Forward Chaining Hierarchical Partial-Order Planning

Andrew Messing and Seth Hutchinson

Georgia Institute of Technology, Atlanta GA
{amessing, seth}@gatech.edu

Abstract. In recent years, the surge in interest in integrated Task and Motion Planning (TAMP) has caused a resurgence in Task Planning research in the robotics community. Meanwhile, in the broader AI community, combining partial-order planning with grounded forward chaining has become popular for temporal task planning, preserving the ability of partial-order planners to deal with concurrent actions while exploiting the speed of modern-day grounded forward search. In this paper, we present two new planning algorithms. The first, FCPOP, combines full utilization of the delayed action ordering commitment of partial-order planning with grounded forward search guided by a temporally informed heuristic. This results in a planner that is flexible in how it structures plans with respect to action parallelism, creates high-quality plans with low makespans, and computes plans quickly. FCPOP is shown empirically to outperform state-of-the-art temporal planners on several benchmark planning problems. The second planning algorithm FCHPOP introduces hierarchical information in the form of abstract actions. This reduces the number of nodes that need to be explored and speeds up the planning process while still generating quality plans.

Keywords: Task Planning and AI Reasoning. Partial-Order Planning. Hierarchical Task Network.

1 Introduction

Recently there has been a resurgence of Task Planning research in the robotics community. This has been drive, in part, by the surge in interest in integrated Task and Motion Planning (TAMP) [7, 9, 14]. Several different approaches have been investigated for improving the Task Planning aspect of TAMP including formal methods [21], satisfiability [7], and AI task planning [9]. In this work, we develop two algorithms that build upon prior AI temporal task planning approaches and can be used with multiple robots.

Partial-order planning (POP) was the most popular method for task planning until the late 90s. With its least-commitment philosophy, POP has several important advantages, such as being more flexible in execution [17], easy to extend for temporal task planning [3], and very suitable for multi-agent planning systems [11]. These advantages led to its use in several task planners including

UCPOP [15], VHPOP [22], and CPT [19]. The flexibility of these plans makes them easier to execute and to repair for real-world systems, as small delays in a robot's actions can be dealt with without modifying the plan. Also, the partial-ordering creates a useful model for concurrent execution, which is useful for multi-robot systems. However, POP is known to be slower than other modern planning methods that use improved solvers, more informed state-based heuristics, and lighter-weight search machinery that allows for much faster backtracking across alternative search states. The speed deficiency led to it becoming unfashionable in the task planning community [4].

The desire for a task planning algorithm that has the flexibility of POP and the speed of modern day planning algorithms has led to recent investigation into combining POP with other planning algorithms. One example of this is combining POP with grounded forward chaining to gain the speed of better state-based heuristics and backtracking mechanisms [4]. This requires a relaxation of the least-commitment philosophy that POP is built on. Another approach that has been explored is combining hierarchical information, similar to what is used in a hierarchical task network (HTN), with partial-order planning in what is known as hybrid planning [2].

In this paper, we present two new task planning algorithms, FCPOP and FCHPOP. The first, FCPOP, combines classical partial-order planning with grounded forward chaining. Unlike most task planning algorithms that combine the two approaches, FCPOP utilizes the full delayed action ordering in order to create more flexible, higher quality plans with shorter makespans and that are suitable for multi-robot systems. The utilization of delayed action ordering also reduces and sometimes eliminates the need for backtracking which causes our planner to be faster than many state-of-the-art temporal planners. In addition, through grounded forward chaining FCPOP can use a temporal state-based heuristic for a more guided search through the plan space and avoiding deadends that might arise due to temporal constraints. Despite the extra computation cost from a more informed heuristic and full delayed action ordering, we empirically show that our planner outperforms state-of-the-art temporal planners on several temporal task planning benchmarks [5].

Our second task planning algorithm, FCHPOP, builds upon the first by including hierarchical information in the form of abstract actions that can be recursively refined. The hierarchical structure of actions and tasks allows expert knowledge about a domain to be included with relative ease in order to guide the search process and speed up planning overall [12]. The action hierarchy can be built so the planning algorithm can plan for teams instead of individual robots. This allows for solutions where robots on the same team can share the same high-level goal and act together as much as possible without the need to plan each action for each individual robot separately [2]. As such, planning with these abstract actions and refining them reduces the number of nodes that need to be visited and explored which speeds up the process while still creating a quality plan. This is empirically shown through comparison with FCPOP on multiple benchmark problems.

2 Background

In this section, we briefly review temporal task planning and partial-order planning. More detailed explanations can be found in [20, 22]. Through this explanation we will use a running example of planning for a robot to move a box from a location l_a to a location l_b. In classical planning, an agent chooses a sequence of actions that taken in order modify the world into a goal state.

Definition 1. *(State Variable [16]) A state variable is an element of the world state. Each state variable, v, is associated to a finite domain, D_v, of mutually exclusive values that refer to objects of the world.*

Definition 2. *(Fluent [16]) A fluent is a tuple of the form $\langle v, d \rangle$, where v is a state variable, and $d \in D_v$ indicates that the variable v has the value d.*

Typically the world is described by a set of state variables that depict high-level properties of physical objects, e.g. that a robot is holding a box, rather than the actual coordinates of the robot and the box. Our example has fluents for **Loc(l)**: the robot is at the location l; **BLoc(l)**: the box is at the location l; and **Holds()**: the robot holds the box.

Actions are represented as state transition functions that modify the values of fluents[10]. Under temporal planning multiple actions can be taken simultaneously, the actions' durations may vary, and there may be complex interdependencies between actions and events. This makes it necessary to model the duration and concurrency of actions and events, and increases the complexity of solving the problem. For our example,we consider three actions with the following schemas:

MOVE(l_{start}, l_{target}):
pre:
Loc(l_{start})
$l_{start} \neq l_{target}$
eff:
Loc(l_{target})
$\neg$ Loc(l_{start})

PICK(l):
pre:
Loc(l)
BLoc(l)
$\neg$ Holds()
eff:
$\neg$ BLoc(l)
Holds()

PLACE(l):
pre:
Loc(l)
Holds()
eff:
BLoc(l)
$\neg$ Holds()

The **eff** clause specifies the fluents that will result from performing the operation if the conditions in the **pre** clause are true before it is executed. There is an grounded action instance of these schemas for each possible binding of parameters.

The temporal planning problem can be represented as a tuple $T = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ where $\mathcal{V}$ is a set of state variables; $\mathcal{A}$ is a set of durative actions; $\mathcal{I}$ is the initial state as a set of fluents; and $\mathcal{G}$ is the goal as a set of fluents [16]. In our example, the initial state is Loc(l_a) $\wedge$ Bloc(l_a) and the goal is Bloc(l_b). The most common metric for plan quality in temporal planning problem is the makespan, which is the time it would take to execute the plan.

Partial-Order Planning (POP), also known as Partial-Order Causal Linking (POCL), is a planning approach that uses a least-commitment philosophy in

which it postpones decisions about action orderings and parameter bindings until a decision is forced. In postponing the decisions about action orderings, only the essential ordering decisions are stored and plans are represented as a partially ordered sequence instead of committing prematurely to a complete totally ordered sequence of actions. Parameter binding decisions are also delayed so action parameters are only instantiated as required for causal links [20].

Definition 3. *(Causal Link) A causal link is a relationship between two actions, a_i and a_j, meaning that a precondition of a_j ($q \in PRE(a_j)$) is supported by an effect of a_i ($q \in EFF(a_i)$). This is represented by $a_i \xrightarrow{q} a_j$.*

Definition 4. *(Partial-Order Plan [20]) A partial-order plan can be represented as a tuple $\Pi = \langle \alpha, \mathcal{C}, \mathcal{L} \rangle$ where α is the set of durative actions in the plan, $\mathcal{C}$ is the set of ordering constraints ($\prec$) on α, and $\mathcal{L}$ is the set of causal links.*

Classical partial-order planners search through a tree where each node represents a partial-order plan and edges represent plan refinement operations such as the addition of an action to the plan [20]. As each node represents a plan, the space that the search is conducted through is known as plan space. The planner starts by creating a fictitious action a_{goal} whose preconditions are the fluents of the goal state and then creates a root node where the partial-order plan it represents contains only a_{goal} ($\Pi_0 = \langle \{a_{goal}\}, \emptyset, \emptyset \rangle$). The planner also has a fictitious action a_{init} whose effects are the fluents of the initial state. In this example, the precondition of a_{goal} would be $\mathrm{BLoc}(l_b)$ and the effects of a_{init} are $\mathrm{Loc}(l_a) \wedge \mathrm{Bloc}(l_a)$.

Definition 5. *(Flaw [10]) A flaw is something that prevents a partial-order plan from being a solution to the planning problem.*

At each stage of planning, a partial-order planner chooses a node in the search tree and determines what flaws that node has. The first type of flaw is an open link in which a precondition q of an action a_j in the partial-order plan does not have a causal link to support it. This can be resolved by creating a causal link from action a_i to action a_j over the fluent q ($a_i \xrightarrow{q} a_j$). The action a_i can either already be in the plan or be a new action that is added. For our example, $\mathrm{BLoc}(l_b)$ is an open condition of a_{goal} and so $\mathrm{PLACE}(l_b)$ could be added to the plan through a causal link over the fluent $\mathrm{BLoc}(l_b)$.

Definition 6. *(Threat) A threat is a conflict between an action, a_k, added to the plan and a causal link ($a_i \xrightarrow{q} a_j$). This conflict is created when an effect of the action a_k modifies a fluent away from the value needed for the casual link and there is no temporal relationship between a_k and either a_i or a_j that would prevent it from doing so.*

The second type of flaw is a threat. A threat from action, a_k on causal link, $a_i \xrightarrow{q} a_j$, can be resolved through either adding an ordering constraint $a_k \prec a_i$, known as a promotion, or $a_j \prec a_k$ known as a demotion.

In our example, MOVE(l_a, l_b) can be added to the plan with a causal link to PLACE(l_b) over the fluent Loc(l_b) and then PICK(l_a) can be added with a causal link to PLACE(l_b) over the fluent Holds(b_1). Next, a_{init} could be added to the plan through a causal link to PICK(l_a) over the fluent Loc(l_a). However, this causes a threat as there is no constraint between PICK(l_a) and MOVE(l_a, l_b) and one of MOVE(l_a, l_b)'s effects, $\neg$ Loc(l_a), contradicts the causal link between a_{init} and PICK(l_a). In order to resolve this MOVE(l_a, l_b) can be promoted so that it is constrained to happen after PICK(l_a).

Partial-order planners search by heuristically determining which node in the tree of partial plans to expand, which flaw to resolve for that node, and then how to resolve the selected flaw. The result of the resolution creates a new node in the tree. A solution is found when a partial-order plan is created that contains a_{init} and there are no remaining flaws. That is, all actions' preconditions have been satisfied through causal links and there are no remaining threats. As such, POP performs a plan-based, backward search, refining partial-order plans through the addition of actions, causal links, and ordering constraints [16].

3 Forward Chaining with Partial-Order Planning (FCPOP)

As temporal planning has become a more central interest of the planning community, investigation began into ways to improve the speed of POP due to its abilities to create plans that are more temporally flexible and to better handle concurrent actions than state based planners. Combining partial-order planning with grounded forward chaining has shown to maintain some of the benefits of POP while exploiting the speed and informed state heuristics of forward chaining.

Definition 7. *(Schedule) A schedule is an assignment of starting and finishing times to each action in a plan.*

Definition 8. *(Frontier State) A frontier state is a world state that can be created by generating a schedule for a partial-order plan and applying the actions at the times specified.*

POPF [4] combines POP and grounded forward chaining by creating a frontier state at each step of the plan to determine the applicable actions and to use for a temporal state-based heuristic. POPF utilizes a relaxed version of the least commitment philosophy as the grounding removes the delayed parameter binding and then the delayed action ordering is relaxed as the frontier state is used to add temporal constraints over actions, thus ignoring alternative choices that would come before it. OPTIC [3], the latest version of POPF, has been demonstrated to be one of the most effective planners in many domains due to its speed at generating successor states during search and the use of effective temporal domain-independent heuristics.

FLAP [16, 17] also combines POP and grounded forward chaining, but fully utilizes POP's delayed action ordering. In FLAP, the frontier state is not used for determining what actions are applicable and thus new actions can be added anywhere in the plan.

FCPOP is our variant of combining forward chaining with partial-order planning. FCPOP removes the delayed parameter binding of POP by grounding the actions before search, but fully utilizes the delayed action ordering commitment from POP and does not create a frontier state for determining which actions can be added to a plan. This allows actions to be added to any point in the plan. As such, the forward search can be viewed as a commitment to a set of actions and not to the order of their application [16].

FCPOP implements an A* search through plan space where each node of the tree is a partial-order plan. For each timed initial literal, a fictitious action a_{til} is created. This action starts at time 0 and lasts until the time at which the timed initial literal would happen. The action has no preconditions and a single effect for setting the specific fluent from the timed initial literal. FCPOP also creates fictitious actions for the initial state (a_{init}) and the goal state (a_{goal}). The search starts at the root node which contains a partial-order plan $\Pi_0 = \langle \{a_{init}\}, \emptyset, \emptyset \rangle$ with the fictitious action a_{init} and no causal links or ordering constraints. The processes of expanding a node for FCPOP is shown in Algorithm 1.

During expansion of a node Π, the fictitious action a_{goal} is checked for whether it can be added to the partial-order plan from the expanding node. If causal links can be created from the actions currently in Π to support the preconditions of a_{goal} then a new partial-order plan is created for each possi-

Algorithm 1: FCPOP Node Expansion

```
   Data: Π_e, The node to expand; A, The set of grounded actions
 1 if causal links can be created from the actions in Π_e to support action a_goal
     then
 2     P ← list of partial-order plans from adding a_goal to Π_e using Algorithm 2;
 3     if the length of P > 0 then
 4         Π_best ← null; c_best ← ∞;
 5         foreach partial-order plan Π ∈ P do
 6             if a temporally consistent STN can be computed from Π then
 7                 S ← the consistent STN computed from Π;
 8                 c ← the time taken to execute the plan as computed by S;
 9                 if c < c_best then
10                     Π_best ← Π; c_best ← c;
11         if Π_best ≠ null then
12             Π_best is the solution to the planning problem
13 foreach action a ∈ A do
14     P ← a list of partial-order plans from adding a to Π_e through Algorithm 2;
15     foreach partial-order plan Π ∈ P do
16         add Π as child node of Π_e;
```

ble combination of causal links that supports a_{goal}. Each of these partial-order plans is converted to a Simple Temporal Network (STN) [8] which will be explained below. From the STN, the fastest schedule that can be created from the new partial-order plan is calculated. The partial-order plan that can create the fastest schedule and now contains a_{goal} is the solution.

If a_{goal} cannot be added to the partial-order plan of Π then FCPOP determines which actions can be added. An action is only feasible to add if causal links can be created from the actions currently in Π to support the new action's preconditions. It is possible there are more than one combination of causal links that support adding the new action. For each of these combinations, the planner determines what threats are created by adding the action, and then if all of the threats are resolvable through ordering constraints. For each combination where all threats are resolvable, a new child node is created with the addition of the new action, the specific combination of causal links, and the ordering constraints needed to resolve the resulting threats.

The process of adding an action to a partial-order plan is shown in Algorithm 2. This process uses elements of POP in adding causal links and resolving threats through ordering constraints; however, it performs a forward chaining instead of backwards reasoning. Also, each node in the search tree, unlike traditional POP, is a flaw-free partial-order plan.

For each new node, a Simple Temporal Network (STN) [8], as a labeled directed graph, is created. In creating this STN, each durative action from the current partial-order plan is split into two instantaneous actions representing the start and end of the action. A vertex is created for each of these instantaneous actions. Edges are added between the start and end instantaneous actions from the same durative action for the lower and upper bounds on the action's

Algorithm 2: FCPOP Adding an action to a partial-order plan

```
   Data: Π, A partial-order plan; a, A durative action
   Result: P, A list of partial-order plans
 1 P ← [];
 2 if causal links can be created from the actions in Π to support a then
 3     L ← a list of the causal link combinations for a in Π;
 4     foreach causal link combination l ∈ L do
 5         T ←a list of the threats from adding a and l to Π;
 6         C ← [];
 7         foreach threat t ∈ T do
 8             if t can be resolved through an ordering constraint then
 9                 add the ordering constraint that resolves t to C;
10             else
11                 break;
12         if All threats were resolvable then
13             Π_new ← copy of Π;
14             add a, l, C to Π_new;
15             add Π_new to P;
```

duration. Edges are also added between an end instantaneous action and a start instantaneous action if there is an ordering constraint or causal link between them. Computing a shortest path from the vertex representing a_{init} identifies the earliest possible time at which the instantaneous action could occur. This can be used to determine a timestamp for each of the instantaneous actions and to create a schedule. If a negative cycle is found in the STN then the plan is inconsistent. In this case, the cost of the node is set to infinite and a heuristic is not calculated, so that the node is never be expanded. If the plan is consistent then the path cost for the node is the total time taken by executing the schedule. Simulating the effects of the actions in the schedule creates a frontier state which can be used for heuristic evaluation.

The frontier state is used as the initial state for a temporal state-based heuristic. FCPOP uses a modified variant of the Temporal Relaxed Planning Graph (TRPG) from [6] in which layers can have fluents that contain values from finite domains of mutually exclusive values. For these fluents, the label contains timestamps of the transition between values and a fact is assumed to simultaneously be all achieved values. This still relaxes the planning and the heuristic results in a relaxed plan. The amount of time needed to execute the relaxed plan is computed and used as the heuristic value for the current node.

During the search process it is possible for a node to be generated that has the same plan as a node already in the tree which is a common problem in most forward-search planners [16]. FCPOP uses a memoization technique to avoid this issue. As all the successors are generated for each expanded node, FCPOP's planning algorithm is sound and complete.

4 Forward Chaining with Hierarchical Partial-Order Planning

While classical and temporal planners focus on achieving a goal state, hierarchical planners focus on taking an initial plan of abstract tasks or actions and decomposing them into less abstract actions until they create a plan of actions that can no longer be decomposed. As part of the planning process the planners are provided with plan fragments, also known as methods, that are intended to implement a specific abstract action. These methods usually also contain further abstract actions, so substitutions of methods for abstract actions has to be performed recursively. This substitution step is called a decomposition of the abstract action and creates a corresponding hierarchy on the actions such that actions in the method are considered less abstract than the substituted abstract action. When an action cannot be decomposed any further it is called a primitive action, which corresponds to the action from classical or temporal planning. Generally, there are more than one possible method for an abstract action. Over time, this form of planning has been subsumed under the label Hierarchical Task Network (HTN) planning [18].

Recently elements from HTNs have been used to improve upon POP in what has become known as hybrid planning. DPOCL [23] introduced decomposition

of actions in the context of POP. This later inspired hybrid planners such as PANDA [18], HIEPPR-POP [13], and HiPOP [2]. Most hybrid planners allow adding an abstract action the same way traditional POP adds a primitive action. They also define a new type of flaw that needs to be resolved. This new flaw is a plan having an abstract action. They also add one or more resolvers for fixing this new type of flaw by decomposing the flaw in a similar way to a HTN. The heuristic used by the algorithm to determine which flaw to resolve at each step is also modified so that it considers this new type of flaw.

Definition 9. *(Abstract Action) An abstract action is the same as a primitive action with the addition of a set of partial-order plans, known as methods, that instantiate the abstract action.*

Forward chaining with hierarchical partial-order planning (FCHPOP) extends FCPOP to use hierarchical information. Previously described durative

Algorithm 3: FCHPOP Adding the method from an abstract action to a partial-order plan

Data: Π, A partial-order plan; m, A method from an abstract action
Result: $\mathcal{P}$, A list of partial-order plans

$\Pi_m \leftarrow$ the partial-order plan from m;
$\mathcal{A}_m \leftarrow$ the list of actions (both abstract and primitive) from Π_m;
$\mathcal{L}_m \leftarrow$ the list of causal links from Π_m;
$\mathcal{C}_m \leftarrow$ the list of ordering constraints from Π_m;
$\Pi_e \leftarrow$ copy of Π;
add $\mathcal{L}_m$ and $\mathcal{C}_m$ to Π_e;
sort $\mathcal{A}_m$ based on $\mathcal{C}$ and $\mathcal{L}$;
$\mathcal{P} \leftarrow []$;
add Π_e to $\mathcal{P}$;
for *action* $a \in \mathcal{A}_m$ **do**
    $\mathcal{L}_a \leftarrow$ a list of combinations of causal links to support the unsupported preconditions of a ;
    $\mathcal{P}_a \leftarrow []$;
    foreach *partial-order plan* $\Pi_i \in \mathcal{P}$ **do**
        foreach *causal link combination* $l \in \mathcal{L}_a$ **do**
            $\mathcal{T}_l \leftarrow$a list of the threats from adding a and l to Π_i;
            $\mathcal{C}_l \leftarrow []$;
            foreach *threat* $t \in \mathcal{T}_l$ **do**
                if *t can be resolved through an ordering constraint* **then**
                    add the ordering constraint that resolves t to $\mathcal{C}_l$;
                else
                    break;
            if *All threats were resolvable* **then**
                $\Pi_l \leftarrow$ copy of Π_i;
                add $a, l, \mathcal{C}_l$ to Π_l;
                add Π_l to $\mathcal{P}_a$;
    $\mathcal{P} \leftarrow \mathcal{P}_a$

Algorithm 4: FCHPOP Node Expansion

Data: Π_e, The node to expand; $\mathcal{A}_\mathcal{P}$, The set of grounded primitive actions; $\mathcal{A}_\mathcal{B}$, The set of grounded abstract actions

```
add primitive actions using Algorithm 1;
add abstract actions as if they were primitive action using Algorithm 1;
A_Π ← the abstract actions in Π_e;
foreach abstract action a ∈ A_Π do
    M ← the methods of a;
    foreach method m ∈ M do
        P ← a list of partial-order plans by adding m to Π_e using Algorithm 3;
        foreach partial-order plan Π ∈ P do
            add Π as a child node of Π_e
```

actions will be known as primitive actions for FCHPOP. FCHPOP also uses a higher-level actions known as abstract actions which can be refined into a partial-order plan of primitive actions.

FCHPOP, as with FCPOP, builds a tree of nodes that contain partial-order plans. The main difference is when FCHPOP expands a node it can, in addition to adding primitive actions to partial-order plans, add the highest-level abstract actions to them, and decompose already included abstract actions as shown in Algorithm 4. FCHPOP still adds primitive actions as part of the expansion in order to maintain the ability to solve problems that cannot be solved solely with the highest level abstract actions.

An abstract action can have multiple methods. The abstract action can be refined into each of these methods. Adding a specific method to a partial-order plan is shown in Algorithm 3. Each method m contains a partial-order plan Π_m containing actions (both abstract and primitive), causal links, and ordering constraints. When a method m is added to a partial-order plan Π, the causal links and ordering constraints from Π_m are added to Π. The list of actions from Π_m are sorted based on the causal links and ordering constraints. Each action is added to Π in a similar manner to Algorithm 2. The main difference is an action originally from Π_m may have preconditions that are supported by the causal links from Π_m and so new causal links to support these preconditions are not created. Adding these actions may still cause threats upon the causal links that were originally from Π and so new ordering constraints may need to be added. Each of the partial-order plans generated from this process is added a child node to Π. The path cost and heuristic for these nodes are calculated in the same way as for FCPOP using the STN and modified TRPG.

5 Evaluation

In order to evaluate the performance of FCPOP, we use eight of the nine benchmarks from the 2018 International Planning Competition's (IPC) temporal track [5] and compare the results of FCPOP against state-of-the-art temporal planners in OPTIC and TFLAP, the version of FLAP used for the 2018 IPC's tempo-

Domain	FCPOP	TFLAP	OPTIC
Airport	9	9	3
Cushing	7	3	10
Floortile	6	3	0
Mapanalyser	9	8	0
Parking	10	10	8
Quantum Circuit	9	8	8
Sokoban	5	4	3
Trucks	10	10	10
Total (80)	65	55	42
Coverage	79.2%	68.75%	52.5%

Table 1. Number of problems solved for each domains

ral track. The *road traffic accident* domain was removed as neither OPTIC nor TFLAP completed any of the problems for it in the competitions. Each domain has 10 problems, so there are 80 benchmark problems in all. As with the 2018 IPC competition, each planner was given 30 minutes and 8 GB of available memory to utilize for each experiment.

Table 1 shows how many problems each of the three planners (FCPOP, TFLAP, and OPTIC) solved for each domain as well as the total number of problems that each planner solved. Figure 1 shows how the plans created by TFLAP and OPTIC compared to the plans created by FCPOP in terms of makespan. In order to make the data easier to understand, the difference in the makespan of the plan created by either OPTIC or TFLAP and the makespan of the plan created by FCPOP is displayed. As such, $y = 0$ corresponds to the makespan of plans created by FCPOP and points above $y = 0$ are plans created by TFLAP or OPTIC with longer makespans and below are plans they created with shorter makespans. Figure 2 shows the planning speed of TFLAP and OPTIC compared to the planning speed of FCPOP. The difference in the time taken to plan by TFLAP or OPTIC and the time taken to plan by FCPOP is shown. $y = 0$ corresponds to the planning time of FCPOP and points above $y = 0$ are problems in which TFLAP or OPTIC were slower and below are problems in which they were faster. For both figures, only problems that FCPOP and at least one other planner solved are shown.

As can be observed in Table 1, OPTIC solved more problems in the *cushing* domain than FCPOP; however, FCPOP solved the same number or more problems in all other domains and solved more problems overall. Also on problems that both FCPOP and OPTIC solved, FCPOP created plans with a better makespan on 66.5% of them and made plans that on average had 83.6% of the makespan of the ones that OPTIC created. These results are likely due in part to FCPOP using a non-relaxed delayed action ordering which makes FCPOP more flexible in how it constructs plans. As it can add an action anywhere in the plan, as opposed to only after the frontier state, this allows for less backtracking and plans with shorter makespans.

OPTIC uses enhanced hill climbing (EHC) as its primary search algorithm, which lets it create a fast, but often lower quality first plan [16]. If EHC cannot find a solution then it switches to a best first search, which unlike FCPOP's A*

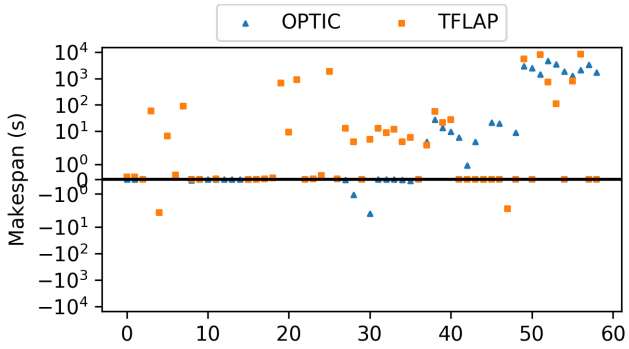

Fig. 1. $y = 0$ corresponds to the makespan of plans created by FCPOP and points above $y = 0$ are plans created by TFLAP or OPTIC with longer makespans and below are plans they created with shorter makespans.

search, does not take the cost of a node into account when deciding if it should be expanded. This results in nodes that are heuristically close to the goal to be expanded even if the cost of the corresponding partial-order plan is large.

While FCPOP created better plans more than half of the time, OPTIC was faster for 69.2% of the problems that both planners solved. This is likely due to extra computation time being needed by FCPOP in expanding a node. First, OPTIC can check whether the precondition of an action holds in the frontier much quicker than FCPOP can determine if the action can be placed anywhere in the plan. Second, OPTIC does not need to check for and resolve threats when adding a new action, as the new action is placed after any other actions that it could have conflicts with. Third, the number of actions that can be supported by a frontier state is fewer than the number of actions that be supported throughout a partial-order plan and so OPTIC's branching facter is smaller. These all lead to FCPOP taking about 2.5x the amount of time OPTIC took to plan on average.

In all domains, FCPOP solved the same number or more problems than TFLAP resulting in FCPOP solving more problem overall. Also, FCPOP created

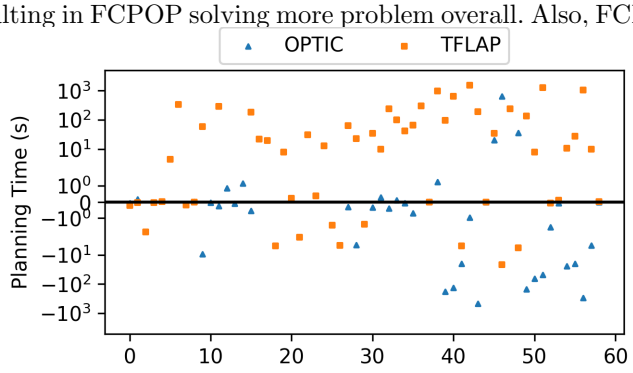

Fig. 2. $y = 0$ corresponds to the planning time of FCPOP and points above $y = 0$ are problems in which TFLAP or OPTIC were slower and below are problems in which they were faster.

a plan for 94.5% of the problems that TFLAP solved. While TFLAP similarly does not relax the delay action ordering, it does not use a temporal heuristic and uses a different algorithm for scheduling. TFLAP uses heuristics and path costs that are based on the number of actions in the plan. This can lead to TFLAP creating a plan with fewer actions, but a longer makespan. Also, due to a lack of temporal information in its search guidance, domains with dead-ends cause TFLAP to encounter plateaus and get stuck in areas of the plan space. FCPOP was also faster than TFLAP on 70.9% of the problems that both solved. FCPOP took on average 86.4% of the time that TFLAP took to solve a problem. This also is likely due to FCPOP having a temporal heuristic that allows it a more guided search and lets it search less of the space before finding a solution.

FCHPOP was tested on the *satellite* domain first introduced in the 2002 IPC [18], the *survivors* domain presented in [2], and the *turn and open* domain which was in the 2011 IPC [1]. For each of the three domains a random generator was used to create a non-hierarchical version and a hierarchical version of the domain and problem.

The *satellite* domain involves gathering data from a number of scientific imaging instruments on a number of satellites and sending the data to earth. The random generator varies the number of satellites, instruments, stars, planets, interesting phenomena and imaging modes. The hierarchical actions allow for domain knowledge to be given to the planner and for it to use premade procedures for sets of actions that would be repeated.

The *survivors* domain involves sending teams of robots to rescue survivors of a natural disaster and bring them to hospitals. In this domain, the hierarchical actions allow for planning for the teams as opposed to planning for the individual robots and for zones instead of individual locations. The random generator varies the numbers of teams, hospitals, and survivors as well as the positions of the hospitals and the survivors.

The *turn and open* domain has a number of robots with two gripper hands, multiple rooms with balls that need to be transported, and doors that must be opened between the rooms. The random generator varies the number of robots, balls, and rooms, which adjacent rooms have doors connecting them, and whether doors are open or closed in the initial state.

For each domain, 20 problems were randomly generated. For each problem FCHPOP planned for the hierarchical version of the problem and FCPOP planned for the problem with just the primitive actions. These experiments were all given 30 minutes to plan and 8 GB of memory.

FCHPOP on average took 28.1% of the time that FCPOP did for the same problem. FCHPOP also explored 20.8% of the number of nodes and visited 9.2% of the number of nodes that FCPOP did. Using the abstract actions significantly reduces the number of nodes the FCHPOP needs to visit and explore to solve the problem, which in turn reduces the amount of time need to solve the problem.

One of the reasons for this is that abstract actions provide domain specific information about how a procedure could be done. In a two level hierarchy such as in the *satellite* domain, adding and refining an abstract action involves two

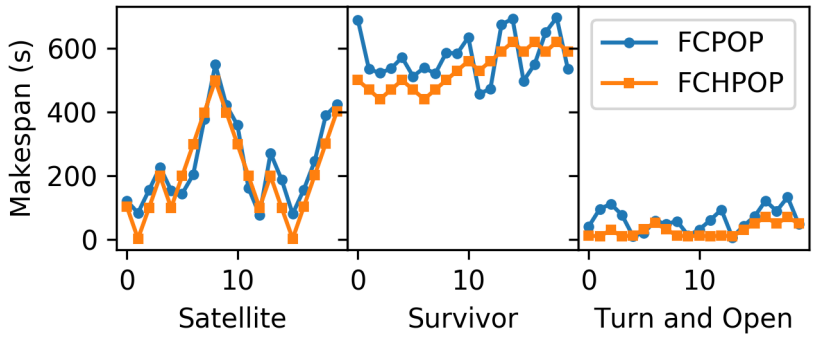

Fig. 3. Comparison of FCPOP and FCHPOP in terms of plan makespan

node expansions, whereas creating the same effect on the plan without abstract action would require an expansion per action to be added. This effect compounds if an abstract action is added multiple times. In the satellite domain for instance many of the plans repeatedly turn a satellite towards a target, calibrate an image instrument, collect an image, point towards Earth, and then send the data to Earth. For each time this is done, FCPOP has to expand four nodes whereas FCHPOP only needs to expand two. With different combinations of hierarchical levels and number of actions in a method, the ratio of nodes FCHPOP to nodes FCPOP need to expand to solve the same subproblem changes, but generally FCHPOP needs fewer expansions. This is an advantage of providing expert domain knowledge. The drawback to this is that if there is a better way to solve a subproblem than the methods in FCHPOP's abstract action then FCHPOP may ignore the better way to solve it and use the precomputed methods of an abstract action. This led to FCPOP creating plans with better makespans in some of the problems.

Another reason that FCHPOP was generally faster than FCPOP in the *survivors* domain was using the hierarchy of the abstract actions to plan for teams of robots instead of individual robots and for zones instead of individual locations. By creating the hierarchy in this way teams of robots could share a common goal and move together without needed to compute each individual robot's ac-

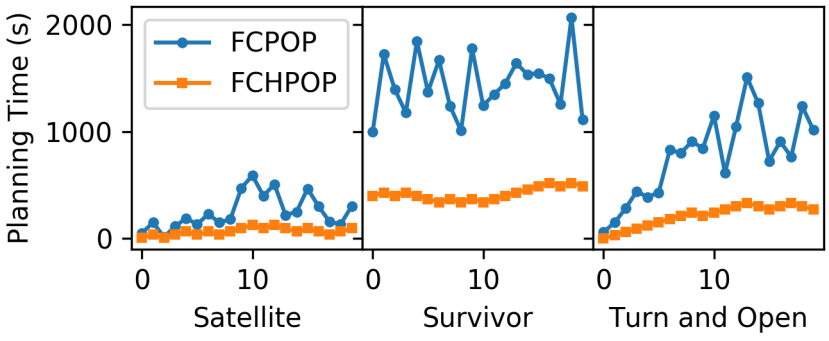

Fig. 4. Comparision of FCPOP and FCHPOP in terms of planning time

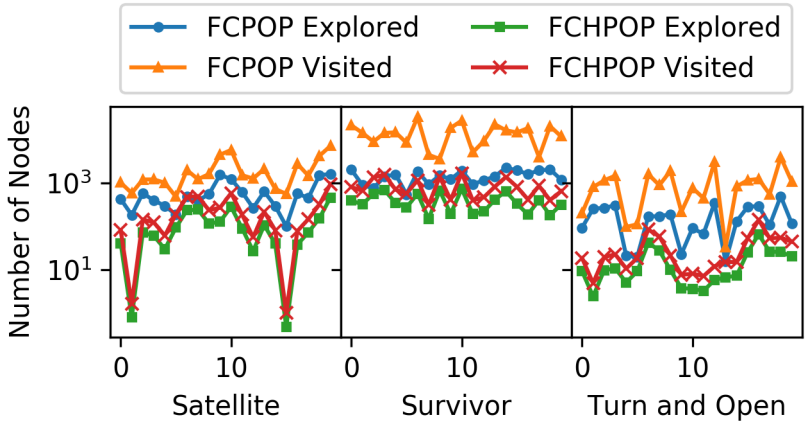

Fig. 5. Comparison of FCPOP and FCHPOP in terms of number of nodes visited and explored

tions separately. This combined with the previously mentioned procedure led to significant speed up in the speed of planning as well as reducing the number of nodes visited and explored.

6 Discussion

We presented two new planning algorithms in FCPOP and FCHPOP, which generate time flexible plans with concurrent actions. FCPOP combines grounded forward chaining with partial-order planning, fully utilizes the delayed action ordering component of the least-commitment philosophy, and uses a temporal state-based heuristic in the Temporal Relaxed Planning Graph. This results in our planner being able to create plans with shorter makespans while still being fast. FCPOP is shown empirically to make plans with equal or shorter makespan than state-of-the-art temporal planners most of the time. FCHPOP builds upon FCPOP by adding hierarchical information in the form of abstract actions that can be recursively refined. This is shown to speed up the planning process and reduce the number of nodes that need to be visited and explored, all while still generating quality plans.

There are several ways to build on this work including executing the hierarchical partial-order plan on teams of robots, repairing the plan to overcome execution failures, adding components to improve scalability in the number of robots that can be quickly planned for, and learning from plans generated by FCPOP to create abstract actions for FCHPOP.

References

1. International Planning Competition 2011 Temporal Tracks. URL http://www.plg.inf.uc3m.es/ipc2011-deterministic/DomainsTemporal.html

2. Bechon, P., Barbier, M., Infantes, G., Lesire, C., Vidal, V.: HiPOP: Hierarchical Partial-Order Planning. Tech. rep. (2014)
3. Benton, J., Coles, A., Coles, A.: Temporal planning with preferences and time-dependent continuous costs. In: 22nd International Conference on Automated Planning and Scheduling, pp. 2–10 (2012)
4. Coles, A., Coles, A., Fox, M., Long, D.: Forward-chaining partial-order planning. In: 20th International Conference on Automated Planning and Scheduling, pp. 42–49. AAAI (2010)
5. Coles, A., Coles, A., Martinez, M.: International Planning Competition 2018 Temporal Tracks. URL https://ipc2018-temporal.bitbucket.io/
6. Coles, A.A., Coles, A.A.: A temporal relaxed planning Graph heuristic for planning with envelopes. In: Proceedings International Conference on Automated Planning and Scheduling, pp. 47–55. AAAI (2017)
7. Dantam, N.T., Kingston, Z.K., Chaudhuri, S., Kavraki, L.E.: An incremental constraint-based framework for task and motion planning. International Journal of Robotics Research **37**(10), 1134–1151 (2018)
8. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence **49**(1-3), 61–95 (1991)
9. Garrett, C., Lozano-Perez, T., Kaelbling, L.: FFRob: Leveraging Symboling Planning for Efficient Task and Motion Planning
10. Ghallab, M., Nau, D., Traverso, P.: Automated Planning and Acting. Tech. rep. (2016)
11. Kvarnström, J.: Planning for Loosely Coupled Agents Using Partial Order Forward-Chaining. In: 21st International Conference on Automated Planning and Scheduling
12. Lallement, R., de Silva, L., Alami, R.: HATP: An HTN Planner for Robotics (2014)
13. Lee-Urban, S.: Hierarchical Planning Knowledge for Refining Partial-Order Plans (2012)
14. Pack Kaelbling, L., Lozano-Pérez, T.: Hierarchical Task and Motion Planning in the Now *. Tech. rep.
15. Penberthy, J.S., Weld, D.: UCPOP: A Sound, Complete, Partial Order Planner for ADL. Proc. KR-92 (1992)
16. Sapena, O., Onaindia, E., Torreño, A.: FLAP: Applying Least-Commitment in Forward-Chaining Planning
17. Sapena, O., Torrenõ, A., Onaindiá, E.: Parallel heuristic search in forward partial-order planning. Knowledge Engineering Review **31**(5), 417–428 (2016)
18. Schattenberg, B.: Hybrid Planning and Scheduling. Ph.D. thesis (2016)
19. Vidal, V., Geffner, H.: Branching and pruning: An optimal temporal POCL planner based on constraint programming. In: Proceedings of the National Conference on Artificial Intelligence, pp. 570–577 (2004)
20. Weld, D.S.: An Introduction to Least Commitment Planning. AI magazine **15**(4), 27–27 (1994)
21. Wolff, E.M., Topcu, U., Murray, R.M.: Automaton-guided controller synthesis for nonlinear systems with temporal logic. In: IEEE International Conference on Intelligent Robots and Systems, pp. 4332–4339 (2013)
22. Younes, H.L., Simmons, R.G.: VHPOP: Versatile heuristic partial order planner. Journal of Artificial Intelligence Research **20**, 405–430 (2003)
23. Young, R.M., Moore, J.D.: DPOCL: A Principled Approach to Discourse Planning. Proceedings of the 7th International Workshop on Natural Language Generation pp. 13–20 (1994)