Planning for Loosely Coupled Agents using Partial Order Forward-Chaining

Jonas Kvarnström
Department of Computer and Information Science
Linköping University, SE-58183 Linköping, Sweden (jonkv@ida.liu.se)

Abstract

Partially ordered plan structures are highly suitable for centralized multi-agent planning, where plans should be minimally constrained in terms of precedence between actions performed by different agents. In many cases, however, any given agent will perform its own actions in strict sequence. We take advantage of this fact to develop a hybrid of temporal partial order planning and forward-chaining planning. A sequence of actions is constructed for each agent and linked to other agents' actions by a partially ordered precedence relation as required. When agents are not too tightly coupled, this structure enables the generation of partial but strong information about the state at the end of each agent's action sequence. Such state information can be effectively exploited during search. A prototype planner within this framework has been implemented, using precondition control formulas to guide the search process.

1 Introduction

A major earthquake has struck in the middle of the night, devastating vast parts of the countryside. Injured people are requesting medical assistance, but clearing all roadblocks will take days. There are too few helicopters to immediately transport medical personnel to all known wounded, and calling in pilots will take time. Fortunately, we also have access to a fleet of unmanned aerial vehicles (UAVs) that can rapidly be deployed to send prepared packages of medical supplies to those less seriously wounded. Some are quite small and carry single packages, while others move carriers containing many packages for subsequent distribution. In preparation, a set of ground robots can move packages out of warehouses and possibly onto carriers.

Given the properties of this somewhat dramatic scenario as well as the robotic agents involved, what types of automated planning could we use to generate high-quality plans?

One option would be to rely on distributed cooperative planning techniques, where each agent is responsible for its own actions but also coordinates its local plan with other agents in order to achieve a shared goal.

An alternative is the use of centralized planning, where a single agent generates a complete plan coordinating activities at a higher level before distributing subplans to individual agents. Plan execution then proceeds in a distributed manner, with either centralized or distributed synchronization between agents. This alternative requires all agents to be fully cooperative, which appears reasonable to assume for the application at hand.

Each of these choices has its own advantages. Distributed planning can for example be more flexible, potentially allowing individual agents to renegotiate parts of the plan during execution. Furthermore, it does not require full cooperation between agents. Having a centralized authority can facilitate the generation of high-quality plans and allows a ground operator to approve or modify a complete plan before execution, which may be a requirement in some cases. Thus, each alternative is likely to be better in some situations and is worth pursuing for its own qualities. For the purposes of this paper, we will proceed under the assumption that centralized planning has been chosen.

Returning to the scenario, we see that action durations are not likely to be perfectly predictable, but it is often possible to specify an approximate expected duration for the case where no failures occur during execution. This information should be taken into account during planning by preferring plans that are expected to require less time to execute. We also prefer plans to be minimally constrained in terms of precedence between actions performed by different agents, in order to avoid unnecessary waiting.

To some extent these properties can be treated after plan generation, for example by inferring less constraining precedence relations from a sequential plan [3]. However, this entails hiding important aspects of the domain and of our concept of plan quality from the planner, which can decrease the quality of the final plan. For example, a sequential planner has no concept of which actions can be executed in parallel and might therefore assign all actions to the same agent, as long as this does not lead to using a greater *number* of actions. Parallelizing such plans after the fact is non-trivial: It requires recognizing subplans that can be assigned to other agents in a meaningful manner, selecting suitable agents for reassignment, and generally also replanning for the selected agents, which may not perform tasks in exactly the same manner as the original agent. Similarly, a non-temporal planner might decide to use as few actions as possible even when these actions are very time-consuming. Treating this after plan generation also requires changing the actions in the plan, as opposed to simply adding temporal durations to each action. We would therefore prefer to work directly with a plan structure capable of expressing these aspects, such as a temporal partial-order plan.

Planners generating such plans do not necessarily have to be explicitly aware of agents, as long as they can express mutual exclusion conditions between actions that cannot be executed concurrently. Indeed, the scenario above is quite similar to the standard logistics benchmark domain, where agents are typically modeled as arguments to actions. Therefore it would be possible to generate the required plans using temporal versions of standard partial order causal link (POCL[1]) planners, agent-aware or not.

Partial order causal link planning was initially conceived as a means of increasing the efficiency of plan generation. Through *late commitment*, avoiding “premature commitments to a particular [action] order” [10], less backtracking was required during the search for a plan. Once a solution was generated, the assumption was that it would be executed in sequence by a single agent. Though a number of POCL planners are also able to generate concurrent plans, the desire to delay commitments to action precedence for performance purposes usually remains one of the primary reasons for the use of partial orders.

However, late commitment is far from the only means of improving planning performance. For example, many recent planners build on the use of forward-chaining state-space search, which generates considerably richer information about the state of the world at any given point in a plan compared to POCL planning. This information can then be exploited in state-based heuristics [4, 8] or in the evaluation of domain-specific control formulas [2, 9]. Since the use of state information has led to a number of very successful total-order planners, it would be interesting to also investigate to what extent one can generate rich state information when generating *partially* ordered plans.

We cannot adopt an unmodified version of forward-chaining search, since we do desire flexible plan structures. However, this desire is entirely motivated by the presence of multiple agents whose capability for concurrent plan execution should be utilized to the greatest extent possible. Therefore, an alternative to POCL planning would be to retain partial ordering (and temporal flexibility) between actions executed by *different* agents, while generating the actions for each *individual* agent in sequential temporal order. Each agent-specific action sequence can then be used to generate partial agent-specific states to be used in heuristics or control formulas.

In this paper, we therefore begin our investigations into alternative methods for generating partial-order plans by adopting certain aspects of the standard forward-chaining paradigm in a hybrid *partial order forward-chaining* (POFC) framework that sacrifices some of the positive aspects of late commitment in order to gain the benefits of richer state information.

In Section 2, we discuss the ideas behind the partial order forward-chaining framework and its applicability to centralized multi-agent planning. In Section 3, we go on to present one concrete planner operating within the POFC framework. We then discuss related work in Section 4 and present our conclusions in Section 5.

[1]We assume a basic familiarity with partial order causal link planning and refer the reader to Weld [12] for an overview of the associated concepts and terminology.

2 Partial Order Forward-Chaining

Our discussion of the fundamental ideas underlying partial order forward-chaining (POFC) begins with an analysis of common execution constraints for multi-agent plans and how these constraints affect the desired plan structure. We then continue by showing how these ideas and structures allow us to take advantage of richer state information than is generally available to partial order causal link (POCL) planners. This results in a hybrid planning framework taking advantage of certain aspects of forward-chaining in the generation of partial order plans.

A significant degree of variation is possible in terms of the exact plan structures and planning algorithms used within this framework. Consequently, the concepts introduced in this section must be defined at a comparatively high level of abstraction. In the next section we present a detailed definition of one concrete POFC planner, which also provides additional intuitions regarding the high-level framework.

Plan Structures. As noted in the introduction, our interest in the generation of partial order plans is grounded in a desire to support the type of concurrency that is inherent in many execution mechanisms. In particular, centralized planning for multiple agents yields plans that are naturally concurrent: Each agent can generally perform its actions in parallel with other agents, and should not be forced to wait for other agents unless this is required due to causal dependencies, resource limitations, or similar constraints.

In Figure 1, for example, the first two actions of uav4 are independent of the actions of the ground robot robot3. Sequential plans cannot model this fact, since they do not permit concurrency at all. Temporal plans, where each action is assumed to be executed during a specific interval of time, do allow concurrency but are only guaranteed to achieve the goal if actions start and end in the predicted order. Partially ordered plans are more complex to handle, as one must prove during planning that any action order consistent with the partial order will satisfy the goal. On the other hand, the result is a considerably greater degree of flexibility during execution.

However, the fact that there exist actions that can be performed concurrently does not mean that *arbitrary* concur-

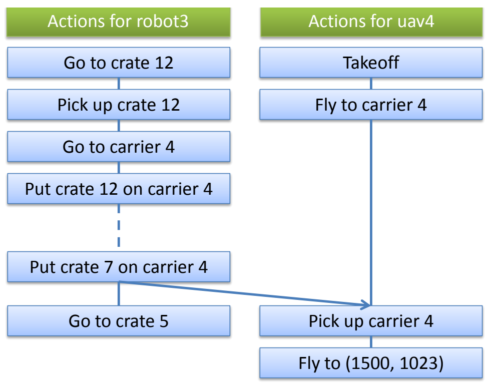

Figure 1: Example POFC plan structure

rency is possible. The actions assigned to any given agent can often only be performed in sequence, or in some cases, through a small and fixed number of sequential *threads* of execution. For example, a UAV would only be able to perform its own flight and package delivery actions in strict sequence and not in parallel, as it cannot be in several places at the same time. Another thread of execution could be used for camera control actions that are also performed sequentially, but in parallel with the flight actions. It is clear that in terms of execution flexibility, the greatest gains come from allowing partial orders across different threads as opposed to allowing partial orders within a particular thread.

The plan structures used in partial order forward-chaining are directly based on this model of execution. Each action is therefore assumed to be explicitly associated with a specific thread belonging to a specific agent. A plan structure then corresponds to one totally ordered action sequence for each thread, with a partial ordering relation between actions belonging to different threads.

Figure 1 shows an example for two threads, one for each of two agents. A UAV flies to a carrier, but is only allowed to pick it up after a ground robot has loaded a number of crates. The ground robot can immediately continue to load another carrier without waiting for the UAV. Depending on the expressivity of the planner, POFC plan structures may also include metric temporal relations between actions, mutual exclusion relations, or other similar information. A concrete plan structure for a specific prototype planner will be formally defined in Section 3.2.

To simplify the remainder of the presentation, we assume without loss of generality that each agent supports a single sequential thread of execution. Multiple threads for each physical agent can be supported either through a minor extension to these definitions or simply by modeling each thread as a separate "virtual" agent.

Sequential Search Order. Since actions for any given agent must be sequentially ordered, it appears natural to also *add* these actions in sequential order during search. In Figure 1, a new action for robot3 would then have to be added strictly after the action of going to crate 5, while a new action for uav4 would have to be added after the action of flying to the position (1500, 1023). Note that this does not prevent a new action from being added before existing actions belonging to *other* agents. For example, the next action added for robot3 could be constrained to occur before uav4 picks up carrier 4, or even before it begins flying to carrier 4.

To some extent this search process results in an earlier commitment to action orderings than in POCL planning. However, the precedence between a new action for one agent and existing actions belonging to other agents only has to be as strong as is required to ensure that preconditions are satisfied and actions do not interfere. Actions belonging to distinct agents can therefore be independent of each other to the same extent as in a standard partial order plan. This allows POFC plans to retain the essential flexibility that is desired for concurrent execution.

State Generation. The more we know about the execution order for a plan, the more information we can infer about the state of the world after any given action in the plan. For example, standard sequential forward-chaining yields a completely defined order of execution. In this case, applying an action in a completely specified world state always yields a new completely specified state. This is very useful when determining which actions are applicable in the next step. Rich state information also facilitates the use of expressive operator specifications as well as state-based heuristics or control formulas [2, 9] guiding forward-chaining search.

Since our plans are partially ordered, we cannot expect to be able to generate complete state information. For any given agent, though, actions are generated in sequential order. POFC planning can therefore be seen as performing a variation of forward-chaining search for each individual agent. We can take advantage of this to generate considerably more state information than is typically available to POCL planners, where one typically aims to restrict action ordering as little as possible and where new actions can be inserted "between" existing actions.

In particular, many state variables are generally associated with a specific agent and are only affected by the agent itself. For example, this holds for the location of an agent (unless some agents actively move others) and for the fact that an ground robot is carrying a particular object (unless one agent can place objects in another agent's gripper). Similarly, agent-specific resources such as fuel or energy levels are rarely directly affected by other agents. Due to the requirement for "local" total ordering, we can easily generate complete information about such "agent-specific" state variables at any point along an agent's action sequence. This information is particularly useful for the agent itself, since actions performed by one agent are likely to depend largely on its own agent-specific variables: Whether it is possible for uav1 to fly to a particular location depends on its own current location, its own fuel level, and its own altitude.

Not all state variables are completely agent-specific.

However, agents are in many cases comparatively loosely coupled [6]: Direct interactions with other agents are relatively few and occur comparatively rarely. For example, a ground robot would require a long sequence of actions to load a set of boxes onto a carrier. Only after this sequence is completed will there be an interaction with the UAV that picks up the carrier. This means that for extended periods of time, agents will mostly act upon and depend upon state variables that are not *currently* affected or required by other agents.

As will be shown in Section 3.5, this also provides opportunities for generating strong and useful partial states by carrying state information from one agent to another along precedence constraints when interactions do occur. In Figure 1, for example, the takeoff action will have little information about the state of the carrier, as one cannot know in advance which actions robot3 will have the time to perform before or during takeoff. However, the action of picking up carrier 4 must occur after the carrier is fully loaded. This information can therefore be carried over from robot3 to uav4 along the cross-agent precedence constraint.

Finally, as in any planner, we also have access to state variables representing static facts such as the locations of stationary objects and the capabilities of individual agents.

Thus, POFC planning enables us to generate quite extensive agent-specific information about the state that will hold after any given action is executed. This information generally will not be total, but is in many cases sufficient to determine whether a particular agent can add a particular action to its sequence. We also expect the information to be useful for the development of new state-based heuristics for POFC planning.

In some cases, additional information that is currently local to another agent will be required in order to determine the executability of an action. Acquiring such information involves the addition of precedence constraints to the plan. Returning once more to Figure 1, picking up a carrier might only be possible if the carrier is fully loaded. We know that carrier 4 is fully loaded after robot3 loads crate 7 onto the carrier. Proving this to be true when uav4 picks up the carrier requires a cross-agent precedence constraint. In the following section, we will present one potential mechanism for generating such constraints.

3 A Prototype POFC Planner

A variety of planning algorithms and search spaces can be realized within the general framework of partial order forward-chaining, differing along several dimensions.

Partial order causal link planners allow actions to be inserted in arbitrary order, without guaranteeing that their preconditions are satisfied or that their effects are compatible with existing actions. Such "flaws" in a plan must be corrected at a later time through the introduction of additional actions and precedence constraints. For example, the planner could insert the action of picking up a fully loaded carrier before adding the actions required for actually loading crates onto the carrier, as long as these actions were then constrained to be executed in the required order.

Similar methods could be used for partial order forward-chaining, with one limitation. It is by definition impossible to insert a new action for one agent before another action belonging to the same agent. However, as mentioned before, a new action can be inserted before an action belonging to *another* agent, allowing previously unsatisfied preconditions of the latter action to be satisfied. For example, the planner could first insert the action of uav4 picking up a fully loaded carrier, and then the actions required for robot3 to load the carrier.

In our initial investigations, we have instead chosen to explore a search space where adding a new action to a plan with a given set of precedence constraints is only permitted if this results in an executable plan without flaws. The preconditions of the new action must be satisfied at the point where it is inserted in the current plan structure, its effects must not interfere with existing actions in the plan, and mutual exclusion relations must be satisfied. In this sense, the planner is closer to forward-chaining planning.

We also choose to achieve goal-directedness through the use of domain-specific precondition control formulas [1, 2, 9] as explained below. This can be very effective due to the comparatively rich state information afforded by the POFC plan structure. Thus, we do not currently make use of means-ends analysis as in standard POCL planning, or state-based domain-independent heuristics as in many forward-chaining planners.

3.1 Domains and Problem Instances

For our first partial order forward-chaining planner, we assume a typed finite-domain state-variable representation of planning domains. State variables will also be called *fluents*. For example, loc(*package*) might be a location-valued fluent taking a package as its only parameter.

An *operator* has a list of typed parameters, where the first parameter always specifies the executing *agent*. For example, the act of flying between two locations may be modeled as the operator fly(*uav*,*from*, *to*), where the *uav* is the executing agent. An *action* is a fully instantiated (grounded) operator. Since finite domains are assumed, any operator is associated with a finite set of actions.

Each operator is associated with a *precondition* formula and a set of *precondition control* formulas, both of which may be disjunctive and quantified. We often use "conditions" to refer to both preconditions and control formulas.

Precondition control represents conditions that are not "physically" required for execution, but should be satisfied for an action to be meaningful for the given domain [1, 2]. For example, flying a fully loaded carrier to a location far from where its packages should be delivered is possible but pointless, and can be prevented using a suitable control formula. Given the search method used in this planner, precondition control will not introduce new subgoals that the planner will attempt to satisfy. Instead, the formulas will be used effectively to prune the search space.

An operator has a strictly positive *duration*, a temporal expression specifying the expected amount of time required to execute the action. The duration may be dependent on the state in which an action is invoked. We currently assume that the true duration of the action is strictly positive and cannot be controlled directly by the executing agent. Apart from this, we assume no knowledge of upper or lower bounds for execution times, though support for such information may be added in the future.

A set of *mutexes* can be associated with every operator[2]. Mutexes are acquired throughout the duration of an action to prevent concurrent use of resources. For example, an action loading a crate onto a carrier may acquire a mutex associated with that crate to ensure that no other agent is allowed to use the same crate at the same time. Mutexes must also be used to prevent actions that are associated with different agents and that have mutually inconsistent effects from being executed in parallel. Thus, we do not model mutual exclusion between actions by deliberately introducing inconsistent effects, as in some planning formalisms.

For simplicity, we initially assume single-step operators, where all *effects* take place in a single effect state. Effects are conjunctive and unconditional, with the expression $\mathsf{f}(\overline{v}) := v$ stating that the fluent $\mathsf{f}(\overline{v})$ has been assigned the value v when execution ends. Both v and all terms in $\overline{v}$ must be either value constants or variables from the formal parameters of the operator. For example, the operator $\mathsf{fly}(uav, from, to)$ may have the effect $\mathsf{loc}(uav) := to$.

For any given problem instance, the *initial state* must provide a complete definition of the values of all fluents. The *goal* is typically conjunctive, but may also be disjunctive. The construct $\mathsf{goal}(\phi)$ can be used in precondition control formulas to test whether ϕ is entailed by the goal. For example, we should only load boxes that must be moved according to the goal.

3.2 Plan Structures

We associate each action a in a plan with an *invocation node* $inv(a)$ where conditions must hold and where mutexes are acquired, and an *effect node* $eff(a)$ where effects take place and mutexes are released. All mutexes belonging to the action are considered to be held by the associated agent in the entire interval of time between its invocation node and its effect node[3]. Invocation nodes and effect nodes are called *plan nodes*.

[2]A mutex is an object that can only be acquired by a single thread, or in this case agent, at any given point in time.

[3]Thus, the planning algorithm must generate a partial order that is sufficiently strong to ensure that no two actions a, a' belonging to distinct agents can hold the same mutex at the same time. This is done at the end of Section 3.6.

A *plan* is then a tuple $\langle A, N, L, O\rangle$ whose components are defined as follows.

- A is the set of actions occurring in the plan.
- N contains one invocation node and one effect node for every action in A.
- L is a set of ground causal links $n_i \xrightarrow{\mathsf{f}=v} n_j$ representing the commitment that the effect node n_i will achieve the condition $\mathsf{f} = v$ for the invocation node n_j.
- O contains a set of ordering constraints on N whose transitive closure is a partial order denoted by $\preceq$, where we define $n_i \prec n_j$ iff $n_i \preceq n_j$ and $n_i \neq n_j$.

For any action a, we implicitly require that $inv(a) \prec eff(a)$: An action is always invoked before it has its effects. Additionally, given that there are no upper or lower bounds on action durations, it is not possible to directly control the time at which an action finishes by any other means than by delaying its invocation. Therefore, if a plan requires $eff(a_1) \prec eff(a_2)$, it is implicitly required that $eff(a_2) \prec inv(a_2)$. Finally, by the definition of partial order forward-chaining, the nodes associated with any given agent must be totally ordered by O.

Similar to standard POCL planning, we assume a special initial action $a_0 \in A$ without conditions or mutexes, whose expected duration is 0 and whose effects provide a complete definition of the initial state. For all other actions $a_i \neq a_0 \in A$, we must have $eff(a_0) \prec inv(a_i)$. Due to the use of forward-chaining techniques instead of means-ends analysis, there is no need for an action whose preconditions represent the goal, as in standard POCL planning.

Note that this plan structure is defined for the expressivity supported by our initial POFC planner. Given different levels of expressivity, different structures may be appropriate. For example, if upper and lower bounds on action durations are supported, a plan may have to include a temporal network or a similar structure, thereby allowing the planner to efficiently query the implicit action precedence constraints that follow from these bounds. As our initial planner has no bounds on durations, precedence can be completely determined by the constraints in O.

3.3 Executable Plans and Solutions

If a partially ordered plan will always be executed sequentially, it can be considered executable if and only if all action sequences satisfying the partial order are executable.

For POFC planners, as well as some POCL planners, the assumption of concurrent execution is fundamental. This leads to the possibility of one agent beginning or finishing executing an action *while* another agent is in the process of executing another action. The need to consider such cases is the reason why our precedence relation is defined relative to invocation and effect nodes, not relative to entire actions. A POFC plan should therefore be considered executable if

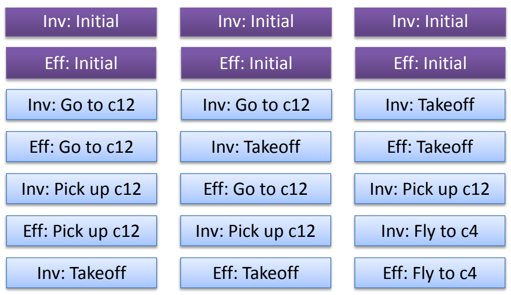

Figure 2: Three node sequences

and only if every *node* sequence satisfying the associated partial order is executable. Figure 2 shows three node sequences compatible with the plan defined in Figure 1, with the addition of the special initial action used in this particular POFC planner.

The executability of a single node sequence is defined in the standard way. Observe that the first node in such a sequence must be the invocation node of the initial action a_0, which has no preconditions or effects. The second node is the effect node of a_0, whose effects completely define the initial state. After this prefix, invocation nodes and effect nodes may alternate, or many nodes of the same type may occur in sequence, depending on the order in which actions are assumed to begin and end. Effect nodes update the current state. For the plan to be executable, an effect node must not have internally inconsistent effects. For example, it must not assign two different values to the same fluent. Invocation nodes contain preconditions and precondition control formulas that must be satisfied in the "current" state. Finally, executability also requires that no mutex is held by more than one agent in the same interval of time.

An executable plan is a *solution* iff every compatible node sequence results in a final state satisfying the goal.

3.4 Search Space

POCL planners add actions first and resolve unsatisfied conditions later, thereby searching through the space of partially ordered (and not necessarily executable) sets of actions. In contrast, forward-chaining planners begin with an empty executable plan, and actions can only be added after being proven executable. Forward-chaining can thus be viewed as searching in the space of *executable plans*, with a single plan modification step consisting of adding one new action at the end of the current plan.

Given the plan structure and expressivity defined above, a similar search space can be used for POFC planning. The initial search node then corresponds to the "empty" executable plan $\langle \{a_0\}, \{inv(a_0), \mathit{eff}(a_0)\}, \emptyset, \{inv(a_0) \prec \mathit{eff}(a_0)\}\rangle$, where a_0 is the initial action whose effects define the initial state. Each child of a search node adds a single new action to the end of one agent's action sequence, together with a set of precedence constraints and causal links ensuring that the plan remains executable.

This claim implies that we do not lose completeness by requiring every intermediate search node to correspond to an *executable* plan, as opposed to an arbitrary action set as in POCL planning. Intuitively, this holds because there can be no circular dependencies between actions, where adding several actions at the same time could lead to a new executable plan but adding any single action is insufficient.

More formally, let $\pi = \langle A, N, L, O\rangle$ be an arbitrary executable plan. Let $a \in A$ be an action which is not guaranteed to precede any other action in the plan (for example, the action of going to crate 5 in Figure 1). In other words, let $a \in A$ be an action such that there exists no other action $b \in A$ where $\mathit{eff}(a) \prec inv(b)$. Such an a must exist, or the precedence relation would be circular and consequently not a partial order, and π would not have been executable.

Since a is not the predecessor of any other action, it cannot have been used to support the preconditions and control formulas of other actions in π. Removing it from the plan will therefore have no negative effects in this respect. Similarly, a cannot be required for mutual exclusion to be satisfied: Removing a can only lead to fewer mutexes being allocated, which can only improve executability.

Consequently, removing a and the associated plan nodes, causal links and precedence constraints from π must lead to a new executable plan π'. We see inductively that any finite executable plan can be reduced to the initial plan through a sequence of such reduction steps, where each step results in an executable plan. Conversely, any executable plan can be constructed from the initial plan through a sequence of action additions, each step resulting in an executable plan.

Since we assume finite domains, solution plans must be of finite size and can be constructed from the initial plan through a finite number of action addition steps.

Given finite domains, the action set must also be finite. Furthermore, when any particular action is added to a plan, there must be a finite number of ways to introduce new precedence constraints and causal links ensuring that the plan remains executable. Any search node must therefore have a finite number of children, and the search space can be searched to any given depth in finite time.

Thus, given a method for generating all valid child nodes (finding all applicable actions), we can incrementally construct and traverse a search space. Given a method for testing goal satisfaction and a complete search method such as iterative deepening or depth first search with cycle detection, we have a complete planner. These issues will be considered in more detail in the following subsections.

3.5 Partial States

Forward-chaining planners find applicable actions by evaluating preconditions in the current completely defined state. For partially ordered plans there is no unique "current" state, and we can rarely infer complete information about the world even for a specific plan node or action.

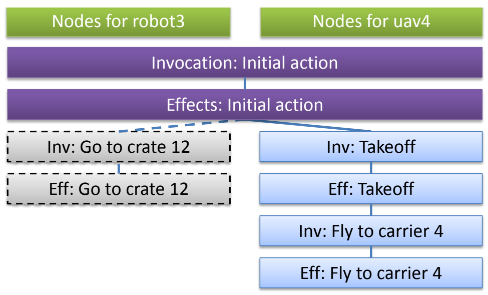

Figure 3: Example POFC plan structure

However, as discussed previously it *is* possible to infer and store *partial* state information for any plan node.

A variety of structures can be used for this purpose, each having its own strengths and weaknesses. For example, associating each plan node with a set of possible states would allow any fact that can be inferred from the current plan to be represented, including arbitrary disjunctive facts such as $\mathsf{at(uav4, pos1)} \vee \mathsf{at(uav5, pos1)}$. However, such structures tend to require considerable space and may take a considerable amount of time to update when new actions are added.

Instead, we currently use partial states represented as a finite set of possible values for each fluent: $\mathsf{f} \in \{v_1, \ldots, v_n\}$. The evaluation procedure defined below resolves as many parts of a formula as possible through this partial state for efficiency. Should this not be sufficient to completely determine the truth or falsity of the formula, the procedure falls back on an explicit traversal of the plan structure for those parts of the formula that remain unknown. This grounds evaluation in the initial state and the explicit effects in the plan for completeness.

The Initial State. The initial plan consists of a single action a_0, whose invocation node is associated with the empty state and whose effect node directly and completely defines the initial state of the planning problem at hand.

Updating States. When a new action is added to a plan, states associated with existing nodes must be incrementally updated to reflect the changes that this might have caused. For example, consider the situation in Figure 3 and assume that robot3 is initially at depot1. Before we add the action of going to crate12, the plan includes no movement for the robot, and the invocation node for takeoff will include the fact that robot3 remains at depot1. When the new action is added, it is temporally unconstrained relative to the takeoff action. Then we only know that when takeoff is invoked, robot3 will be either at depot1 or at crate12.

State updates must generate "sound" states: When a particular node is reached during execution, each fluent must be guaranteed to take on a value included in the state of the node. However, updates do not have to yield the *strongest* information that can be represented in the state structure, since formula evaluation will be able to fall back on explicit plan traversal. Thus, a tradeoff can be made between the strength and the efficiency of the update procedure.

For example, a sound state update procedure could weaken the states of all existing nodes in the plan: If a state claims that $\mathsf{f} \in V$ and the new action has the effects $\mathsf{f} := v$, the state would be modified to claim $\mathsf{f} \in V \cup \{v\}$. On the other hand, it is clear that no effect node can interfere with the states of its own ancestors. In Figure 1, for example, the effect node for the action of picking up carrier 4 has ancestor nodes belonging to robot3 as well as uav4 and cannot interfere with the states of these nodes. Therefore, weakening the states of all *non-ancestors* is sufficient.

Generating New States. When a new plan node n is added, it always has at least one immediate predecessor – a node $p \prec n$ such that there exists no intermediate node where $p \prec n' \prec n$. In Figure 3, for example, the invocation node of going to crate12 has a single immediate predecessor: The effect node of the initial action. If an action is also constrained to start after an action belonging to another agent, it can have multiple immediate predecessors.

Let n be a new node and p one of its immediate predecessors. It is clear that the facts that hold in p will still hold in n except when there is explicit interference from intervening effects. Therefore, taking the state associated with p and "weakening" it with all effects that *may* occur between p and n, in the same manner as in the state update procedure, will result in a new partial state that is valid for n.

For example, let n be the invocation node of going to crate12 in Figure 3, and let p be the effect node of the initial action. We can then generate a state for n by taking the state of p and weakening it with the effects associated with taking off and flying to carrier 4, since these are the only effect nodes that might intervene between p and n.

Now suppose that we apply this procedure to two immediate predecessors p_1 and p_2, resulting in two states s_1 and s_2 both describing facts that must hold at the new node n. If s_1 claims that $\mathsf{f} \in V_1$ and s_2 claims that $\mathsf{f} \in V_2$ for some fluent f, then both of these claims must be true. We therefore know that $\mathsf{f} \in V_1 \cap V_2$. This can be extended to an arbitrary number of immediate predecessors, resulting in stronger state information when a new node is created. Note that given that agents are loosely coupled, a node generally has very few immediate predecessors, which limits the time required for state generation.

Conjoining information from multiple predecessors often results in gaining "new" information that was not previously available for the current agent. For example, if robot3 loads boxes onto a carrier, incrementally updating a total-weight fluent, other agents will only have partial information about this fluent. When uav4 picks up the carrier, this action must have the last load action of robot3 as an immediate predecessor. The UAV thereby gains complete information about weight and can use this efficiently in future actions.

Finally, if the new node is an effect node, its own effects must also be applied to the new state.

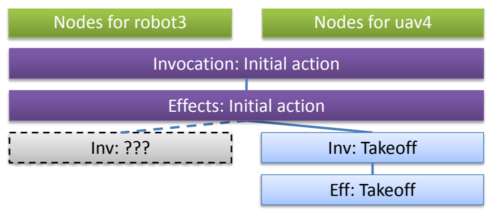

Figure 4: A new invocation node is being created

3.6 Searching for Applicable Actions

When searching for applicable actions, we first determine which agent to use. Several heuristics can be used, such as reusing agents to the greatest extent possible or distributing actions evenly across all available agents. In the latter case, we can calculate the timepoint at which we expect each agent to finish executing its actions in the current plan and test agents in this order. This selection must naturally be a backtrack point, to which the planner can return in order to try a different choice of agent. The same applies to many other choices below.

The intention is then to make use of the previously discussed procedure for generating state information in order to quickly detect most inapplicable actions for the selected agent. For some of the remaining actions, we may have to introduce precedence constraints in order to actively *make* the actions applicable. For example, suppose we are testing whether uav4 can pick up carrier4. This may only be possible if the pickup action is constrained to occur after robot3 loaded the carrier.

Satisfying Preconditions. When a specific agent has been chosen, we create a new invocation node n that is not yet associated with a particular action. This node is appended to the end of the selected agent's node sequence, or after the initial action a_0 if no agent-specific actions have been added previously. In Figure 4, for example, a new invocation node for robot3 is being created. At this point, n will always have a single immediate predecessor.

We then generate a "temporary" partial state s for n according to the procedure discussed previously. This state includes many of the facts that must hold when the new action is invoked, regardless of which action happens to be selected. Consequently it can be used to efficiently separate potential new actions for the current agent into three sets, where preconditions and precondition control formulas are definitely satisfied (A_+), definitely *not* satisfied (A_-), and potentially satisfied ($A_?$), respectively.

For actions in $A_?$, there is currently insufficient information in s to determine whether the required conditions hold. This may be due to incomplete state updates or because it is inherently impossible to determine the value of a particular fluent given the current partial order. For example, robot3 may only be able to move to launchpad1 if uav4 has already taken off, which requires n to be ordered strictly after the effect node of takeoff. Thus, new precedence constraints may be required to ensure that an action is executable, which may give n additional immediate predecessors. This can only strengthen the information previously provided in s, never invalidate it.

Given sufficiently loose coupling, together with the existence of agent-local and static facts, $A_?$ will be comparatively small. Nevertheless, actions in this set must also be handled. For this purpose we define the procedure *make-true*(α, n, π), which recursively determines whether a formula α can be made to hold in n, and if so, which precedence constraints need to be added for this to be the case. Subformulas are evaluated in the partial state of n whenever possible.

The procedure returns a set of *extensions* corresponding to the minimally constraining ways in which the precedence order can be constrained to ensure that α holds in n. Each extension is a tuple $\langle P, C\rangle$ where P is a set of precedence constraints to be added to O and C is a set of causal links to be added to L. Thus, if α is proven false regardless of which precedence constraints are added, $\varnothing$ is returned: There exists no valid extension. If α is proven true without the addition of new constraints, $\{\langle\varnothing, C\rangle\}$ is returned for some suitable set of causal links L. In this case, the state of n can be updated accordingly, providing better information for future formula evaluation.

We will now describe the *make-true* procedure. Certain aspects of the procedure have been simplified below to improve readability while retaining correctness. A number of optimizations to this basic procedure can and have been applied, several of which will be discussed below the main procedure description.

Assume that we call *make-true*(α, n, π), where $\pi = \langle A, N, L, O\rangle$, and let s be the partial state of n.

Let us first consider the base case, where α is the atomic formula $\mathsf{f} = v$. If this is true according to s, we determine which node n' generated the supporting value for f and return $\{\langle\varnothing, \{n' \xrightarrow{\mathsf{f}=v} n\}\rangle\}$. If the formula is false according to s, we return $\varnothing$. Otherwise, s contains insufficient information to determine whether the formula holds. We then find all effect nodes $E = \{e_1, \ldots, e_m\}$ in π that assign the value $\mathsf{f} = v$. This set may be empty, in which case we must return $\varnothing$. If $|E| > 0$, then for each effect node $e_i \in E$, we generate all sets $P_{i,j}$ of minimally constraining new precedence constraints that we could use to ensure that the relevant effect cannot be interfered with between e_i and n. Each set $P_{i,j}$, together with the associated causal links, forms one valid extension. The set of all these extensions is returned.

The case where α has the form $\mathsf{f} \neq v$ is handled similarly.

If α is a negated formula $\neg\beta$, the negation is pushed inwards using standard equivalences. For example, *make-true*$(\neg(\beta \wedge \gamma), n, \pi)$ = *make-true*$(\neg\beta \vee \neg\gamma, n, \pi)$.

If α is a conjunction $\beta \wedge \gamma$, then both conjuncts must be satisfied. We first determine how β can be satisfied by recursively calling E_1 = *make-true*(β, n, π). If this returns $\varnothing$, we immediately return $\varnothing$: If we cannot satisfy the first conjunct, we cannot satisfy the conjunction. Otherwise, we need to determine how the extensions in E_1 can be further extended so that γ is also satisfied. For every extension $\langle P_i, C_i \rangle \in E_1$, we let $\pi_i = \langle A, N, L \cup C, O \cup P \rangle$ and call *make-true*(γ, n, π_i). We take the union of all results, remove all extensions that are not minimal in terms of precedence constraints, and return the remaining extensions.

If α is a disjunction $\beta \vee \gamma$, then it is sufficient that one disjunct is satisfied. We therefore calculate *make-true*$(\beta, n, \pi) \cup$ *make-true*(γ, n, π), corresponding to all ways of satisfying either disjunct. We then remove all extensions that are not minimal in terms of precedence constraints and return the remaining extensions.

Finally, if α is a quantified formula, we iterate over the finite set of values in the domain of the quantified variable. Universal quantification can then essentially be considered equivalent to conjunction, while existential quantification is equivalent to disjunction.

This procedure may seem quite complex. However, any POCL planner must also resolve unsupported conditions in a similar manner, searching for existing actions that support the conditions or possibly adding new actions for support. Apart from the order of commitment, the main differences are that the POFC planner uses a partial state to quickly filter out most candidate actions and is restricted to searching for existing actions supporting conditions as opposed to adding new actions.

Similarly, though the evaluation procedure may seem to lead to a combinatorial explosion, recall that we are essentially doing forward search. We must therefore find existing support for all conditions *in the current plan*, which tends to yield a reasonably sized set of consistent extensions.

A number of optimizations can also be applied.

For example, instead of calculating *all* possible extensions in a single call, extensions can be returned incrementally as they are found.

It is possible to store and efficiently update a map associating each fluent with the nodes affecting it, for efficiency when searching for support for an atomic condition.

The evaluation order can be altered so that one always evaluates those parts of a formula that can be resolved in the current partial state before those parts that require support from effect nodes. This is useful in cases such as when the first conjunct in a conjunction is not determined by the partial state but the second conjunct is definitely false.

As a final example, we can structure the process of finding all applicable instances of a particular operator so that large sets of instances can be ruled out in a single evaluation. For example, suppose that flying between two locations is only possible when a UAV is already in the air, represented as the fluent flying(uav). Whether this condition holds depends on the agent but is independent of the locations in question. If a particular UAV is not in the air, there is therefore no need to iterate over all combinations of locations.

Once the evaluation procedure has ensured that preconditions and precondition control formulas will hold, we continue by adding precedence constraints ensuring that no mutex is used twice concurrently. We then ensure that the effects of the new action cannot interfere with existing causal links in the plan. If this entire procedure proceeds, the action was applicable and one of the possible sets of precedence constraints and causal links can be added to the plan.

Finally, we should note that **goal satisfaction** can be tested in a manner equivalent to the *make-true* procedure. The goal formula is then evaluated in a new node having all other nodes as ancestors. Any extension returned by *make-true* corresponds to one possible way in which the current plan can be extended with new precedence constraints to ensure that the goal is satisfied after the execution of all actions.

4 Related work

The ability to create temporal partially ordered plans is far from new. A variety of such planners exist in the literature and could potentially be applied in multi-agent settings. Some of these planners also explicitly focus on multi-agent planning. For example, Boutilier and Brafman [5] focus on modeling concurrent interacting actions, in a sense the opposite of the loosely coupled agents we aim at.

However, the main focus of this paper is to investigate the possibility of taking advantage of certain aspects of *forward-chaining* when generating partially ordered plans for multiple agents. In this area, very little appears to have been done. An extensive search through the literature reveals two primary examples.

First, a multi-agent planner presented by Brenner [7] does combine partial order planning with forward search. However, the planner does not explicitly separate actions by agent and does not keep track of agent-specific states. Instead, it evaluates conjunctive preconditions relative to those value assignments that must hold after *all* actions in the current plan have finished. This is significantly weaker than the evaluation procedure defined in this paper. In fact, as Brenner's evaluation procedure cannot introduce new precedence constraints, the planner is incomplete.

Second, the FLECS planner [11] uses means-ends analysis to add relevant actions. A FLExible Commitment Strategy determines when an action should be moved to the end of a totally ordered plan prefix, allowing its effects to be determined and increasing the amount of state information available to the planner. Actions that have not yet been added to this prefix remain partially ordered.

Though there is some similarity in the combination of total and partial orders, FLECS uses a completely different search space and method for action selection. Also, whereas we strive to generate the weakest partial order possible between actions performed by different actions, any action that FLECS moves to the plan prefix immediately becomes totally ordered relative relative to all other actions.

FLECS therefore does not retain a partial order between actions belonging to distinct agents.

Thus, we have found no existing planners taking advantage of agent-specific forward-chaining in the manner described in this paper.

5 Conclusions

We have presented a hybrid planning framework combining interesting properties of temporal partial order and forward-chaining planning. We have also described one of many possible planners operating within this framework. We view this as an interesting variation of POCL planning worthy of further exploration, and believe that future investigations will show that each framework has its own strengths and applications.

An early prototype implementation of the suggested planner has been developed. As we are still in the exploration phase, the current implementation is written for readability and ease of extension rather than for performance. For example, many data structures can and will be replaced with considerably more efficient ones. Therefore, standard benchmark tests would provide no meaningful information about the strengths of POFC planners as compared to other temporal partial order planners.

However, the basic structure of this particular POFC planning method can also be evaluated by observing search patterns, such as the strength of pruning when precondition control formulas are used. Though a final judgment has to await more extensive testing in multiple domains, initial experiments indicate a pruning strength very similar to that of standard forward-chaining planners based on control formulas for pruning, which is very promising.

Several extensions are planned for the near future, including support for incompletely specified initial states and the generation of conformant plans. We are also very interested in investigating the use of domain-independent heuristics for the new plan structure. Finally, we may develop alternative search procedures more similar to POCL planners in the sense that actions with currently unsupported conditions can be added, resulting in flaws that can be resolved through means-ends analysis.

Acknowledgements

This work is partially supported by grants from the Swedish Research Council (50405001, 50405002) and CENIIT, the Center for Industrial Information Technology (06.09).

References

[1] F. Bacchus and M. Ady. Precondition control. Available at http://www.cs.toronto.edu/~fbacchus/Papers/BApre.pdf, 1999.

[2] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

[3] C. Bäckström. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, 9 (99):137, 1998.

[4] B. Bonet and H. Geffner. HSP: Heuristic search planner. *AI Magazine*, 21(2), 2000.

[5] C. Boutilier and R. I. Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.

[6] R. I. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 28–35, Sydney, Australia, 2008.

[7] M. Brenner. Multiagent planning with partially ordered temporal plans. In *Proc. IJCAI*, 2003.

[8] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[9] J. Kvarnström and P. Doherty. TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, 30:119–169, June 2000.

[10] E. D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, pages 206–214. Morgan Kaufmann Publishers Inc., 1975.

[11] M. Veloso and P. Stone. FLECS: Planning with a flexible commitment strategy. *Journal of Artificial Intelligence Research*, 3:25–52, 1995.

[12] D. S. Weld. An introduction to least commitment planning. *AI magazine*, 15(4):27, 1994.