

Admissible Abstractions for Near-optimal Task and Motion Planning*

William Vega-Brown and Nicholas Roy
Massachusetts Institute of Technology
{wrvb, nickroy}@mit.edu

Abstract

We define an admissibility condition for abstractions expressed using angelic semantics and show that these conditions allow us to accelerate planning while preserving the ability to find the optimal motion plan. We then derive admissible abstractions for two motion planning domains with continuous state. We extract upper and lower bounds on the cost of concrete motion plans using local metric and topological properties of the problem domain. These bounds guide the search for a plan while maintaining performance guarantees. We show that abstraction can dramatically reduce the complexity of search relative to a direct motion planner. Using our abstractions, we find near-optimal motion plans in planning problems involving 10^{13} states without using a separate task planner.

1 Introduction

Consider a problem domain like the one shown in figure 1. A holonomic two-dimensional agent is tasked with navigating to a specified goal region as quickly as possible. The path is blocked by doors that can only be opened by pressing the appropriate switch. Planning the sequence of switches to toggle requires combinatorial search; deciding if a path exists to each switch requires motion planning. As in many real-world planning domains, such as object manipulation or navigation among movable objects, the combinatorial search and motion planning problems are coupled and cannot be completely separated.

A standard approach to making such problems computationally tractable is to use abstraction to reason about the properties of groups of primitive plans simultaneously. For example, we could choose a sequence of high-level operations using a task planner, ignoring the details of the underlying motion plan. If we later determine that we cannot find a motion plan consistent with our high-level plan, we can use that information to modify our high-level plan. For example, [Gravot *et al.*, 2005] describe an integrated approach that

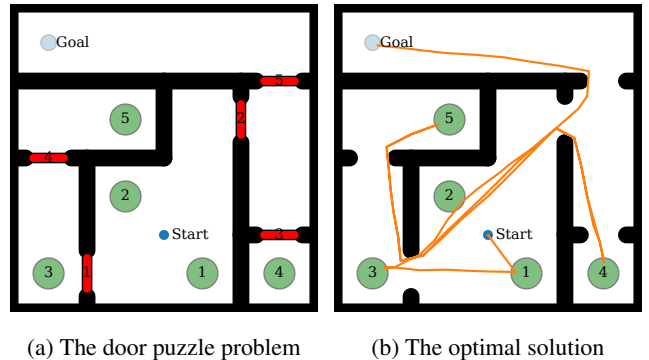


Figure 1: The door-switch problem, an example task and motion planning domain. A two-dimensional robot must navigate from the start location to a goal location, but the way is obstructed by doors that can only be opened by toggling a corresponding switch. The optimal solution to this problem instance is to toggle the switches in the order (1, 3, 2, 4, 5) and then go to the goal set. Because the size of the configuration space grows exponentially with the number of doors, planning is computationally challenging. Abstraction can render such planning problems tractable.

relies on a heuristic search for a high-level plan and uses motion planners as subroutines to deal with detailed geometry. [Kaelbling and Lozano-Pérez, 2011] use a hierarchy to guide high-level decision making, resolving low-level decisions arbitrarily and trusting in the reversibility of the system to ensure hierarchical completeness. Although these and other approaches (e.g., [Garrett *et al.*, 2015; Cambon *et al.*, 2009; Srivastava *et al.*, 2013]) vary in how they deal with the interaction between geometric planning and combinatorial search, they share a common weakness: they can only make guarantees about the plans they generate relative to the abstraction they are provided. Even optimizing approaches ([Wolfe *et al.*, 2010]) are generally limited to guarantees of hierarchical optimality.

Angelic semantics ([Marthi *et al.*, 2008]) provide a way to describe an abstraction that preserves optimality, but it is not clear what criteria an angelic abstraction must satisfy in order to make guarantees about the quality of synthesized plans. In this paper, we describe conditions under which an abstraction will preserve the ability to find the optimal motion plan while accelerating planning. We derive abstractions for two con-

*This document is an extended version of [Vega-Brown and Roy, 2018] containing additional proofs and exposition.

tinuous planning domains, and using these abstractions we can dramatically reduce the complexity of search relative to a direct motion planner. We find near-optimal motion plans in planning problems involving 10^{13} states without using a separate task planner.

2 Problem Formulation

We are interested in planning problems involving some underlying continuous configuration space \mathcal{X} , such as the position of a robot or the configuration of its joints. Our task is to find a path through free space that starts in a specified state s_0 and ends in a goal set S_{goal} . This goal set may be specified implicitly, as the set of all states satisfying some constraint.

A path is a continuous map $p : [0, 1] \rightarrow \mathcal{X}$. We define a concatenation operator \circ for paths.

$$(p_1 \circ p_2)(s) = \begin{cases} p_1(2t) & \text{if } t \leq \frac{1}{2} \\ p_2(2t - 1) & \text{if } \frac{1}{2} < t \leq 1 \end{cases} \quad (1)$$

Let $\mathcal{P}_{\mathcal{X}}(S, S')$ be the set of all paths starting in $S \subset \mathcal{X}$ and ending in $S' \subset \mathcal{X}$. Let $c : \mathcal{X} \times T\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ be a cost function, where $T\mathcal{X}$ is the tangent space of \mathcal{X} . We can define an associated cost functional $\mathcal{C} : P \rightarrow \mathbb{R}_{\geq 0}$.

$$\mathcal{C}[p] = \int_0^1 c(p(t), \dot{p}(t)) dt \quad (2)$$

Because \mathcal{C} is additive, $\mathcal{C}[p_1 \circ p_2] = \mathcal{C}[p_1] + \mathcal{C}[p_2]$. We define the *optimal cost function* $c^* : 2^{\mathcal{X}} \times 2^{\mathcal{X}} \rightarrow \mathbb{R}_{\geq 0}$ as

$$c^*(S, S') = \inf\{\mathcal{C}(p) : p \in \mathcal{P}_{\mathcal{X}}(S, S')\}. \quad (3)$$

We define the ϵ -approximate planning problem as the search for a path $\hat{p} \in \mathcal{P}_{\mathcal{X}}(\{s_0\}, S_g)$ with cost less than $(1 + \epsilon)$ the optimal cost for any $\epsilon \in \mathbb{R}_{\geq 0} \cup \{\infty\}$.

$$\hat{p} \in \{p \in \mathcal{P}_{\mathcal{X}}(\{s_0\}, S_g) : \mathcal{C}(\hat{p}) \leq (1 + \epsilon)c^*(s_0, S_g)\} \quad (4)$$

The case where $\epsilon = \infty$, when we wish to find any feasible path to the goal set, is the problem of *satisficing* planning. The case where $\epsilon = 0$ is optimal planning.

The set $\mathcal{P}_{\mathcal{X}}(\mathcal{X}, \mathcal{X})$ of all possible paths from all possible start and goal locations is continuous and topologically complex. To simplify planning, we assume we have available a finite set \mathcal{A}_0 of *primitive operators*, low-level actions that can be executed in the real world. The problem of constructing such a set of operators in continuous motion planning domains is well studied; in this document, we will assume the set of operators are given by the edges in a probabilistic roadmap (PRM*). That is, we randomly sample a finite set of configurations $\mathcal{V}_n \subset \mathcal{X}$, and for each such configuration v , we define an operator p_v . The operator p_v ensures that the robot will end at the state v if executed from any state in the open ball of radius r_n around v , where $r_n \propto (\log n/n)^{1/d}$ is a radius that increases slowly with the size of the discretization. Any feasible plan can be well-approximated by a sequence of these randomly sampled operators as the number of sampled configurations tends to infinity. For example, we can show that if $\mathcal{A}_{0,n}^*$ is the set of all paths through a PRM* with n sampled configurations, then

$$\lim_{n \rightarrow \infty} \inf\{\mathcal{C}[p] : p \in \mathcal{A}_{0,n}^* \cap \mathcal{P}_{\mathcal{X}}(\{s_0\}, S_g)\} = \inf\{\mathcal{C}[p] : p \in \mathcal{P}_{\mathcal{X}}(\{s_0\}, S_g)\}. \quad (5)$$

This was proven by [Karaman and Frazzoli, 2011] for the case where the system is subject to analytic differential constraints, and by [Vega-Brown and Roy, 2016] when the system has piecewise-analytic differential constraints (as in object manipulation problems).

Because the set of primitive operators can grow quite large, especially in problems with high-dimensional configuration spaces, a direct search for primitive plans is computationally intractable. Instead, we will use angelic semantics to encode bounds on the cost of large groups of plans. We can use these bounds to plan efficiently while preserving optimality.

3 Angelic Semantics

An *abstract operator* \mathbf{a} represents a set $\mathbf{a} \subset \mathcal{P}_{\mathcal{X}}$ of primitive plans. Because the space of plans is infinite, we define operators implicitly, using constraints on the underlying primitive plans. For example, in a navigation problem, we might define an operator as any primitive plan that remains inside a given set of configuration space and ends in a different set of configuration space. This is depicted graphically in figure 2a: the operator \mathbf{a}_{23} contains every path that is contained in region 2 and ends in region 3.

The concatenation of two operators $\mathbf{a}_i \circ \mathbf{a}_j$ is an abstract plan containing all possible concatenations of primitive plans in the operators.

$$\mathbf{a}_i \circ \mathbf{a}_j = \{p_i \circ p_j : p_i \in \mathbf{a}_i, p_j \in \mathbf{a}_j, p_i(1) = p_j(0)\} \quad (6)$$

The condition $p_i(1) = p_j(0)$ is necessary to enforce that only feasible plans are contained in $\mathbf{a}_i \circ \mathbf{a}_j$. In a problem with nontrivial dynamic constraints, the condition would need to be more complex. In figure 2b, we show samples from the plan $\mathbf{a}_{12} \circ \mathbf{a}_{23} \circ \mathbf{a}_{34} \circ \mathbf{a}_{4g}$, which contains paths that move from region 1 to 2 to 3 to 4 to the goal. The concatenation operation allows us to express complicated sets of plans in a compact way.

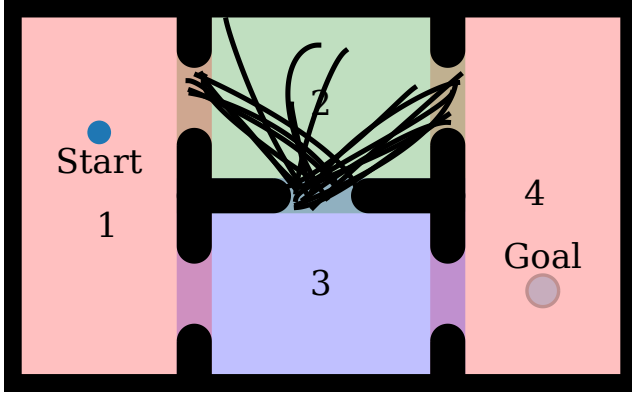
Because our operators are defined implicitly, it can be difficult to find the best plan in the abstract plan, or even to decide if there exists a plan consistent with the constraints of an abstract plan. Note that it is easy to write down abstract plans that are empty; in the toy navigation example in figure 2, the plan $\mathbf{a}_{12} \circ \mathbf{a}_{34}$ contains no primitive plans, as the intersection of regions 1, 2, and 3 is empty. For planning with abstract operators to be feasible, we need a way to reason about the primitive plans contained by an abstract plan *without* first enumerating those primitive plans.

Specifically, we will develop a way to compare abstract plans, and we will show this comparison is sufficient for planning. We do this using the *valuation* of an operator or plan. A *valuation* $V[\mathbf{a}]$ for an operator or plan \mathbf{a} is the unique map $V[\mathbf{a}] : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ that takes a pair of states and gives the lowest cost path between the pair.

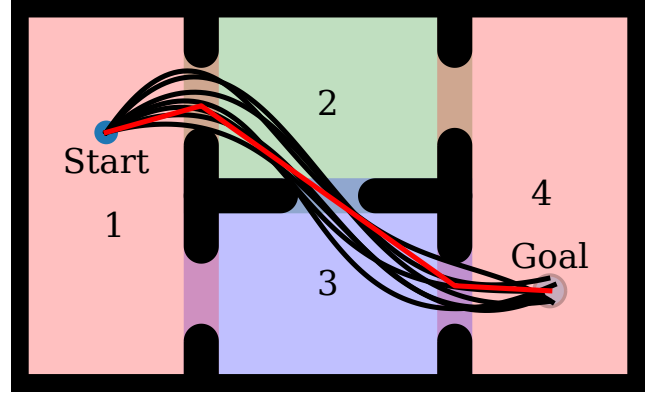
$$V[\mathbf{a}](s_1, s_2) = \inf\{\mathcal{C}(\sigma) : \sigma \in \mathbf{a}, \sigma(0) = s_1, \sigma(1) = s_2\} \quad (7)$$

Note that if there are no paths in \mathbf{a} linking s_1 and s_2 , then $V[\mathbf{a}](s_1, s_2) = \inf \emptyset = \infty$.

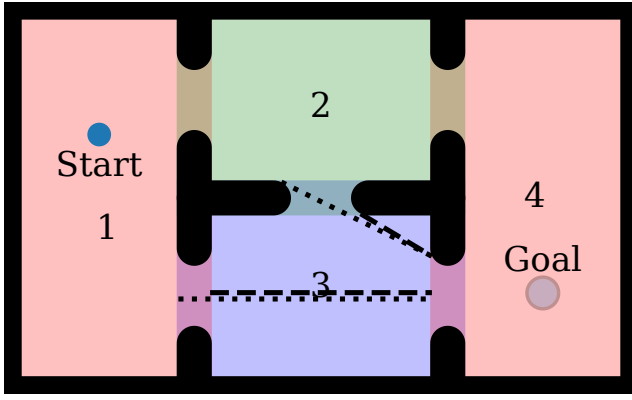
Valuations allow us to compare abstract plans without reference to the primitive plans they contain. Given two abstract plans \mathbf{p} and \mathbf{p}' , if we can prove that for any pair of states x, x' ,



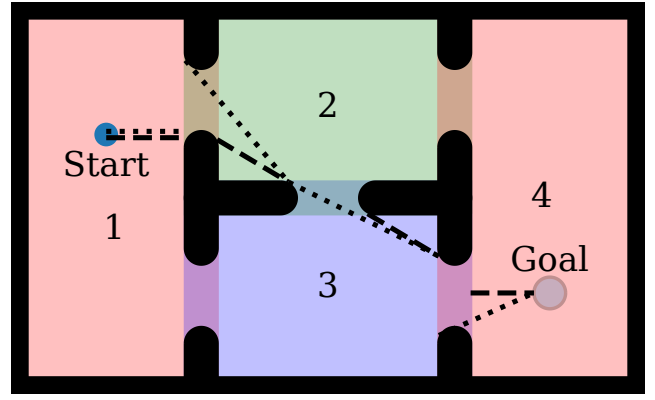
(a) Plans in operator a_{23}



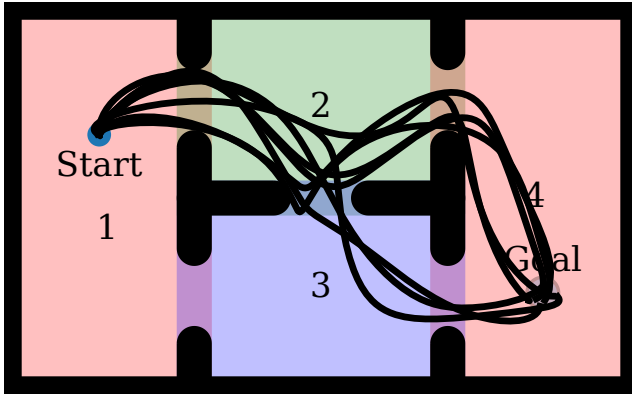
(b) Plans in $a_{12} \circ a_{23} \circ a_{34} \circ a_{4g}$



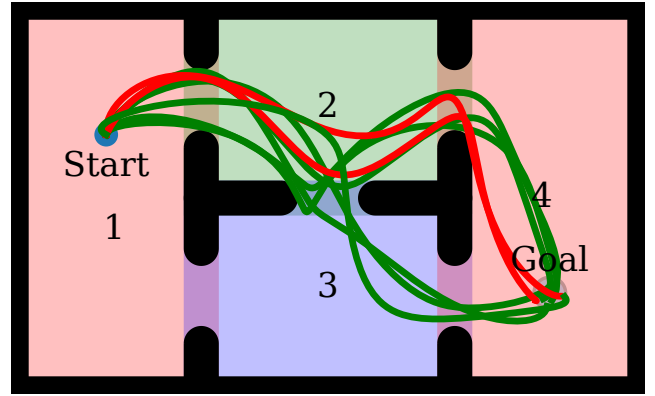
(c) Bounds for $V[a_{34}]$



(d) Bounds for $V[a_{12} \circ a_{23} \circ a_{34} \circ a_{4g}]$



(e) Plans in $a_{12} \circ ACT$



(f) Plans in $a_{12} \circ a_{23} \circ ACT$

Figure 2: A schematic description of angelic semantics. Abstract operators (a) are sets of primitive plans, and can be defined implicitly in terms of constraints. For example, the operator a_{23} contains all plans that end in region 3 and do not leave region 2. We can sequence abstract operators into abstract plans (b). The red lines link the centroids of successive regions, while the black lines are randomly sampled primitive plans representative of the abstract plan that move from regions 1 to 2 to 3 to 4 to the goal. We can use domain-specific information to compute bounds on the cost of any plan in an operator stating from a specific set of states (c). Here, lower bounds are drawn using dashed lines, while upper bounds are drawn in dotted lines. Note the dependence on the initial state: the cost of a plan starting in $1 \cap 3$ is strictly higher than the cost of a plan stating in $2 \cap 3$. We can sequence these bounds (d) to compute bounds on the cost of an abstract plan. Finally, a refinement of an abstract plan p (e) is a less abstract plan (f) $p' \subset p$. Primitive plans in p' are shown with heavy lines, while plans in $p \setminus p'$ are shown with finer lines.

either $V[\mathbf{p}](x, x') < V[\mathbf{p}'](x, x')$ or $V[\mathbf{p}'](x, x') = \infty$, then either there is a solution to our planning problem in \mathbf{p} , or there is no solution in \mathbf{p} or \mathbf{p}' . Either way, we do not need to consider any plan in \mathbf{p}' ; we can prune \mathbf{p}' from our search space. Under such a condition, we say that \mathbf{p} *dominates* \mathbf{p}' and we write $V[\mathbf{p}] \prec V[\mathbf{p}']$. Similarly, if either $V[\mathbf{p}](x, x') \leq V[\mathbf{p}'](x, x')$ or $V[\mathbf{p}'](x, x') = \infty$, then we say that \mathbf{p} *weakly dominates* \mathbf{p}' and we write $V[\mathbf{p}] \preceq V[\mathbf{p}']$.

Unfortunately, determining the valuation of an operator is itself an optimization problem, and one that is not necessarily any easier than the planning problem we are trying to solve. In many cases, however, we can derive a computational advantage from reasoning about *bounds* on the valuation of an abstract operator. By representing these bounds *symbolically*, we will be able to reason without reference to the underlying states or plans.

We first define bounds on the valuation of an operator over a set of states.

$$V_L[\mathbf{a}](s, s') = \inf\{\inf\{V[\mathbf{p}](s, s') : s' \in s'\} : s \in s\} \quad (8)$$

$$V_U[\mathbf{a}](s, s') = \sup\{\inf\{V[\mathbf{p}](s, s') : s' \in s'\} : s \in s\} \quad (9)$$

A symbolic valuation bound $\hat{V}[\mathbf{a}]$ is a set of tuples $\{(s, s', l, u)\}$, where s, s' are symbolic states and $l < u \in \mathbb{R}_{\geq 0} \cup \{\infty\}$. A bound $\hat{V}[\mathbf{a}]$ is *admissible* if

$$\exists(s, s', l, u) \in \hat{V}[\mathbf{a}] : \quad l \leq V_L[\mathbf{a}](s, s') \quad (10)$$

$$\forall(s, s', l, u) \in \hat{V}[\mathbf{a}] : \quad u \geq V_U[\mathbf{a}](s, s'). \quad (11)$$

In words, a bound (s, s', l, u) is *admissible* if for any state x in s there exists a plan p ending in some state x' in s' with cost $c = \mathcal{C}[p]$ bounded above by u and below by l . We can also interpret a symbolic valuation bound \hat{V} as a bound over sets of states.

$$\hat{V}_L[\mathbf{a}](s, s') = \inf\{l : (s_0, s_1, l, u) \in \hat{V}[\mathbf{a}], s \cap s_0 \neq \emptyset, s' \cap s_1 \neq \emptyset\} \quad (12)$$

$$\hat{V}_U[\mathbf{a}](s, s') = \inf\{u : (s_0, s_1, l, u) \in \hat{V}[\mathbf{a}], s \subseteq s_0, s' \subseteq s_1\}. \quad (13)$$

Note that if $\hat{V}[\mathbf{a}]$ is *admissible*, then $\hat{V}_L[\mathbf{a}](s, s') \leq V_L[\mathbf{a}](s, s')$ and $V_U[\mathbf{a}](s, s') \leq \hat{V}_U[\mathbf{a}](s, s')$ for all abstract state pairs s, s' (see appendix A, proposition 1).

This observation has important consequences in a few interesting limiting cases. A bound $\hat{V}[\mathbf{a}]$ contains at least one element (s, s', l, u) where u is finite only if then there must be some plan in the operator \mathbf{a} . A bound $\hat{V}[\mathbf{a}]$ does not contain an element (s, s', l, u) where l is finite only if \mathbf{a} is empty. Similarly, $\hat{V}_U[\mathbf{a}](s, s') < \infty$ implies \mathbf{a} contains feasible plans connecting each state in s to some state in s' , while $\hat{V}_L[\mathbf{a}](s, s') = \infty$ implies \mathbf{a} contains no plan connecting a state in s to a state in s' . In addition, if $\hat{V}[\mathbf{p}]$ and $\hat{V}[\mathbf{p}']$ are *admissible*, then $\hat{V}[\mathbf{p} \cup \mathbf{p}'] = \hat{V}[\mathbf{p}] \cup \hat{V}[\mathbf{p}']$ is *admissible* (see appendix A, proposition 3 for a proof).

It is also important to recognize the state-dependence of valuation bounds. Consider the operator \mathbf{a}_{34} in figure 2c; the operator is defined as containing any plan contained in region

3 that ends in region 4. Because regions 3 and 4 intersect, the global lower bound on the cost of a plan in this operator is zero. However, we can compute nontrivial bounds for specific states, or for specific sets of states. For example, paths achieving lower and upper bounds are drawn from the abstract states $R_2 \cap R_3$ and $R_1 \cap R_3$ to the termination set of the operator.

As we will see in sections 4.2 and 4.3, for many domains we will not need to write down a valuation explicitly. Instead, we can use domain information to make metric computations and generate the necessary elements of a valuation procedurally. Moreover, by working with symbolic bounds we can efficiently compute bounds on the cost of plans consisting of sequences of abstract operators, without reference to a dense discretization of the underlying space of plans. For example, if we have bounds on a plan $\hat{V}[\mathbf{a}]$ and an operator $\hat{V}[\mathbf{a}']$, we can compute a bound $\hat{V}[\mathbf{a} \circ \mathbf{a}']$.

$$\begin{aligned} \hat{V}[\mathbf{a} \circ \mathbf{a}'] = & \{(s, s''', l + l', u + u') : (s, s', l, u) \in \hat{V}[\mathbf{a}], \\ & (s'', s''', l', u') \in \hat{V}[\mathbf{a}'], s' \subseteq s''\} \cup \\ & \{(s, s''', l + l', u) : (s, s', l, u') \in \hat{V}[\mathbf{a}], \\ & (s'', s''', l', \infty) \in \hat{V}[\mathbf{a}'], s' \cap s'' \neq \emptyset\} \quad (14) \end{aligned}$$

If $\hat{V}[\mathbf{a}]$ and $\hat{V}[\mathbf{a}']$ are *admissible*, then $\hat{V}[\mathbf{a} \circ \mathbf{a}']$ is *admissible* as well (see appendix A, proposition 2 for a proof). We call this process *propagation*. This process is depicted graphically in figure 2d.

4 Admissible Abstractions

We will use angelic semantics to specify abstractions that enable efficient planning. Suppose that \mathbf{p}, \mathbf{p}' are abstract plans, with $\mathbf{p} \subset \mathbf{p}'$. Then $\mathbf{p}' \preceq \mathbf{p}$, since any plan in \mathbf{p} is also in \mathbf{p}' —but because \mathbf{p} is a smaller set than \mathbf{p}' , our bounds may be tighter. If $\hat{V}_U[\mathbf{p}'] \prec \hat{V}_L[\mathbf{p}]$, then we can also conclude that $\mathbf{p}' \setminus \mathbf{p} \prec \mathbf{p}'$. We can incrementally construct an increasingly accurate estimate of $V[\mathbf{p}]$ by iteratively considering smaller and smaller subsets of an operator \mathbf{p} and pruning those subsets that cannot contain an optimal plan. This is depicted graphically in figures 2e and 2f.

We can make precise the construction of these increasingly fine subsets by introducing a *refinement relation* $\mathcal{R} \subset \mathcal{A}^* \times \mathcal{A}^*$, where $*$ denotes the Kleene closure. The elements of \mathcal{R} are ordered pairs $(\mathbf{p}, \mathbf{p}')$ such that $\mathbf{p}' \subset \mathbf{p}$. We can construct a relation \mathcal{R} by defining a procedure to generate plans \mathbf{p}' given a plan \mathbf{p} . First, define an operation $\text{HEAD} : \mathcal{A}^* \rightarrow \mathcal{A}$, which takes a plan \mathbf{p} and selects a single operator \mathbf{a} from it to replace with a more constrained refinement. We then define operations $\text{BASE} : \mathcal{A}^* \rightarrow \mathcal{A}^*$ and $\text{EXT} : \mathcal{A}^* \rightarrow \mathcal{A}^*$ that return the part of \mathbf{p} before and after $\text{HEAD}(\mathbf{p})$, respectively. Together, the three operators split a plan \mathbf{p} into three segments so that $\mathbf{p} = \text{BASE}(\mathbf{p}) \circ \text{HEAD}(\mathbf{p}) \circ \text{EXT}(\mathbf{p})$. Finally, we define a domain-specific relation $\mathcal{R} \subset \mathcal{A} \times \mathcal{A}^*$; this can be thought of as a function mapping an abstract operators to a set of abstract plans. Then $(\mathbf{p}, \mathbf{p}') \in \mathcal{R}$ if and only if $\mathbf{p}' = \text{BASE}(\mathbf{p}) \circ \mathbf{p}'' \circ \text{EXT}(\mathbf{p})$ and $(\text{HEAD}(\mathbf{p}), \mathbf{p}'') \in \mathcal{R}$. If $(\mathbf{a}, \mathbf{p}) \in \mathcal{R}$, we call \mathbf{p} a *refinement* of \mathbf{a} ; similarly, if $(\mathbf{p}, \mathbf{p}') \in \mathcal{R}$, we call \mathbf{p}' a *refinement* of \mathbf{p} .

We can combine these elements into an *abstraction* over a problem domain $(\mathcal{X}, c, s_0, S_g)$. Formally, an abstraction is a tuple $(\mathcal{S}, \mathcal{A}, \bar{\mathcal{R}}, \hat{V})$, where

- \mathcal{S} is a collection of propositional symbols,
- \mathcal{A} is a collection of operators, including a distinguished top-level operator Act ,
- $\bar{\mathcal{R}} \subset \mathcal{A} \times \mathcal{A}^*$ is a refinement relation, and
- \hat{V} is a symbolic valuation bound.

The valuation bound encodes both the cost and the dynamics of our problem domain. The refinement relation structures the space of abstract plans.

Angelic planning algorithms accept an abstraction as an argument in much the same way that the A* search algorithm [Hart *et al.*, 1968] accepts a heuristic. This raises an important question: under what circumstances will an abstraction $(\mathcal{S}, \mathcal{A}, \bar{\mathcal{R}}, \hat{V})$ allow us to find the optimal primitive plan for a domain $(\mathcal{X}, c, s_0, S_g)$, and to prove we have done so? We will generalize the idea of an admissible heuristic to define an *admissible* abstraction. As we will show in section 5, two properties suffice.

Definition 1. An abstraction $(\mathcal{S}, \mathcal{A}, \bar{\mathcal{R}}, \hat{V})$, defined over a planning domain (\mathcal{X}, c) , is *admissible* if

1. For each abstract operator $\mathbf{a} \in \mathcal{A}$, for each primitive plan p in \mathbf{a} , there is a refinement \mathbf{p} of \mathbf{a} such that $p \in \mathbf{p}$, i.e.,

$$\forall \mathbf{a} \in \mathcal{A}, \forall p \in \mathbf{a}, \exists (\mathbf{a}, \mathbf{p}) \in \bar{\mathcal{R}} : p \in \mathbf{p}. \quad (15)$$
2. \hat{V} is *admissible*, i.e., $\hat{V}_L[\mathbf{p}] \preceq V[\mathbf{p}] \preceq \hat{V}_U[\mathbf{p}]$ for each abstract operator $\mathbf{p} \in \mathcal{A}$.

The first property ensures that we do not “lose track” of any primitive plans while refining a plan. Plans are only removed from consideration when they are deliberately pruned. The second property ensures that if abstract plans $\mathbf{p}, \mathbf{p}' \in P$, where P is a collection of abstract plans, and $\hat{V}_U[\mathbf{p}] \prec \hat{V}_L[\mathbf{p}']$, then no optimal plan is in \mathbf{p}' and thus the best plan in p is also in the set $P' = P \setminus \{\mathbf{p}'\}$. Taken together, these properties ensure that if P' is the result of refining and pruning a collection of plans P , then for every plan in P there is a plan that is no worse in P' . If we start with the set $P_0 = \{\text{Act}\}$, no sequence of refinement and pruning operations will discard an optimal solution. This ensures completeness. To construct planning algorithms, we simply need to choose an order in which to refine and prune, and keep track of bounds to know when we can terminate the search.

4.1 The Flat Abstraction for Graph Search

We illustrate the construction of an admissible abstraction with graph search. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where each edge $e \in \mathcal{E}$ has an associated cost c_e . Suppose our objective is to find the shortest path to a goal node $v_g \in \mathcal{V}$ and we have an admissible heuristic $h : \mathcal{V} \rightarrow \mathbb{R}_{\geq 0}$. Then the abstraction $\mathcal{A}_{\mathcal{G}} = (\mathcal{V}, \mathcal{E} \cup \{\text{ACT}\}, \hat{V}, \bar{\mathcal{R}})$ is admissible, where

- $\hat{V}[\mathbf{a}_e] = \{(\{e_0\}, \{e_1\}, c_e, c_e)\}$,
- $\hat{V}[\text{ACT}] = \{(\{v\}, \{v_g\}, h(v, v'_g), \infty) : v \in \mathcal{V}\}$

- $\bar{\mathcal{R}}$ is the union of $\{(\text{ACT}, e \circ \text{ACT}) \mid e \in \mathcal{E}\}$ and $\{(\text{ACT}, e) \mid e \in \mathcal{E} : e_1 = v_g\}$.

Admissibility of \hat{V} follows immediately from the admissibility of h , and the admissibility of $\bar{\mathcal{R}}$ is easily proven. By definition, any primitive plan p is contained in ACT . Every primitive plan in the abstract plan $p \circ \text{ACT}$ is of the form $p \circ p'$. Suppose the first primitive operator in p' is e . For each such p and p' , $(\text{ACT}, e \circ \text{ACT}) \in \bar{\mathcal{R}}$. Therefore $\bar{\mathcal{R}}$ is admissible, and so $\mathcal{A}_{\mathcal{G}}$ is admissible.

This demonstrates that the machinery of angelic abstractions is at least as general as heuristics in graph search: every graph search problem can be reformulated as an abstract search, using the edges to define a refinement operation and an admissible heuristic to define lower bounds. Often, however, we can use domain-specific information to devise even more informative abstractions. In the remainder of this section, we will provide concrete examples of admissible abstractions for a pair of simple *continuous* planning problems.

4.2 An Abstraction for Navigation

A common problem in robotics is navigating to some specified goal location in a structured environment. Simple heuristics like the Euclidean distance to the goal work well in environments that are cluttered but largely unstructured, where the distance is a good proxy for the true cost. In highly structured environments, however, the Euclidean distance can be quite a bad proxy for cost. Consider the example in figure 9, in which the robot starts just on the other side of a wall from the goal. Using A* with a Euclidean heuristic requires searching almost the entire space.

We can plan more efficiently by taking advantage of structure in the environment. Suppose we have a decomposition of the environment into a finite set of overlapping regions, as in figure 2, and we know which regions overlap. These regions can be derived from a semantic understanding of the environment, such as rooms and doorways, or they can be automatically extracted using (for example) the constrained Delaunay triangulation. Then any plan can be described by the sequence of regions it moves through. We can use this region decomposition to define an abstraction.

Let $\mathcal{S} = \{R_i\}$, where $\cup_i R_i = \mathcal{X}$, and let $\mathcal{A} = \mathcal{A}_0 \cup \{\mathbf{a}_{ij} : R_i \cap R_j \neq \emptyset\} \cup \{\text{ACT}\}$, where $p \in \mathbf{a}_{ij}$ if $p(s) \in R_i \forall s \in [0, 1) \wedge p(1) \in R_j$. The refinement can be defined as follows.

$$\begin{aligned} \bar{\mathcal{R}} = \bigcup_{ij} \{ & (\text{ACT}, \mathbf{a}_{ij} \circ \text{ACT}), (\text{ACT}, \mathbf{a}_{ij}) \} \cup \\ & \{(\mathbf{a}_{ij}, a \circ \mathbf{a}_{ij}) : a(t) \in R_i \forall t\} \cup \\ & \{(\mathbf{a}_{ij}, a) : a(t) \in R_i \forall t, a(1) \in \text{cl}(R_j)\} \end{aligned} \quad (16)$$

We can use spatial indices like k-D trees and R-trees to quickly find the operators that are valid from a particular state. It is straightforward to show this refinement relation is admissible (see appendix A, proposition 4).

If the cost function is path length, then we can compute bounds using geometric operations. Executing the action \mathbf{a}_{ij} from a state in $R_k \cap R_i$ would incur a cost at least as great as the set distance $\inf\{\|s - s'\| : s \in R_i \cap R_k, s' \in R_i \cap R_j\}$. If the intersections between sets are small and well-separated,

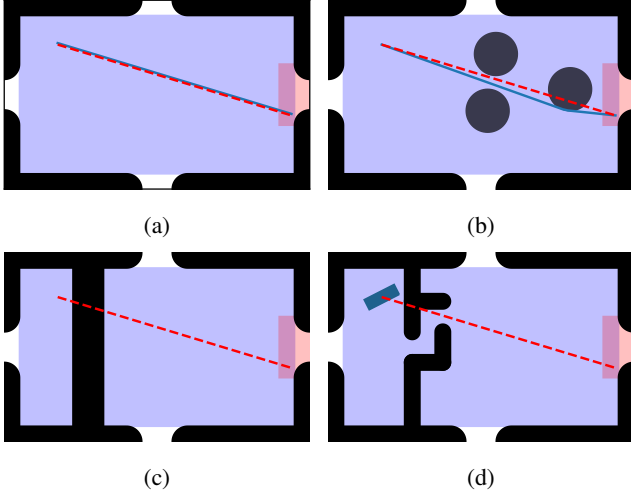


Figure 3: Useful regions for defining abstract operators are nearly convex. In all four examples here, the lower bound is given by the Euclidean distance in work space. In a convex region (a), the gap between the lower bound on the cost of a plan and the true optimal cost is zero. In an ϵ -convex region (b), the gap between the lower bound l on the cost of a plan and the true optimal cost c is small: $l < c < (1 + \epsilon)l$. Some regions are not ϵ -convex for any finite ϵ ; for example, the region might not be connected (c). This can happen even if the region is connected in the work space (d) if it is not connected in configuration space. Here, the object cannot fit through the narrow gap, and so the region is not ϵ -convex. In the presence of dynamic constraints, regions can fail to be path-connected even if they are connected in the workspace.

this lower bound will be an accurate estimate. This has the effect of heuristically guiding the search towards the next region, allowing us to perform a search in the (small) space of abstract plans rather than the (large) space of primitive plans. The Euclidean heuristic can deal with things like clutter and unstructured obstacles, while the abstraction can take advantage of structure in the environment.

Note that we have made no reference to the shape of the regions, nor even to their connectedness. If regions can be disconnected, for instance by an obstacle, abstract operators can have no upper bound, which can lead the search to be inefficient. On the other hand, if we require the regions to be convex, then executing the action a_{ij} from a state in $R_k \cap R_i$ would incur a cost no greater than the Hausdorff distance $d_H(R_i \cap R_k, R_i \cap R_j)$, where

$$d_H(X, Y) = \max(\sup_{x \in X} \inf_{y \in Y} \|x - y\|, \sup_{y \in Y} \inf_{x \in X} \|x - y\|). \quad (17)$$

Convexity is quite a strong requirement. In a cluttered environment, a convex representation may need to contain many regions. We can relax the requirement of convexity, and generalize to costs besides path length, by defining ϵ -convexity. A region R is ϵ -convex if

$$\inf_{p \in \mathcal{P}_R(x, x')} C[p] \leq (1 + \epsilon) \|x - x'\|. \quad (18)$$

This is shown graphically in figure 3. Intuitively, a region is ϵ -convex if the shortest path between any two points is

only slightly longer than the distance between the points. For example, if $\mathcal{X} \subset \mathbb{R}^n$, a convex region R cluttered with convex objects of diameter less than d is ϵ -convex, with $\epsilon = \pi \sqrt{\frac{n}{2(n+1)}}$; this is an elementary consequence of Jung’s theorem [Jung, 1899].

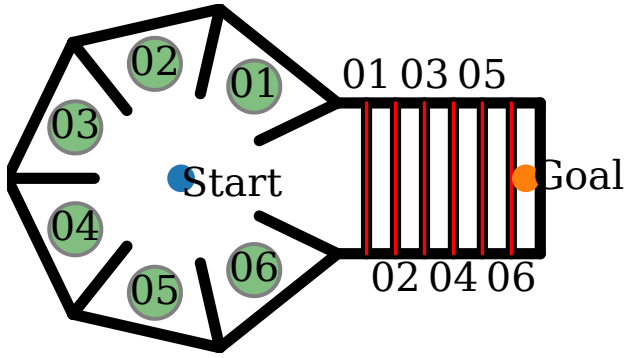
4.3 An Abstraction for the Door Puzzle

The door puzzle introduced in the introduction combines the motion-planning aspects of navigation with a high-level task planning problem: the choice of which doors to open and in which order. Unlike in the navigation problem, the configuration space for the door problem involves discrete components: $\mathcal{X} \subset \mathbb{R}^2 \times \{0, 1\}^N$, where N is the number of doors. This creates an element of combinatorial search that is not present in the navigation experiment.

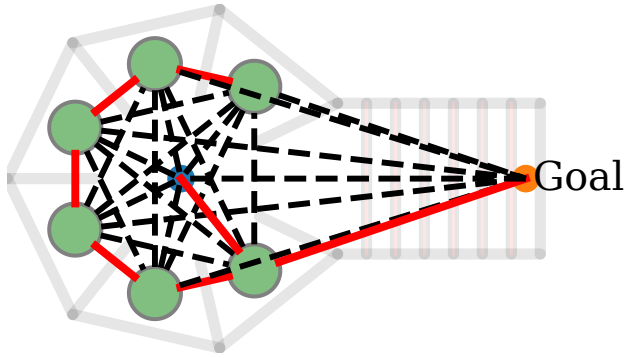
We use the same region-based abstraction to guide the search for motion plans, and construct a relaxed representation of the effects of toggling switches in PDDL by omitting geometric constraints like collision. Using this representation, we can quickly compute a partial ordering on the sequence of switches that need to be pressed in order to reach the goal. For example, in figure 4, the path to the goal is blocked by six doors. Before we can move towards the goal, we must move to and press each of the six switches. This leaves us with the task of computing a lower bound on the cost to reach and toggle each switch. We can find such a bound in two steps. First, we construct a directed graph whose vertices are the possible effects of executing each operator, and whose edges have weights that lower bound the cost of executing each operator. This graph, and the minimum spanning tree, are drawn in figure 4b. This reduces the problem of finding a lower bound to solving a travelling salesperson problem (TSP). While solving a TSP requires exponential time, we can compute a lower bound on the cost of the optimal solution by computing a minimum spanning tree of the directed graph—and this is a computation that can be done in polynomial time with standard methods. Although this bound neglects possible interactions between the operators, it is admissible; in fact, it is an admissible special case of the more general (and inadmissible) h_{add} heuristic [Haslum and Geffner, 2000]. We can use this bound to guide the search for a more detailed motion plan (figure 4c).

5 Algorithms

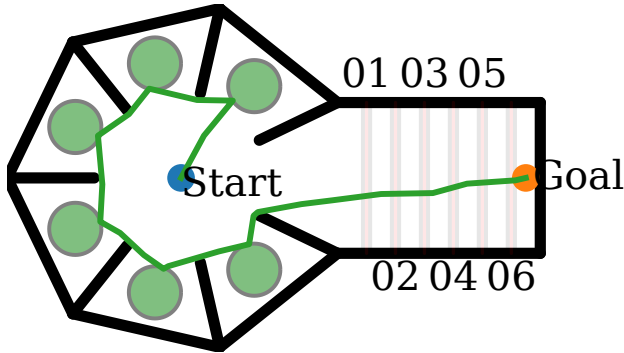
We now describe several algorithms that leverage an admissible angelic abstraction to search efficiently, even in high-dimensional continuous spaces. Our algorithms are all derived from the angelic hierarchical A* algorithm developed by [Marthi *et al.*, 2008]. We begin by reviewing this algorithm (section 5.1), then discuss a subtle variation that dramatically improves efficiency in some common degenerate or nearly degenerate cases (section 5.2). Finally, we discuss an extension that solves the approximately optimal planning problem, embracing the key insights of weighted A* (section 5.3).



(a) Problem



(b) Lower bound



(c) Solution

Figure 4: In the problem shown in (a), it is easy to conclude that all $N = 6$ doors must be opened before the robot can reach the goal. However, there are $N! = 720$ possible orders in which we might press the switches. We can bound the cost of any sequence by solving a travelling salesperson problem (b, dotted lines), where the edge costs are the minimal distance the robot must travel to move between switches. Although this is an NP-hard problem, we can compute a lower bound on the cost of a solution in polynomial time by computing a minimum spanning tree (b, solid red line). This allows the planner to quickly find a near-optimal solution (c).

5.1 Angelic A*

Angelic A* (algorithm 1) is a reformulation of the angelic hierarchical A* algorithm [Marthi *et al.*, 2008]. This algorithm solves the optimal planning problem using a best-first forward search over abstract plans.

The primary data structure maintained by our algorithm is a tree. Each node in the tree is a tuple $(a, p_-, \text{BASE}(p), \hat{V}[p])$ representing a plan $p = p_- \circ a$, where

- a is an abstract operator,
- p_- is a pointer to the predecessor of the node,
- $\text{BASE}(p)$ is a pointer to the base plan, which is used in choosing refinements, and
- $\hat{V}[p]$ is an admissible bound on the valuation of p .

The root of the tree is the node $(\emptyset, \emptyset, \emptyset, \{(\{x_s\}, \{x_s\}, 0, 0)\})$, representing the start of any plan.

The main entry point for the algorithm is the SEARCH routine, which first constructs the root plan node (line 2) then computes an initial abstract plan that includes all possible primitive plans (line 5). This abstract plan is then added to the plan queue (line 6).

Then, as long as a plan remains on the queue, AA* repeatedly finds the abstract plan in Q with the lowest lower bound (line 8). If this plan is dominated by a previously discovered plan, then the algorithm returns successfully, as any remaining plan on the queue is also dominated. Otherwise, AA* expands the active plan by computing its successors and adding them to the queue if they cannot be pruned (lines 12-17). If the queue becomes empty without discovering a primitive plan that reaches the goal, then no plan exists and the algorithm returns failure.

AA* generates successors to a plan using the refinement relation. It then constructs a set of *child* plans by selecting one operator from the plan and replacing it with its refinements. Any successor plan that cannot possibly contain an acceptable solution is pruned, while any plan that could contain an acceptable solution is added to the priority queue. The algorithm terminates when we remove a plan from the queue that is dominated by a previously expanded primitive plan.

We compute the valuation of each new plan incrementally (line 28). If that new plan does not optimistically reach some state with lower cost than a previously explored plan ending with the same extension, we discard it (line 37). Otherwise, we update the bounds on any plan with the current extension to include the new plan (line 39). Next, if the upper bound on the cost of reaching the goal under the new plan is better than any previous plan, we record this new plan as the best yet found (line 16). Finally, if the lower bound on the cost of reaching the goal under the new plan is better than the upper bound under any previous plan, we add it to the set Q of active plans.

Marthi *et al.* showed this algorithm will return the optimal refinement of the top-level operator ACT after a finite number of iterations, provided the lower bound on the cost of every operator is greater than zero.

Theorem 1. *Algorithm 1 will return the optimal primitive refinement of the abstract plan ACT, provided the lower bound*

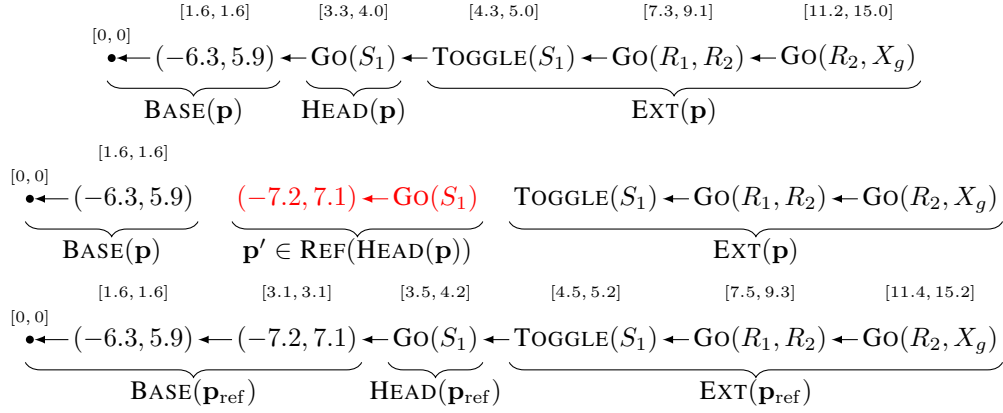


Figure 5: A schematic illustration of the process by which we construct successor plans. A plan is represented by a collection of nodes representing operators. Each node has a pointer to its predecessor, and represents the concatenation of the predecessor with its operator. Each node also has a pointer to a *base* node. To form the successors of a plan, we first break plan into three pieces: the base, the node after the base (called the head), and the rest of the plan (called the extension). We then replace the head with a valid refinement, chosen to be optimistically feasible. Finally, we propagate, creating new nodes corresponding to the operators in the refinement and the extension.

on the cost of every operator is strictly positive [Marthi et al., 2008].

However, if the abstraction is admissible, we can prove the following stronger claim.

Theorem 2. *If the abstraction \mathcal{A} is admissible and a feasible plan exists, then algorithm 1 returns an optimal sequence of primitive operators in finite time, provided the lower bound on the cost of every operator is greater than zero.*

Corollary 1. *If the set of primitive operators $\mathcal{A}_{0,n}$ is asymptotically optimal (equation 5), then*

$$\lim_{n \rightarrow \infty} \Pr(C[\text{SEARCH}(\mathcal{A}_n)] < (1 + \epsilon)c^*) = 1. \quad (19)$$

Proof. See appendix A, theorem 2. \square

The distinction between these claims is subtle, but important. Theorem 1 implies hierarchical optimality: if a plan is returned, no better plan can be expressed as a refinement of the top-level operator. Theorem 2 implies primitive optimality: if a plan is returned, no better plan exists. If we can ensure our abstraction is admissible, then using our abstraction provides the same guarantees as a direct search over the space of primitive plans, but may be much faster.

5.2 Acyclic Angelic A*

Algorithm 1 requires strictly positive lower bounds on the cost of any operator. In discrete problems, this is reasonable restriction, but it presents challenges in continuous problems. For example, suppose we have a plan consisting of two operators $\mathbf{a}_{ij} \circ \mathbf{a}_{i'j'}$ from our navigation abstraction. If the destination regions intersect—if $R_j \cap R'_j \neq \emptyset$ —then the largest possible lower bound for the valuation of $\mathbf{a}_{i'j'}$ is zero. This phenomenon can lead to a zero-cost cycle: a sequence of operators that can optimistically returns to a given state with zero cost (figure 8a). Even positive cost-cycles are problematic, if the lower bound l on the cost of a cycle is much smaller than the upper bound u : the algorithm can only prune a plan

if it executes the cycle $\lceil u/l \rceil$ times (figure 8b). Unfortunately, we cannot simply discard any abstract plan with a cycle: the optimal plan may leave and return to an abstract state if the state is non-convex, even if the state is connected (figure 8c-8e). Often, this indicates a poor choice of abstraction, but it arise even with natural choices of abstraction, especially in domains with topologically complex configuration spaces. We can deal with such edge cases while still avoiding cycles with a minor modification to the algorithm.

We define an acyclic plan as any plan \mathbf{p} that cannot be partitioned into two plans $\mathbf{p}_0 \circ \mathbf{p}_1$ such that $\hat{V}_L[\mathbf{p}_0] \leq \hat{V}_L[\mathbf{p}]$ (algorithm 3). When we compute the successors of a plan \mathbf{p} , if we find the extension \mathbf{p}_{ext} would create a cyclic (i.e. not acyclic) plan when propagated on top of $\text{BASE}(\mathbf{p})$, we do not add $\mathbf{p} \circ \mathbf{p}_{\text{ext}}$ to the set of successors. Instead, we add $(\text{BASE}(\mathbf{p}), \mathbf{p}_{\text{ext}})$ to the set of deferred plans (algorithm 2, line 12).

When any descendent of \mathbf{p} is expanded, we consider activating any deferred extension of \mathbf{p} by propagating it on top of the descendent plan. If the resulting plan is no longer cyclic, we add it to the set of successors (line 19). This ensures that only acyclic plans will ever be added to the queue of plans, while also ensuring all plans that are not pruned will eventually be considered.

Theorem 3. *If the abstraction \mathcal{A} is admissible and a feasible plan exists, then the acyclic angelic A* algorithm returns a sequence of primitive operators with cost no greater than $c^*(\{x_s\}, X_g)$ in finite time.*

Corollary 2. *If $\mathcal{A}_{0,n}$ is asymptotically optimal, then*

$$\forall \epsilon \geq 0, \lim_{n \rightarrow \infty} \Pr(C[\text{SEARCH}(\mathcal{A}_n)] < (1 + \epsilon)c^*) = 1. \quad (20)$$

Proof. See appendix A, theorem 3. \square

5.3 Approximate Angelic A*

Even with a good abstraction, finding an optimal solution may be intractable for many problems. By modifying the

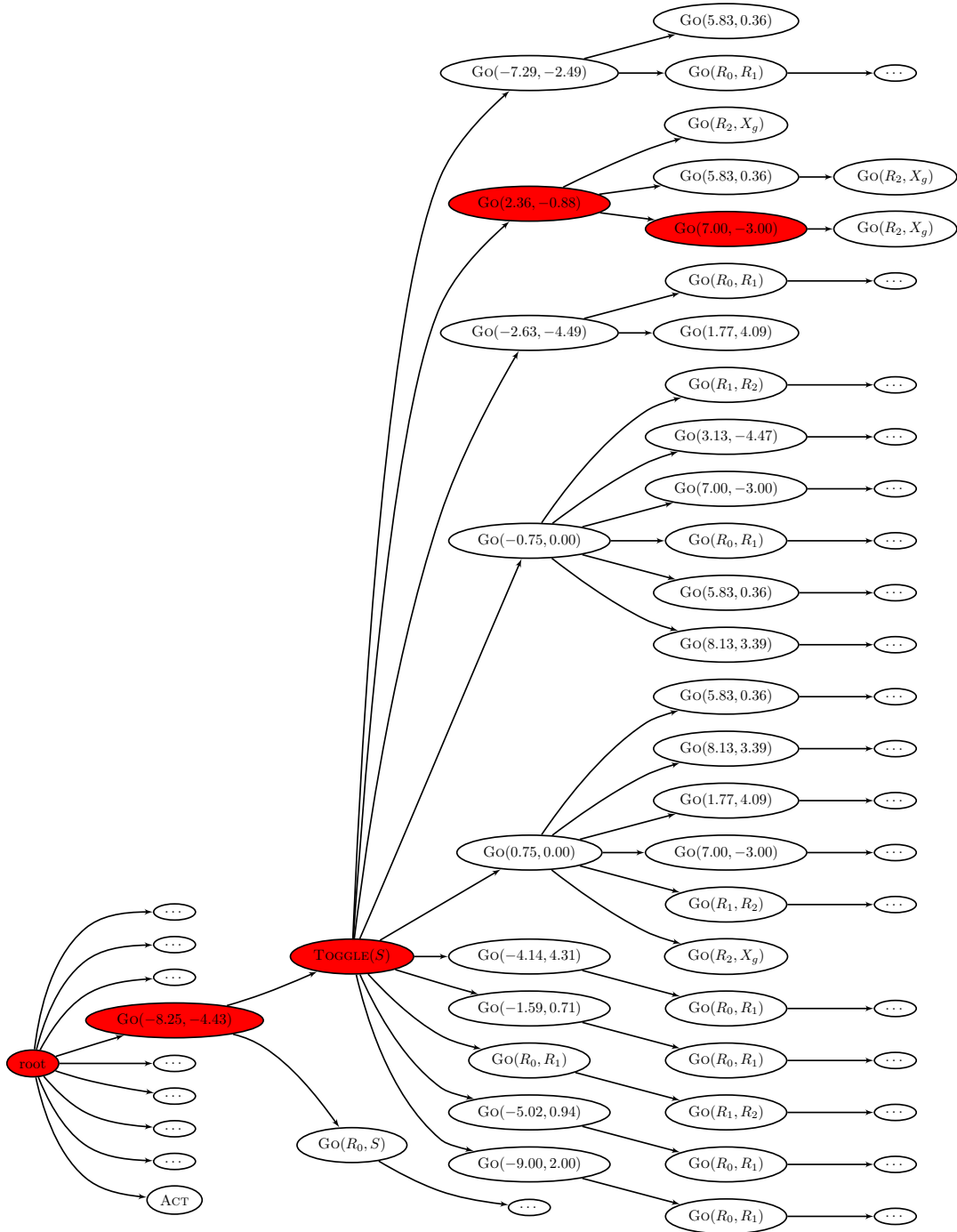


Figure 6: Part of the plan tree constructed by AA* for the problem shown in figure 7. Each node represents a plan; edges link a node to its predecessor. Nodes that are part of the optimal plan are highlighted in red. Branches of the tree not drawn are indicated with an ellipsis. The act of opening the door is referred to as $\text{TOGGLE}(S)$. Primitive motion operators are referred to as $\text{Go}(x, y)$, where x and y are coordinates. An abstract motion to a region R_j through a region R_i is referred to as $\text{Go}(R_i, R_j)$. The top level operator is labelled ACT.

Algorithm 1 Angelic A*

```

1: function SEARCH(abstraction  $(S, \mathcal{A}, \bar{\mathcal{R}}, \hat{V})$ )
2:   root =  $(\emptyset, \emptyset, \emptyset, \{(x_s, x_g, 0, 0)\})$ 
3:    $\mathbf{p}^* = \emptyset$ 
4:   BOUND( $\emptyset$ ) =  $\hat{V}[\text{ACT}]$ 
5:    $\mathbf{p}_0 = \text{PROPAGATE}(\text{root}, [\text{ACT}])$ 
6:    $Q = \{\mathbf{p}_0\}$ 
7:   while  $|Q| > 0$  do
8:      $\mathbf{p} = \arg \min \{\hat{V}[\mathbf{p}](\{x_s\}, X_g) : \mathbf{p} \in Q\}$ 
9:     if PRIMITIVE( $\mathbf{p}^*$ ) and  $\hat{V}_U[\mathbf{p}^*] < \hat{V}_L[\mathbf{p}]$  then
10:      return  $\mathbf{p}^*$ 
11:     else
12:        $Q \leftarrow Q \setminus \{\mathbf{p}\}$ 
13:        $S \leftarrow \text{SUCCESSORS}(\mathbf{p})$ 
14:       for  $\mathbf{p}' \in S$  do
15:         if  $\hat{V}_U[\mathbf{p}'] < \hat{V}_U[\mathbf{p}^*]$  then
16:            $\mathbf{p}^* \leftarrow \mathbf{p}'$ 
17:        $Q \leftarrow Q \cup \{\mathbf{p}' \in S : \neg \hat{V}_U[\mathbf{p}^*] < \hat{V}_L[\mathbf{p}]\}$ 
18:   return  $\emptyset$ 
19: function SUCCESSORS(plan node  $\mathbf{p}$ )
20:   POST(BASE( $\mathbf{p}$ )) =  $\{s' : (s, s', l, u) \in \hat{V}[\text{BASE}(\mathbf{p})]\}$ 
21:    $\mathbf{a} = \text{OPERATOR}(\text{HEAD}(\mathbf{p}))$ 
22:    $S = \emptyset$ 
23:   for  $\mathbf{p}' : (\mathbf{a}, \mathbf{p}') \in \bar{\mathcal{R}}, \exists s \in \text{POST}(\text{BASE}(\mathbf{p})) : \text{HEAD}(\mathbf{p}') \cap s \neq \emptyset$  do
24:      $\mathbf{p}_{\text{ref}} \leftarrow \text{PROPAGATE}(\text{BASE}(\mathbf{p}), \mathbf{p}' \circ \text{EXT}(\mathbf{p}))$ 
25:     if  $\hat{V}_L[\mathbf{p}_{\text{ref}}](x_s, X_g) < \infty$  then
26:        $S \leftarrow S \cup \{\mathbf{p}_{\text{ref}}\}$ 
27:   return  $S$ 
28: function PROPAGATE(base node  $\mathbf{p}$ , list  $\mathbf{p}_{\text{ext}}$ )
29:    $\mathbf{b} \leftarrow \mathbf{p}$ 
30:   while  $\mathbf{p}_{\text{ext}}$  is not empty do
31:      $\mathbf{a} \leftarrow \text{POP}(\mathbf{p}_{\text{ext}})$ 
32:     if  $\mathbf{a}$  is more primitive than OPERATOR( $\mathbf{p}$ ) then
33:        $\mathbf{b} \leftarrow \mathbf{p}$ 
34:      $\mathbf{p} \leftarrow (\mathbf{a}, \mathbf{p}, \mathbf{b}, \hat{V}[\mathbf{p} \circ \mathbf{a}])$ 
35:     if  $\hat{V}[\mathbf{p}] = \emptyset$  then return
36:     else if BOUND( $\mathbf{p}_{\text{ext}}$ )  $< \hat{V}_L[\mathbf{p}]$  then
37:       return  $\emptyset$ 
38:     else
39:       BOUND( $\mathbf{p}_{\text{ext}}$ )  $\leftarrow \text{BOUND}(\mathbf{p}_{\text{ext}}) \cup \hat{V}[\mathbf{p}]$ 
40:   return  $\mathbf{A}$ 

```

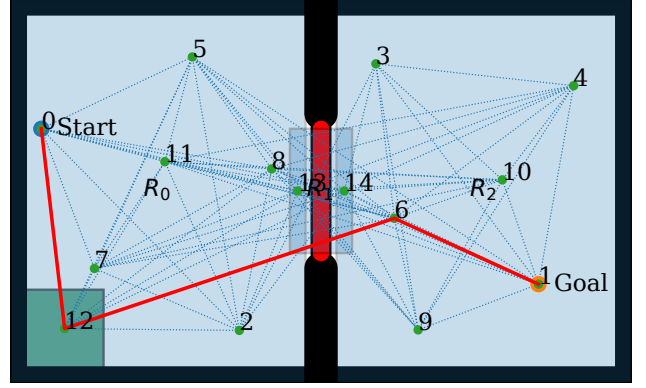


Figure 7: The problem solved by the tree in figure 6. The plan first toggles the switch in the lower left hand corner, then moves through the opened door to reach the goal. Primitive operators (edges) are displayed as dotted lines, while the optimal plan is highlighted in red.

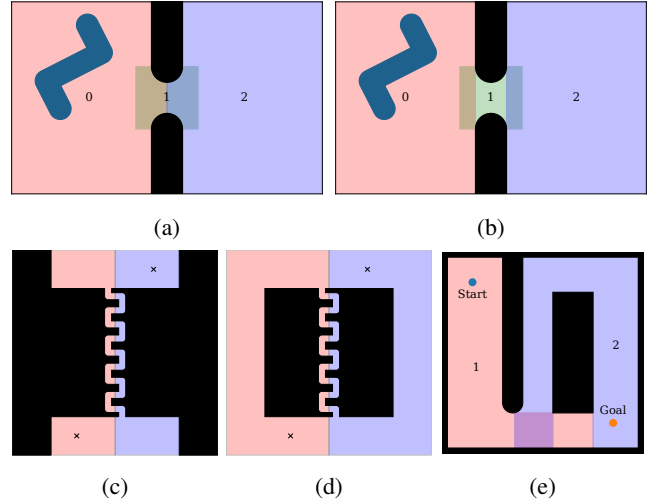


Figure 8: An illustration of the problems with cyclic paths. Many natural operators in continuous domains have a cost with a lower bound of zero and no upper bound. For example, deciding whether the irregularly-shaped object (a) can reach the blue region requires detailed geometric analysis. Since regions 0 and 2 touch, the greatest lower bound on the cost of a plan in $\mathbf{a}_{01} \circ \mathbf{a}_{12}$ is the same as the bound on $\mathbf{a}_{01} \circ \mathbf{a}_{12} \circ \mathbf{a}_{10} \circ \mathbf{a}_{12}$. Any number of repetitions of the cycle $\mathbf{a}_{10} \circ \mathbf{a}_{12}$ will have the same cost, and so if $\mathbf{a}_{01} \circ \mathbf{a}_{12}$ is ever selected for expansion, the algorithm will only ever refine this infinite sequence of cyclic plans. Separating the regions (b) eliminates the infinite recursion, but remains inefficient; each cycle of $\mathbf{a}_{10} \circ \mathbf{a}_{12}$ adds only a small cost ϵ to the lower bound, meaning that if the next plan on the queue has a cost δ greater, the planner will consider $\lceil \delta/\epsilon \rceil$ cycles before considering the next acyclic plan. We cannot simply ignore these ‘cyclic’ plans; in some scenarios, the best plan (c) or any feasible plan (d) is cyclic. This can occur even if the regions defining our operators are connected in configuration space: in the diagram in (e), although there is a feasible plan in $\mathbf{a}_{12} \circ \mathbf{a}_{21} \circ \mathbf{a}_{2g}$, the optimal plan is in $\mathbf{a}_{12} \circ \mathbf{a}_{21} \circ \mathbf{a}_{2g}$.

Algorithm 2 Acyclic angelic A*

```

1: function SUCCESSORS(plan node  $\mathbf{p}$ )
2:    $\triangleright D$  is a global variable, initially set to  $\emptyset$ .
3:    $\text{POST}(\text{BASE}(\mathbf{p})) = \{s' : (s, s', l, u) \in \hat{V}[\text{BASE}(\mathbf{p})]\}$ 
4:    $S = \emptyset$ 
5:    $\mathbf{a} = \text{OPERATOR}(\text{HEAD}(\mathbf{p}))$ 
6:   for  $\mathbf{p}' : (\mathbf{a}, \mathbf{p}') \in \bar{\mathcal{R}}, \exists s \in \text{POST}(\text{BASE}(\mathbf{p})) : \text{HEAD}(\mathbf{p}') \cap s \neq \emptyset$  do
7:      $\mathbf{p}_{\text{ref}} \leftarrow \text{PROPAGATE}(\text{BASE}(\mathbf{p}), \mathbf{p}' \circ \text{EXT}(\mathbf{p}))$ 
8:     if  $\hat{V}_L[\mathbf{p}_{\text{ref}}](x_s, X_g) < \infty$  then
9:       if  $\text{ACYCLIC}(\mathbf{p}_{\text{ref}}, \emptyset)$  then
10:         $S \leftarrow S \cup \{\mathbf{p}_{\text{ref}}\}$ 
11:       else
12:         $D \leftarrow D \cup \{(\text{BASE}(\mathbf{p}), \text{EXT}(\mathbf{p}_{\text{ref}}))\}$ 
13:    $\mathbf{p}_a \leftarrow \mathbf{p}$ 
14:   while  $\text{BASE}(\text{PARENT}(\mathbf{p}_a)) \neq \emptyset$  do
15:      $\mathbf{p}_a \leftarrow \text{BASE}(\text{PARENT}(\mathbf{p}_a))$ 
16:     for  $\mathbf{p}_{\text{ext}} : (\mathbf{p}_a, \mathbf{p}_{\text{ext}}) \in D$  do
17:        $\mathbf{p}_{\text{ref}} \leftarrow \text{PROPAGATE}(\text{BASE}(\mathbf{p}), \mathbf{p}_{\text{ext}})$ 
18:       if  $\text{ACYCLIC}(\mathbf{p}_{\text{ref}}, \emptyset)$  and  $\hat{V}_L[\mathbf{p}_{\text{ref}}] < \infty$  then
19:         $S \leftarrow S \cup \{\mathbf{p}_{\text{ref}}\}$ 
20:   return  $S$ 

```

Algorithm 3 Acyclic angelic A*

```

1: function ACYCLIC(plan nodes  $\mathbf{p}, \mathbf{p}'$ )
2:   if  $\mathbf{p} = \emptyset$  then
3:     return true
4:   else if  $\mathbf{p}' = \emptyset$  then
5:      $\mathbf{p}_- \leftarrow \text{PREDECESSOR}(\mathbf{p})$ 
6:     return  $\text{ACYCLIC}(\mathbf{p}_-, \emptyset) \wedge \text{ACYCLIC}(\mathbf{p}_-, \mathbf{p})$ 
7:   else
8:      $\mathbf{p}_- \leftarrow \text{PREDECESSOR}(\mathbf{p})$ 
9:     return  $\neg(\hat{V}_L[\mathbf{p}] \preceq \hat{V}_L[\mathbf{p}']) \wedge \text{ACYCLIC}(\mathbf{p}_-, \emptyset)$ 

```

order in which plans are expanded and the conditions under which the algorithm terminates, we can accelerate the search process while still ensuring approximate optimality. This modification is described in algorithm 4.

Often, admissible valuations are unduly optimistic: the lower bound $L[\mathbf{p}]$ on the cost of a plan is much less than the true optimal cost $c^*(\{x_s\}, X_g)$. This problem is well-understood in the context of graph search, where it is often mitigated by using an admissible heuristic that has been inflated, as in weighted A* ([Pohl, 1970]). WA* keeps a queue of states, and expands the state minimizing

$$\text{KEY}_{\text{WA}^*}(x; w) = g(x) + wh(x), \quad (21)$$

where $g(x)$ is the estimated cost to reach a state x and $h(x)$ is an admissible estimate of the cost to reach the goal from a state x . This biases the search towards plans that pessimistically reach states close to the goal. If $h(x) - c^*(\{x\}, X_g)$ has only shallow local minima, this will explore far fewer states before finding a path to the goal than A* would. Moreover, when a path to the goal is found, it will have cost less than $wc^*(\{x_s\}, X_g)$.

Unfortunately, we cannot directly apply this computation of an inflated priority in the context of angelic search. When we use angelic semantics, we may not have a distinct cost $g(x)$ to reach a state; we only have bounds on the cost of plans. In order to apply the idea of WA* to angelic search, we need to compute a priority that satisfies the same properties as $\text{KEY}_{\text{WA}^*}(x; w)$. A naïve approach, such as inflating the lower bound on each operator, does not have the desired effect: it would inflate the priority of all plans equally and would not affect the order in which plans are expanded.

A more reasonable approach might be to inflate the lower bounds on each nonprimitive operator, which would exactly equal the priority computed by WA* for a flat abstraction—but this approach does not properly take upper bounds into account. Consider an operator \mathbf{a} for which $\hat{V}_U[\mathbf{a}](s, s') = (1 + \varepsilon)L[\mathbf{a}](s, s')$ for some abstract state pair (s, s') . If ε is zero, the operator would be treated as primitive and its lower bound would not be inflated. If it is small but positive, it would be treated as nonprimitive. This would artificially bias the search away from operators that are almost, but not quite, primitive.

To avoid this undesirable bias, we recursively compute a priority $\text{KEY}(\mathbf{p}, w)$. If $\mathbf{p}_- = \text{PREDECESSOR}(\mathbf{p})$,

$$\text{KEY}(\mathbf{p}; w) = \min(\text{KEY}(\mathbf{p}_-) + w(\hat{V}_L[\mathbf{p}] - \hat{V}_L[\mathbf{p}_-]), \hat{V}_U[\mathbf{p}]), \quad (22)$$

with $\text{KEY}(\emptyset; w) = 0$. $\text{KEY}(\mathbf{p}; w)$ has several useful properties. For each plan \mathbf{p} , $\text{KEY}(\mathbf{p}; w)$ is no greater than the upper bound $U[\mathbf{p}]$ or the inflated lower bound $wL[\mathbf{p}]$. If $w = 1$, then $\text{KEY}(\mathbf{p}; w) = L[\mathbf{p}]$. For a primitive plan, $\text{KEY}(\mathbf{p}; w)$ equals the cost of the plan. In a flat abstraction, $\text{KEY}(\mathbf{p}; w)$ is precisely equal to the cost estimate used by WA*.

We refer to this approach as *approximate* angelic A*, and present pseudocode in algorithm 4. The pseudocode is substantially similar to algorithm 1, and identical subroutines have been omitted. Changes are highlighted in red.

This algorithm is approximately optimal, in the sense that any plan it returns has cost less than w times the cost of the

Algorithm 4 Approximate angelic A*

```

1: function SEARCH(abstraction  $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \hat{V})$ , weight  $w$ )
2:   root =  $(\emptyset, \emptyset, \emptyset, \{(x_s, x_s, 0, 0)\})$ 
3:    $\mathbf{p}^* = \emptyset$ 
4:   BOUND( $\emptyset$ ) =  $\hat{V}[\text{ACT}]$ 
5:    $\mathbf{p}_0 = \text{PROPAGATE}(\text{root}, [\text{ACT}])$ 
6:    $Q = \{\mathbf{p}_0\}$ 
7:   while  $|Q| > 0$  do
8:      $\mathbf{p} = \arg \min \{\text{KEY}(\mathbf{p}, w) : \mathbf{p} \in Q\}$ 
9:     if PRIMITIVE( $\mathbf{p}^*$ ) and  $\hat{V}_U[\mathbf{p}^*] < \hat{V}_L[\mathbf{p}]$  then
10:      return  $\mathbf{p}^*$ 
11:     else
12:       $Q \leftarrow Q \setminus \{\mathbf{p}\}$ 
13:       $S \leftarrow \text{SUCCESSORS}(\mathbf{p})$ 
14:      for  $\mathbf{p}' \in S$  do
15:        if  $\hat{V}_U[\mathbf{p}'] < \hat{V}_U[\mathbf{p}^*]$  then
16:           $\mathbf{p}^* \leftarrow \mathbf{p}'$ 
17:       $Q \leftarrow Q \cup S$ 
18:   return  $\emptyset$ 

```

Algorithm 5 Approximate angelic search priority

```

1: function KEY(node  $\mathbf{p}$ , weight  $w \in \mathbb{R}_{\geq 1}$ )
2:   if  $\mathbf{p} = \emptyset$  then
3:     return 0
4:   else
5:      $\mathbf{p}_- \leftarrow \text{PREDECESSOR}(\mathbf{p})$ 
6:     return  $\min(\text{KEY}(\mathbf{p}_-) + w(\hat{V}_L[\mathbf{p}] - \hat{V}_L[\mathbf{p}_-]), \hat{V}_U[\mathbf{p}])$ 

```

optimal plan. If we combine the acyclic generation of successor plans with the approximate search, the resulting algorithm is approximately optimal even if there are zero-cost operators.

Theorem 4. *If the abstraction \mathcal{A} is admissible and a feasible plan exists, then algorithm 4 returns a sequence of primitive operators with cost no greater than $w \cdot c^*(\{x_s\}, X_g)$ in finite time.*

Corollary 3. *If $\mathcal{A}_{0,n}$ is asymptotically optimal, then*

$$\forall \epsilon \geq 0, \lim_{n \rightarrow \infty} \Pr(C[\text{SEARCH}(\mathcal{A}_n)] < (1 + \epsilon)w \cdot c^*) = 1. \quad (23)$$

Proof. See appendix A, theorem 4. \square

If we use this idea in the angelic A* algorithm, rather than the acyclic angelic A* algorithm, the same theorem holds with the additional constraint that all operators must have a strictly positive lower-bound.

6 Results

We implemented algorithms 1-4 and the abstractions described in sections 4.2 and 4.3 in the Python programming language. We then compared the performance of the planner to the original angelic A* search algorithm ([Marthi *et al.*, 2008]) and to a search without abstraction using A*.

In the navigation domain, we constructed a random discretization with 10^4 states. Examples of the search trees constructed by A* and by algorithm 2 are given in figure 9. By

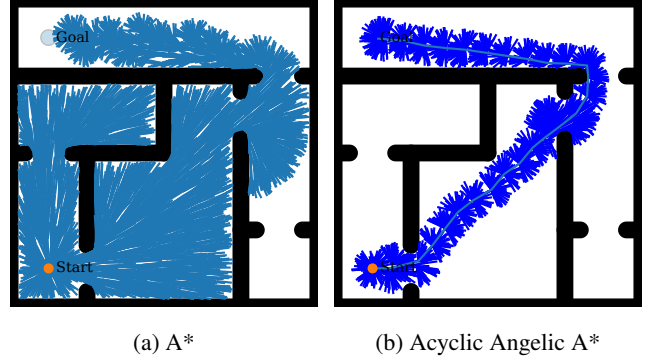


Figure 9: The search trees constructed by A* (a) and by algorithm 2 (b). Note that the A* search needs to explore almost the entire space, due to limitations of the Euclidean distance as a heuristic. In contrast, when provided with a decomposition of the world into nearly-convex regions, angelic A* can find a path to the goal while exploring far fewer states. By avoiding plans with cycles, our modified angelic planning algorithm can explore these states while expanding far fewer plans.

	cost	time	plans	states
A*	33.430	42.119	11807	7948
Angelic A*	33.430	160.256	25770	4758
AAA* (w=1.)	33.430	4.159	706	3068
AAA* (w=2.5)	35.586	0.697	48	1443

Table 1: Quantitative performance on a problem instance in the navigation domain. The discretized state space includes 10^4 sampled configurations. We see that abstraction and approximation result in exploring fewer plans and states, yielding a faster search and optimal or nearly optimal results.

using the abstraction, the algorithm can avoid exploring large parts of the configuration space. Our quantitative results bear this out: using abstraction allows us to reduce the number of states explored by a factor of three and the number of plans considered by several orders of magnitude.

Using abstraction in the door puzzle domain resulted in even larger speedups. Even in easy problem instances with only a few doors, search without abstraction quickly became infeasible (figure 10). Using abstraction reduced the number of states explored by orders of magnitude. However, the unmodified angelic search spent a great deal of time exploring plans with cycles. By deferring these plans, our algorithms were able to reduce the number of plans expanded by an order of magnitude. In fact, only our algorithm was able to solve problem instances with more than ten doors. We were able to find 2-optimal plans for instances with up to 32 doors and 10^4 sampled configurations (corresponding to a discretized state space with approximately 40 trillion states). Unfortunately, software limitations prevented us from experimenting on states with more than 32 doors.

7 Related Work

There is a long history of using abstraction to solve robotic planning problems [Nilsson, 1984; Lozano-Pérez *et al.*,

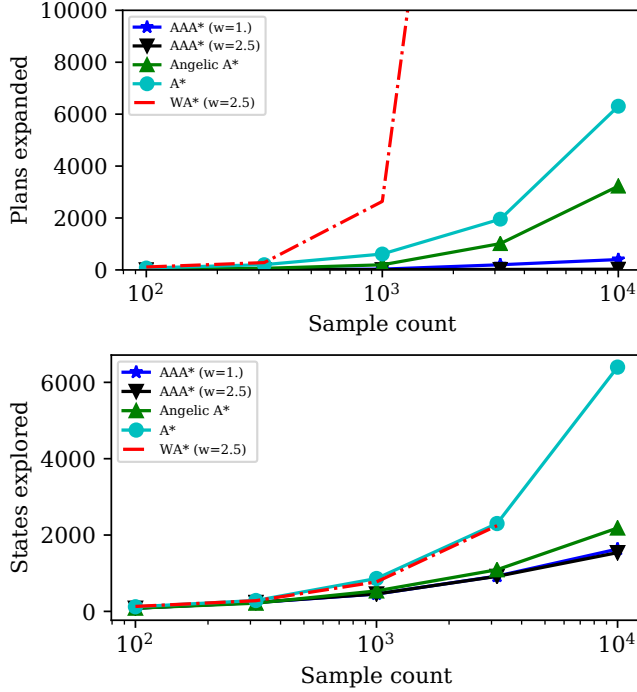


Figure 10: Quantitative evaluation on an easy instance of the door puzzle domain with only two doors. More difficult instances could not be solved by any algorithm considered except algorithm 2. The abscissa measures the number of randomly sampled states in the discretization of the configuration space. The ordinate axes measure the number plans expanded by each algorithm and the number of distinct configurations explored during search.

1987], and although our formulation is different from most standard approaches to task or motion planning, many authors [Alami *et al.*, 1990; Siméon *et al.*, 2004] have employed our underlying approach of searching for paths through a graph of configurations connected by feasible motion plans. Practical algorithms often overcome the high computational cost of searching these planning graphs using clever heuristics. For example, aSyMov [Cambon *et al.*, 2009] and FFRob [Garrett *et al.*, 2016] both employ the fast-forward heuristic, augmented with information derived from the geometric and kinematic computation. Like these approaches, our work is built atop a heuristically-guided search; however, angelic semantics allow us to define upper bounds which can be used to prune away abstract plans, and allow for admissible hierarchies of arbitrary depth.

Our definition of abstract plans is closely related to the notion of “plan skeletons” considered by several authors [Erdogan and Stilman, 2013; de Silva *et al.*, 2013; Lozano-Pérez and Kaelbling, 2014]. Plan skeletons fix a sequence of operators but leave continuous parameters undefined. There are many approaches to determining the feasibility of a given skeleton; for example, [Toussaint, 2015] uses continuous optimization techniques to search for optimal values of the real-valued variables. [Lozano-Pérez and Kaelbling, 2014] fix a discretization of the continuous variables then find feasible values by formulating and solving a constraint satisfaction problem. [Lagriffoul *et al.*, 2014] use linear programming to find valid values of the free variables or prove that none exist. The primary difference between our approach and these plan skeletons is the choice of formalism. By defining our abstract operators as implicitly defined *sets* of primitive motion plans, we can reason about plans at varying levels of abstraction in a unified way, which is essential to the generality of our guarantees.

Another approach to task and motion planning represents geometric information in a way amenable to search using classical AI search techniques. For example, [Dornhege *et al.*, 2010] model geometric information as predicates that can be resolved by solving motion planning problems during the task planning process. More recently, [Ferrer-Mestres *et al.*, 2017] show that by fixing a discretization, in some domains all geometric information can be represented compactly in planning languages more expressive than PDDL, avoiding the need to make geometric queries during the planning process. Other authors [Erdem *et al.*, 2011; Srivastava *et al.*, 2013; Dantam *et al.*, 2016] use the task planner as a partial or approximate representation of the underlying geometric task, which can be improved during search. For instance, [Erdem *et al.*, 2011] use a high-level task planner to find an optimal task plan, then use a motion planner to attempt to find a kinematically feasible primitive solution to that task plan. If no feasible solution exists, additional kinematic constraints are extracted from the motion planner and provided to the task planner, and the process is repeated.

Many authors have devised planning algorithms tailored to more specific task and motion planning domains. For example, the problem of navigation among movable obstacles has long been of practical interest, and probabilistically complete solutions have been known since 2008 [Stilman and

Kuffner, 2008; Nieuwenhuisen *et al.*, 2008]. Planning for non-prehensile manipulation has been addressed by [Dogar and Srinivasa, 2011] and by [Barry *et al.*, 2013]. Our work could provide a new analytical tool with which to study these special classes of problems, and perhaps formulate new algorithms with stronger performance guarantees.

8 Conclusions

We have defined conditions on an abstraction that allow us to accelerate planning while preserving the ability to find an optimal or near-optimal solution to complex motion planning problems. We motivate these conditions by deriving two admissible abstractions and showing they improve the efficiency of search without adversely affecting the quality of the resulting solutions. We view this work as a proof of concept, demonstrating that a good abstraction can render optimal planning feasible even on large problems. The classical planning community has developed several powerful families of admissible heuristics [Haslum and Geffner, 2000]; by reformulating these heuristics to employ angelic abstractions, we may be able to obtain optimal or near-optimal solutions to practical manipulation planning problems.

Acknowledgements

This research was sponsored by Northrop Grumman and by the Robotics Collaborative Technology Alliance (RCTA) of the US Army. Their support is gratefully acknowledged.

References

- [Alami *et al.*, 1990] Rachid Alami, Thierry Simeon, and Jean-Paul Laumond. A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps. In *Proceedings of the International Symposium on Robotics Research*, pages 453–463. MIT Press, 1990.
- [Barry *et al.*, 2013] Jennifer Barry, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. A hierarchical approach to manipulation with diverse actions. In *International Conference on Robotics and Automation*, pages 1799–1806. IEEE, 2013.
- [Cambon *et al.*, 2009] Stephane Cambon, Rachid Alami, and Fabien Gravot. A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research*, 28(1):104–126, 2009.
- [Dantam *et al.*, 2016] Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems*, pages 1–6, 2016.
- [de Silva *et al.*, 2013] Lavindra de Silva, Amit Kumar Pandey, Mamoun Gharbi, and Rachid Alami. Towards combining HTN planning and geometric task planning. *arXiv preprint arXiv:1307.1482*, 2013.
- [Dogar and Srinivasa, 2011] Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. In *Robotics: Science and Systems*, volume 1, 2011.
- [Dornhege *et al.*, 2010] Christian Dornhege, Patrick Eyerich, Thomas Keller, Michael Brenner, and Bernhard Nebel. Integrating task and motion planning using semantic attachments. In *Proceedings of the First AAAI Conference on Bridging the Gap Between Task and Motion Planning*, pages 10–17. AAAI Press, 2010.
- [Erdem *et al.*, 2011] Esra Erdem, Kadir Haspalamutgil, Can Palaz, Volkan Patoglu, and Tansel Uras. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *International Conference on Robotics and Automation*. IEEE, 2011.
- [Erdogan and Stilman, 2013] Can Erdogan and Mike Stilman. Planning in constraint space: Automated design of functional structures. In *International Conference on Robotics and Automation (ICRA)*, pages 1807–1812. IEEE, 2013.
- [Ferrer-Mestres *et al.*, 2017] Jonathan Ferrer-Mestres, Guillem Francès, and Hector Geffner. Combined task and motion planning as classical AI planning. *arXiv preprint arXiv:1706.06927*, 2017.
- [Garrett *et al.*, 2015] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. FFROB: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI*, pages 179–195. Springer, 2015.
- [Garrett *et al.*, 2016] Caelan Reed Garrett, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, page 0278364917739114, 2016.
- [Gravot *et al.*, 2005] Fabien Gravot, Stephane Cambon, and Rachid Alami. asymov: a planner that deals with intricate symbolic and geometric problems. In *Robotics Research. The Eleventh International Symposium*, pages 100–110. Springer, 2005.
- [Hart *et al.*, 1968] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
- [Haslum and Geffner, 2000] Patrik Haslum and Héctor Geffner. Admissible heuristics for optimal planning. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 140–149, 2000.
- [Jung, 1899] Heinrich Jung. *Über die kleinste kugel die eine räumliche figur einschliesst*. PhD thesis, Universität Marburg, 1899.
- [Kaelbling and Lozano-Pérez, 2011] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *International Conference on Robotics and Automation*, pages 1470–1477. IEEE, 2011.
- [Karaman and Frazzoli, 2011] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

- [Lagriffoul *et al.*, 2014] Fabien Lagriffoul, Dimitar Dimitrov, Julien Bidot, Alessandro Saffiotti, and Lars Karlsson. Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research*, 2014.
- [Lozano-Pérez and Kaelbling, 2014] Tomás Lozano-Pérez and Leslie Pack Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *International Conference on Intelligent Robots and Systems*, pages 3684–3691. IEEE, 2014.
- [Lozano-Pérez *et al.*, 1987] Tomás Lozano-Pérez, Joseph Jones, Emmanuel Mazer, Patrick O’Donnell, W. Eric L. Grimson, Pierre Tournassoud, and Alain Lanusse. Handey: A robot system that recognizes, plans, and manipulates. In *International Conference on Robotics and Automation*, volume 4, pages 843–849. IEEE, 1987.
- [Marthi *et al.*, 2008] Bhaskara Marthi, Stuart Russell, and Jason Wolfe. Angelic hierarchical planning: Optimal and online algorithms. In *International Conference on Automated Planning and Scheduling*, pages 222–231, 2008.
- [Nieuwenhuisen *et al.*, 2008] Dennis Nieuwenhuisen, A Frank van der Stappen, and Mark H Overmars. An effective framework for path planning amidst movable obstacles. In *Algorithmic Foundation of Robotics VII*, pages 87–102. Springer, 2008.
- [Nilsson, 1984] Nils J Nilsson. Shakey the robot. Technical report, DTIC Document, 1984.
- [Pohl, 1970] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4):193–204, 1970.
- [Siméon *et al.*, 2004] Thierry Siméon, Jean-Paul Laumond, Juan Cortés, and Anis Sahbani. Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):729–746, 2004.
- [Srivastava *et al.*, 2013] Siddharth Srivastava, Lorenzo Riano, Stuart Russell, and Pieter Abbeel. Using classical planners for tasks with continuous operators in robotics. In *International Conference on Automated Planning and Scheduling*, pages 27–35, 2013.
- [Stilman and Kuffner, 2008] Mike Stilman and James Kuffner. Planning among movable obstacles with artificial constraints. *The International Journal of Robotics Research*, 27(11-12):1295–1307, 2008.
- [Toussaint, 2015] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence*, 2015.
- [Vega-Brown and Roy, 2016] William Vega-Brown and Nicholas Roy. Asymptotically optimal planning under piecewise-analytic constraints. In *Workshop on the Algorithmic Foundations of Robotics*, 2016.
- [Vega-Brown and Roy, 2018] William Vega-Brown and Nicholas Roy. Admissible abstractions for near-optimal task and motion planning. In *International Joint Conference on Artificial Intelligence*, 2018.
- [Wolfe *et al.*, 2010] Jason Wolfe, Bhaskara Marthi, and Stuart Russell. Combined task and motion planning for mobile manipulation. In *International Conference on Automated Planning and Scheduling*, pages 254–258, 2010.

A Proofs

Proposition 1. *Define operator bounds*

$$V_L[\mathbf{a}](\mathbf{s}, \mathbf{s}') = \inf\{\inf\{V[\mathbf{p}](s, s') : s' \in \mathbf{s}'\} : s \in \mathbf{s}\} \quad (24)$$

$$V_U[\mathbf{a}](\mathbf{s}, \mathbf{s}') = \sup\{\inf\{V[\mathbf{p}](s, s') : s' \in \mathbf{s}'\} : s \in \mathbf{s}\} \quad (25)$$

$$\hat{V}_L[\mathbf{a}](\mathbf{s}, \mathbf{s}') = \inf\{l : (\mathbf{s}_0, \mathbf{s}_1, l, u) \in \hat{V}[\mathbf{a}], \\ \mathbf{s} \cap \mathbf{s}_0 \neq \emptyset, \mathbf{s}' \cap \mathbf{s}_1 \neq \emptyset\} \quad (26)$$

$$\hat{V}_U[\mathbf{a}](\mathbf{s}, \mathbf{s}') = \inf\{u : (\mathbf{s}_0, \mathbf{s}_1, l, u) \in \hat{V}[\mathbf{a}], \\ \mathbf{s} \subseteq \mathbf{s}_0, \mathbf{s}' \subseteq \mathbf{s}_1\}. \quad (27)$$

If $\hat{V}[\mathbf{a}]$ is admissible, then $\hat{V}_L[\mathbf{a}](\mathbf{s}, \mathbf{s}') \leq V_L[\mathbf{a}](\mathbf{s}, \mathbf{s}')$ and $V_U[\mathbf{a}](\mathbf{s}, \mathbf{s}') \leq \hat{V}_U[\mathbf{a}](\mathbf{s}, \mathbf{s}')$ for all abstract state pairs \mathbf{s}, \mathbf{s}' .

Proof.

$$\forall x \in \mathbf{s}, x' \in \mathbf{s}', \\ \exists (\mathbf{s}_0, \mathbf{s}_1, l, u) \in \hat{V}[\mathbf{a}] : l \leq V[\mathbf{a}](x, x'), x \in \mathbf{s}, x' \in \mathbf{s}' \quad (28)$$

$$x \in \mathbf{s} \wedge x' \in \mathbf{s}_0 \implies \mathbf{s} \cap \mathbf{s}_0 \neq \emptyset \quad (29)$$

$$x' \in \mathbf{s}' \wedge x' \in \mathbf{s}_1 \implies \mathbf{s}' \cap \mathbf{s}_1 \neq \emptyset \quad (30)$$

$$\hat{V}_L[\mathbf{a}](\{x\}, \{x'\}) < V[\mathbf{a}](x, x') \quad (31)$$

$$\therefore \hat{V}_L[\mathbf{a}](\mathbf{s}, \mathbf{s}') \leq V_L[\mathbf{a}](\mathbf{s}, \mathbf{s}') \quad (32)$$

$$\forall x \in \mathbf{s}, x' \in \mathbf{s}', \\ \forall (\mathbf{s}_0, \mathbf{s}_1, l, u) \in \hat{V}[\mathbf{a}] : x \in \mathbf{s}, x' \in \mathbf{s}' : u \geq V[\mathbf{a}](x, x') \quad (33)$$

$$\mathbf{s} \subseteq \mathbf{s}_0 \wedge x \in \mathbf{s} \implies x \in \mathbf{s}_0 \quad (34)$$

$$\mathbf{s}' \subseteq \mathbf{s}_1 \wedge x' \in \mathbf{s}' \implies x' \in \mathbf{s}_1 \quad (35)$$

$$\hat{V}_U[\mathbf{a}](\{x\}, \{x'\}) \geq V[\mathbf{a}](x, x') \quad (36)$$

$$\therefore \hat{V}_U[\mathbf{a}](\mathbf{s}, \mathbf{s}') \geq V_U[\mathbf{a}](\mathbf{s}, \mathbf{s}') \quad (37)$$

Then $\hat{V}_L[\mathbf{a}](\mathbf{s}, \mathbf{s}') \leq V_L[\mathbf{a}](\mathbf{s}, \mathbf{s}')$ and $V_U[\mathbf{a}](\mathbf{s}, \mathbf{s}') \leq \hat{V}_U[\mathbf{a}](\mathbf{s}, \mathbf{s}')$ for all abstract state pairs \mathbf{s}, \mathbf{s}' . \square

Proposition 2. *Suppose $\hat{V}[\mathbf{a}]$ and $\hat{V}[\mathbf{a}']$ are admissible. Then the valuation bound*

$$\hat{V}[\mathbf{a} \circ \mathbf{a}'] = \{(\mathbf{s}, \mathbf{s}', l + l', u + u') : (\mathbf{s}, \mathbf{s}', l, u) \in \hat{V}[\mathbf{a}], \\ (\mathbf{s}'', \mathbf{s}''', l', u') \in \hat{V}[\mathbf{a}'], \mathbf{s}' \subseteq \mathbf{s}''\} \cup \\ \{(\mathbf{s}, \mathbf{s}', l + l', u) : (\mathbf{s}, \mathbf{s}', l, u') \in \hat{V}[\mathbf{a}], \\ (\mathbf{s}'', \mathbf{s}''', l', \infty) \in \hat{V}[\mathbf{a}'], \mathbf{s}' \cap \mathbf{s}'' \neq \emptyset\} \quad (38)$$

is admissible.

Proof. Let $p \circ p'$ be a plan in $\mathbf{a} \circ \mathbf{a}'$, where $p \in \mathbf{a}$ and $p' \in \mathbf{a}'$. Then because $\hat{V}[\mathbf{a}]$ and $\hat{V}[\mathbf{a}']$ are admissible,

$$\exists (\mathbf{s}, \mathbf{s}', l, u) \in \hat{V}[\mathbf{a}] : p(0) \in \mathbf{s}, p(1) \in \mathbf{s}', l \leq \mathcal{C}[p] \quad (39)$$

$$\exists (\mathbf{s}'', \mathbf{s}''', l', u') \in \hat{V}[\mathbf{a}'] : p'(0) \in \mathbf{s}'', p'(1) \in \mathbf{s}''', l' \leq \mathcal{C}[p'] \quad (40)$$

Because $p \circ p'$ is continuous, $\mathbf{s}' \cap \mathbf{s}''$ intersect.

$$p(1) = p'(0) \implies \mathbf{s}' \cap \mathbf{s}'' \neq \emptyset \quad (41)$$

Because the cost function is additive, $l + l'$ is a lower bound on the cost of $p \circ p'$.

$$\mathcal{C}[p \circ p'] = \mathcal{C}[p] + \mathcal{C}[p'] \geq l + l' \quad (42)$$

By construction, it follows that $(\mathbf{s}, \mathbf{s}', l + l', u + u') \in \hat{V}[\mathbf{a} \circ \mathbf{a}']$, and therefore

$$\therefore \forall p \circ p' \in \mathbf{a} \circ \mathbf{a}', \exists (\mathbf{s}, \mathbf{s}', l, u) \in \hat{V}[\mathbf{a} \circ \mathbf{a}'] : \\ p(0) \in \mathbf{s}, p(1) \in \mathbf{s}', l \leq \mathcal{C}[p \circ p']. \quad (43)$$

Thus $\hat{V}[\mathbf{a} \circ \mathbf{a}']$ is an admissible lower bound.

Let x, x, x'' be primitive states.

$$V[\mathbf{a} \circ \mathbf{a}'](x, x'') \\ \leq V[\mathbf{a} \circ \mathbf{a}'](x, x') + V[\mathbf{a} \circ \mathbf{a}'](x', x'') \quad (44)$$

$$\leq V[\mathbf{a}](x, x') + V[\mathbf{a}'](x', x'') \quad (45)$$

$$\leq V_U[\mathbf{a}](\mathbf{s}, \mathbf{s}') + V_U[\mathbf{a}'](\mathbf{s}'', \mathbf{s}''') \forall \mathbf{s}' \subseteq \mathbf{s}'' \quad (46)$$

$$\leq \hat{V}_U[\mathbf{a}](\mathbf{s}, \mathbf{s}') + \hat{V}_U[\mathbf{a}'](\mathbf{s}'', \mathbf{s}''') \quad (47)$$

$$\forall \mathbf{s}' \subseteq \mathbf{s}'', x \in \mathbf{s}, x' \in \mathbf{s}', x'' \in \mathbf{s}''' \\ \leq u + u' \quad (48)$$

$$\forall (\mathbf{s}, \mathbf{s}', l, u) \in \hat{V}[\mathbf{a}] : x \in \mathbf{s}, x' \in \mathbf{s}', \\ \forall (\mathbf{s}'', \mathbf{s}''', l', u') \in \hat{V}[\mathbf{a}'] : x' \in \mathbf{s}'', x'' \in \mathbf{s}''' \quad (49)$$

$$\leq \inf\{u : (\mathbf{s}, \mathbf{s}'', l, u) \in \hat{V}[\mathbf{a} \circ \mathbf{a}']\} \\ = \hat{V}_U[\mathbf{a} \circ \mathbf{a}'](x, x'') \quad (50)$$

Thus $\hat{V}[\mathbf{a} \circ \mathbf{a}']$ is an admissible upper bound. \square

Proposition 3. *If $\hat{V}[\mathbf{p}]$ and $\hat{V}[\mathbf{p}']$ are admissible, then $\hat{V}[\mathbf{p} \cup \mathbf{p}'] = \hat{V}[\mathbf{p}] \cup \hat{V}[\mathbf{p}']$ is admissible.*

Proof. Observe that

$$V[\mathbf{p} \cup \mathbf{p}'](s, s') = \min(V[\mathbf{p}](s, s'), V[\mathbf{p}'](s, s')). \quad (51)$$

Then

$$\hat{V}_U[\mathbf{p} \cup \mathbf{p}'](\{s\}, \{s'\}) \geq \hat{V}_U[\mathbf{p}](\{s\}, \{s'\}) \quad (52)$$

$$\geq V[\mathbf{p}](s, s') \quad (53)$$

$$\geq V[\mathbf{p} \cup \mathbf{p}'](s, s') \quad (54)$$

so $\hat{V}[\mathbf{p} \cup \mathbf{p}']$ is a valid upper bound. Note this argument applies to $\hat{V}_U[\mathbf{p}](\{s\}, \{s'\})$ as well.

Then

$$\hat{V}_L[\mathbf{p} \cup \mathbf{p}'](\{s\}, \{s'\}) \\ = \min(\hat{V}_L[\mathbf{p}](\{s\}, \{s'\}), \hat{V}_L[\mathbf{p}'](\{s\}, \{s'\})) \quad (55)$$

$$\leq \min(V[\mathbf{p}](s, s'), V[\mathbf{p}'](s, s')) \quad (56)$$

$$= V[\mathbf{p} \cup \mathbf{p}'](s, s') \quad (57)$$

so $\hat{V}[\mathbf{p} \cup \mathbf{p}']$ is a valid lower bound. \square

Proposition 4. Let $S = \{R_i\}$ be a decomposition of the state space \mathcal{X} into N regions, such that $\cup_{i \in [N]} R_i = \mathcal{X}$. The refinement relation

$$\begin{aligned} \bar{\mathcal{R}} = & \bigcup_{ij} \{(\text{ACT}, \mathbf{a}_{ij} \circ \text{ACT}), (\text{ACT}, \mathbf{a}_{ij})\} \cup \\ & \{(\mathbf{a}_{ij}, a \circ \mathbf{a}_{ij}) : a(t) \in R_i \forall t\} \cup \\ & \{(\mathbf{a}_{ij}, a) : a(t) \in R_i \forall t, a(1) \in \text{cl}(R_j)\} \end{aligned} \quad (58)$$

is admissible.

Proof. Choose any $p \in \text{ACT}$. Because $\cup_{i \in [N]} R_i = \mathcal{X}$, there exists some R_i such that $p(0) \in R_i$. If $p(t) \in R_i \forall t$, then $p \in \text{aii}$. Otherwise, there is some $t' \in [0, 1]$ such that $p(t') \in \partial R_i$ and $p(t) \in R_i \forall t < t'$. Because $\cup_{j \in [N]} R_j = \mathcal{X}$, there exists some R_j such that $p(t') \in \text{cl}(R_j)$. Then we can partition p into $p_1 \circ p_2$, such that $p_1(t) \in R_i \forall t$ and $p_2(0) \in \text{cl}(R_j)$. Then $p_1 \in \mathbf{a}_{ij}$ and $p_2 \in \text{ACT}$, so $p \in \mathbf{a}_{ij} \circ \text{ACT}$.

Choose any $p \in \mathbf{a}_{ij}$. Because $p \in \mathcal{A}_0^*$, either we can partition p into $a \circ p'$, or $p = a$. If $p = a$, then $(\mathbf{a}_{ij}, p) \in \bar{\mathcal{R}}$. Otherwise, $p' \in \mathbf{a}_{ij}$, so $(\mathbf{a}_{ij}, a \circ p') = (\mathbf{a}_{ij}, p) \in \bar{\mathcal{R}}$. \square

Theorem 2. If the abstraction \mathcal{A} is admissible and a feasible plan exists, then algorithm 1 returns an optimal sequence of primitive operators in finite time, provided the lower bound on the cost of every operator is greater than zero.

Proof. Define the condition $I_1(p)$ as

$$I_1(p) \iff \exists \mathbf{p} \in Q : p \in \mathbf{p} \vee \hat{V}_U[\mathbf{p}] \prec V[p]. \quad (59)$$

We will first show that if $I_1(p)$ holds before line 12, it holds after line 17. If $I_1(p)$ holds initially, there is a plan $\mathbf{p} \in Q$ such that $p \in \mathbf{p}$. If $\mathbf{p} \neq \arg \min\{\hat{V}_L[\mathbf{p}] : \mathbf{p} \in Q\}$, then $\mathbf{p} \in Q$ after line 17. Otherwise, because the abstraction is admissible, there is a plan \mathbf{p}' such that $p \in \mathbf{p}'$ and $(\mathbf{p}, \mathbf{p}') \in \mathcal{R}$. If $\mathbf{p} \prec \mathbf{p}'$, there is a plan $\mathbf{p}'' \in Q$ such that $\hat{V}_U[\mathbf{p}''] \prec V[p]$. Otherwise, \mathbf{p}' will be added to the queue before line 17. Therefore if $I_1(p)$ holds before line 12, it holds after line 17.

Initially, $Q = \{\text{ACT}\}$. By construction, the operator ACT contains every valid sequence of primitive operators. Therefore the proposition $\forall p : I_1(p)$ holds after line 6. If $I_1(p)$ holds at line 8, it holds before line 12; therefore $\forall p : I_1(p)$ holds every time line 9 is reached.

If $\forall p : I_1(p)$ and $\hat{V}_U[\mathbf{p}^*] \prec \hat{V}_L[\mathbf{p}] \forall \mathbf{p} \in Q$, then $\forall p \exists p^* \in \mathbf{p}^* : V[p^*] \preceq V[p]$ and thus there exists an optimal primitive plan $p^* \in \mathbf{p}^*$. Then any time line 10 is reached, \mathbf{p}^* is an optimal primitive plan. In other words, if algorithm 1 returns a plan, that plan is an optimal plan.

If there is a feasible plan, it has finite cost, and therefore the cost c^* of an optimal plan is finite. Let $N \in \mathbb{N}$ be the number of plans with lower bound less than c^* ; because the abstraction is finite and every operator has a positive lower bound, N is finite. Each plan is added to the queue at most once, and is therefore removed at most once. Then after N iterations, the lower bound on any plan in Q is greater than c^* , and algorithm 1 will return a plan. \square

Theorem 3. If the abstraction \mathcal{A} is admissible and a feasible plan exists, then the acyclic angelic A^* algorithm returns a sequence of primitive operators with cost no greater than $c^*(\{x_s\}, X_g)$ in finite time.

Proof. Define the parent of a plan as

$$\text{PARENT}(\mathbf{p}') = \mathbf{p} \iff (\mathbf{p}, \mathbf{p}') \in \mathcal{R}. \quad (60)$$

Define the ancestors of a plan recursively.

$$\text{ANCESTOR}(\mathbf{p}, 0) = \mathbf{p} \quad (61)$$

$$\text{ANCESTOR}(\mathbf{p}, n) =$$

$$\text{BASE}(\text{PARENT}(\text{ANCESTOR}(\mathbf{p}, n-1))) \quad (62)$$

$$\text{ANCESTORS}(\mathbf{p}) = \{\text{ANCESTOR}(\mathbf{p}, n) : n \in \mathbb{Z}\} \quad (63)$$

Define the condition $I_2(Q, D, p)$ as true if one of

- $\exists \mathbf{p} \in Q : p \in \mathbf{p}$, or
- $\exists \mathbf{p} \in Q : \exists \mathbf{p}' \in \text{DEF}(\mathbf{p}, D) : \exists v \in \mathbf{p} : p \in v \circ \mathbf{p}'$, or
- $\exists \mathbf{p} \in Q : \hat{V}_U[\mathbf{p}] \prec V[p]$

is true.

First, we show that if I_2 holds before removing a plan from the queue, it holds after the successors to that plan have been added to the queue. If $I_2(Q, D, p)$ holds before removing a plan \mathbf{p} , then either

1. $\exists \mathbf{p}' \neq \mathbf{p} \in Q : p \in \mathbf{p}'$
2. $\exists \mathbf{p}' \neq \mathbf{p} \in Q : \mathbf{p}' \prec \mathbf{p}$
3. $\exists \mathbf{p}' \in Q : \exists \mathbf{p}'' \in \text{ANC}(\mathbf{p}', D) : \exists \mathbf{p}''' \in \text{DEF}(\mathbf{p}'') : p \in \text{BASE}(\mathbf{p}''') \circ \mathbf{p}'''$, or
4. $p \in \mathbf{p}$, or
5. $\exists \mathbf{p}_a \in \text{ANC}(\mathbf{p}, D) : \exists \mathbf{p}'' \in \text{DEF}(\mathbf{p}') : p \in \mathbf{p}' = \text{BASE}(\mathbf{p}') \circ \mathbf{p}''$

In cases 1, 2, or 3, $\mathbf{p}' \in Q$ after expansion and the invariant holds. In case 4, because the abstraction is admissible, there is a plan \mathbf{p}' such that $p \in \mathbf{p}'$ and $(\mathbf{p}, \mathbf{p}') \in \mathcal{R}$; this plan will be considered for expansion. In case 5, the plan $\mathbf{p}' = \text{BASE}(\mathbf{p}) \circ \mathbf{p}''$ will be considered for expansion. If \mathbf{p}' is acyclic, then either $\mathbf{p}' \in Q$ after expansion and the invariant holds, or \mathbf{p}' is dominated by some plan already in Q and the invariant holds. If \mathbf{p}' is cyclic, its extension will be added to $\text{DEF}(\text{PAR}(\mathbf{p}))$. Since there is at least one refinement of \mathbf{p} that will be added to the queue, there will be a plan on the queue with \mathbf{p} as an ancestor, and thus the invariant will hold.

Since I_2 holds initially, and holds after each expansion, it holds until the algorithm terminates. If $\forall p : I_2(Q, D, p)$ and $\hat{V}_U[\mathbf{p}^*] \prec \hat{V}_L[\mathbf{p}] \forall \mathbf{p} \in Q$, then $\forall p \exists p^* \in \mathbf{p}^* : V[p^*] \preceq V[p]$ and thus there exists an optimal primitive plan $p^* \in \mathbf{p}^*$. Then any time line 10 is reached, \mathbf{p}^* is an optimal primitive plan. In other words, if algorithm 2 returns a plan, that plan is an optimal plan.

If there is a feasible plan, it has finite cost, and therefore the cost c^* of an optimal plan is finite. Let $N \in \mathbb{N}$ be the number of acyclic plans with lower bound less than c^* ; because the abstraction is finite and every acyclic plan has a cost strictly greater than its predecessor, N is finite. Only acyclic plans are added to the queue; each plan is added at most once, and

is therefore removed at most once. Then after N iterations, the lower bound on any plan in Q is greater than c^* , and algorithm 1 will return a plan. \square

Theorem 4. *If the abstraction \mathcal{A} is admissible and a feasible plan exists, then algorithm 4 returns a sequence of primitive operators with cost no greater than $w \cdot c^*(\{x_s\}, X_g)$ in finite time.*

Proof. Two invariants hold while the algorithm is running. First, invariant I_2 still holds; second, we know the minimal lower bound of any plan on the queue is less than w times the lowest priority on the queue.

$$\forall p : I_2(Q, D, p) \quad (64)$$

$$\text{KEY}(\mathbf{p}, w) \leq w \hat{V}_L[\mathbf{p}] \forall \mathbf{p} \in Q \quad (65)$$

When the algorithm terminates,

$$\hat{V}_U[\mathbf{p}^*](x_s, X_g) \prec \text{KEY}(\mathbf{p}, w) \forall \mathbf{p} \in Q. \quad (66)$$

Consequently,

$$\hat{V}_U[\mathbf{p}^*](x_s, X_g) \prec w \hat{V}_L[\mathbf{p}](x_s, X_g). \quad (67)$$

Since the algorithm must terminate by the same argument as for theorem 3, algorithm 4 will return a w -optimal solution in finite time. \square