Admissible abstractions for near-optimal task and motion planning

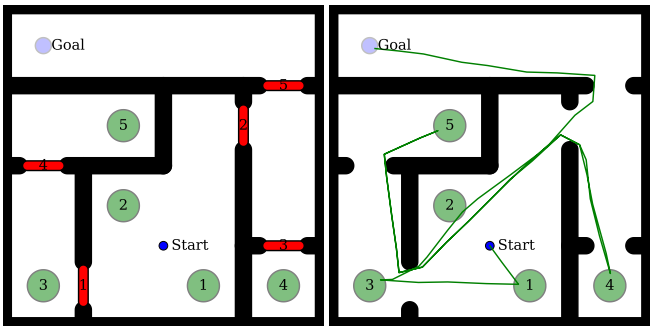
William Vega-Brown and Nicholas Roy
Massachusetts Institute of Technology
{wrvb, nickroy}@mit.edu

***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 we can use these abstractions to simultaneously search for semantic plans and primitive motion plans, 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.**

I. 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 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 semantically-meaningful high level plan 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, Cambon, and Alami (2005) describe an integrated approach that 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, Lozano-Pérez, and Kaelbling 2015; Cambon, Alami, and Gravot 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, Marthi, and Russell 2010) are generally limited to guarantees of hierarchical optimality.



(a) The door puzzle problem (b) An example solution

Fig. 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.

Angelic semantics (Marthi, Russell, and Wolfe 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 continuous 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.

II. 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 $p : [0, 1] \to \mathcal{X}$. We define a concatenation operator $\circ$ for paths.

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

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

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

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

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

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

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

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 $\mathcal{P}(\mathcal{X}, \mathcal{X})$ 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

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

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.

III. Angelic semantics

We define a *semantic operator* $\mathbf{a} \subset \mathcal{P}(\mathcal{X}, \mathcal{X})$ as a set of primitive plans. Because the space of plans is infinite, we define operators implicitly, in terms of 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.

The concatenation of two operators $a_1 \circ a_2$ is a semantic plan containing all possible concatenations of primitive plans in the operators.

$$\mathbf{a}_1 \circ \mathbf{a}_2 = \{p_1 \circ p_2 : p_1 \in a_1, p_2 \in a_2, p_1(1) = p_2(0)\} \tag{6}$$

Concatenations of a well-chosen small set of semantic operators can express very complicated plans in a compact way.

In order to use these semantic operators for planning, we need a way to compare semantic plans. 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 $V[\mathbf{a}] : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ that takes a pair of states and gives the lowest cost path between the pair.

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

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

Valuations allow us to compare semantic plans without reference to the primitive plans they contain. Given two semantic plans $\mathbf{p}$ and $\mathbf{p}'$, if we can prove that for any pair of states s, s', either $V[\mathbf{p}](s, s') < V[\mathbf{p}'](s, s')$ or $V'(s, s') = \infty$, 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 $\mathbf{p} \prec \mathbf{p}'$. Similarly, if either $V[\mathbf{p}](s, s') \leq V[\mathbf{p}'](s, s')$ or $V'(s, s') = \infty$, then we say that $\mathbf{p}$ *weakly dominates* $\mathbf{p}'$ and we write $\mathbf{p} \preceq \mathbf{p}'$.

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. The computational advantage comes from reasoning about bounds on the valuation of an semantic operator. By representing these bounds symbolically, we are able to reason without reference to the underlying states or plans.

A symbolic valuation bound is a set of tuples $\{(\mathbf{s}, \mathbf{s}', v)\}$. A bound $L[\mathbf{p}]$ is optimistic, written $L[\mathbf{p}] \prec V[\mathbf{p}]$, if

$$l \leq \inf\{\inf\{V[\mathbf{p}](s, s') : s \in \mathbf{s}\} : s' \in \mathbf{s}'\} \quad \exists (\mathbf{s}, \mathbf{s}', l) \in L[\mathbf{p}]. \tag{8}$$

A bound is pessimistic, written $V[\mathbf{p}] \prec U[\mathbf{p}]$, if

$$u \geq \sup\{\inf\{V[\mathbf{p}](s, s') : s \in \mathbf{s}\} : s' \in \mathbf{s}'\} \quad \forall (\mathbf{s}, \mathbf{s}', u) \in U[\mathbf{p}]. \tag{9}$$

By construction, if $U[\mathbf{p}]$ and $U[\mathbf{p}']$ are bounds on the valuations of $\mathbf{p}$ and $\mathbf{p}'$, then $U[\mathbf{p}] \cup U[\mathbf{p}']$ is a bound on the valuation of $\mathbf{p} \cup \mathbf{p}'$.

As we will see in sections IV-A and IV-B, 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.

By working with symbolic bounds, we can efficiently compute bounds on the cost of plans consisting of sequences of semantic operators, without reference to a dense discretization of the underlying space of plans. For example, if we know bounds $L[\mathbf{p}]$ and $U[\mathbf{p}]$ on the valuation of a plan $\mathbf{p}$, we can compute bounds on the valuation of the plan $\mathbf{p} \circ \mathbf{a}$.

$$L[\mathbf{p} \circ \mathbf{a}] = \{(l + l', \mathbf{s}, \mathbf{s}'') : (l, \mathbf{s}_0, \mathbf{s}_1) \in L[\mathbf{p}], (l', \mathbf{s}'_0, \mathbf{s}'_1) \in L[\mathbf{a}], \mathbf{s}'_0 \cap \mathbf{s}_1 \neq \varnothing\} \quad (10)$$

$$U[\mathbf{p} \circ \mathbf{a}] = \{(u + u', \mathbf{s}, \mathbf{s}'') : (u, \mathbf{s}_0, \mathbf{s}_1) \in U[\mathbf{p}], (u', \mathbf{s}'_0, \mathbf{s}'_1) \in U[\mathbf{a}], \mathbf{s}_1 \subseteq \mathbf{s}'_0\} \quad (11)$$

IV. Admissible abstractions

We can use angelic semantics to specify an abstraction that will enable efficient planning. Suppose that $\mathbf{p}, \mathbf{p}'$ are semantic 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 tighter. If our bounds allow us to conclude that $U[\mathbf{p}'] \prec L[\mathbf{p}]$, 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.

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 several simple operations. First, we define operations $\textsc{Base} : \mathcal{A}^* \to \mathcal{A}^*$, $\textsc{Head} : \mathcal{A}^* \to \mathcal{A}$, and $\textsc{Ext} : \mathcal{A}^* \to \mathcal{A}^*$ that split a plan into three segments so that $\mathbf{p} = \textsc{Base}(\mathbf{p}) \circ \textsc{Head}(\mathbf{p}) \circ \textsc{Ext}(\mathbf{p})$. Second, we define a relation $\bar{\mathcal{R}} \subset \mathcal{A} \times \mathcal{A}^*$. Then $(\mathbf{p}, \mathbf{p}') \in \mathcal{R}$ if and only if $\mathbf{p}' = \textsc{Base}(\mathbf{p}) \circ \mathbf{p}'' \circ \textsc{Ext}(\mathbf{p})$ and $(\textsc{Head}(\mathbf{p}), \mathbf{p}'') \in \bar{\mathcal{R}}$.

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 $(\mathcal{S}, \mathcal{A}, \bar{\mathcal{R}}, L, U)$, 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
- L and U are lower and upper bounds, respectively, on the valuation of each operator.

The valuation bounds encode both the cost and the dynamics of our problem domain. The refinement relation structures the space of semantic plans.

Angelic planning algorithms accept an abstraction as an argument in much the same way that the A* search algorithm (Hart, Nilsson, and Raphael 1968) accepts a heuristic. This raises an important question: under what circumstances will an abstraction $(\mathcal{S}, \mathcal{A}, \bar{\mathcal{R}}, L, U)$ 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.

In fact, two properties suffice.

1) For each semantic operator $\mathbf{a} \in \mathcal{A}$, for each primitive plan in $\mathbf{a}$, there is a refinement $\mathbf{p}$ of $\mathbf{a}$ such that $p \in \mathbf{p}$.
$$\forall \mathbf{a} \in \mathcal{A}, \forall p \in \mathbf{a}, \exists (\mathbf{a}, \mathbf{p}) \in \bar{\mathcal{R}} : p \in \mathbf{p} \quad (12)$$
2) L and U are valid bounds. That is, $L[\mathbf{p}] \preceq V[\mathbf{p}] \preceq U[\mathbf{p}]$ for each semantic plan $\mathbf{p} \in \mathcal{A}^*$.

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 semantic plans $\mathbf{p}, \mathbf{p}' \in P$, where P is a collection of abstract plans, and $U[\mathbf{p}] \prec L[\mathbf{p}']$, 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 $P_0 = \{\mathrm{Act}\}$, 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.

Not every admissible abstraction is useful for planning. A good abstraction must be concise: it must be informative enough to enable us to explore the space of plans quickly. In general, designing useful abstractions for a given domain is a complex exercise. In the remainder of this section, we will provide concrete examples of admissible abstractions for a pair of simple continuous planning problems.

A. 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 3, 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, and we know which regions overlap. Then any plan can be described by the sequence of regions it moves through.We can use this to define an abstraction. Let $\mathcal{S} = \{R_i\}$, where $\cup_i R_i = \mathcal{X}$,and let $\mathcal{A} = \mathcal{A}_0 \cup \{\mathbf{a}_{ij} : R_i \cap R_j \neq \varnothing\} \cup \{\mathrm{Act}\}$, where $p \in \mathbf{a}_{ij}$ if $p(t) \in R_i \forall t \in [0, 1) \wedge p(1) \in R_j$. Define the refinement relation as follows.

$$\begin{aligned} \bar{\mathcal{R}} = \bigcup_{ij} & \{(\mathrm{Act}, \mathbf{a}_{ij} \circ \mathrm{Act})\} \cup \\ & \{(\mathrm{Act}, \mathbf{a}_{ij}) : R_j \cap \mathcal{X}_g \neq \varnothing\} \cup \\ & \{(\mathbf{a}_{ij}, a \circ \mathbf{a}) : a(t) \in R_i \forall s\} \cup \\ & \{(\mathbf{a}_{ij}, a) : a(t) \in R_i \forall t, a(1) \in R_j\} \end{aligned} \quad (13)$$

In practice, we will not iterate over all possible refinements, but will instead use spatial indices like k-D trees and R-trees to find the operators that are valid from a particular state. It is straightforward to show this satisfies the completeness property.

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, 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 semantic 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, semantic operators can have no upper bound, which can lead the search to be inefficient. If we additionally require the regions to be convex, then we can use the Hausdorff distance between sets as an *upper* bound. Executing the action $\mathbf{a}_{ij}$ 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

$$d_H(X, Y) = \max(\sup_{x \in X} \inf_{y \in Y} \|x - y\|, \sup_{y \in Y} \inf_{x \in X} \|x - y\|). \quad (14)$$

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

$$\inf_{p \in \mathcal{P}_R(x, x')} \mathcal{C}[p] \leq (1 + \epsilon)\|x - x'\|. \quad (15)$$

B. 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 where N is the number of doors. 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. 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 2, 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 reduces the problem of finding a lower bound to solving a travelling salesperson problem (TSP). While it is believed that 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. This bound is not tight, and even if it were, it neglects possible interactions between the operators. However, it is a valid lower bound, and as such we can use this bound to guide the search for a complete, ordered semantic plan. Once we find a sequence of semantic operators, we can use it to guide the search for a full primitive motion plan.

V. Acyclic approximate angelic search

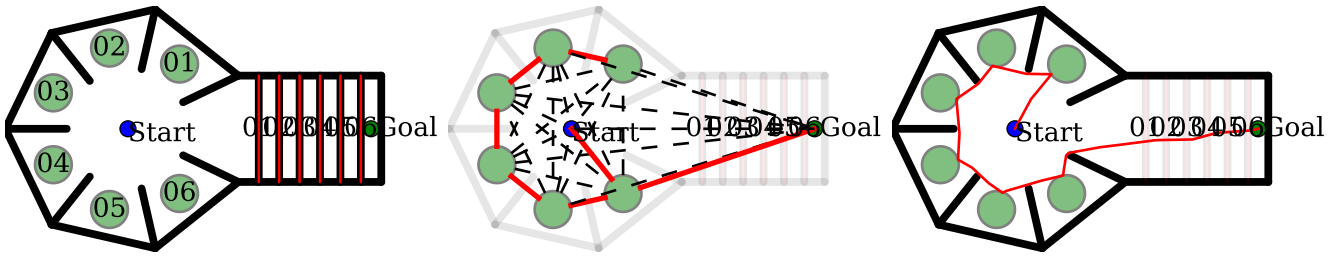
Algorithm 1 is a reformulation and extension of the angelic hierarchical A* algorithm developed by Marthi, Russell, and Wolfe (2008). Angelic A* is effectively a best-first forward search over semantic plans. It accepts an abstraction and a positive weight $w \geq 1$, and maintains a queue of plans to expand. The algorithm repeatedly removes from the queue the plan with the lowest lower bound on the cost to reach the goal (lines 8-12). 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.

The primary data structure maintained by our algorithm is a tree. Each node in the tree represents a plan as the concatenation of a predecessor plan $\mathbf{p}_-$ and an operator $\mathbf{a}$. Nodes store the following information:

- an semantic operator $\mathbf{a}$,
- the predecessor plan $\mathbf{p}_-$,
- the parent plan, from which this plan was refined,
- the base plan $\textsc{Base}(\mathbf{p})$, which is used in choosing refinements, and
- upper and lower bounds on the valuation of the plan represented by the node.

The root of the tree is a node that contains no operator, predecessor, or base, and whose upper and lower valuation bounds are zero at the start state and infinite elsewhere.

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. We provide two key modifications. First, instead of expanding the plan with the smallest lower bound, we order the queue with a priority that is no greater than w times the lower bound on the cost a plan. This allows us to exchange optimality for approximate optimality and an accelerated search, in much the same fashion as weighted A* (Pohl 1970). Inflating the lower bound may accelerate search by providing a more accurate estimate of the true value of a semantic plan. This is especially true in domains where formulating a semantic plan is difficult: inflating our lower bounds can allow us to focus our search on a single, reasonable semantic plan, rather than considering every possible semantic plan.



(a) Problem (b) Lower bound (c) Solution

Fig. 2: 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).

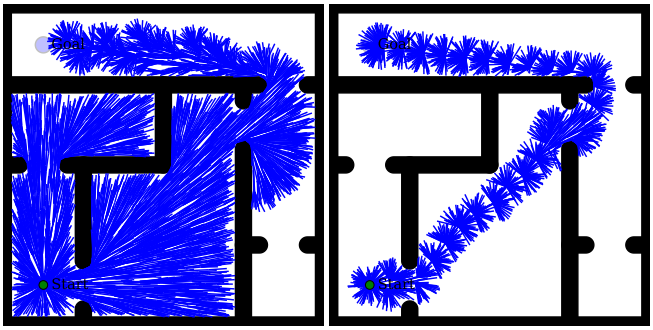
Second, the original formulation of angelic A* required strictly positive lower bounds on the cost of any operator. In discrete problems, this is reasonable restriction, but it presents challenges in continuous problems. For example, suppose we have a plan consisting of two operators $\mathbf{a}_{ij} \circ \mathbf{a}_{i'j'}$ from our navigation abstraction. If the destination regions intersect—if $R_j \cap R'_j \neq \varnothing$—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. 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. Unfortunately, we cannot simply discard any semantic plan with a cycle: the optimal plan may leave and return to an abstract state if the state is non-convex. Typically, this indicates a poor choice of abstraction, but 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 $L[\mathbf{p}_0] \preceq L[\mathbf{p}]$ (algorithm 1, lines 58-66). If when computing the successors of a plan $\mathbf{p}$, we find the extension $\mathbf{p}_{\text{ext}}$ would create a deferred plan when propagated on top of $\textsc{Base}(\mathrm{p})$, we do not add $\mathbf{p} \circ \mathbf{p}_{\text{ext}}$ to the set of successors. Instead, we add $(\textsc{Base}(\mathbf{p}), \mathbf{p}_{\text{ext}})$ to the set of deferred plans (algorithm 1, line 30). 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 acyclic, we add it to the set of successors (line 37). This ensures that only acyclic plans will ever be added to the queue of plans without sacrificing completeness.

VI. Results

We implemented algorithm 1 and the abstractions described in sections IV-A and IV-B in the Python programming language. We then compared the performance of the planner to the original angelic A* search algorithm (Marthi, Russell, and Wolfe 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 1 are given in figure 3.



(a) A* (b) Acyclic Angelic A*

Fig. 3: The search trees constructed by A* (a) and by algorithm 1 (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.

By 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 4). 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.

Algorithm 1 Angelic Approximate A*

function SEARCH(abstraction $(\mathcal{S}, \mathcal{A}, \bar{\mathcal{R}}, \hat{V})$, weight w)
  root $= (\varnothing, \varnothing, \varnothing, \{(x_s, x_s, 0, 0)\})$
  $\mathbf{p}^* = \varnothing$
  BOUND$(\varnothing) = \hat{V}$[ACT]
  $\mathbf{p}_0 =$ PROPAGATE(root, [ACT])
  $Q = \{\mathbf{p}_0\}$
  while $|Q| > 0$ **do**
    $\mathbf{p} = \arg\min\{$KEY$(\mathbf{p}, w) : \mathbf{p} \in Q\}$
    if PRIMITIVE$(\mathbf{p}^*)$ and $\hat{V}_U[\mathbf{p}^*] \prec \hat{V}_L[\mathbf{p}]$ **then**
      return $\mathbf{p}^*$
    else
      $Q \leftarrow Q \setminus \{\mathbf{p}\}$
      $S \leftarrow$ SUCCESSORS$(\mathbf{p})$
      for $\mathbf{p}' \in S$ **do**
        if $\hat{V}_U[\mathbf{p}'] < \hat{V}_U[\mathbf{p}^*]$ **then**
          $\mathbf{p}^* \leftarrow \mathbf{p}'$
      $Q \leftarrow Q \cup \{\mathbf{p}' \in S : \neg \hat{V}_U[\mathbf{p}^*] \prec \hat{V}_L[\mathbf{p}]\}$
  return $\varnothing$
function SUCCESSORS(plan node $\mathbf{p}$)
  ▷ D is a global variable, initially set to $\varnothing$.
  POST(BASE$(\mathbf{p})) = \{\mathbf{s}' : (\mathbf{s}, \mathbf{s}', l, u) \in \hat{V}$[BASE$(\mathbf{p})]\}$
  $S = \varnothing$
  $\mathbf{a} =$ OPERATOR(HEAD$(\mathbf{p})$)
  for $\mathbf{p}' : (\mathbf{a}, \mathbf{p}') \in \bar{\mathcal{R}}, \exists \mathbf{s} \in$ POST(BASE$(\mathbf{p})) :$ HEAD$(\mathbf{p}') \cap \mathbf{s} \neq \varnothing$ **do**
    $\mathbf{p}_{\text{ref}} \leftarrow$ PROPAGATE(BASE$(\mathbf{p}), \mathbf{p}' \circ$ EXT$(\mathbf{p}))$
    if $\hat{V}_L[\mathbf{p}_{\text{ref}}](x_s, X_g) < \infty$ **then**
      if ACYCLIC$(\mathbf{p}', \varnothing)$ **then**
        $S \leftarrow S \cup \{\mathbf{p}'\}$
      else
        $D \leftarrow D \cup \{($BASE$(\mathbf{p}),$ EXT$(\mathbf{p}'))\}$
  $\mathbf{p}_a \leftarrow \mathbf{p}$
  while BASE(PARENT$(\mathbf{p}_a)) \neq \varnothing$ **do**
    $\mathbf{p}_a \leftarrow$ BASE(PARENT$(\mathbf{p}_a))$
    for $\mathbf{p}_{\text{ext}} : (\mathbf{p}_a, \mathbf{p}_{\text{ext}}) \in D$ **do**
      $\mathbf{p}' \leftarrow$ PROPAGATE(BASE$(\mathbf{p}), \mathbf{p}_{\text{ext}})$
      if ACYCLIC$(\mathbf{p}', \varnothing)$ and $\hat{V}_L[\mathbf{p}'] < \infty$ **then**
        $S \leftarrow S \cup \{\mathbf{p}'\}$
  return S
function PROPAGATE(base node $\mathbf{p}$, list $\mathbf{p}_{\text{ext}}$)
  $\mathbf{b} \leftarrow \mathbf{p}$
  while $\mathbf{p}_{\text{ext}}$ is not empty **do**
    $\mathbf{a} \leftarrow$ POP$(\mathbf{p}_{\text{ext}})$
    if $\mathbf{a}$ is more primitive than OPERATOR$(\mathbf{p})$ **then**
      $\mathbf{b} \leftarrow \mathbf{p}$
    $\mathbf{p} \leftarrow (\mathbf{a}, \mathbf{p}, \mathbf{b}, \hat{V}[\mathbf{p} \circ \mathbf{a}])$
    if $\hat{V}[\mathbf{p}] = \varnothing$ **then return**
    else if BOUND$(\mathbf{p}_{\text{ext}}) \prec \hat{V}_L[\mathbf{p}']$ **then**
      return $\varnothing$
    else
      BOUND$(\mathbf{p}_{\text{ext}}) \leftarrow$ BOUND$(\mathbf{p}_{\text{ext}}) \cup \hat{V}[\mathbf{p}]$
  return A
function KEY(node $\mathbf{p}$, weight $w \in \mathbb{R}_{\geq 1}$)
  $\mathbf{p}_- \leftarrow$ PREDECESSOR$(\mathbf{p})$
  if $\mathbf{p}_- = \varnothing$ **then**
    return 0
  else
    return $\min($KEY$(\mathbf{p}_-) + w(\hat{V}_L[\mathbf{p}] - \hat{V}_L[\mathbf{p}_-]), \hat{V}_U[\mathbf{p}])$
function ACYCLIC(plan nodes $\mathbf{p}, \mathbf{p}'$)
  if $\mathbf{p} = \varnothing$ **then**
    return true
  else if $\mathbf{p}' = \varnothing$ **then**
    $\mathbf{p}_- \leftarrow$ PREDECESSOR$(\mathbf{p})$
    return ACYCLIC$(\mathbf{p}_-, \varnothing) \wedge$ ACYCLIC$(\mathbf{p}_-, \mathbf{p})$
  else
    $\mathbf{p}_- \leftarrow$ PREDECESSOR$(\mathbf{p})$
    return $\neg(\hat{V}_L[\mathbf{p}] \preceq \hat{V}_L[\mathbf{p}']) \wedge$ ACYCLIC$(\mathbf{p}_-, \varnothing)$

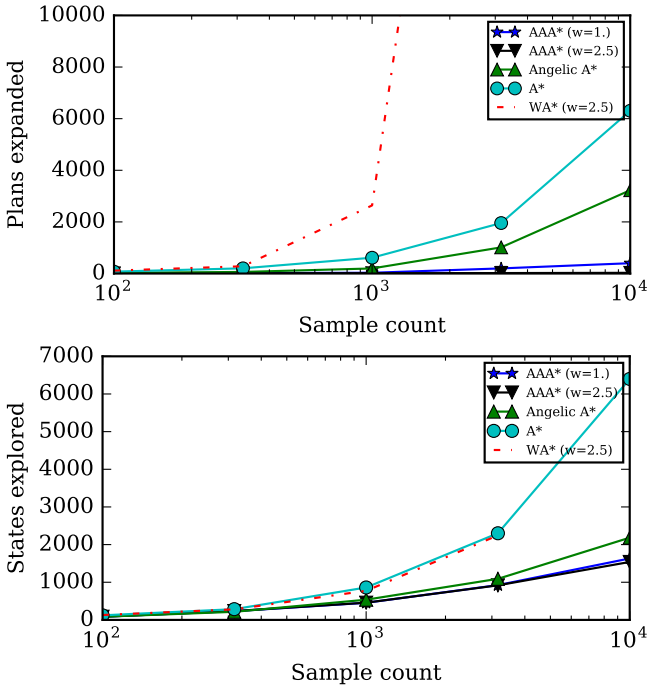


Fig. 4: Quantitative evaluation on an easy instance of the door puzzle domain with only two doors. More difficult instances could not be solved by any algorithm considered except algorithm 1. The abscissa measures the number of randomly sampled states in the discretization of the configuration space. The ordinate axes measure the number plans expanded by each algorithm and the number of distinct configurations explored during search.

	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 I: 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 expanding fewer plans and exploring fewer states, yielding a faster search and optimal or nearly optimal results.

VII. 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. It may provide a path forward for domains where conventional abstraction strategies have proven ineffective.

References

Cambon, Stephane, Rachid Alami, and Fabien Gravot (2009). "A hybrid approach to intricate motion, manipulation and task planning". In: *IJRR* 28.1, pp. 104–126.

Garrett, Caelan Reed, Tomás Lozano-Pérez, and Leslie Pack Kaelbling (2015). "FFROB: An efficient heuristic for task and motion planning". In: *WAFR*.

Gravot, Fabien, Stephane Cambon, and Rachid Alami (2005). "aSyMov: a planner that deals with intricate symbolic and geometric problems". In: *ISRR*.

Hart, Peter, Nils Nilsson, and Bertram Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Kaelbling, Leslie Pack and Tomás Lozano-Pérez (2011). "Hierarchical task and motion planning in the now". In: *ICRA*.

Karaman, Sertac and Emilio Frazzoli (2011). "Sampling-based algorithms for optimal motion planning". In: *IJRR* 30.7, pp. 846–894.

Marthi, Bhaskara, Stuart Russell, and Jason Wolfe (2008). "Angelic Hierarchical Planning: Optimal and Online Algorithms". In: *ICAPS*.

Pohl, Ira (1970). "Heuristic search viewed as path finding in a graph". In: *Artificial intelligence* 1.3-4, pp. 193–204.

Srivastava, Siddharth et al. (2013). "Using classical planners for tasks with continuous operators in robotics". In: *ICAPS*.

Vega-Brown, William and Nicholas Roy (2016). "Asymptotically optimal planning under piecewise-analytic constraints". In: *WAFR*.

Wolfe, Jason, Bhaskara Marthi, and Stuart Russell (2010). "Combined Task and Motion Planning for Mobile Manipulation". In: *ICAPS*.