Finding concise plans: Hardness and algorithms

Jason M. O'Kane and Dylan A. Shell

***Abstract*— This paper addresses the problem of generating the simplest plans that solve robotic planning problems. Most robotic planning algorithms are concerned with producing plans that minimize execution cost, or generalizations of such costs. Motivated by circumstances with severe computational resource limits (*e.g.,* memory or communication constrained settings), we instead address the problem of producing *concise* plans. In this work, conciseness is a measure of plan size that reflects the complexity of representing the plan explicitly. We seek a plan with minimal representational size, subject to correctness and completeness. We introduce a planning algorithm that generates concise plans for planning problem that may involve both non-determinism and partial observability, and also show that finding the most concise plan is an NP-hard problem, excusing the possible sub-optimality of our algorithm's output. We describe an implementation of the algorithm, along with empirical results on the run time and solution quality for both manipulation and navigation problem domains.**

I. Introduction

Broadly speaking, autonomous task-oriented behavior requires robots to select and execute actions on the basis of the limited information available to them. This information includes the history of what has been sensed, the history of actions executed in the past, and any prior knowledge that might be available. The overwhelming majority of existing work in robotic planning seeks plans that optimize some measure (such as time, energy consumption, or safety) of a plan's execution cost. This paper considers an orthogonal view of the planning process, in which the objective is to optimize *expression complexity* of the generated plans. The underlying question is "What is the most concise plan the solves a given planning problem?"

At least three factors motivate a search for concise plans:

1) In situations where robots have severe memory limitations (such as those stemming from extremely small space, weight, or energy budgets), finding a concise plan may be more imperative than finding one whose execution cost is low.
2) Plan size is also important when the plan is being relayed over a noisy channel. This case may be familiar to anyone who has communicated driving directions to another human: A common strategy is to provide instructions that minimize the number of turns, in lieu of instructions that follow a faster but more complex route.
3) Finally, understanding the size and structure of concise plans for a given family of problems may provide valuable insights into those problems. As a simple example, one might assess the value of a particular sensing or actuation primitive by comparing the size of the most concise plans that respectively use or omit that capability.

Jason M. O'Kane (`jokane@cse.sc.edu`) is with the Department of Computer Science and Engineering, University of South Carolina, Columbia, South Carolina, USA. Dylan A. Shell (`dshell@cse.tamu.edu`) is with the Department of Computer Science and Engineering, Texas A&M University, College Station, Texas, USA.

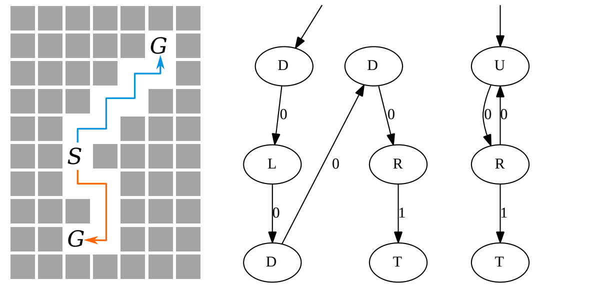

Fig. 1: [left] A planning problem in which a robot with a goal detector moves from S to G. [middle] A plan graph for this problem that minimizes execution time. [right] A plan graph for this problem of minimal size.

An illustrative example of this concept, in the context of an idealized robot moving on a grid and in possession of a goal detection sensor, is shown in Figure 1. The robot's goal is to travel from its starting location (marked 'S') either of the two goals (marked 'G'). In this case, the plan with the smallest execution time travels directly to the lower goal. However this plan—informally, "Down, left, down, down, right, stop"—is more complex than the alternative that travels to the upper goal using a plan informally expressed as "Alternate up and right until reaching the goal." The objective of this paper is formalize this idea, and to investigate methods for generating concise plans that solve a very general class of robotic planning problems.

After reviewing related work (Section II) and formally defining the concise planning problem (Section III), this paper makes two major contributions. First, we prove in Section IV that that the problem of finding the *smallest* plan that solves a given problem is NP-hard. The proof, which uses a reduction from the problem of 3-coloring a graph, extends and generalizes the authors' earlier result on filter minimization [12].

Second, we present in Section V an algorithm that rapidly generates concise plans, albeit without any guarantee of optimality. Our approach involves two pieces: (*i*) reduction of a given plan to express it as concisely as possible, and (*ii*) an incremental search for plans that exploits structure in the solution space by reusing sub-parts of plans. The planning process is similar Dijkstra's algorithm in the sense that it constructs a sequence of sub-plans designed to reach the goal from vertices that are increasingly distant from the goal. A key complication is that a globally concise plan may result from sub-plans that, for their sub-problems, are not maximally concise. As a consequence, our algorithm stores a collection of candidate sub-plans at each vertex, including separate containers for plans that are themselves concise (a local criterion) and plans than score well on a heuristic estimate of their reusability (a global criterion). The number of plans associated with each vertex is bounded by an algorithm parameter that encodes a tradeoff between solution quality and the time and memory consumed by the planner.

A further important difference is that, because constructing and reducing a sub-plan is a relatively expensive process, it behooves one to leverage that effort as much as possible. Our algorithm accomplishes this by associating each generated sub-plan with *all* of the I-state graph vertices for which it is a correct solution. In this way, partial solutions are treated as first-class objects, and there is a many-to-many relationship between sub-plans and I-state graph vertices.

Section VI describes an implementation of this algorithm and shows its effectiveness an a collection of planning problems, including instances of both manipulation and navigation problems. The paper then concludes with a discussion of future work in Section VII.

II. Related Work

The idea of understanding problems by examining the representational complexity needed to solve them can be traced at least as far Kolmogorov's definition of the complexity of a sequence in terms of size of the smallest problem that outputs that string [10]. Another family of well-known results considers the "power" of various sensors, such as abstract compasses [2] and pebbles [1], for exploration tasks. In that work, the power of a sensor is measured in terms of the amount of memory (finite, finite augmented with a single counter, *etc.*) required for an agent to explore its environment using that sensor.

The class of planning problems we consider in this paper is equivalent to the class of nondeterministic graphs that appears in Erdmann's recent topological conditions on the existence of plans that succeed in such graphs [5]. Such graphs, commonly represented as AND-OR graphs, have received attention by AI researchers employing heuristics to find a solution to reach a goal [3], [9]. The results we present here are orthogonal, in the sense that our results are algorithmic, and focused constructing on optimally concise, rather than merely extant, plans. Likewise, the kinds plan graphs we use here to represent the robot's strategy as a finite state machine have also been used in the context of POMDPs [8].

As discussed in the introduction, prior work by the authors addressed the related problem of *filter reduction*. Given an I-space partition, the goal is to find the smallest finite state machine that maintains enough information to identify the cell in the partition that a robot's current I-state resides in. We showed that this problem is NP-complete and provided an algorithm for solving it efficiently. The intuition of that algorithm is to compress filters by recognizing vertices that must remain distinct in an correct filter and forcing them to be separated, but permitting any others to be merged. Generating a partition that obeys these constraints becomes a graph vertex coloring problem where vertices which have the same color are identified, forming a more concise expression of the given filter. This algorithm is used as a subroutine in Algorithm 3 to reduce candidate sub-plans by treating them as filters.

III. Definitions and Problem Formulation

We consider problems in which a robot interacts with its environment by executing *actions*, selected from a finite *action space* U. We assume that the action space contains a special *termination action* u_{T}, which signals that the robot has completed its execution. In response to each action, the robot receives an *observation*, selected from a finite *observation space* Y, from its sensors.

A. Information state graphs

The robot may have *prediction uncertainty*—that is, uncertainty about the results of its actions—and *sensing uncertainty*—uncertainty arising from incomplete sensor data, along with uncertainty about its initial conditions. We encapsulate all three forms of uncertainty using the *information space (I-space)* formalism, which was codified by LaValle [11]. This approach uses the term *information state (I-state)* to refer to any representation of the (generally incomplete) knowledge available to the robot. As the robot executes actions and receives observations, it can update its current I-state to reflect new knowledge that can be inferred from those events.

In discrete time systems in which both the action space and observation space are finite, including the systems we consider in this paper, we can model the progression of I-states as a walk on an I-state graph.

Definition 1: An I-state graph $\mathbf{I} = (V_{\mathrm{u}} \cup V_{\mathrm{y}}, E_{\mathrm{u}} \cup E_{\mathrm{y}})$ *is a bipartite directed multigraph in which*

1) *the vertex set, of which each member is called an* I-state *, can be partitioned into a set of* action vertices V_{u} *and a set of* observation vertices V_{y},
2) *the edge set can be partitioned into a set of* action edges $E_{\mathrm{u}} \subseteq V_{\mathrm{u}} \times V_{\mathrm{y}}$ *and a set of* observation edges $E_{\mathrm{y}} \subseteq V_{\mathrm{y}} \times V_{\mathrm{u}}$,
3) *each action edge* e *is labeled with an action* $u(e)$,
4) *each observation edge* e *is labeled with an observation* $y(e)$, *and*
5) *no pair of distinct edges (neither action edges nor observation edges) share both a source vertex and a label.*

B. Plan graphs

Given an I-state graph, we can trace the evolution of the robot's I-state by following the appropriately labeled edge each time the robot executes an action. Notice that this formulation does not require the I-state graph to be "complete," in the sense that each action vertex does not necessarily have an out-edge for each action in the action space; those missing actions are considered "illegal" from those I-states. Likewise, an observation node need not have out-edges for each observation in the observation space, which can occur if the underlying structure of the problem dictates that certain observations cannot occur from a given I-state.

Note that, because we are interested plans that succeed even in the worst case, we do not attach any probability models to the observations; any observation for which an observation edge exists is considered possible, and all such observations are treated equally by our algorithms. We discuss the potential for probabilistic extensions in Section VII.

Definition 2: A planning problem *is a 3-tuple* $\mathcal{P} = (\mathbf{I}, v_{\mathrm{s}}, V_{\mathrm{g}})$, *in which* $\mathbf{I} = (V_{\mathrm{u}} \cup V_{\mathrm{y}}, E_{\mathrm{u}} \cup E_{\mathrm{y}})$ *is an I-state graph,* $v_{\mathrm{s}} \in V_{\mathrm{u}}$ *is called the* start node *and* $V_{\mathrm{g}} \subseteq V_{\mathrm{u}}$ *is called the* goal region.

The objective is to generate a strategy that, when executed starting from v_{s}, will terminate at some I-state in V_{g}, regardless of the observations received along the way. Such strategies, which operate in discrete time and with finite memory, are naturally expressed as transition graphs.

Definition 3: A plan graph $\mathbf{P} = (V_{\mathrm{p}}, E_{\mathrm{p}})$ *is a directed graph in which*

1) *one vertex* $v_{\mathrm{s}} \in V_{\mathrm{p}}$ *is designated as a* start plan vertex.
2) *each vertex* $v \in V_{\mathrm{p}}$ *is labeled with an action* $u(v) \in U$,
3) *each edge* $e \in E_{\mathrm{p}}$ *is labeled with an observation* $y(e) \in Y$, *and*
4) *no pair of distinct edges share both a source vertex and a label.*

To execute the plan described by such a graph the robot should, starting from v_{s}, execute the action $u(v_{\mathrm{s}})$, and then follow the edge corresponding to the observation received. This process repeats until:

1) The plan attempts to execute an action that is not allowed at the robot's current I-state (indicated by the absence of the corresponding edge in the I-state graph), or the plan lacks an edge labeled with the robot's observation outgoing from its current vertex. In either case, the result of the plan for that execution is a *failure*.
2) The plan executes action u_{T}. In this case, the plan's execution is a *success* if the current I-state is a member of the goal region, or a *failure* otherwise.

We are interested in plan graphs that succeed in the *worst case* for a given planning problem:

Definition 4: A plan graph $\mathbf{P}$ solves *a planning problem* $\mathcal{P} = (\mathbf{I}, v_{\mathrm{s}}, V_{\mathrm{g}})$ *if there exists an integer* k*, such that every execution of* $\mathbf{P}$ *successfully terminates in* V_{g} *after at most* k *steps.*

Finally, notice that the size of a plan graph is a direct indicator of the plan's conciseness. This motivates the core problem addressed in this paper:

Problem: Concise Planning (CP)

Input: A planning problem $\mathcal{P}$.
Output: A plan graph $\mathbf{P}$ that solves $\mathcal{P}$, such that the number of vertices in $\mathbf{P}$ is minimal.

IV. Hardness of Concise Planning

In this section, we prove that the concise planning problem introduced in Section III is NP-hard. In keeping with the usual practice in complexity theory, our approach starts from the related decision problem:

Decision Problem: Concise Planning (CP-DEC)

Input: A planning problem $\mathcal{P}$ and an integer k.
Output: *True* if there exists a plan graph $\mathbf{P}$ of at most k vertices that solves the $\mathcal{P}$; *False* otherwise.

We show that CP-DEC is NP-complete, which directly implies that CP is NP-hard. To accomplish this, we first show that CP-DEC is in class NP (Section IV-A), and then show, via reduction from a graph coloring problem, that CP-DEC is NP-hard (Section IV-B).

A. Concise Planning is in NP

To show that CP-DEC is in NP, it is sufficient to find a polynomial-time algorithm that determines, given a planning problem $\mathcal{P}$, an integer k, and a plan graph $\mathbf{P}$, whether (i) G has at most k nodes and (ii) G solves the planning problem. The former condition requires a simple count of the vertices. A technique to check the latter condition appears as Algorithm 1.

The intuition of the algorithm is to enumerate all reachable I-state/plan node pairs via a forward search, and to return True only if the set of reachable pairs is exhausted without finding any failures or incorrect terminations.

It is straightforward to see that, for each iteration of the outer while loop, the algorithm does work bounded by the number of observations. The outer while loop can perform no more than one iteration per unique pair of plan and I-state graph vertices, and therefore the whole algorithm has time-complexity in $O(|Y||V_u||V_p| + |Y||V_y||V_p|)$. Because this algorithm exists and executes in polynomial time, we have the desired result.

Lemma 1: CP-DEC *is in complexity class NP.*

B. Concise Planning is NP-complete

To show that CP-DEC is NP-complete, we next present a reduction from the standard problem of 3-coloring a graph:

Decision Problem: Graph 3-Coloring (GRAPH-3C)

Input: An undirected graph $\mathbf{G}$.
Output: *True* if there exists coloring of $\mathbf{G}$ using at most three colors, such that no pair of adjacent vertices shares the same color; *False* otherwise.

This problem is known to be NP-complete [4], so it only remains to give a polynomial time reduction from GRAPH-3C to CP-DEC. Given an instance of GRAPH-3C, namely an

Algorithm 1 Verify Plan Correctness

Input:

A problem $\mathcal{P} = (\mathbf{I}, v_{\mathrm{s}}, V_{\mathrm{g}})$ and a plan graph .

Output:

True if $\mathbf{P}$ solves $\mathcal{P}$; False otherwise.

$Q \leftarrow$ empty queue
$Q.\mathrm{insert}(v_{\mathrm{s}}, v_{\mathrm{s}}(\mathbf{P}))$
while Q is not empty **do**
  $(v_i, v_p) \leftarrow Q.\mathrm{pop}()$
  if (v_i, v_p) is its own ancestor **then**
    return False {Plan may never terminate.}
  end if
  if $u(v_p) = u_{\mathrm{T}}$ **then**
    if $v_i \notin V_{\mathrm{g}}$ **then**
      return False {Plan terminates outside of goal.}
    end if
  else
    for each out edge $v_i \xrightarrow{y} v_i'$ of v_p **do**
      if $\mathbf{P}$ has an edge $v_p \xrightarrow{y} v_p'$ and (v_i', v_p') has not be inserted into Q yet **then**
        $Q.\mathrm{insert}(v_i', v_p')$
      else
        return False {Plan is not prepared for observation.}
      end if
    end for
  end if
end while
return True {No incorrect terminations or failures.}

undirected graph $\mathbf{G} = (V, E)$, we construct an instance $(\mathbf{I}, v_{\mathrm{s}}, V_{\mathrm{g}}), k$ of CP-DEC with the following elements in $\mathbf{I}$:

1) A starting action node v_{s}.
2) An observation node v_1 and an edge $v_{\mathrm{s}} \xrightarrow{u_0} w_s$ connecting it to v_{s}.
3) For each vertex a of $\mathbf{G}$, an action node v_a, and observation node w_a, and edges $w_1 \xrightarrow{y_a} v_a$ and $v_a \xrightarrow{u_1} w_a$.
4) Two action nodes v_+ and v_- and two observation nodes w_+ and w_-, along with action edges $v_+ \xrightarrow{u_+} w_+$ and $v_- \xrightarrow{u_-} w_-$.
5) For each edge $a \to b$ of $\mathbf{G}$, two observation edges $w_a \xrightarrow{y_{ab}} v_+$ and $w_b \xrightarrow{y_{ab}} v_-$.
6) An action node v_g, and two observation edges $w_+ \xrightarrow{y_g} v_g$ and $w_- \xrightarrow{y_g} v_g$.

We complete the CP-DEC instance by choosing v_{s} and $\{v_g\}$ for the start node and goal region respectively, and setting $k = 7$.

Figure 2 shows an example of this construction. The intuition is that only two action sequences allow the robot to successfully reach the goal, namely $u_0, u_1, u_+, u_{\mathrm{T}}$ and $u_0, u_1, u_-, u_{\mathrm{T}}$. Moreover, in any given execution, only one of these two choices will succeed. The construction forces any successful plan graph to "remember" enough to know

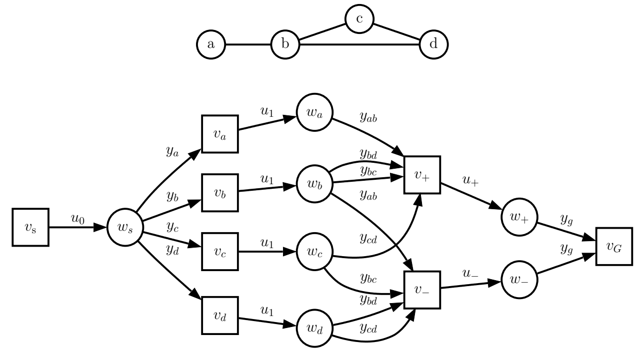

Fig. 2: [top] An example instance of 3-coloring. [bottom] The I-state graph constructed from that instance.

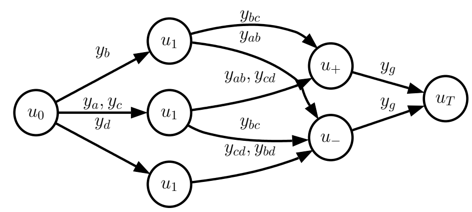

Fig. 3: A plan that solves the planning problem in Figure 2. Because the original graph is 3-colorable, the problem can be solved by plan with only 7 vertices.

whether u_+ or u_- is the correct choice.

The time to perform this construction is linear in the size of $\mathbf{G}$. We must now argue that the constructed planning problem is equivalent to the original graph, in the sense that the graph has 3-coloring if and only if the planning problem admits a solution of at most 7 vertices.

Lemma 2: For any instance $\mathbf{G} = (V, E)$ of GRAPH-3C *for which the correct output is "True," the correct output of the* CP-DEC *instance described above is also "True."*

Proof: Let $c : V \to \{1, 2, 3\}$ denote a 3-coloring of $\mathbf{G}$. Let $\mathbf{P}$ denote the plan graph of exactly 7 vertices with the following elements:

1) A start vertex v_{s} labeled with action u_0.
2) Three vertices v_1, v_2, and v_3 (one for each of the colors of $\mathbf{G}$), all labeled with action u_1.
3) Three vertices v_+ and v_-, and v_g, labeled with action u_+, u_-, and u_{T} respectively, along with edges $v_+ \xrightarrow{y_g} v_g$ and $v_- \xrightarrow{y_g} v_g$.
4) For each vertex a of $\mathbf{G}$, an edge $v_{\mathrm{s}} \xrightarrow{y_a} v_{c(a)}$.
5) For each edge $a \to b$ of $\mathbf{G}$, edges $v_{c(a)} \xrightarrow{y_{ab}} u_+$ and $v_{c(b)} \xrightarrow{y_{ab}} u_-$.

Figure 3 illustrates this construction for the example introduced in Figure 2.

To show that $\mathbf{P}$ is indeed a plan graph, we must confirm that none of its vertices has multiple outgoing edges labeled with the same observation. The only vertices at which this could occur are v_0, v_1, and v_2. Suppose such a vertex v exists, with two distinct outgoing edges for observation y_{ab}. Because v_+ and v_- are the only two possible targets for edges outgoing from v, these two edges must connect those two vertices.

By construction, v must also have incoming edges from v_{s} for observations y_a and y_b. Note that because observations y_a

and y_b both lead to v, we know that in the coloring of $\mathbf{G}$, we have $c(v_a) = c(v_b)$. However, the existence of edges labeled with observation y_{ab} implies that $\mathbf{G}$ has an edge between v_a and v_b. Since v_a and v_b are adjacent in $\mathbf{G}$ but have the same color in c, we have a contradiction to the supposition that c is a proper 3-coloring of $\mathbf{G_1}$. Therefore $\mathbf{P}$ is a legitimate plan graph.

Finally, it is straightforward to see that $\mathbf{P}$ correctly solves the planning problem by examining each of the finitely many possible execution traces. □

Lemma 3: For any instance $\mathbf{G}$ *of* GRAPH-3C *for which the correct output is "False," the correct output of the* CP-DEC *instance described above is also "False."*

Proof: Prove by contrapositive. Suppose there exists a seven-node plan graph $\mathbf{P}$ that solves this planning problem, in order to show that there exists a 3-coloring of the original $\mathbf{G}$.

First, note that any correct plan for this problem must contain at least one distinct node labeled with each of u_0, u_+, u_-, and u_T. Moreover, because each of these actions is executed at most once in any correct plan, we can (without loss of generality) assume that each of these actions is the label for *exactly* one vertex in $\mathbf{P}$. Therefore, there are a most three vertices of $\mathbf{P}$ labeled with u_1. Denote these vertices v_1, v_2, and v_3. Let v_s denote the $\mathbf{P}$ vertex labeled with u_0, which must be the start vertex of $\mathbf{P}$.

For each vertex a in $\mathbf{G}$, note that there must exist in $\mathbf{P}$ a unique edge $v_\mathrm{s} \xrightarrow{y_a} v_j$ to some v_j labeled with u_1. Let $c : V \to \{1, 2, 3\}$ denote the vertex-labeling of $\mathbf{G}$ that maps each vertex a to the index of the target vertex of this associated edge. Since v_1, v_2, and v_3 are the only candidates for v_j, this labeling uses only three colors.

Let us prove by contradiction that c is a proper coloring of $\mathbf{G}$. Suppose not, and let (a, b) denote an edge of $\mathbf{G}$ with $c(a) = c(b)$. By construction, $\mathbf{P}$ has edges $v_\mathrm{s} \xrightarrow{y_{ab}} v_{c(a)}$ and $v_\mathrm{s} \xrightarrow{y_{ab}} v_{c(b)}$. Observe that the target vertices of these two edges are identical. However, notice that in a correct plan, the observation sequence $y_a y_{ab}$ must lead to the plan node labeled u_+, whereas the observation sequence $y_b y_{ab}$ must lead to the plan node labeled u_-. In $\mathbf{P}$, these observation sequences lead to same plan node. Therefore, $\mathbf{P}$ is not a correct solution to the planning problem. This contradiction demonstrates that c is a proper 3-coloring of $\mathbf{G}$.

□

C. Statement of results

The partial results in Section IV-A and IV-B lead directly to our main hardness results.

Lemma 4: CP-DEC *is NP-hard.*

Proof: Combine Lemmas 2 and 3. □

Theorem 5: CP-DEC *is NP-complete.*

Proof: Combine Lemmas 1 and 4. □

Theorem 6: CP *is NP-hard.*

Proof: This is a direct consequence of Lemma 4. □

Algorithm 2 TRYSUBPLAN

Input:
A plan graph $\mathbf{P}$ *and an action node* v.

Compute metadata for $\mathbf{P}$.
for $i \in \{1, 2\}$ **do**
    $s_i(v).\mathrm{insert}(\mathbf{P})$
    if $s_i(v)$ holds more than k plans **then**
        remove worst plan, according to H_i, from $s_i(v)$
    end if
end for
if $\mathbf{P}$ remains in any $s_i(v)$ **and** each out-neighbor of v holds at least one plan **then**
    $Q.\mathrm{push}(v)$
end if

Algorithm 3 PLANCONCISELY

Input:
A problem $\mathcal{P} = (\mathbf{I}, v_\mathrm{s}, V_\mathrm{g})$.
Output:
A plan graph $\mathbf{P}$ *that solves* $\mathcal{P}$.

$Q \leftarrow$ empty set of observation nodes
$\mathbf{P}_\mathrm{T} \leftarrow$ single vertex labeled u_T
for each action node $v \in V_\mathrm{g}$ **do**
    TRYSUBPLAN$(v, \mathbf{P}_\mathrm{T})$
end for
while Q is not empty **do**
    $w \leftarrow Q.\mathrm{pop}()$
    for each in-neighbor $v \in V_\mathrm{u}$ of w **do**
        build candidate plans starting at v through w.
        for each candidate plan $\mathbf{P}$ and each $v' \in V_\mathrm{u}$ **do**
            TRYSUBPLAN$(v', \mathbf{P})$
        end for
    end for
end while
return smallest reduced plan stored at v_s, if any.

V. ALGORITHM DESCRIPTION

The previous section proved that, unless $P = NP$, no efficient algorithm can optimally solve the concise planning problem CP. Therefore, we turn our attention now to a new algorithm that solves the problem approximately, in the sense that the plans the generate remain correct in the worst-case sense, but cannot be guaranteed to be optimally concise.

The idea of the algorithm is to use the structure of the I-state graph to generate a series of candidate plans, each of which can successfully reach the goal from at least one I-state. This process starts with a trivial "Terminate immediately" plan, which is correct from the goal region. From there, the algorithm maintains a collection of observation nodes for which all of the out-neighbors have at least one associated plan, and repeatedly constructs new plans that pass though each successive observation node extracted from that set. The plans generated in this way—all of which have the form of a rooted tree with leaves labeled u_T—each undergo a plan reduction step, which mutates a given plan

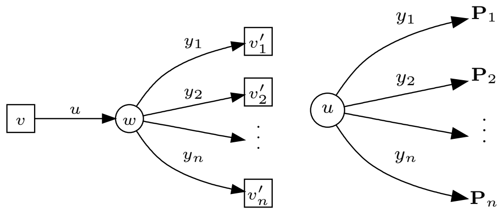

Fig. 4: [left] A fragment of an I-state graph, for which a new plan can be constructed, as long as all of $v_1, v_2, \ldots, v_n$ have at least one associated plan. [right] The constructed plan copies and grafts the existing plans into a tree rooted at a new node with action u.

into an approximately-optimally-concise plan equivalent to the original.

As this process proceeds, the algorithm maintains, for each action node, a small finite collection of such sub-plans that reach the goal from that node. Our algorithm prioritizes the plans to retain based on heuristic evaluations of both the local concision and global applicability of each sub-plan.

The algorithm terminates when its queue is exhausted, at which time it returns the most concise plan associated with the start vertex of the I-state graph. Pseudocode for this approach appears as Algorithms 2 and 3. The following subsections explain and clarify the details.

A. Candidate plan construction

Beyond the initial trivial plan, Algorithm 3 constructs additional plans in the following way. It identifies action nodes v for which *(i)* there exists an action edge $v \xrightarrow{u} w$, and *(ii)* all of the out-neighbors $v'_1, \ldots, v'_n$ of w have at least one associated plan. In this situation, we can form a new plan graph that reaches the goal from v as follows: Start with a vertex that executes action u, and attach plan out-edges for each of the out-edges $w \xrightarrow{y} v'$ of w in the I-state graph. Each of these edges connects to a copy of a plan associated with v', which by construction reaches the goal from there. See Figure 4.

To efficiently locate portions of the I-state graph at which this construction is possible, the algorithm maintains, for each observation node w, a set of "incomplete" (that is, planless) out-neighbors, and inserts w into the global Q each time a new plan is stored at one of its out neighbors, provided that w's incomplete list is empty. In this way, the algorithm ensures that every plan it generates is complete and correct for the v at which it is generated.

B. Plan evaluation heuristics

As mentioned above, Algorithm 3 maintains a bounded size collection of "promising" plans for each action node. Specifically, at action node v, we store *two* plan sets $s_1(v)$ and $s_2(v)$, each of which holds at most k plans, in which k is a tunable algorithm parameter.

Below, we introduce heuristic functions H_1 (which measures the *local* conciseness of a plan) and H_2 (which measures the global reusability of a plan). As the algorithm proceeds, $s_1(v)$ always contains the k or fewer plans that maximize H_1, across all generated plans that reach the goal from v. The set s_2 likewise stores the k best plans according to H_2. The subsequent sections introduce H_1 and H_2.

1) Local heuristic: Reduced plan size: Notice that each of the plans constructed as described in Section V-A will have the form of a rooted tree but that, in most cases, concise plans have cycles. In fact, there is no reason to suspect that these trees will be concise plans. Figure 1 illustrates this idea. As a result, as part of the "compute metadata step" in Algorithm 2 (line 1), we use a *plan reduction* algorithm whose input is the original rooted tree plan graph $\mathbf{P}$, and the intended output is the smallest plan graph $r(\mathbf{P})$ that produces identical behavior.

This problem is equivalent to the *filter reduction* problem from the authors' prior work [12]—the only difference is that the description of the existing algorithm refers to the vertex labels as abstract "colors" instead of actions—and we employ the algorithm from that paper to reduce plans. Note that because filter reduction is NP-hard, we settle for reduced plans that can be generated efficiently and are guaranteed to be correct, but are only approximately optimal.

The size of these reduced plans represents local, greedy measurement of the usefulness of a plan. Therefore, we define the local heuristic as

$$H_1(\mathbf{P}) = -\operatorname{size}(r(\mathbf{P}))$$

Notice that, at the conclusion of the algorithm, the smallest plan in the set $s_1(v_\mathrm{s})$ represents the most concise plan start-to-goal we have found. As a result, this plan becomes the final output of the algorithm.

2) Global heuristic: Reuse potential: Unfortunately, the local heuristic introduced in Section V-B.1 is not sufficient, because it cannot account for the idea of choosing actions at one action node expressly because those plan nodes can be re-used in other portions of the I-state graph. This notion of the reuse of plan fragments motivates our second, global heuristic.

The idea is to compute the *outcome function* $\mathcal{O}_\mathbf{P} : V_\mathrm{u} \to 2^{V_\mathrm{u}}$ of a plan $\mathbf{P}$. This function considers the potential results from executing $\mathbf{P}$ starting at each action node v, mapping each to the set of action nodes at which that plan might terminate, or to the empty set if the plan might fail when executed from v. This function can be computed by a forward search of reachable action node/plan node pairs, very similar to Algorithm 1.

The appeal of the outcome function is that it shows, from a global perspective, how much potential for reuse a plan possesses. For H_2 we use a straightforward measure of reusability based on the total average distance across $\mathbf{I}$ that a plan can achieve. Specifically, we define

$$H_2(\mathbf{P}) = \sum_{v \in \{V_\mathrm{u} | \mathcal{O}_{r(\mathbf{P})}(v) \neq \emptyset\}} \left(\frac{1}{\mathcal{O}_{\mathbf{r}(\mathbf{P})}(v)} \sum_{v' \in \mathcal{O}_{r(\mathbf{P})}(v)} d(v, v') \right),$$

in which $d(v, v')$ denotes the number of edges in the shortest directed path connecting v to v' in $\mathbf{I}$.

C. Algorithm summary

This completes the overall picture of Algorithm 3. To summarize, it tracks a set of observation nodes through which new complete plans can be constructed. As long as this set is not empty, it removes an arbitrary observation node, w. It then constructs new plans that pass through w starting from each of its in-neighbors, using all combinations of plans stored in both the s_1 and s_2 sets of the resulting action nodes. For each such plan $\mathbf{P}$, we compute the heuristic functions H_1 and H_2, and insert $\mathbf{P}$ into the s_1 and/or s_2 sets at *every* action node for which $\mathbf{P}$ successfully reaches the goal and improves upon the existing plans and Q is updated appropriately. This process continues until Q is exhausted, at which point the best start-to-goal reduced plan is returned.

VI. Experimental Results

We implemented Algorithm 3 to test its efficiency and the concision of the plans it produces for both manipulation (Section VI-A) and navigation (Section VI-B) domains. Our implementation uses C++ and all of the executions used a single core of a 2.5GHz quad-core processor.

A. Manipulation

In the spirit of the established techniques for (nearly-)sensorless manipulation [7], [13], we executed the algorithm on a family of problems in which the goal is to orient a polygonal shape using a series of squeezes from a parallel-jaw gripper. Given a description of the convex-hull of an object we followed the steps (detailed in [6]) for treating such problems: (1) we computed the diameter function for the polygon, (2) identified minima in this function, giving the stable orientations that occur after a squeeze operation, and (3) computed the so-called squeeze function mapping a pre-squeeze orientation into the post-squeeze orientation. For these problems we considered small sets of actions of the form "rotate gripper by x and squeeze." This is sufficient to construct an I-state graph for sensorless problems.

The left part of Figure 5 gives an example using one of the objects we evaluated: the "fourgon" shape. The figure provides intuition for how the local minima in the plot represent stable orientations after a squeeze operation is performed by the frictionless parallel-jaw gripper. Figure 5 shows the form of the I-state graph generated for the problem of orienting the fourgon using rotations of only 5° and 65°, and squeeze operations; the resulting plan produced by the algorithm is also shown. Note how, although the geometry of the object is simple, determining a concise open-loop plan given those actions remains far from obvious.

Sensing information can also be incorporated in this problem. We considered a simple setting in which one can specify a set of binary sensors, each determining whether the distance between jaws of the gripper exceed some threshold or not. Figure 6 extends the preceding example by adding a single diameter threshold sensor with distance 10.5cm. The I-state graph observation edges (in blue) are now labeled with the output from the threshold sensor. The resulting plan exploits this information and is smaller than the sensorless plan, which has been seen in plans for orienting other objects too.

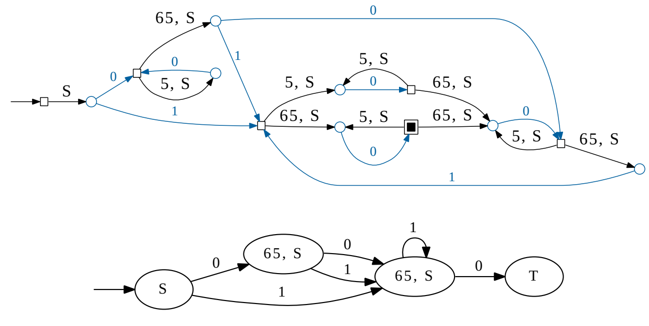

Fig. 6: [top] The I-state graph for the problem of orienting the Fourgon polygon with a binary sensor measuring whether the jaws of the gripper are more than x =10.5cm apart or not. Observation arcs are labeled '0' or '1'; the latter is returned when the diameter of object in its stable orientation exceeds the distance threshold, and '0' is returned otherwise. (The remainder of the graph follows the format of Fig 5.) [bottom] The most concise plan found.

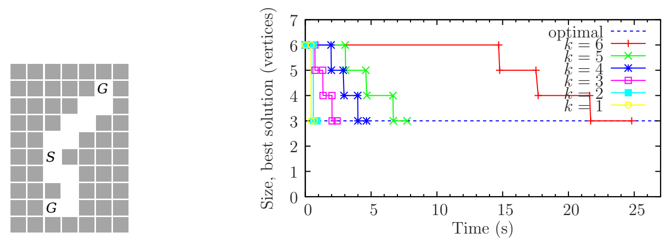

Fig. 7: An example with multiple goals. The same (optimal) solution is found with all values of parameter k, which bounds the number of subplans stored at each action node.

B. Navigation

Second, we considered a simplified navigation domain in which a robot moves within a grid of discrete cells using actions `up`, `down`, `left`, and `right`, each of which reliably moves a single cell in the desired direction unless an obstacle impedes that motion. The robot's observation space is $Y = \{00, 01, 10, 11\}$, in which the first bit indicates whether the robot bumped on obstacle on its previous move, and the second bit is the output of a goal-detect sensor.

This family of problems is interesting because many instances admit very concise plans. In particular, small

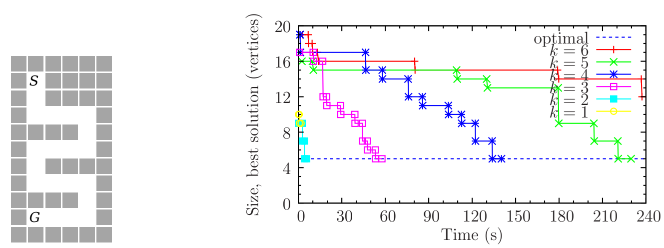

Fig. 8: A switch-back pattern affords opportunity for plan compression.

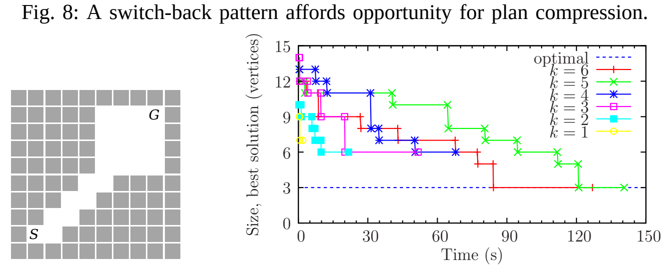

Fig. 9: An example in which many concise plans can navigate through the open field, but only one can navigate the narrow corridor.

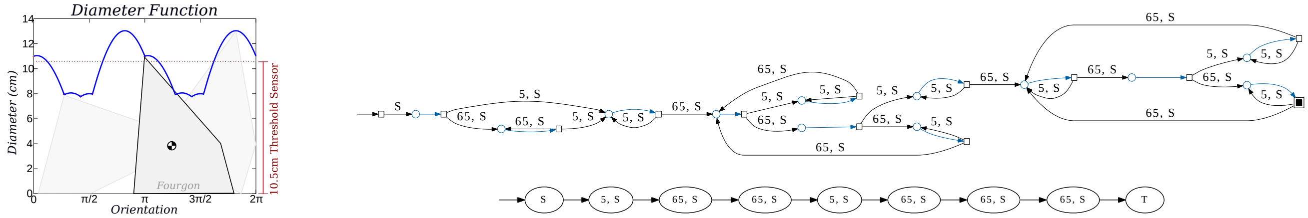

Fig. 5: [left] The diameter function computed for the Fourgon figure (shown inset in three orientations). Local minima in the plot represent stable orientations for the object when squeezed by a parallel-jaw gripper. [right, top] The I-state graph for the problem of orienting the Fourgon polygon without sensors, constructed automatically using classical techniques. The black arcs leading from square vertices are where an action must be selected, the blue arrows represent transitions based on a observation (but are empty in this example); the shaded vertex is the goal. The 'S' symbol denotes a squeeze action by the gripper; '5' and '65' denote rotations of the gripper by those many degrees, respectively. [right, bottom] An open-loop plan found believed to be the most concise solution, found using Algorithm 3.

plans tend to exist for instances in which short sequences of actions, terminated by appropriate observations, can be repeated to make progress toward the goal. We constructed several grid navigation problems in order to evaluate Algorithm 3. Space limits us to the three examples in Figures 7—9. The figures show the performance of the method with different limits on the number of plans associated with each action node. This is indicated with the value of the k parameter (used on Line 4 of Algorithm 2).

Figure 7 is the same environment as Figure 1, but the I-space differs because the earlier example had no bump detector. The plot shows that an optimal 3 state solution is found for each value of k. Figure 8 is an environment with the potential for comparatively long sequences to be simplified in a concise plan. This example has a hierarchical structure to the repetitions: there are corridor navigation pieces (repeated `left`/`right` actions until a bump) and repetitive sequences composed of corridor navigation subplans. The shortest solution involves 5 states, and was found for values of k greater than one. Figure 9 is an instance where many conceivable concise subplans can be generated in the open field in the upper right, but only a single plan (the most concise one, with 3 states) is concise and navigates the corridor too. In this example, the optimal solution is found only when $k \geq 5$.

The plots of "most concise plan so far" vs. running-time show the same behavior. A larger k causes slower progress, because the search phase keeps more subplans thereby increasing the search space, but it also increases the likelihood of finding the highest quality solution ($k = \infty$ is a search of the complete plan space). The lowest value of k which results in the optimal plan gives an informal idea of the comparative complexity of the problems, suggesting that Figure 7 is easiest and Figure 9 most difficult of the three. We hypothesize that this result occurs because of the open field in the top left of Figure 9, through which many distinct plans successfully travel. In a counterintuitive contrast to the narrow corridor problem faced by typical motion planning algorithms, it appears that concise planning problems are more difficult in the presence of wide corridors.

VII. Conclusions

In this paper, we introduced the concise planning problem, proved that solving it optimally is NP-hard, and presented an implemented algorithm that solves it approximately. A few avenues for future work are apparent.

First, one might relax the requirement of worst case correctness and, after assigning a probability distribution over the out-edges of each observation node, instead form plans whose success probability is less than unity. An interesting question is to understand the impact of increasing the required success probability on plan conciseness.

Additionally, it is worth noting that generally one wishes to have an understanding of the tradeoff between plan execution cost and expression cost. Existing work emphasizes the latter factor, but the algorithm presented ignores the former aspect. Joint consideration both criteria—via scalarization or Pareto optimality concepts—would be useful results.

Acknowledgments This material is based upon work supported by the National Science Foundation under Grant No. IIS-0953503.

References

[1] M. A. Bender, A. Fernández, D. Ron, A. Sahai, and S. Vadhan, "The power of a pebble: exploring and mapping directed graphs," in *Proc. IEEE Symposium on Foundations of Computer Science*, 1998, pp. 269–278.

[2] M. Blum and D. Kozen, "On the power of the compass (or, why mazes are easier to search than graphs)," in *Proc. IEEE Symposium on Foundations of Computer Science*, 1978, pp. 132–142.

[3] B. Bonet and H. Geffner, "An algorithm better than AO*?" in *Proceedings of the AAAI National Conference on Artificial Intelligence (AAAI'97)*, 2005, pp. 1343–1347.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[5] M. Erdmann, "On the topology of discrete planning with uncertainty. in advances in applied and computational topology," in *Proc. Symposia in Applied Mathematics*, A. Zomorodian, Ed., vol. 70. American Mathematical Society, 2012.

[6] K. Y. Goldberg, "Orienting polygonal parts without sensors," *Algorithmica*, vol. 10, pp. 201–225, 1993.

[7] K. Goldberg and M. T. Mason, "Bayesian grasping," in *Proc. International Conference on Robotics and Automation*, 1990, pp. 1264–1269.

[8] E. A. Hansen, "Solving POMDPs by searching in policy space," in *Proc. Conference on Uncertainty in Artificial Intelligence*, 1998.

[9] E. A. Hansen and S. Zilberstein, "LAO*: A heuristic search algorithm that finds solutions with loops," *Artificial Intelligence*, vol. 29, no. 1-2, pp. 35–62, 2001.

[10] A. N. Kolmogorov, "On tables of random numbers," *Theoretical Computer Science*, vol. 207, pp. 387–395, 1998, reprinted from Sankhyā: The Indian Journal of Statistics, Series A. Vol. 25, Part 4 (1963).

[11] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[12] J. M. O'Kane and D. A. Shell, "Automatic reduction of combinatorial filters," in *Proc. IEEE International Conference on Robotics and Automation*, 2013, to appear.

[13] R. H. Taylor, M. T. Mason, and K. Y. Goldberg, "Sensor-based manipulation planning as a game with nature," in *International Symposium on Robotics Research*, R. C. Bolles and B. Roth, Eds. Cambridge MA: MIT Press, 1988.