Heuristic Search in Infinite State Spaces Guided by Lyapunov Analysis

Theodore J. Perkins and **Andrew G. Barto**
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003, USA
{perkins,barto}@cs.umass.edu

Abstract

In infinite state spaces, many standard heuristic search algorithms do not terminate if the problem is unsolvable. Under some conditions, they can fail to terminate even when there are solutions. We show how techniques from control theory, in particular Lyapunov stability analysis, can be employed to prove the existence of solution paths and provide guarantees that search algorithms will find those solutions. We study both optimal search algorithms, such as A*, and suboptimal/real-time search methods. A Lyapunov framework is useful for analyzing infinite-state search problems, and provides guidance for formulating search problems so that they become tractable for heuristic search. We illustrate these ideas with experiments using a simulated robot arm.

1 Introduction

As the boundaries become less distinct between artificial intelligence and fields more reliant on continuous mathematics, such as control engineering, it is being recognized that heuristic search methods can play useful roles when applied to problems with infinite state spaces (e.g., Boone [1997], Davies *et al.* [1998]). However, the theoretical properties of heuristic search algorithms differ greatly depending on whether the state space is finite or infinite.

For finite state space problems, a variety of well-understood algorithms are available to suit different needs. For example, the A* algorithm finds optimal solutions when they exist, and is also "optimally efficient"—no other heuristic search algorithm has better worst-case complexity. Variants, such as IDA*, allow more memory-efficient search at the cost of greater time complexity. Conversely, suboptimal search methods, such as depth-first search or best-first search with an inadmissible heuristic, can often produce some solution, typically suboptimal, more quickly than A* can find an optimal solution [Pearl, 1984; Russell and Norvig, 1995]. The RTA* algorithm is able to choose actions in real-time, while still guaranteeing eventual arrival at a goal state [Korf, 1990]. However, if the state space is infinite, none of these algorithms is guaranteed to have the same properties. A* is complete only if additional conditions hold on the costs of search operators, and it does not terminate if the problem admits no solution [Pearl, 1984]. Suboptimal search methods may not terminate, even if a closed list is maintained. RTA* is not guaranteed to construct a path to a goal state [Korf, 1990].

A further difficulty when infinite state spaces are considered is that the most natural problem formulations often include infinite action spaces. To apply heuristic search, one must select a finite subset of these actions to be explored in any given state. In doing so, the possibility arises that an otherwise reachable goal set becomes unreachable.

Some of these difficulties are unavoidable in general. With an infinite number of possible states, a search problem may encode the workings of a Turing machine, including its memory tape. Thus, the question of whether or not an infinite-state search problem has a solution is in general undecidable. However, it is possible to address these difficulties for useful subclasses of problems. In this paper we examine how Lyapunov analysis methods can help address these difficulties when it is applicable. A Lyapunov analysis of a search problem relies on domain knowledge taking the form of a Lyapunov function. The existence of a Lyapunov function guarantees that some solution to the search problem exists. Further, it can be used to prove that various search algorithms will succeed in finding a solution. We study two search algorithms in detail: A* and a simple iterative, real-time method that incrementally constructs a solution path. Our main goal is to show how Lyapunov methods can help one analyze and/or formulate infinite-state search problems so that standard heuristic search algorithms are applicable and can find solutions.

The paper is organized as follows. In Section 2 we define heuristic search problems. Section 3 covers some basics of Lyapunov theory. Sections 4 and 5 describe how Lyapunov domain knowledge can be applied to prove that a search algorithm will find a solution path. The relationship between Lyapunov functions and heuristic evaluation functions is also discussed. Section 6 contains a demonstration of these ideas on a control problem for a simulated robot arm. Section 7 concludes.

2 State Space Search Problems

Definition 1 *A state space search problem (SSP) is a tuple* $(S, G, s_0, \{O_1, \ldots, O_k\})$, *where:*

- *S is the state set. We allow this to be an arbitrary set.*

- *$G \subset S$ is the set of goal states.*

- *$s_0 \notin G$ is the initial, or start, state.*

- *$\{O_1, \ldots, O_k\}$ is a set of search operators. Some search operators may not be applicable in some states. When a search operator O_j is applied to a state $s \notin G$, it results in a new state $Succ_j(s)$ and incurs a cost $c_j(s) \geq 0$.*

A solution to an SSP is a sequence of search operators that, when applied starting at s_0, results in some state in G. An optimal solution to an SSP is a solution for which the total (summed) cost of the search operators is no greater than the total cost of any other solution.

For infinite state spaces, the infimal cost over all solution paths may not be attained by any path. Thus, solutions may exist without there being any optimal solutions. This possibility is ruled out if there is a universal lower bound $c_{low} > 0$ on the cost incurred by any search operator in any non-goal state [Pearl, 1984]. We will use the phrase "the SSP's costs are bounded above zero" to refer to this property.

Note that our definition of an SSP includes a finite set of search operators $\{O_1, \ldots, O_k\}$, some of which may be unavailable in some states. We have made this choice purely for reasons of notational convenience. The theory we present generalizes immediately to the case in which there is an infinite number of search operators, but the number of operators applicable to any particular non-goal state is finite.

3 Control Lyapunov Functions

Lyapunov methods originated in the study of the stability of systems of differential equations. These methods were borrowed and extended by control theorists. The techniques of Lyapunov analysis are now a fundamental component of control theory and are widely used for the analysis and design of control systems [e.g., Vincent and Grantham 1997].

Lyapunov methods are tools for trying to identify a Lyapunov function for a control problem. In search terms, a Lyapunov function is most easily understood as a descent function (the opposite of a hill-climbing function) with no local minima except at goal states.

Definition 2 *Given an SSP, a control Lyapunov function (CLF) is a function $L : S \mapsto \Re$ with the following properties:*

1. *$L(s) \geq 0$ for all $s \in S$.*

2. *There exists $\delta > 0$ such that for all $s \notin G$ there is some search operator O_j such that $L(s) - L(Succ_j(s)) \geq \delta$.*

The second property asserts that at any non-goal state some search operator leads to a state at least δ down on the CLF. Since a CLF is non-negative, a descent procedure cannot continue indefinitely. Eventually it must reach a state where a δ step down on L is impossible; such a state can only be a goal state.

A CLF is a strong form of domain knowledge, but the benefits of knowing a CLF for a search problem are correspondingly strong. The existence of solutions is guaranteed, and (suboptimal) solutions can be constructed trivially. Numerous sources discuss methods for finding Lyapunov functions

(e.g., Vincent and Grantham [1997], Krstić *et al.* [1995]). For many important problems and classes of problems, standard CLFs have already been developed. For example, linear and feedback linearizable systems are easily analyzed by Lyapunov methods. These systems include almost all modern industrial robots [Vincent and Grantham, 1997]. Path planning problems similarly yield to Lyapunov methods [Connolly and Grupen, 1993]. Many other stabilization and control problems, less typically studied in AI, have also been addressed by Lyapunov means, including: attitude control of ships, airplanes, and spacecraft; regulation of electrical circuits and engines; magnetic levitation; stability of networks or queueing systems; and chemical process control (see Levine [1996] for references). Lyapunov methods are relevant to many important and interesting applications.

The definition of a CLF requires that at least one search operator leads down by δ in any non-goal state. A stronger condition is that all search operators descend on the CLF:

Definition 3 *Given an SSP, the set of search operators descends on a CLF L if for any $s \notin G$ there exists at least one applicable search operator, and every applicable search operator O_j satisfies $L(s) - L(Succ_j(s)) \geq \delta$ for some fixed $\delta > 0$.*

4 CLFs and A*

In this section we establish two sets of conditions under which A* is guaranteed to find an optimal solution path in an SSP with an infinite state space. We then discuss several other search algorithms, and the relationship of CLFs to heuristic evaluation functions.

Theorem 1 *If there exists a CLF L for a given SSP, and either of the following conditions are true:*

1. *the set of search operators descends on L, or*

2. *the SSP's costs are bounded above zero,*

then A* search will terminate, finding an optimal solution path from s_0 to G.

Proof: Under condition 1, there are only a finite number of non-goal states reachable from a given start state s_0. The only way for an infinite number of states to be reachable is for them to occur at arbitrarily large depths in the search tree. But since every search operator results in a state at least δ lower on L and $L \geq 0$ everywhere, no state can be at a depth greater than $\lceil L(s_0)/\delta \rceil$. Since the CLF implies the existence of at least one solution and there are only a finite number of reachable states, it follows (e.g., from Pearl [1984], section 3.1), that A* must terminate and return an optimal solution path.

Alternatively, suppose conditions 2 holds. The CLF ensures the existence of at least one solution path. Let this path have cost C. Since each search operator incurs at least c_{low} cost, the f-value of a node at depth d is at least $d \cdot c_{low}$. A* will not expand a node with f-value higher than C. Thus, no node at depth $d > \lceil C/c_{low} \rceil$ will ever be expanded. This means A* must terminate and return an optimal solution. QED.

These results extend immediately to variants of A* such as uniform cost search or IDA*. Under condition 1, not only is

the number of reachable states finite, but these states form an acyclic directed graph and all "leaves" of the graph are goal states. Thus, a great many search algorithms can safely be applied in this case. Under condition 2, depth-first branch-and-bound search is also guaranteed to terminate and return an optimal solution, if it is initialized with the bounding cost C. This follows from nearly the same reasoning as the proof for A*.

How do CLFs relate to heuristic evaluation functions? In general, a CLF seems a good candidate for a heuristic function. It has the very nice property that it can be strictly monotonically decreased as one approaches the goal. Contrast this, for example, to the Manhattan distance heuristic for the 8-puzzle [Russell and Norvig, 1995]. An 8-puzzle solution path typically goes up and down on the Manhattan distance heuristic a number of times, even though the ultimate effect is to get the heuristic down to a value of zero.

However, a CLF might easily fail to be admissible, overestimating the optimal cost-to-goal from some non-goal state. Notice that the definition of a CLF does not depend on the costs of search operators of an SSP. It depends only on S, G, and the successor function. CLFs capture information about the connectivity of the state space more than the costs of search operators.

A possible "fix" to the inadmissibility of a CLF is to scale it. If a CLF is multiplied by any positive scalar, defining a new function $L' = \alpha L$, then L' is also a CLF. If a given CLF overestimates the cost-to-goal for an SSP, then it is possible that a scaled-down CLF would not overestimate. We will not elaborate on the possibility of scaling CLFs to create admissible heuristics in this paper. We will, however, use the scalability of a CLF in the next section, where we discuss real-time search.

5 Real-Time Search

Lyapunov domain knowledge is especially well suited to real-time search applications. As mentioned before, one can rapidly generate a solution path by a simple descent procedure. A generalization of this would be to construct a path by performing a depth-limited search at each step to select the best search operator to apply next. "Best" would mean the search operator leading to a leaf for which the cost-so-far plus a heuristic evaluation is lowest. We will call this algorithm "repeated fixed-depth search" or RFDS.

Korf [1990] described this procedure in a paper on real-time search and, rightly, dismissed it because in general it is not guaranteed to produce a path to a goal state, even in finite state spaces. However, the procedure is appealing in its simplicity and appropriate for real-time search in that the search depth can be set to respond to deadlines for the selection of search operators. In this section we examine conditions under which this procedure can be guaranteed to construct a path to a goal state.

Theorem 2 *Given an SSP and a CLF, L, for that SSP, if the set of search operators descends on L then RFDS with any depth limit $d \geq 1$ and any heuristic evaluation function will terminate, constructing a complete path from s_0 to G.*

Proof: At each step, any search operator that is appended to the growing solution will cause a step down on L of at least δ, which inevitably leads to G. QED.

A more interesting and difficult case is when some of the operators may not be descending, so that the solution path may travel up and down on the CLF.

Theorem 3 *Given an SSP whose costs are bounded above zero and a CLF, L. Suppose that L is used as a heuristic evaluation of non-goal leaves, and that $L(s) - L(Succ_1(s)) \geq c_1(s)$ for all $s \notin G$. Then RFDS with any depth limit $d \geq 1$ will terminate, constructing a complete path from s_0 to G.*

Note that the theorem assumes that O_1 has the special property $L(s) - L(Succ_1(s)) \geq c_1(s)$. More generally, in each state there must be some search operator that satisfies this property. It is only for notational convenience that we have assumed that O_1 satisfies the property everywhere.

To prove Theorem 3, we require some definitions. Suppose RFDS generates a path $s_0, s_1, \ldots, s_N$, where s_N may or may not be a goal state but the other states are definitely non-goal. At the tth step, $t \in \{0, \ldots, N\}$, RFDS generates a search tree to evaluate the best operator to select next. Let g_t be the cost of the search operators leading up to state s_t; let l_t be the leaf-state with the best (lowest) evaluation in the tth search tree; let $c_{t,1}, \ldots, c_{t,n(t)}$ be the costs of the search operators from s_t to l_t; and let h_t be the heuristic evaluation of l_t—zero if the leaf corresponds to a goal state, and $L(l_t)$ otherwise. Define $f_t = g_t + \sum_{i=1}^{n(t)} c_{t,i} + h_t$.

Lemma 1 *Under the conditions of Theorem 3, $f_t \geq f_{t+1}$ for $t \in \{0, \ldots, N-1\}$.*

Proof: $f_t = g_t + \sum_{i=1}^{n(t)} c_{t,i} + h_t = g_{t+1} + \sum_{i=2}^{n(t)} c_{t,i} + h_t$ because s_{t+1} is one step along the path to the best leaf of iteration t. Thus, the inequality we need to establish is $\sum_{i=2}^{n(t)} c_{t,i} + h_t \geq \sum_{i=1}^{n(t+1)} c_{t+1,i} + h_{t+1}$. The right side of this inequality is simply the cost of some path from s_{t+1} to l_{t+1} plus a heuristic evaluation. The left side corresponds to a path from s_{t+1} to l_t, plus a heuristic evaluation.

First, suppose l_t is a goal state. The path from s_{t+1} to l_t will be included among those over which RFDS minimizes at the tth step. Thus, in this case the inequality holds.

If l_t is not a goal state, then the search at iteration $t+1$ expands l_t. The evaluation of the path from s_{t+1} to $Succ_1(l_t)$ is

$$
\begin{aligned}
&\sum_{i=2}^{n(t)} c_{t,i} + c_1(l_t) + h(Succ_1(l_t)) \\
\leq\ &\sum_{i=2}^{n(t)} c_{t,i} + c_1(l_t) + L(Succ_1(l_t)) \\
\leq\ &\sum_{i=2}^{n(t)} c_{t,i} + L(l_t) \\
=\ &\sum_{i=2}^{n(t)} c_{t,i} + h_t.
\end{aligned}
$$

Thus, this path's cost meets the inequality, and since RFDS minimizes over that and other paths, the inequality must hold for the best path of iteration $t+1$. QED.

Proof of Theorem 3: Suppose RFDS runs forever without constructing a path to a goal state. From the previous lemma, $f_t \leq f_0$ for all $t \geq 0$. Since the SSP's costs are bounded above zero, $f_t \geq g_t \geq t \cdot c_{low}$. For sufficiently large t, this contradicts $f_t \leq f_0$. Thus RFDS does construct a path to goal. QED.

Theorem 3 requires a relationship between the decrease in a CLF caused by application of O_1 and the cost incurred

by O_1. A given CLF may not satisfy this property, or one may not know whether the CLF satisfies the property or not. We propose two methods for generating a CLF guaranteed to have the property, assuming that applying O_1 to any non-goal state causes a decrease in the original CLF of at least δ, for some fixed $\delta > 0$.

The first method is scaling, which is applicable when there is a maximum cost c_{high} that operator O_1 may incur when applied to any non-goal state. In this case, consider the new CLF $L' = \frac{c_{high}}{\delta}L$. For any $s \notin G$, $L'(s) - L'(\text{Succ}_1(s)) = (c_{high}/\delta) \cdot (L(s) - L(\text{Succ}_1(s))) \geq (c_{high}/\delta) \cdot \delta = c_{high} \geq c_1(s)$. Thus, L' is a CLF satisfying the conditions of Theorem 3.

A problem with this scheme is that one or both of the constants c_{high} and δ_1 may be unknown. An overestimate of the constant $\frac{c_{high}}{\delta_1}$ will produce a new CLF satisfying the theorem. If an overestimate cannot be made directly, then a scaling factor can be determined on-line, as RFDS runs. Suppose one defines $L' = \alpha L$ for any initial guess α. One can than perform RFDS, checking the condition $L'(s) - L'(\text{Succ}_1(s)) \geq c_1(s)$ every time O_1 is applied to some state. If the condition is violated, update the scaling factor, e.g., as $\alpha \leftarrow \frac{c_1(s)}{L'(s) - L'(\text{Succ}_1(s))} + \varepsilon$, for some fixed $\varepsilon > 0$. This update rule ensures that the guess α is consistent with all the data observed so far and will meet or exceed $\frac{c_{high}}{\delta_1}$ in some finite number of updates. If RFDS does not construct a path to a goal state by the time that a sufficient number of updates are performed, then Theorem 3 guarantees the completion of the path afterward.

A second method for generating a CLF that meets the conditions of Theorem 3 is to perform roll-outs [Tesauro and Galperin, 1996; Bertsekas *et al.*, 1997]. In the present context, this means defining $L'(s) =$ "the cost of the solution path generated by repeated application of O_1 starting from s and until it reaches G". The function L' is evaluated by actually constructing the path. The reader may verify that L' meets the definition of a CLF and satisfies the requirements of Theorem 3. Performing roll-outs can be an expensive method for evaluating leaves because an entire path to a goal state needs to be constructed for each evaluation. However, roll-outs have been found to be quite effective in both game playing and sequential control [Tesauro and Galperin, 1996; Bertsekas *et al.*, 1997].

6 Robot Arm Example

We briefly illustrate the theory presented above by applying it to a problem requiring the control of the simulated 3-link robot arm, depicted in Figure 1. The state space of the arm is $\Re^6$, corresponding to three angular joint positions and three angular joint velocities. We denote the joint position and joint velocity column vectors by θ and $\dot{\theta}$. The set G of goal states is $\{s \in S : \|s\| \leq 0.01\}$, which is a small hyper-rectangle of states in which the arm is nearly stationary in the straight-out horizontal configuration.

The dynamics of mechanical systems such as a robot arm are most naturally described in continuous time. A standard
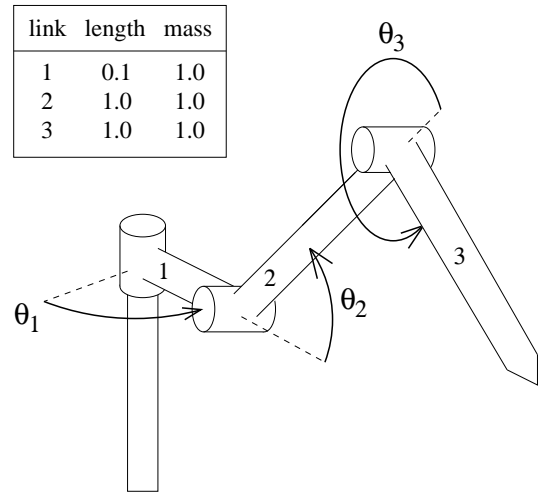
link	length	mass
1	0.1	1.0
2	1.0	1.0
3	1.0	1.0

Figure 1: Diagram of the 3-Link Robot Arm

model for a robot arm is [Craig, 1989]:

$$\frac{d}{dt}s = \frac{d}{dt}\left[\begin{array}{c} \theta \\ \dot{\theta} \end{array}\right] = \left[\begin{array}{c} \dot{\theta} \\ H^{-1}(\theta, \dot{\theta})(\tau - V(\theta, \dot{\theta}) - g(\theta)) \end{array}\right]$$

where g represents gravitational forces, V represents Coriolis and other velocity-dependent forces, H is the inertia matrix, and τ is the actuator torques applied at the joints.

We develop a set of continuous time controllers—rules for choosing τ—based on feedback linearization and linear-quadratic regulator (LQR) methods [e.g., Vincent and Grantham, 1997]. These controllers form the basis of the search operators described in the next section. Feedback linearization amounts to reparameterizing the control torque in terms of a vector u, where $\tau = Hu + V + g$. This puts the robot dynamics into the particularly simple linear form

$$\frac{d}{dt}\left[\begin{array}{c} \theta \\ \dot{\theta} \end{array}\right] = \left[\begin{array}{c} \dot{\theta} \\ u \end{array}\right]$$

to which LQR control design is applicable. An LQR controller is a rule for choosing u, and thus τ, based on θ and $\dot{\theta}$. LQR design yields two main benefits: 1) a simple controller that can asymptotically bring the arm to any specified target state, 2) a Lyapunov function that can be used as a CLF for the problem of getting to a goal state. We define five basic controllers:

- C_1: An LQR controller with $\theta = (0,0,0)$ as the target configuration. This configuration is within G, so C_1 alone is able to bring the arm into G from any initial position. We use the associated Lyapunov function as a CLF for the SSP we are constructing: $L_{arm} = s^T Q s$, where Q is a symmetric positive definite matrix produced during the standard computations for the LQR design.

- C_2: Chooses τ as an LQR controller with target configuration $\theta = (0,0,0)$ except that u is multiplied by 2. This tends to cause higher control torques that bring the arm towards the target configuration faster.

- C_3: Similar to C_2, but u is multiplied by 0.5 instead of 2.

- C_4: An LQR controller that does not apply any torque to the first joint, but regulates the second two joints to configuration $(\theta_2, \theta_3) = (\pi/4, -\pi/2)$. This tends to fold the arm in towards the center, so it can swing around the first joint more easily.

- C_5: Similar to C_4, but regulating the second two joints to the configuration $(\theta_2, \theta_3) = (\pi/2, -\pi)$.

6.1 Search Operators

We construct two sets of five search operators. In the first set of search operators, Ops_1, each operator corresponds to one of the five controllers above. $\text{Succ}_j(s)$ is defined as the state that would result if the arm started in state s and C_j controlled the arm for $\Delta = 0.25$ time interval.

We define the cost of applying a search operator to be the continuous time integral of the quantity $\|\theta(t)\|^2 + \|\tau(t) - \tau_0\|^2$, where $\theta(t)$ are the angles the arm goes through during the Δ-time interval, $\tau(t)$ are the joint torques applied, and τ_0 is the torque needed to hold the arm steady, against gravity, at configuration $\theta = (0,0,0)$. This kind of performance metric is standard in control engineering and robotics. The term $\|\theta(t)\|^2$ reflects the desire to move the arm to G; the term $\|\tau(t) - \tau_0\|^2$ penalizes the use of control torques to the extent that they differ from τ_0. Defined this way, the costs of all search operators are bounded above zero.

The second set of search operators, Ops_2, is also based on controllers C_1 through C_5, but the continuous-time execution is somewhat different. When a controller C_j, $j \in \{2,\ldots,5\}$ is being applied for a Δ-time interval, the time derivative of L_{arm}, $\dot{L}_{\text{arm}}$, is constantly computed. If $\dot{L}_{\text{arm}}$ is greater than -0.1, the torque choice of C_1 is supplied instead. Under C_1, $\dot{L}_{\text{arm}} \leq \delta_1 < 0$ for some unknown δ_1. In this way, $\dot{L}_{\text{arm}}$ is bounded below zero for all the Ops_2 search operators, and thus they all step down L_{arm} by at least the (unknown) amount $\delta = \Delta \cdot \min(0.1, \delta_1)$ with each application. The costs for applying these search operators is defined in the same way as for Ops_1.

6.2 Experiments and Results

We tested A* and RFDS on the arm with both sets of search operators, averaging results over a set of nine initial configurations $\theta_0^T = (x, y, -y)$ for $x \in \{-\pi, -2\pi/3, -\pi/3\}$ and $y \in \{-\pi/2, 0, +\pi/2\}$. Theorem 1 guaranteed that A* would find an optimal solution within either set of search operators. We ran RFDS at a variety of depths and with three different heuristic leaf evaluation functions: zero (RFDS-Z), roll-outs (RFDS-R), and a scaled version of L_{arm} (RFDS-S). Because we did not know an appropriate scaling factor for L_{arm}, for each starting configuration we initialized the scaling factor to zero and updated it as discussed in Section 5, using $\varepsilon = 0.01$. All of the RFDS runs under Ops_2 were guaranteed to generate solutions by Theorem 2. For the RFDS-R and RFDS-S runs with Ops_1, Theorem 3 provided the guarantee of producing solutions.

The results are shown in Figures 2 and 3, which report average solution cost, nodes expanded by the search, and the amount of virtual time for which the robot arm dynamics were simulated during the search. Simulated time is closely related to the number of nodes expanded, except that the RFDS-R figures also account for the time spent simulating the roll-out path to the goal. The actual CPU times used were more than an order of magnitude smaller than the simulated times, and our code has not been optimized for speed. The "C_1 only" row gives the average solution cost produced by controlling the arm using C_1 from every starting point. Achieving this level of performance requires no search at all. A* with either set of search operators found significantly lower cost solutions than those produced by C_1. The A* solution qualities with Ops_1 and Ops_2 are very close. We anticipated that Ops_2 results might not be as good, because the paths are constrained to always descend on L_{arm}, whereas Ops_1 allows the freedom of ascending. For this problem, the effect is small. A* with Ops_2 involved significantly less search effort than with Ops_1, in terms of both the number of expanded nodes and the amount of arm simulation time required.

The RFDS-Z – Ops_1 combination is the only one not guaranteed to produce a solution, and indeed, it did not. Under Ops_2, RFDS-Z is guaranteed to produce a path to a goal for any search depth. At low search depths, the solutions it produced were worse than those produced by C_1. However, at greater depth, it found significantly better solutions, and with much less search effort than required by A*.

RFDS-R with either set of search operators produced excellent solutions at depth one. This result is quite surprising. Apparently the roll-out yields excellent information for distinguishing good search operators from bad ones. Another unexpected result is that greater search depths did not improve over the depth-one performance. Solution quality is slightly worse and search effort increased dramatically. At the higher search depths, RFDS-R required more arm simulation time than did A*. This primarily resulted from the heuristic leaf evaluations, each of which required the simulation of an entire trajectory to G.

RFDS-S produced good solutions with very little search effort, but greater search effort did not yield as good solutions as other algorithms were able to produce. RFDS-S seems to be the most appropriate algorithm for real-time search if very fast responses are needed. It produced solutions that significantly improved over those produced by C_1, using search effort that was less than one tenth of that of the shallowest roll-out runs, and less than one hundredth of the effort required by A*.

6.3 Discussion

The arm simulation results reflect some of the basic theory and expectations developed earlier in the paper. The algorithms that were guaranteed to find solutions did, and those that were not guaranteed to find solutions did not. These results depend, among other things, on our choice of the "duration" of a search operator, Δ. We ran the same set of experiments for four other choices of Δ: 1.0, 0.75, 0.5, and 0.1. Many of the qualitative results were similar, including the excellent performance of roll-outs with a depth-one search.

One difference was that for higher Δ, A* became more competitive with RFDS in terms of search effort. Con-

Algorithm	Sol'n Cost	Nodes Expanded	Sim. Time
C_1 only	435.0	0	0
A*	304.0	11757.4	14690.7
RFDS-Z depths 1-5	∞	∞	∞
RFDS-R depth 1	305.7	41.3	667.6
RFDS-R depth 2	307.1	158.6	2001.1
RFDS-R depth 3	307.1	477.9	5489.8
RFDS-R depth 4	307.1	1299.0	13214.6
RFDS-R depth 5	307.1	3199.6	27785.5
RFDS-S depth 1	354.3	32.9	49.0
RFDS-S depth 2	371.0	163.2	210.7
RFDS-S depth 3	359.4	675.3	848.8
RFDS-S depth 4	343.4	2669.4	3337.6
RFDS-S depth 5	334.4	9565.3	11952.1

Figure 2: Ops_1 Results

Algorithm	Sol'n Cost	Nodes Expanded	Sim. Time
C_1 only	435.0	0	0
A*	305.0	4223.3	3492.8
RFDS-Z depth 1	502.8	45.0	40.6
RFDS-Z depth 2	447.8	113.2	94.7
RFDS-Z depth 3	385.2	226.8	192.9
RFDS-Z depth 4	360.9	434.9	374.0
RFDS-Z depth 5	326.7	743.6	648.0
RFDS-R depth 1	306.4	38.2	479.8
RFDS-R depth 2	307.8	118.7	1424.1
RFDS-R depth 3	307.8	333.1	4047.0
RFDS-R depth 4	307.8	920.7	10085.3
RFDS-R depth 5	307.8	2359.6	21821.6
RFDS-S depth 1	355.3	33.0	28.2
RFDS-S depth 2	370.7	105.1	87.9
RFDS-S depth 3	359.2	324.3	284.3
RFDS-S depth 4	342.9	1019.6	920.9
RFDS-S depth 5	330.3	3011.4	2758.3

Figure 3: Ops_2 Results

versely, at $\Delta = 0.1$, A* exhausted the computer's memory and crashed. The complexity of A* generally grows exponentially with the length of the optimal solution. For RFDS, which iteratively constructs a solution, effort is linear in the length of the solution it produces. The complexity of RFDS is more governed by the search depth, which can be chosen independently. Another difference observed at $\Delta = 1.0$ is that RFDS-Z succeeded in producing solutions from all starting positions when the search depth was four or higher.

For all algorithms, solution quality was lower for higher Δ. This is simply because lower Δ allows finer-grained control over the trajectory that the arm takes. The freedom to choose Δ suggests an alternative means for performing real-time search: one could perform a sequence of A* searches, initially with a high Δ and then decrease Δ to find solutions at finer temporal resolutions.

7 Conclusion

We demonstrated how domain knowledge in the form of a Lyapunov function can help one analyze and formulate infinite-state search problems. Lyapunov methods guarantee the existence of solutions to a problem, and they can be used to show that both optimal and suboptimal/real-time search procedures will find those solutions. These results provide a theoretical basis for extending the range of problems to which heuristic search methods can be applied with a guarantee of results.

Acknowledgments

References

[Bertsekas *et al.*, 1997] D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu. Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 1997.

[Boone, 1997] G. Boone. Minimum-time control of the acrobot. In *1997 International Conference on Robotics and Automation*, pages 3281–3287, 1997.

[Connolly and Grupen, 1993] C. I. Connolly and R. A. Grupen. The applications of harmonic functions to robotics. *Journal of Robotics Systems*, 10(7):931–946, 1993.

[Craig, 1989] J. J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley, 1989.

[Davies *et al.*, 1998] S. Davies, A. Ng, and A. Moore. Applying online search techniques to continuous-state reinforcement learning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 753–760, 1998.

[Korf, 1990] R. E. Korf. Real-time heuristic search. *Artificial Intelligence*, (42):189–211, 1990.

[Krstić *et al.*, 1995] M. Krstić, I. Kanellakopoulos, and P. Kokotović. *Nonlinear and Adaptive Control Design*. John Wiley & Sons, Inc., New York, 1995.

[Levine, 1996] W. S. Levine, editor. *The Control Handbook*. CRC Press, Inc., 1996.

[Pearl, 1984] J. Pearl. *Heuristics*. Addison Wesley Publishing Company, Inc., Reading, Massachusetts, 1984.

[Russell and Norvig, 1995] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1995.

[Tesauro and Galperin, 1996] G. Tesauro and G. R. Galperin. On-line policy improvement using monte-carlo search. In *Advances in Neural Information Processing: Proceedings of the Ninth Conference*. MIT Press, 1996.

[Vincent and Grantham, 1997] T. L. Vincent and W. J. Grantham. *Nonlinear and Optimal Control Systems*. John Wiley & Sons, Inc., New York, 1997.