Sampling-Based Methods for Discrete Planning

Jason M. O’Kane  Steven M. LaValle
Dept. of Computer Science
University of Illinois
Urbana, IL 61801 USA
{ jokane, lavalle } @cs.uiuc.edu

Introduction

Methods for automated planning have been of continued interest for several decades, with roots extending as far back as (Newell & Simon 1963). Since that time, extensions and refinements have been made to the problem definition – concurrent actions and partial ordering, resource constraints, uncertainty etc— but the core problem has remained essentially the same since (Fikes & Nilsson 1971): In the context of a set of states and actions, we are given an initial state and a goal region. The task is to find a sequence of actions that move the state of the system into the goal region. The distinguishing feature of planning problems considered historically by the AI community has been a discrete search space. Only recently have continuous quantities been tackled seriously.

In contrast, research in other communities has been devoted to methods for planning in continuous search spaces. The field of motion planning deals with selecting actions for physical systems to complete tasks in a world in which geometry is relevant. Perhaps the earliest attempt is that of (Nilsson 1969), which introduced the idea of a multi-resolution grid. Contemporary motion planning generally uses multi-resolution search and randomization to search complex, high-dimensional spaces (Hwang & Ahuja 1992, Latombe 1991). These methods are mature and of general usefulness for planning in continuous spaces with complex obstacles and constraints.

Our work is based on the observation that these two long lines of research share conceptually similar problem sets. Moreover, it is our contention that these problems are similar enough to merit consideration as a unified whole. In this abstract, we begin to develop a case for such a unified view of planning and present preliminary results on one particular approach – that of borrowing the sampling-based planning ideas that have proven quite effective in continuous domains. Specifically, we describe a generalization of the Rapidly-exploring Random Tree (RRT) algorithm (LaValle & Kuffner 2000) to solve discrete planning problems. Our long-term goal is to discover a set of essential features for classifying planning problems and to develop a methodology for selecting solution methods based on these features. We believe that this approach will yield improvements for a wide range of planning problems.

Aside from the wealth of research in each of the two communities from which we draw, there is relatively little apparent precedent for this approach. A start at a similar approach is given by (Dorst, Mandhyan, & Trovato 1991), in which planning is reduced to the problem of finding integral curves of a vector field placed over the state space. The field of hybrid systems deals with systems that mix continuous and discrete elements (Branicky 1995). In (Branicky et al. 2003), a hybrid systems approach is taken to the problems we consider, but that work lacks the crucial notion of local search operator that we introduce here.

The remainder of this abstract is structured as follows. First, we will define planning problems in a general fashion. Next, we will describe the basic RRT algorithm. Then we will outline the extensions to apply it in discrete domains and illustrate these extensions with an implementation for a simple puzzle problem. We present experiments comparing this implementation to FF, a contemporary general-purpose planner. Finally, we mention the difficulties to be overcome in order to use these methods for traditional STRIPS-style propositional planning.

Problem definition

In this section we formalize the idea of a planning problem. Our definition is not novel, except in our emphasis on minimizing the set of assumptions we make. We intentionally avoid explicit reference to the representations of states and actions in order to emphasize the generality of this definition. Formally, a planning problem is a 6-tuple \((X, x_{\text{init}}, X_{\text{goal}}, U, f, X_{\text{obst}})\), composed of:

- A search space \(X\). We consider it typical to choose the underlying state space as the search space \(X\). However we emphasize that, several important exceptions illustrate the generality of this formulation. For example, certain kinds of state uncertainty may be handled by a search in belief space rather than state space (Russell & Norvig 2003).
- An initial state \(x_{\text{init}} \in X\).
- A goal region \(X_{\text{goal}} \subseteq X\). The goal region need not be explicitly represented in the problem statement.
- For each state \(x \in X\), set \(U(x)\) of available actions.
A state transition equation defining action effects. Generally, time consists of discrete steps and we have a function $f : X \times U \rightarrow X$. If time is continuous, then state transitions are governed by a differential equation $dx/dt = f(x,u)$. To make this more manageable, choose a $\Delta t > 0$ and discretize time into intervals of this length. Under this discretization, a discrete step transition function can be defined by integrating $f$ over intervals of length $\Delta t$.

- A set $X_{\text{obst}} \subseteq X$ of illegal states that encodes a set of constraints on acceptable states.

To define the notion of solution to such a problem, consider a sequence of actions $u_1, \ldots, u_k$. Use this sequence to inductively define a sequence of states $x_1, \ldots, x_{k+1}$:

$$x_1 = x_{\text{init}}$$

$$x_i = f(x_{i-1}, u_{i-1})$$

We call the sequence $u_1, \ldots, u_k$ a solution if for $x_{k+1} \in X_{\text{goal}}$ and $(x_1, \ldots, x_{k+1}) \cap X_{\text{obst}} = \emptyset$.

### Sampling-based planning

Motion planning is almost universally approached as a search through the configuration space (C-space) of the system of interest. Informally, the C-space is a continuous space with one dimension for each degree of freedom of the underlying system. Each point in C-space completely describes a single configuration of the system. Within this space, the obstacle set $X_{\text{obst}}$ consists of configurations containing collisions. Combinatorial methods such as (Schwartz & Sharir 1983, Canny 1988) that compute the obstacle regions explicitly have extremely poor time bounds and have also proven exceptionally difficult to implement correctly. For this reason, an large family of sampling-based planners has been developed that avoid building an explicit representation of $X_{\text{obst}}$. For example, see the planners of (Mazer, Ahuactzin, & Bessi`ere 1998, Kavraki et al. 1996, Amato & Wu 1996). For a more complete survey of the use of sampling-based algorithms in motion planning, see (Lindemann & LaValle 2004). Since the state space of any nontrivial discrete planning problem will be hopelessly large, efficient planners must also avoid building an explicit representation of $X$. This suggests that sampling-based planning may be effective in these domains as well. For the remainder of this abstract, we focus on one particular sampling-based planning algorithm, the Rapidly-exploring Random Tree.

### Rapidly-exploring random trees

The Rapidly-exploring Random Tree (RRT) algorithm was originally developed for robot motion planning in high dimensional configuration spaces. We present the basic algorithm here and refer the reader to (LaValle & Kuffner 2000) for a more comprehensive presentation. The intuition is to use a sequence of sample states to bias the expansion of a search tree toward unexplored regions of the search space. RRTs have been used in an extremely wide variety of applications (see (Branicky et al. 2003, Williams et al. 2001, Lim & Ostrowski 2003) for a few examples).

---

**Algorithm 1 RRT**

1: $T \leftarrow \{x_{\text{init}}\}$
2: for $k \leftarrow 1 \ldots K$ do
3: $x_s \leftarrow \text{SAMPLESTATE}()$
4: $x_n \leftarrow \text{NEARESTNEIGHBOR}(x_s, T)$
5: $T \leftarrow \text{EXTENDTREE}(T, x_n, x_s)$
6: end for

---

**Figure 1:** An RRT exploring a planar search space after 10, 100, 1000, and 5000 iterations.

Algorithm 1 outlines the basic RRT for exploring a search space. In the most general case, we have a search space $X$ in which we build a search tree rooted at $x_{\text{init}}$. Consider $X$ a metric space by equipping it with a distance function $\rho : X \times X \rightarrow \mathbb{R}$. At each step of the search process, we select a random sample $x_s$. The nearest node to $x_s$ in the tree is selected for expansion. In EXTENDTREE, we add a new child node to $x_n$ by moving closer to $x_s$. More precisely, we choose an action $u = \arg \min_{u \in U(x_s)} \rho(f(x_n, u), x_s)$. For each potential new node $x$, we perform a membership test to determine whether $x \in X_{\text{obst}}$. If so, we discard $x$ and continue without expanding the tree.

A common variant is to apply this $u$ several times in an attempt to connect the tree to $x_s$. This variant will play an important role in our generalization of the algorithm.

The basic algorithm only "explores" a search space without regard to the goal region. For planning, information about the goal region can be incorporated in one of two ways. We may introduce goal bias into our sample sequence, or we may grow a pair of trees, as in bidirectional search. At each iteration, we extend one tree toward a sample state, then extend the other tree toward the newly added node. Thus, unlike typical bidirectional methods, which are essentially a combination of independent forward and backward chaining processes, bidirectional RRTs actively seek to connect the initial and goal trees.

**Voronoi bias** How can we characterize the choices made of nodes for expansion? The Voronoi diagram of a finite set of points $T$ in a space $X$ is a piecewise-constant function $V : X \rightarrow T$ in which $x \mapsto \text{NEARESTNEIGHBOR}(x, S)$. RRTs exhibit Voronoi bias in the sense that the probability that a state $s$ in the tree will be selected for expansion is related to the volume of its Voronoi region. If $E$ is a random variable indicating which node in the tree will be expanded, then for any node $x_n \in T$, we have

$$P(E = x_n) = \frac{\text{Vol}(V^{-1}(x_n))}{\text{Vol}(X)}$$
Thus, we can imagine large unexplored regions of the search space generating attracting forces that "pull" the search tree into them with strength proportional to their volume.

Another way to view the RRT algorithm is as a variation on standard tree search methods, but with a careful scheme for selecting nodes in the tree for expansion. It is distinguished from other methods by its reliance on the metric $\rho$. With this distance measure, it makes sense to think of large, connected regions of the search space that do not contain any nodes of $T$. This algorithm seeks out such regions and extends $T$ into them, thereby quickly achieving good coverage of the space in a relatively small number of iterations.

RRTs in a discrete space

Having outlined the notion of sampling-based planning in general and RRTs in particular, we now examine some of the difficulties that result from its application to discrete search spaces.

Metrics

The most important observation to make is that RRTs depend crucially on the existence of a metric function $\rho$. Without such a function, the nearest neighbor operators make no sense. This requirement will generally not present major difficulties, however, because nearly all standard heuristics and heuristic extraction procedures are already stated in terms of some target state. More precisely, suppose we have a family of admissible heuristic functions $H_x : X \rightarrow \mathbb{R}$ in which $H_x(x')$ denotes an underestimate of the cost to travel from $x'$ to $x$. Then a metric function $\rho : X \times X \rightarrow \mathbb{R}$ can be defined in an obvious way: $(x_1, x_2) \mapsto H_{x_1}(x_2)$. Of course, it remains to show that $X$ with $\rho$ is a metric space. This is not true in general, but we expect the performance penalty for using a $\rho$ that is not a true metric to be modest and the degradation to be graceful.

Local search operators

In the RRT algorithm, each iteration of the search is a time-consuming operation, primarily because of the nearest-neighbor query that must be made of the tree at every step. Therefore, to build an effective planner, we should maximize the amount of progress made at each iteration. In continuous spaces, this has been done by an enhanced $\text{EXTEND}$ step that applies the selected action for a period of time longer than $\Delta t$ in hopes of making more progress toward $x_s$. A similar extension can be made for discrete problems. Given a sample state $x_s$ and its nearest neighbor in the tree $x_{n}$, rather than choosing a single action to move toward $x_s$, we would like to take multiple steps toward $x_s$ at once. We emphasize that we are not particularly interested in actually reaching $x_s$, but only moving toward it and into the large empty Voronoi region containing it.

To this end, we choose a local search operator to perform the $\text{EXTEND}$ step in which we build a path starting at $x_n$ and moving toward $x_s$. We envision using the RRT algorithm to "direct" the search while the local search operator handles smaller-scale details. In principle, we allow any search or planning technique as a local search operator. We require only that it make progress toward its goal in a short period of time. Two obvious first choices are hill climbing (repeatedly choose the action that decreases most the distance to $x_s$) and an iteration-limited truncation of $A^*$. Perhaps more interestingly, any existing state space search technique may be used, in particular those that incorporate domain-specific knowledge. The experiments in the next section include a preliminary investigation of these operators.

An illustration

As an illustrative example, consider the $n^2 - 1$ puzzle. We have $n^2 - 1$ numbered tiles in $n \times n$ grid with one open space. Actions consist of swapping the open space with any adjacent tile. The initial state is an arbitrary solvable state; the goal region is a single state, generally the one in which the tiles are ordered according to their numbers, left to right and top to bottom. We attempt to solve this problem with a minimum of problem-specific information. Although the solution of simple puzzles like $n^2 - 1$ is not of direct interest, this problem provides a simple, easily implemented and easily visualized testbed for preliminary testing of our ideas.

If we denote each tile by its number and let $n$ denote the blank space, then the state space $X$ is simply the permutation group $S_n$. The action sets $U(x)$ consist of the appropriate transpositions of the blank with a adjacent tile; $f$ is defined by composition of permutations. For $\rho$, we adopt the aggregate Manhattan distance heuristic suggested by (Russell & Norvig 2003). Since $\rho$ is the sum of $n^2$ independent $L_1$ distances, it can easily be shown to be a metric itself.

We have implemented the discrete bidirectional RRT algorithm to solve this problem. For local search operators we consider $A^*$ limited to 100 and 1000 iterations in addition to the trivial operator that selects the single most promising action from each expanded state. For perspective, we compare our results to those obtained by FF, a well-known state-space planner (Hoffmann & Nebel 2001). The results in Fig. 3 suggest that our simple technique may be competitive with FF in both run time and solution quality. We must of course avoid taking this comparison too seriously, since FF is a general-purpose planner operating on a propositional representation while our implementation is problem specific both in its metric and its representation. Nonetheless, these results appear to suggest potential for sampling-based methods to be quite successful for a much wider variety of planning problems than those for which they have in the past been used.

RRTs for propositional planning

We now consider classical propositional planning problems. This section outlines several important issues that must be faced to solve such problems with sampling-based algorithms. Let us impose some degree of structure on $X$, $U$, and $f$. Let $P$ represent a finite set of predicates. Each state $x \in X$ is a subset of $P$, indicating precisely which predicates are true in that state. Each action $u \in U$ is characterized by three lists $\text{Pre}(u)$, $\text{Add}(u)$, $\text{Del}(u) \subseteq P$ which are the precondition list, add list, and delete list, respectively. For each state $x$, the action set is $U(x) = \{u \in U|\text{Pre}(u) \subseteq x\}$. The state transition function is defined by
Figure 2: Several selected instances of the $n^2 - 1$ puzzle.

<table>
<thead>
<tr>
<th>$n$</th>
<th>RRT</th>
<th>RRT/A100</th>
<th>RRT/A1000</th>
<th>FF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>3</td>
<td>0.71</td>
<td>0.01</td>
<td>0.04</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
<td>0.76</td>
<td>0.02</td>
<td>0.05</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4</td>
<td>308.97</td>
<td>0.62</td>
<td>0.93</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4</td>
<td>&gt; 1800</td>
<td>0.62</td>
<td>1.11</td>
</tr>
<tr>
<td>$P_5$</td>
<td>5</td>
<td>&gt; 1800</td>
<td>189.95</td>
<td>41.66</td>
</tr>
<tr>
<td>$P_6$</td>
<td>5</td>
<td>&gt; 1800</td>
<td>260.99</td>
<td>26.86</td>
</tr>
</tbody>
</table>

Figure 3: Results for several planners on the $n^2 - 1$ puzzle instances from Fig. 2. RRT/A100 and RRT/A1000 signify RRT with A* with iterations limited to 100 and 1000, respectively. Results for RRT-based planners are averages over 10 runs. Blanks indicate a failure to solve the problem within 30 minutes.

<table>
<thead>
<tr>
<th>$n$</th>
<th>RRT</th>
<th>RRT/A100</th>
<th>RRT/A1000</th>
<th>FF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>3</td>
<td>24.60</td>
<td>53.00</td>
<td>43.20</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
<td>30.00</td>
<td>56.00</td>
<td>43.00</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4</td>
<td>74.00</td>
<td>186.00</td>
<td>197.40</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4</td>
<td>188.60</td>
<td>211.40</td>
<td>211.40</td>
</tr>
<tr>
<td>$P_5$</td>
<td>5</td>
<td>511.60</td>
<td>579.60</td>
<td>579.60</td>
</tr>
<tr>
<td>$P_6$</td>
<td>5</td>
<td>560.40</td>
<td>475.20</td>
<td>475.20</td>
</tr>
</tbody>
</table>

Figure 4: Histogram of solution qualities for RRT/A100 solving $P_1$ 1000 times. The optimal solution has 21 steps. Note that 27% of the solutions are within a factor of 2 of optimality and 94% are superior to the solution generated by FF.

of propositions in which they disagree.

**Goal regions** The usual bidirectional approach depends on a tree rooted at some $x_{goal} \in X_{goal}$. It is not difficult to select such a representative – we can simply choose a truth value for each proposition whose value is unspecified in the description of $X_{goal}$. The problem is that we have no guarantee that the $x_{goal}$ we select is reachable from $x_{init}$. If $x_{goal}$ is in an unreachable connected component of $X_{goal}$, the search will certainly fail.

If we are unsure of the connectivity of $X_{goal}$, we instead adopt the convention that the root node $x_G$ of the goal tree covers all of $X_{goal}$. Then, given a sample $x_s$, the nearest $x \in X_{goal}$ is the one that agrees with $x_s$ in every dimension that is absent from the goal description. Using this method when finding the nearest neighbor of a sample state $x_s$, we grow the goal tree appropriately from this nearest state in $X_{goal}$.

**Sparse Samples** Our algorithm depends on a sequence of sample states. If there are many propositions, then it may be prohibitively expensive to even generate such a sample explicitly. To circumvent this problem, we propose lazy sample generation to achieve results equivalent to generating a random sample without ever explicitly writing down each bit of $x_s$. Rather than generating a sample completely, we may treat the value in most dimensions as to be announced (TBA). When a value is needed for any TBA bit, it can be generated on-the-fly.

For example, consider the nearest-neighbor operation. To compute the nearest neighbor in $T$ of $x_s$, it is sufficient to assign in $x_s$ values to those propositions that appear in states already in $T$, leaving the values for the other propositions in an implicit TBA state. These undecided values will contribute some additive quantity $C$ to $\rho(x, x_s)$ for every $x \in T$. 

\[ f(x, u) = (x - Del(x)) \cup Add(x). \]

Given two disjoint sets $G^+, G^- \subseteq A$, the goal region is $X_{goal} = \{ x \in X | G^+ \subseteq x, G^- \cap x = \emptyset \}$.

Geometrically, $X$ is a boolean cube of dimension $|P|$. Similarly, $X_{goal}$ is a subcube of dimension $|P| - (|G^+| + |G^-|)$. Thus, we may think of each state as string of bits and actions as manipulating these bit strings directly. In practice, however, we may expect these strings to be quite sparse, because the set of propositions that are true in a typical state of interest to us will be small. As a result, explicit representation of these bit strings seems ill-advised; we must take care to develop methods that are compatible with sparse representations of states.

For concreteness, let us consider a bidirectional RRT with iteration-limited $A^*$ as the local search operator. The most important question to answer is the selection of a metric function. In keeping with our geometric view of the state space, the obvious choice is the Hamming distance, under which the distance between any pair of states is the number
Since we are only concerned with the nearest neighbor to $x_*$, subtracting $C$ from each distance makes no change to the $x_n$ selected.

Of course, these tricks apply directly only to the Hamming metric and to the trivial generalization to any linear function of the propositions true in each state. We are keenly interested in studying how more sophisticated and powerful heuristics from, for example, (Ghallab & Laruelle 1994, Bonet & Geffner 2001, Hoffmann & Nebel 2001) may be adapted to work in this context.

Our near-term goal is to implement a planner based on these ideas. We believe that such a planner will have several important advantages, of which we mention two here. First, there are excellent possibilities for supporting a richer class of goal regions. For example, problems with disjunctive goals may be handled with a runtime penalty at each step only linear in the number of disjuncts. Secondly, since we are using planning technology developed for continuous search spaces, such a planner should be able to incorporate problems with numerical elements by adding additional real-valued dimensions to the search space.

**Conclusion**

We have sketched an algorithm for propositional planning that borrows metric-based ideas that historically have been used for planning high-dimensional continuous domains. Our purpose is not to propose using RRTs for all planning problems, but to unify the approach to search and planning in disparate domains. We believe there is fertile ground for idea-sharing between various planning disciplines.

We will continue this work by implementing the planner sketched above. We are also interested in seeking better understanding and justification for our local search operators and in exploring a posteriori techniques for improving solution quality.

**References**


...