Abstract

An architecture for reactive planning, called an influence network, is presented. Most approaches to planning involve goal decomposition, action generation, and action fusion, the latter being the most difficult. In an influence network, fusion involves influences, rather than fully specified actions. An influence serves to constrain or bias the eventual selection of actions.

Plans to satisfy individual goals generate influences. These influences are validated as they are generated to ensure that they are consistent with influences already accepted. After all influences have been processed, actions are generated which satisfy the influences that have been accepted.

Influence networks provide a simple method of action fusion by delaying commitment while not delaying validation. As a result, plan repair can be accomplished by the individual planning agents as their plans are being constructed, rather than being performed at the system level.

1 Introduction

A planner can be defined as a process which generates actions designed to achieve a specified set of goals within a given environment. (Figure 1.) It is common to classify planning architectures within the spectrum from reflexive to premeditated, the basis for this classification being response time [1]. However, such a classification has the disadvantage of focusing on performance rather than planning approach.

1.1 Reactive vs Predictive

In this paper, we define a planner to be reactive if it is able to generate its next step without having to generate any subsequent steps. Similarly, a planner is predictive if it must generate a sequence of steps before it can determine any one step in the sequence. By this definition, A* [2], for example, is a predictive planning method, regardless of its response time.

Within the context of real-time control, a reactive planner repeats its planning process each decision cycle. In contrast, a predictive planner generates an entire plan initially and then applies a step of that plan each cycle, modifying or regenerating the plan when the environment changes unexpectedly. Clearly, the more uncertain or unpredictable the environment, the more appropriate a reactive planning approach becomes.

1.2 Behavior-Driven vs State-Driven

Reactive and predictive planners have different types of goals. The goal of a predictive planner is a desired state. A reactive planner, on the other hand, is inherently short-term in scope, with goals which can be viewed as desired behaviors — things which are to be achieved every decision cycle, rather than at some point in the future. We can therefore refer to predictive planners as state-driven and reactive planners as behavior-driven.

This dichotomy is useful because it focuses on the requirements of the problem to be solved, rather than on the internal processing used to solve it. In a state-driven planner, the goals describe the desired state and thus are part of the problem definition. (Figure 2.) In contrast, a behavior-driven planner sees the desired state as simply part of the environment, while the goals describe how it is to solve the problem. (Figure 3.)
particular, the goals of a behavior-driven planner remain constant over a given class of problems, whereas the goals of a state-driven planner may change with each new problem.

1.3 Basic Planning Stages

Planning is generally divided into three stages: goal decomposition, action generation, and action fusion. Goal decomposition takes a high-level goal and divides it into smaller subgoals. This is repeated until the subgoals are simple enough that they can be solved directly in the action generation stage. These actions are intended to solve each subgoal in isolation, but the solutions to the various subgoals may interfere with each other. Reconciling these interactions is the task of the action fusion stage and is a fundamental problem of planning architectures.

While a state-driven architecture, such as the hierarchical planner shown in Figure 4, exhibits all three of these stages, a behavior-driven planner needs no decomposition stage. Since the goals do not change from one problem to the next, goal decomposition can be done while the planner is being designed, rather than at execution time. Consider, for example, the subsumption architecture [3] shown in Figure 5. Each of the planners in the generation stage corresponds to a specific type of desired behavior, such as avoiding objects or exploring. The generation of these behavioral goals was done as a part of designing the planner, rather than being performed for each new problem. In fact, the goals in a subsumption architecture are not even represented externally but are inherent in the design of each component planner.

2 Influence Networks

An influence network, shown in Figure 6, is a type of behavior-driven planning architecture. As with subsumption, influence networks have no goal decomposition stage. The action generation and fusion functions are achieved by the interaction of two types of processes: planning agents and resource managers. Briefly, the planning agents process goals to generate influences on the resource managers, which in turn identify conflicting influences and generate the resultant actions. Each of these concepts is explained below.

2.1 Behavioral Goals

Goals within an influence network are prioritized and exist external to the planning processes. Each goal expresses a type of behavior to be achieved and optionally what aspects of the environment the behavior pertains to. A goal can be represented as a list consisting of a verb phrase followed by zero or more noun phrases, such as (remain-quiet) or (avoid-collision-with neighboring-obstacles). Noun phrases may represent queries against the environment, which are instantiated to identify specific objects. Verb and noun phrases may include additional parameters to modify their meaning, such as a noise level or avoidance range, for the above goals.

2.2 Planning Agents and Influence Generation

Each planning agent within an influence network is responsible to attempt to satisfy one or more types of
goal. Unlike conventional planners, an agent generates influences instead of actions. Rather than completely specifying a desired action, an influence merely serves to constrain or bias the selection of an action. Each agent attempts to use the least restrictive set of influences which will ensure the accomplishment of its goal.

2.3 Resources

A resource is any means by which the system can interact with its environment. This includes both desired interactions, such as motion or device settings, and side effects, such as the generation of noise or consumption of fuel.

A resource can be viewed as a decision domain from which one or more actions are selected each decision cycle. Influences serve either to constrain a resource domain, by prohibiting certain actions, or to bias the selection process to favor certain types of actions.

For example, one can treat the motion of a vehicle as a resource where the domain is all possible motion vectors, as explained in [4]. A collision avoidance agent might produce a constraint influence to prohibit motion in a certain set of directions, while an agent responsible for reaching a waypoint could bias the resource toward the selection of a motion which closes on the waypoint as rapidly as possible. Neither agent makes the decision of how the motion resource is to be used (i.e., which motion is to be selected), but both can influence the decision sufficiently to accomplish their individual goals.

2.4 Resource Managers and Plan Fusion

All actions in an influence network are generated by resource managers, in spite of the fact that they have no knowledge of either the goals or the environment, other than those aspects of the environment directly pertaining to the state of their particular resource. Each manager has two principal functions: influence validation and action selection.

Common approaches to least commitment planning involve delaying plan validation [1]. In an influence network, decisions are delayed, but validation is not. As each planning agent generates its plan to accomplish a goal, it determines which resources are needed to ensure success. These requirements are expressed as influences to the appropriate resource managers. As a manager receives an influence, it determines if it is compatible with other influences it has already committed to (i.e., influences generated by planning agents to satisfy higher priority goals.) An incompatible influence is one which contradicts an accepted influence, eliminates the entire remaining decision domain, or attempts to override the current resource bias.

A planning agent is informed as to whether its influences have been was accepted or rejected. An agent can respond to a rejected influence either by trying a different (and perhaps less optimal) way to accomplish the same goal or by simply reporting failure to achieve its goal during that cycle. The key point is that any plan repair is performed by the individual planning agents, where the specialized knowledge resides, rather than being done at the system level.

Once all of the goals have been processed in priority order by their respective agents, each resource manager then selects the actions it will generate for the current cycle. The decision domain for a resource is reduced by constraint influences. Within the remaining domain, the manager uses any bias influences to select the "best" action to take. All resources have a default bias which is used in the event that no bias influence has been received.

It should be noted that not all resources necessarily control actions. Some resources, like noise or fuel, may exist only to allow side effects of other resources to be explicitly represented and thus planned for.

2.5 Resource Interactions

Resource managers function independently of each other. Any dependencies which do exist between resources are handled in one of two ways. If the amount of interaction between two resources is great, they can be combined into a single, higher-level resource, such as in the case of representing a vehicle's course and speed as a single resource of motion. Otherwise, interactions among resources are captured by the planning agents which use them. For example, if taking a certain type of photograph requires that the camera be stationary, a photography agent would need to successfully influence both the camera and motion resources. Failure to obtain commitment from either manager would imply that the agent's plan is untenable. If an agent obtains some of the resources it needs but fails to obtain others, it needs to instruct the resource managers to release the ones it did obtain (analogous to the concept of a database transaction).

3 A Blocks World Example

Although not normally thought of as a reactive planning problem, the blocks world problem in Figure 7 (commonly called Sussman's anomaly) can be used to illustrate the operation of simple influence network.

Traditional approaches to this problem [5] are state-driven, taking the desired state of (on A B) and (on B C) as the goal. Difficulty arises from the fact that optimal plans to achieve each of the two goals separately interfere with each other, necessitating some form of plan repair.

To solve this problem using an influence network, we must identify the environment, the possible actions,
the resource managers, their influences, the agents, and their goals. The environment consists of the table, the blocks, and two relationships: is-on and should-be-on. Note that the desired state, expressed by the should-be-on relationship, is just another aspect of the environment.

There are only two possible actions in this simple world: pick-up and put-on, both of which can be performed as a pair once per decision cycle. Since these two actions are very closely related, a single resource manager, called the arm, is appropriate. The domain for pick-up is any exposed block or nothing, while the domain for put-on is any other exposed block, the table, or nothing. The default bias for pick-up is nothing. The default bias for put-on is the table if something is picked up, nothing otherwise.

The influences for a resource represent the minimal meaningful commitments which can be made. For the arm resource, a reasonable set of influences is:

\[
\begin{align*}
(pick-up \ X) \\
(do\-not\-pick-up \ X) \\
(put-on \ X) \\
(do\-not\-put-on \ X \ Y)
\end{align*}
\]

The second argument to do-not-put-on is optional and means that no block should be put on block X with the exception of block Y.

Since an influence network is behavior-driven, the goals should express desired behavior. Within the context of the blocks world, the following prioritized set of goals can be used:

\[
\begin{align*}
(do\-not\-cover \ misfits) \\
(stack \ stackable-misfits) \\
(uncover \ covered-misfits)
\end{align*}
\]

The noun phrase misfits is a query returning the blocks which participate in a should-be-on relationship but not the corresponding is-on relationship. A block is covered if it has any block on top of it. A stackable block is one which is exposed and should be on a block which is also exposed. For example, in state 3 of Figure 7, blocks A and B are misfits, block C is covered, and block A is stackable.

The first (and thus highest priority) goal above is to prevent making the situation worse by covering misfits. The other two goals seek to improve the situation, by uncovering misfits so they can be acted upon and stacking misfits to get them into their should-be-on relationships. Note that the order of these two goals determines whether the planner would rather create a desired stack or destroy an undesired one, when given the choice. Either order will take the same number of steps, but the one used here serves to minimize the number of blocks on the table at one time.

In this simple example, there is one agent per goal type, and each agent processes blocks in an arbitrary order. The uncover agent generates an influence to pick up the block on top of the stack containing the block it is to expose. The stack agent generates two influences— one to pick up the stackable block and one to put it on its desired support. The do-not-cover agent is more complex. If the block needs to be moved (i.e., is not on want it should be on), no block should be put on it. However, if the block does not need to be moved, it is okay to cover it with the block which should be on it. These objectives are expressed via the two forms of the do-not-put-on influence. All three of these agents are simple algorithms, performing no replanning if an influence is rejected and generating no bias influences.

This simple influence network, shown in Figure 8, was coded by one of the authors in Common Lisp in a few hours. It has given optimal solutions to every blocks problem tried (although no effort has been made to prove that it will always do so). A trace of its operation against the problem in Figure 7 is shown in Table 1. Note that during the planning for state 1, the stack agent wanted to stack B on C but was prohibited from doing so because it conflicted with the higher priority goal to not cover A. In the next cycle, the same agent tried again and succeeded, in spite of having the lowest priority goal. In state 4, there are no misfits, so no agents generated influences, leaving the arm to perform its default action—nothing.

![Figure 8: A Blocks World Influence Network](image-url)
Table 1: Influence Network Solution to the Blocks World Problem

<table>
<thead>
<tr>
<th>State</th>
<th>Goals</th>
<th>Instantiations</th>
<th>Influences</th>
<th>Validation</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(do-not-cover misfits)</td>
<td>(do-not-cover A) (do-not-cover B) (do-not-cover C)</td>
<td>(do-not-put-on C) (do-not-put-on B) (do-not-put-on C B)</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>(stack stackable-misfits)</td>
<td>(stack B) (stack A)</td>
<td>(pick-up C) (pick-up B) (pick-on C)</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>(uncover covered-misfits)</td>
<td>(uncover A)</td>
<td>(pick-up B)</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(do-not-cover misfits)</td>
<td>(do-not-cover A) (do-not-cover B) (do-not-cover C)</td>
<td>(do-not-put-on A) (do-not-put-on B) (do-not-put-on C B)</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>(stack stackable-misfits)</td>
<td>(stack A) (stack B)</td>
<td>(pick-up A) (pick-on B) (pick-on C)</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>(uncover covered-misfits)</td>
<td>(uncover A)</td>
<td>(pick-up B)</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(do-not-cover misfits)</td>
<td>(do-not-cover A) (do-not-cover B) (do-not-cover C)</td>
<td>(do-not-put-on A) (do-not-put-on B A) (do-not-put-on B C)</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>(stack stackable-misfits)</td>
<td>(stack A)</td>
<td>(pick-up A) (pick-on B)</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(uncover covered-misfits)</td>
<td>(uncover A)</td>
<td>(pick-up B)</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

4 A Submarine Control Example

Figure 9 shows an influence network planner called Submarine Tactical Assessment and Response (STAR) developed to automate the tactical control of a submarine [6]. It was implemented in Lucid Common Lisp on a Sun-4 using the FROBS [7] object-oriented environment and is currently under evaluation at the Naval Underwater Systems Center (NUSC) for possible utility as a submarine decision aid.

The influence network approach is well suited to the submarine tactical control domain due to the high degree of uncertainty in the environment. In addition, motion has impact on nearly every tactical objective, including safety, sensor utility, stealth, and weapons deployment. Influences provide a mechanism to accomplish multiple goals simultaneously with a single maneuver. Motion is therefore managed as a resource by the STAR planner, using the methodology of geometric constraint-based reasoning [4,6].

The STAR architecture uses a separate situation assessment phase to allow the specific goal priorities to be dynamically altered in response to significant events. For example, goals which are of interest while performing a transit may be completely irrelevant while under attack. The mission contains a set of tactical goal lists and the criteria under which each list is to be active. Mission assessment is a simple, real-time process that determines each decision cycle which list of tactical goals specified in the mission is to be used in the current situation. In addition to these reactive components, the contents of the mission can be

Figure 9: Submarine Tactical Assessment and Response (STAR) Planner Architecture
modified by a predictive planner (not shown) in response to long-term developments. Note that the influence network does not have to wait for the predictive planner to make a decision but is always operating in real-time against the last set of goals it has been given.

5 Characteristics of Influence Networks

5.1 Robustness

Unexpected states do not pose a problem for an influence network, because, being reactive, it does not really have any "expectations". For example, the blocks world planner described above would have no trouble responding to the arm accidently knocking over a stack. Each decision cycle represents an entirely new problem, regardless of whether it could have been predicted.

While an influence network as a whole is reactive, the individual agents (which are themselves planners) may employ predictive reasoning methods. In fact, some of the planning agents built for the STAR planner are simple rule-based systems. However, detailed plans tend to be of limited use, given that higher priority agents may at any time "consume" needed resources, potentially rendering detailed plans obsolete.

5.2 Efficiency

Clearly, individual agents and managers will each have their own performance characteristics, dependent on the specifics of their implementation. However, the primary function of an influence network is action fusion, which it performs in a fashion that is linear with respect to the number of influences.

5.3 Distributed Control

Rodney Brooks refers to his robots as "out of control", and the same phrase might be applied to an influence network. No single part of the system can be said to be "in control". An agent can only influence the system's behavior, and even then, it is focused on only its particular goal. A manager does control a specific set of actions, but it does so only as directed by the collective influences of the agents and has no knowledge of the other managers or how its actions agree with those of other managers.

While this may look like chaos, the behavior of an influence network is actually very rigidly bounded, due to the strict prioritization of goals. One may not be able to easily predict exactly how an influence network will behave, but we can use the goal priorities to guarantee that it will, for example, never decide to drive off a cliff in order to take a good picture.

Control is distributed throughout the system but is nonetheless explicit and rigid.

5.4 Modularity

An influence network provides for a great deal of independence among the various planning agents. Each agent can be designed and tested independently, allowing agents to be easily added or removed from the network. (An exception to this occurs when one agent is allowed to leverage another agent's expertise by generating an intermediate goal for that agent to solve.)

The use of influences provides for modularity of resource managers, as well. For example, we could easily replace the arm manager of our blocks world planner with a two-armed version, enabling it to respond to two pick-up influences each cycle. As long as the new manager responds to the same influences, the system not only continues to function but actually utilizes the second arm quite effectively.

5.5 Derived vs Canned Tactics

Since each goal is handled by independent planning agents, it might seem that no coherent overall behavior would result. However, the use of resources actually serves to coordinate the actions of the individual agents in ways which are often better suited to the specifics of the situation than would be possible if a global agent attempted to drive the system as a whole.

An instance of this occurred early in the development of the STAR planner. Members of our group coded separate agents to follow a given vessel, to orient the towed array sensors properly to observe a vessel, and to prevent frequent maneuvers (to keep the array straight). Each agent was developed and tested in isolation and approved by our domain experts. When we then combined the agents into a single system and ran the simulation, the submarine began to move in an odd zigzagging fashion. Thinking that we had a bug, we were surprised when the domain experts stated that this behavior was actually quite appropriate.

The key point of this experience is that nowhere in the system was there an agent responsible to see that this type of behavior was employed in such a situation. Instead, the tactics were derived as a natural result of the interaction of the independent agents as they vied for resources. Indeed, these tactics were used by the system for the same reason they might be used by a person — they satisfied multiple goals simultaneously. Similarly competent behavior was observed in far more complicated scenarios and over extended periods of time.

5.6 Explanation of Actions

An interesting benefit of influence networks is that failure to accomplish a given goal can be explained in
terms of resource contention, limitations, and trade-offs. Each failure can be directly attributed to the inability to obtain one or more needed resources. Seeing which higher priority goals "consumed" these resources gives an accurate picture of the trade-offs which were made and why.

6 Areas For Research

Influence networks were developed for use in real-time control. An interesting problem within this domain is attempting to ensure system response within a specific time interval, regardless of the complexity of the situation. Since the goals are prioritized, one possible approach is to process as many of the goals (in priority order) as possible until time runs out. This should have the effect that, as the situation becomes critical, low priority goals would automatically be dropped from consideration.

Another area for research is that of cooperating planners. We are currently experimenting with the idea of allowing one planner to send influences to another. For example, one robot might be planning to photograph a given object. In order to try to ensure that other robots stay out of the way, the planner could broadcast a motion influence. Such an influence could be treated by the other robots as if it came from a one of its own agents. Given the priority of the influence, the other planners would know when they could reject the influence in order to accomplish some more important goal, such as collision avoidance.

References