Heathrow)$ etc. do not appear in an interaction graph. The reason for ignoring these from the interaction graph is to keep the number of edges low. Propositions from I which contain only secondary objects are not represented in the interaction graph. Whether an object is primary or not is assumed to be given.

Examples of interaction graphs are given in Fig. 1 and 2. Blocks are primary objects and table is secondary object. Note that if vertices are connected because only secondary objects like table in blocks world are common, the graph will generally be connected, even when $< I, G, O >$ can be split into independent subproblems like $< I_1, G_1, O_1 >$ and $< I_2, G_2, O_2 >$ where $(G_1 \wedge G_2) = G$, $I = (I_1 \wedge I_2)$ and $(O_1 \cup O_2) \subseteq O$. Note that size of an interaction graph depends only on I and G. It is not affected by O at all.

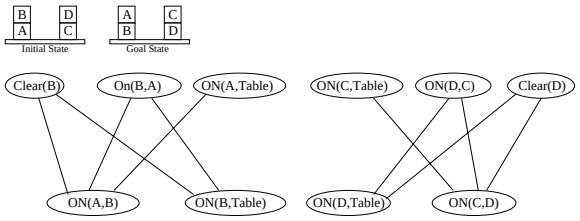

Figure 1. Disconnected interaction graph

The polynomial time algorithm (pg. 274, [5]) can be used to test if an interaction graph is disconnected and find all of its connected components if it is disconnected. IG-DGP uses this algorithm to find if an interaction graph is disconnected. If the interaction graph is disconnected and it has n connected components, then if n is even, two planning subproblems are formed such that each is based on $\frac{n}{2}$ components. If n is odd, one planning subproblem is formed based on $\frac{n+1}{2}$ components and other planning subproblem is formed based on the remaining $\frac{n-1}{2}$ components. So even if a problem is decomposable into more than two subproblems, IG-DGP splits it into only two subproblems. This decision was taken to keep the implementation simple. Splitting into more subproblems can be more beneficial. Different planning problems corresponding to different components of the interaction graph (when it is disconnected) are allocated to different agents. For plans of individual agents to be free of interactions, whether there are shared resources must be considered. We consider a plane to be one kind of resource, truck to be another kind of resource etc. We have the following guideline on decomposition of a problem into p independent subproblems:

Guideline: If an interaction graph of a planning problem $< I, G, O >$ has p connected components and there are at least p resources of each kind, 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.

If fewer resources are available, the interaction graph methodology can still be useful for synthesis of individual plans and one can use efficient resource allocation algorithms for successful merging of these plans. Even if there is no distinction between objects (like primary objects and secondary objects) in a domain, one can still construct a graph based on I and G and connect vertices with common objects to derive a decomposition. If such a decomposition is not good, planning and/or conflict resolution times may be high.

Let us consider the example in Fig. 1. Interaction graph 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 $move(A, B, D)$ will no longer be considered by IG-DGP which uses interaction graph-based decomposition. The loss of such actions is in fact desirable and it does not cause loss of completeness of the planner.

4.2 Connected Interaction Graphs

Even if an interaction graph for problem $< I, G, O >$ is connected, one can find a sequence of actions such that applying this sequence starting in I leads to a new world state I' such that interaction graph for the problem $< I', G, O >$ is disconnected. Such an action sequence could be found by an FSS planner. We are not suggesting FSS search for solving $< I, G, O >$. We suggest construction of an interaction graph for each node in search tree of FSS planner, treating the world state in the node as the initial state, keeping G, O same. This can be repeated till a world state I' is found such that the interaction graph for $< I', G, O >$ is disconnected. For example, the interaction graph in Fig. 2 becomes disconnected after the action of moving block D from B to C is carried out. In fact the new disconnected interaction graph is same as the graph in Fig. 1. FSS search has been shown to be a very efficient planning technique in [4]. Note that FSS search not only finds a world state for which interaction graph is disconnected, but it also finds a part of the plan. Global plan in such a case is obtained by prepending to the plan obtained after merging individual plans, the path in tree of FSS planner that lead from I to I'.

4.3 IG-DGP Algorithm

This consists of the following steps: **(i)** Construct an interaction graph for the given problem. **(ii)** If the interaction graph is connected, carry out breadth-first FSS search until

a new initial world state for which the interaction graph is disconnected is reached. **(iii)** Form two planning subproblems and assign them to the agents, if the conditions in the guideline in section 4.1 are satisfied. **(iv)** Allow both agents to generate individual plans using Graphplan. **(v)** After both agents have generated individual plans, merge them using steps (iii), (iv) and (v) of the DGP algorithm. Step (ii) of IG-DGP involves centralized search.

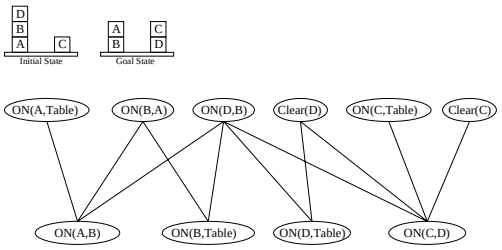

Figure 2. Connected interaction graph

5 Empirical Evaluation

Our empirical results are shown in Fig. 3, 4 and 5. These are obtained by running DGP, IG-DGP and Graphplan on various problems. The experiments were conducted on a Dual Intel Pentium II 400 MHz SunOS 5.7 machine. Times for generation of individual plans, resolving conflicts between the plans and checking the disconnectedness of an interaction graph and doing FSS search are reported in Fig. 3. The times are in cpu seconds, except those stated to be in cpu miliseconds (ms). * denotes that the run was terminated after 15 minutes. - denotes that no data is available because of termination of run. The problems with names starting with bw are from blocks world. bw-large.a is a benchmark problem. Actions in this domain are ground instances of the operator $move(x, y, z)$ where $x \neq y, y \neq z, x \neq z, x \neq Table$. This means moving block x from top of block y or table to top of block z or table. Grippers are not represented in the problem, so multiple blocks whose tops are clear can be moved at same time. Other problems are from the transportation logistics domain in which packages are to be delivered to be appropriate locations using planes and trucks. Trucks can travel between locations within a city and planes can fly from a location x to location y only if both are of type airport. In this figure, T1, T2 denote the times needed by agent 1 and agent 2 respectively, to generate individual plans. C denotes the time needed to resolve conflicts between the individual plans to yield a global plan. T denotes the total time needed to generate global plan. F denotes the time needed to construct an interaction graph plus the time needed to check disconnectedness of the interaction graph plus the time needed to do FSS search. The FSS search time in F dominates other times contributing to F. In some cases, the interaction graph was disconnected and no FSS search was needed. Note that F does not apply to DGP. In case of DGP, T = (max(T1,T2) + C). In case of IG-DGP, T = (max(T1,T2) + F + C). GP denotes Graphplan. Speedup in Fig. 3 is found by dividing the time needed by GP by the total time needed by IG-DGP. The current implementation of DGP algorithm does not contain step (v). However all test problems are such that they can be solved with the current implementation of DGP, if enough time is provided.

The subproblems in DGP had the form $< I, G_1, O_1 >$ and $< I, G_2, O_2 >$ such that $G_1 \wedge G_2 = G$ and $(O_1 \cup O_2) \subseteq O$. Note that $(O_1 \cap O_2)$ was not always empty because some objects had to be accessible to both agents. The goal was manually split to ensure that G_1 and G_2 were as independent as possible. To ensure a fair evaluation of DGP, each agent was given access to as few irrelevant actions as possible. For example, if there were two towers of blocks W_1, W_2 in initial state of a problem such that the goal involved two towers W' and W'' such that W' could be obtained by manipulating blocks from only W_1 and W'' could be obtained by manipulating blocks from only W_2, agent 1 was not given access to blocks from towers W_2, W'' and agent 2 was not given access to blocks from towers W_1, W'. In general, an agent was allowed to move only those blocks contained in its subgoals along with blocks which were in towers on the top of these blocks. Because of this, it became necessary to allow both agents to move a block, resulting in $(O_1 \cap O_2) \neq \phi$. Each agent was allowed to move a block that it could move, to top of any other block or table. Since each block could be moved by some agent, this manual decomposition of action set and goal was completeness preserving. In transportation logistics domain, agents were given different packages to deliver and different vehicles whenever multiple vehicles existed. An agent was given access to vehicles at same location as that of its packages in initial state, whenever possible. Each agent could move its vehicles anywhere and load its packages into any of its vehicles and unload from them anywhere, as long as the preconditions of the actions were true. Since each package and vehicle could be moved by some agent, actions in $(O_1 \cup O_2)$ were sufficient to solve the problems, preserving completeness. The results clearly show that IG-DGP was orders of magnitude faster than GP as well as DGP on several problems. The problem bw-1 was difficult to split even for the automatic decomposition technique, because of several interactions. I in this contained a single tower of blocks $B_1B_2B_3B_4B_5B_6...$ (which means that block B_1 is on top of block B_2, B_2 is on top of block B_3 etc.). The goal contained two towers of blocks which were $B_1B_3B_5....$ and $B_2B_4B_6...$

Some details of global plans and plans of individual agents are reported in Figure 4. A1 and A2 denote the num-

ber of actions in plans of agent 1 and agent 2 respectively. S1 and S2 denote the number of steps in plans of agent 1 and agent 2 respectively. A and S denote the number of actions and steps in global plan. The number of actions and steps in optimal plan are reported in the last column. These are the minimum number of actions needed in a plan with minimum number of steps. For DGP, S = (max(S1,S2) + a), where a is the number of extra steps added by backward global search while resolving conflicts between individual plans. a was always zero since step (v) in the DGP algorithm is not implemented. For IG-DGP, S = (max(S1,S2) + f + a) where f is the number of steps needed by FSS search to lead to a world state I' such that the interaction graph for I' and G was disconnected. The values of f for the problems (in top to bottom order) were 6, 2, 0, 4, 3, 0 , 1, 0 and 0. It can be seen that the average number of steps in global plans found by IG-DGP (9) was close to the average number of steps in optimal plans (8). The average number of actions in global plans found by IG-DGP (28) was however more than the average number of actions in optimal plans (21). Note that Graphplan is guaranteed to find plans with fewest number of steps. It is not guaranteed to find plans with fewest number of actions. The number of steps in plans found by DGP was always same as the number of steps in optimal plans. This was not always the case for IG-DGP because the FSS search applied only one action in a world state even if more could have been applied. The global plans found by IG-DGP can be post-processed to reduce the number of actions and steps.

The maximum number of actions in search spaces of the two agents for DGP and IG-DGP and in the search space of GP are reported in Figure 5, along with information about objects in the problems. The maximum number of actions is found by computing the number of all ground instances based on the objects that an agent had access to. N1 and N2 denote this for agent 1 and agent 2 respectively. It is useful to know this since the worst case size of search space depends on this. The number of objects in blocks world was same as the number of blocks. The data for transportation logistics problems shows the number of planes, cities, packages, trucks, airport locations and non-airport locations respectively. To test the effectiveness of interaction graph technique at reducing the number of actions, we found the percentage reduction in the maximum number of actions using the ratio of (r - (N1 + N2)) and r, r being the maximum number of actions for GP, using N1 and N2 for IG-DGP. The maximum reduction was 86.2 % (logistics-4), the minimum reduction was 49.8 % (bw-4) and the average reduction was 74.1 %. This shows that interaction graphs are very effective at reducing the maximum number of actions. This reduces the worst-case size of the search space by an exponential amount. FSS search is likely to need more steps on problems that are hard to separate. In such a case, the IG-DGP solving times are likely to go up. Advances in heuristic state-space search [4] can be used to reduce the times needed by the centralized FSS search.

Note that one may deal with a connected interaction graph of a problem $< I, G, O >$ by solving a planning problem $< I, G', O >$ such that any world state I' in which G' is true is such that the interaction graph for $< I', G, O >$ is disconnected. For this, one can inspect an interaction graph and detect some edges whose removal will disconnect the graph. The goal G' is such that achieving it will also remove these edges in the graph. Note however that new edges making the graph connected should not be introduced in the process of removal of these edges. For example, if the two edges between $on(D, B)$ and $on(A, B)$, $on(B, Table)$ are removed from the graph in Fig. 2, the interaction graph becomes disconnected. G' that must be fulfilled to achieve this can be derived using the argument that no primary object in one component of the resulting disconnected graph should appear in any relationship with any primary object in the remaining part of the graph. The set of primary objects in two components of the resulting disconnected interaction graph are $\{A, B\}$ and $\{C, D\}$. To reach a state in which there is no such relationship, one can achieve G' which is $(\neg on(A, C) \wedge \neg on(C, A) \wedge \neg on(B, C) \wedge \neg on(C, B) \wedge \neg on(A, D) \wedge \neg on(D, A) \wedge \neg on(B, D) \wedge \neg on(D, B))$. Our implementation of Graphplan allows specification of negated subgoals in goal of a problem. We experimented with this approach as an alternative to FSS search to get disconnected interaction graphs and found it to be very inefficient.

6 Discussion

In this section, we discuss potential extensions of our work. Our work differs from the previous work in distributed planning because our main emphasis is on adapting Graphplan for distributed planning with an automatic problem decomposition based on an interaction graph. Most other distributed planners use a highly manual decomposition of a problem. No domain-independent distributed planners using automatic decomposition are publicly available to the best of our knowledge, to compare IG-DGP with.

DGP algorithm is incomplete because it does not consider actions that do not occur in the planning graphs of individual agents while resolving conflicts. DGP can be made complete by considering actions that appear in the leveled off planning graph for the original problem $< I, G, O >$. IG-DGP can be easily extended to handle problems with goals that contain subgoals with no primary objects. This is because subgoals without primary objects can be easily achieved in most domains without undoing the subgoals containing primary objects. Graphplan or some other centralized planner can be used to achieve such subgoals

Problem	**DGP (T1,T2,C,T)**	**IG-DGP (T1,T2,C,F,T)**	**GP**	**Speedup**
bw-1	5.2, 5.6, *, *	45 ms, 12 ms, 0, 4.24, 4.29	3.481	-
bw-large.a	2.52, 0.8, 32 ms, 2.55	1.04, 3 ms, 0, 0.6, 1.65	12.55	7.6
bw-2	0.74, 0.77, 0, 0.77	0.37, 0.36, 0, 0.06, 0.43	39.22	91.2
bw-3	0.15, 1.88, 5 ms, 1.88	47 ms, 18 ms, 0, 30 ms, 77 ms	1.535	19.9
bw-4	3.37, 10.05, 74.1, 84.1	2 ms, 3.45, 0, 0.63, 4.09	15.98	3.91
logistics-1	0.17, 0.21, 0, 0.21	42 ms, 37 ms, 0, 30 ms, 72 ms	$>$ 30 minutes	$>$ 25000
logistics-2	72 ms, 50 ms, 0, 72 ms	28 ms, 9 ms, 0, 20 ms, 48 ms	0.202	4.21
logistics-3	0.537, 0.534, 0, 0.537	0.119, 0.115, 0, 50 ms, 0.169	$>$ 30 minutes	$>$ 10650
logistics-4	0.936, 0.941, 0, 0.941	0.201, 0.202, 0, 0.14, 0.342	$>$ 30 minutes	$>$ 5263

Figure 3. Empirical results - Planning and other times

Problem	**DGP (A1/S1, A2/S2, A/S)**	**IG-DGP (A1/S1, A2/S2, A/S)**	**GP**	**Optimal**
bw-1	14/8, 12/8, -	3/3, 3/3, 12/9	17/9	11/9
bw-large.a	8/4, 3/3, 11/4	11/6, 2/2, 16/9	10/4	6/4
bw-2	6/6, 6/6, 12/6	6/6, 6/6, 12/6	12/6	12/6
bw-3	9/7, 3/7, 12/7	4/4, 3/3, 11/8	12/7	10/7
bw-4	11/5, 5/5, 16/5	2/2, 12/5, 17/8	16/5	11/5
logistics-1	25/12, 25/12, 50/12	25/12, 25/12, 50/12	-	32/12
logistics-2	16/10, 7/6, 23/10	16/10, 7/6, 24/11	26/10	23/10
logistics-3	29/12, 29/12, 58/12	29/12, 29/12, 58/12	-	40/12
logistics-4	25/11, 25/11, 50/11	25/11, 25/11, 50/11	-	48/11

Figure 4. Empirical results - Number of steps and actions in plans

Problem	**DGP (N1, N2)**	**IG-DGP (N1, N2)**	**GP**	**Objects**
bw-1	294, 180	48, 18	294	7
bw-large.a	504, 360	294, 4	648	9
bw-2	792, 792	180, 180	1584	12
bw-3	224, 448	48, 48	448	8
bw-4	630, 810	4, 448	900	10
logistics-1	1052, 1052	90, 90	1240	2,4,4,4,4,4
logistics-2	360, 360	90, 10	402	1,3,4,3,3,3
logistics-3	1186, 1186	130, 130	1536	2,4,8,4,4,4
logistics-4	5250, 5250	402, 402	5820	2,6,6,6,6,6

Figure 5. Empirical results - Maximum number of actions and objects

from the world state at the end of execution of the plan found by IG-DGP. Interaction graph and planning graph can be viewed as complementary representations. Interaction graphs give an idea about separation of I and G into parts. However they do not give any information about actions relevant to solving a problem. Planning graphs give information about actions relevant to solving a problem, but they do not tell how to separate I or G into parts. Once subproblems based on an interaction graph are derived, one can construct separate planning graphs for these subproblems. Action levels and proposition levels and mutex relations from these planning graphs can be compared to detect dependencies between subproblems. If there are such dependencies, an alternative decomposition can be derived from the interaction graph with/without use of centralized FSS search. This methodology is very useful when resources are fewer and/or objects cannot be classified as primary and secondary. In case resources are inadequate for solving subproblems independently, it can be detected fast from individual planning graphs since proposition levels of some such planning graphs will never contain some subgoals. We have come up with a generalized version of the IG-DGP algorithm which does not make assumptions about I, G, the total number of resources and types and number of resources needed for individual problems. This version only assumes that primary, secondary and resource objects are explicitly specified. It includes an automatic resource allocation and re-allocation and use of a centralized planner if enough resources are not available.

IG-DGP does not construct an interaction graph after a disconnected interaction graph is found. Interaction graphs can be continuously constructed in presence of more agents than the number of components of an interaction graph, to carry out an online distribution of planning in a continuous fashion, to reap maximum benefits from parallel operation of agents. Our work can also be seen as improving Graphplan in a new way, by automatic decomposition of a problem using interaction graphs. Integrating this with existing Graphplan improvement strategies like use of persistent mutexes and goal ordering heuristics can improve Graphplan further. IG-DGP shows that centralized search can be used to process a problem until it is easy to separate it into independent subproblems. The centralized search can be viewed as centralized planning since the search generates a part of the plan. Such a centralized planning either eliminates or significantly reduces cost of resolving interactions among multiple plans. The efficiency of such planners is likely to depend upon how fast the centralized planning component works. Advances in centralized planning can be used to improve such planners. Interaction graph-based decomposition can also benefit distributed versions of planners other than Graphplan.

7 Conclusion

Since several practical planners are distributed planners, potential of Graphplan in distributed planning is worthy of investigation. Though Graphplan has been improved in several ways and adapted to handle conditional effects, metric time, uncertainty and resource quantities, its potential for distributed planning had not been investigated. We developed two variants of Graphplan for distributed planning (DGP and IG-DGP). We also introduced the notion of interaction graphs. Our empirical evaluation shows that both variants perform significantly better than Graphplan. IG-DGP is orders of magnitude faster than Graphplan. IG-DGP uses interaction graphs for intelligently splitting a problem. IG-DGP shows how hybrid planners containing both a centralized planner and a distributed planner can be developed. Though IG-DGP does not construct an interaction graph after a disconnected interaction graph is found, the graphs can be continuously constructed to distribute planning effort continuously, to reap maximum benefits from parallel operation of agents.

Acknowledgement: This work was funded by NSF grant IIS-0119630 to Amol Mali. The authors thank Avrim Blum, Subbarao Kambhampati, Dana Nau, Ichiro Suzuki and Dan Weld for useful comments.

[1] M. Baioletti, S. Marcugini and A. Milani, DPPlan: An algorithm for fast solution extraction from planning graph, Proceedings of international conference on Artificial Intelligence Planning and Scheduling (AIPS), 2000.

[2] T. C. Baker and J. R. Greenwood, Star: An environment for development and execution of knowledge-based planning applications, Proceedings of DARPA knowledge-based planning workshop, Dec. 1987.

[3] Avrim Blum and Merrick Furst, Fast planning through planning graph analysis, Artificial Intelligence 90, 1997, 281-300.

[4] Blai Bonet and Hector Geffner, Heuristic search planner 2.0, AI Magazine, Fall 2001, Volume 22, No. 3, pp. 77-80.

[5] Narsingh Deo, Graph theory with applications to engineering and computer science, Prentice Hall, page 274, 1999.

[6] Maria Fox and Derek Long, Utilizing automatically inferred invariants in graph construction and search, Proceedings of AIPS, CO, 2000.

[7] C. Key, Cooperative planning in the pilot's associate, Proceedings of DARPA Knowledge-based planning workshop, Dec. 1987.

[8] Amol Mali and Subbarao Kambhampati, Distributed Planning, To appear in the Encyclopaedia of Distributed Computing, Kluwer Academic Publishers.

[9] David E. Smith and Daniel S. Weld, Temporal planning with mutual exclusion reasoning, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI),

1999.
[10] David E. Wilkins and Karen L. Myers, A multiagent planning architecture, Proceedings of international conference on artificial intelligence planning systems (AIPS), 1998, pp. 154-162.