# Easy and Hard Testbeds for Real-Time Search Algorithms*

**Sven Koenig** **Reid G. Simmons**
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891
{ skoenig, reids }@cs.cmu.edu

**Abstract**

Although researchers have studied which factors influence the behavior of traditional search algorithms, currently not much is known about how domain properties influence the performance of real-time search algorithms. In this paper we demonstrate, both theoretically and experimentally, that Eulerian state spaces (a superset of undirected state spaces) are very easy for some existing real-time search algorithms to solve: even real-time search algorithms that can be intractable, in general, are efficient for Eulerian state spaces. Because traditional real-time search testbeds (such as the eight puzzle and gridworlds) are Eulerian, they cannot be used to distinguish between efficient and inefficient real-time search algorithms. It follows that one has to use non-Eulerian domains to demonstrate the general superiority of a given algorithm. To this end, we present two classes of hard-to-search state spaces and demonstrate the performance of various real-time search algorithms on them.

## Introduction

Real-time heuristic search algorithms interleave search with action execution by limiting the amount of deliberation performed between actions. (Korf 1990) and (Korf 1993) demonstrated the power of real-time search algorithms, which often outperform more traditional search techniques. Empirical results for real-time search algorithms have typically been reported for domains such as

- sliding tile puzzles (such as the 8-puzzle) (Korf 1987; 1988; 1990; Russell & Wefald 1991; Knight 1993; Korf 1993; Ishida 1995) and
- gridworlds (Korf 1990; Ishida & Korf 1991; Ishida 1992; Pemberton & Korf 1992; Pirzadeh & Snyder 1990; Thrun 1992; Matsubara & Ishida 1994; Stentz 1995; Ishida 1995).

Such test domains permit comparisons between search algorithms. It is therefore important that the performance of real-time search algorithms in test domains be representative of their performance in the domains of interest: test domains should either reflect the properties of the domains that one is interested in or, at least, be representative of a wide range of domains. To this end, one has to understand how properties of state spaces affect the performance of real-time search algorithms.

Although researchers have studied which factors influence the performance of traditional search algorithms, such as A*, (Pearl 1985), not much is known about real-time search algorithms. We investigate two classes of state spaces: a state space is considered easy to search (Type 1) if no real-time search algorithm has a significant performance advantage over other (reasonable) algorithms, otherwise the state space is hard to search (Type 2). Our analysis of several uninformed real-time search algorithms with minimal lookahead that solve suboptimal search problems shows that Eulerian state spaces (each state has an equal number of actions that leave and enter the state) are all Type 1 – even real-time search algorithms that are inefficient, in general, can perform well in Eulerian state spaces. Since sliding tile puzzles and gridworlds are typically Eulerian, these domains are not appropriate for demonstrating how well real-time search algorithms perform, in general (note that the Eulerian property has no effect on the performance of traditional search algorithms). To remedy this, we propose two classes of non-Eulerian testbeds ("reset" and "quicksand" state spaces) that are of Type 2 and, thus, hard to search. Our empirical results on these testbeds demonstrate that they clearly distinguish efficient and inefficient real-time search algorithms.

## Real-Time Search Algorithms

We use the following notation to describe state spaces formally: $S$ denotes the finite set of states of the state space, $G$ with $\emptyset \neq G \subseteq S$ the non-empty set of goal states, and $s_{start} \in S$ the start state. $A(s)$ is the finite set of actions that can be executed in $s \in S$, and $succ(s, a)$ denotes the successor state that results from the execution of $a \in A(s)$ in $s \in S$. The size of the state space is $n := |S|$, and the total number of state-action pairs (loosely called actions) is $e := \sum_{s \in S} |A(s)|$. $gd(s)$

*This research is sponsored in part by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330.

denotes the goal distance of $s \in S$ (measured in action executions).

There exist state spaces in which all of our real-time search algorithms can get trapped in a part of the state space that does not contain a goal state. To exclude these state spaces, we assume that the state spaces are strongly connected. In this paper, for purposes of clarity, we also assume that $e \leq n^2$ (an extremely realistic assumption), since this allows us to state all complexity results in terms of $n$ only.

We study suboptimal search – the task that real-time search algorithms can perform very well. Suboptimal search means looking for any path (i.e., sequence of actions) from the start state to a goal state. In real-time search, the search time is (roughly) proportional to the length of the solution path. Thus, we use the path length to evaluate the performance of real-time search algorithms. When we refer to the complexity of a real-time search algorithm, we mean an upper bound on the total number of actions that it executes until it reaches a goal state, in big-O notation. This bound must hold for all possible topologies of state spaces of a given size, start and goal states, and tie breaking rules among indistinguishable actions.

To make meaningful comparisons, we study algorithms that make similar assumptions and restrict our attention to uninformed real-time search algorithms with minimal lookahead and greedy action selection.[1] Such algorithms maintain information in the form of integer values, $V(s, a)$, which are associated with every state-action pair $(s, a)$. An additional integer value is maintained across action executions in the variable *memory*. The semantics of these values depend on the specific real-time search algorithm used, but all values are zero-initialized, reflecting that the algorithms are initially uninformed. At no point in time can these values contain much information, since the algorithms must be able to decide quickly which actions to execute, and their decisions are based on these values. This requirement prevents the algorithms, for example, from encoding significant portions of the state space in these values.

The algorithms that we consider all fit the skeleton shown in Figure 1. They consist of a termination checking step (line 2), an action selection step (line 3), a value update step (line 4), and an action execution step (line 5). First, they check whether they have already reached a goal state and thus can terminate successfully (line 2). If not, they decide on the action to execute next (line 3). For this decision, they can consult the value stored in their memory and the values $V(s, a)$ associated with the actions in their current state $s$. Then, they update the value of this action and their memory, possibly also using the values associated with the actions in their new state (line 4). Finally, they execute the selected action (line 5) and iterate this procedure (line 6).

[1] A note of caution: While this restriction is convenient for studying properties of state spaces, it would be unfair to compare real-time search algorithms with each other solely on the basis of our study, since some algorithms are better in incorporating initial knowledge of the state space or allowing for larger lookaheads. We relax some of these assumptions in the penultimate section of the paper, where we discuss real-time search algorithms with larger lookaheads.

The real-time search algorithm starts in state $s_{start}$. Initially, $memory = 0$ and $V(s, a) = 0$ for all $s \in S$ and $a \in A(s)$.

1. $s :=$ the current state.
2. If $s \in G$, then stop successfully.
3. Choose an $a$ from $A(s)$ possibly using $memory$ and $V(s, a')$ for $a' \in A(s)$.
4. Update $memory$ and $V(s, a)$ possibly using $memory$, $V(s, a)$, and $V(succ(s, a), a')$ for $a' \in A(succ(s, a))$.
5. Execute action $a$, i.e. change the current state to $succ(s, a)$.
6. Go to 1.

Figure 1: Skeleton of the studied algorithms

## Theoretical Analysis

We first study the complexity of real-time search algorithms over all state spaces. In this case, one can freely choose the state space that maximizes the number of action executions of a given real-time search algorithm from all state spaces with the same number of states. Later, we restrict the possible choices and study the search complexity over a subset of all state spaces. In both cases, we are interested in the complexity of efficient and inefficient real-time search algorithms. The smaller the difference in the two complexities, the stronger the indication that search problems in such domains are of Type 1. (All proofs can be found in (Koenig & Simmons 1996b).)

### General State Spaces

In this section, we introduce a particular search algorithm (min-LRTA*) and compare its complexity in general state spaces to the most efficient and less efficient real-time search algorithms.

**LRTA*-Type Search** Korf's Learning Real-Time A* (LRTA*) algorithm (Korf 1990) is probably the most popular real-time search algorithm (Ishida 1995; Knight 1993; Koenig & Simmons 1995). The version we use here is closely related to Q-learning, a widely-used reinforcement learning method, see (Koenig & Simmons 1996a). We call it LRTA* with minimalistic lookahead

(min-LRTA*), because the search horizon of its action selection step is even smaller than that of LRTA* with lookahead one. (We analyze Korf's original version of LRTA* with lookahead one in the section on "Larger Lookaheads.")

The following table presents the action selection step (line 3) and value update step (line 4) of min-LRTA*. We use two operators with the following semantics: Given a set $X$, one-of $X$ returns one element of $X$ according to an arbitrary rule. $\arg\min_{x \in X} f(x)$ returns the set $\{x \in X : f(x) = \min_{x' \in X} f(x')\}$.

| Min-LRTA* | |
|---|---|
| line 3 | $a :=$ one-of $\arg\min_{a' \in A(s)} V(s, a')$ |
| line 4 | $V(s, a) := 1 + \min_{a' \in A(succ(s,a))} V(succ(s, a), a')$ |

The action selection step selects the state-action pair with the smallest value. The value update step replaces $V(s, a)$ with the more accurate lookahead value $1 + \min_{a' \in A(succ(s,a))} V(succ(s, a), a')$.

Min-LRTA* always reaches a goal state with a finite number of action executions. The following complexity result was proved in (Koenig & Simmons 1996a).

**Theorem 1** *Min-LRTA* has a tight complexity of $O(n^3)$ action executions.*

**Efficient Search Algorithms** No real-time search algorithm that fits our framework (Figure 1) can distinguish between actions that have not been executed, since it does not look at the successor states of its current state when choosing actions (and initially all actions have the same value). This implies the following lower bound on their complexity, which follows from a result in (Koenig & Simmons 1996a).

**Theorem 2** *The complexity of every real-time search algorithm that fits our real-time search skeleton is at least $O(n^3)$ action executions.*

Thus, no real-time search algorithm can beat min-LRTA*, since none can have a complexity smaller than $O(n^3)$.

**An Inefficient Search Algorithm** Particularly bad search algorithms are ones that do not remember were they have already searched. Random walks are examples of such search algorithms. We can easily derive a real-time search algorithm that shares many properties with random walks, but has finite complexity – basically, by "removing the randomness" from random walks.

| Edge Counting | |
|---|---|
| line 3 | $a :=$ one-of $\arg\min_{a' \in A(s)} V(s, a)$ |
| line 4 | $V(s, a) := 1 + V(s, a)$ |

Random walks execute all actions in a state equally often in the long run. The action selection step of edge counting always chooses the action that has been executed the least number of times. This achieves the same result as random walks, but in a deterministic way. One particular tie breaking rule, for example, is to execute all actions in turn. Shannon used this algorithm as early as in the late 1940's to implement an exploration behavior for an electronic mouse that searched a maze (Sutherland 1969). To the best of our knowledge, however, its relationship to random walks has never been pointed out, nor has its complexity been analyzed.

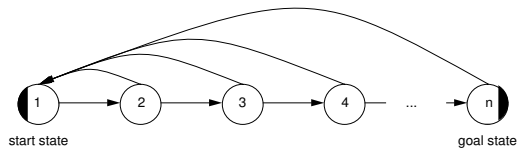


Figure 2: A reset state space

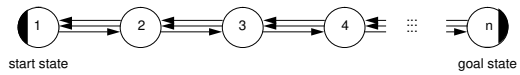


Figure 3: A quicksand state space

In (Koenig & Simmons 1996b), we prove that edge counting always reaches a goal state with a finite number of action executions, but its complexity can be exponential in the size of the state space.

**Theorem 3** *The complexity of edge counting is at least exponential in $n$.*

To demonstrate this, we present two classes of state spaces for which edge counting needs a number of action executions in the worst case that is exponential in $n$. These are Type 2 spaces since, by Theorem 1, in these domains min-LRTA* needs only a polynomial number of action executions.

- **Reset State Spaces**: A reset state space is one in which all states (but the start state) have an action that leads back to the start state (in general, the problem occurs if the "reset" actions are beyond the search horizon of the algorithm). For the reset state space in Figure 2, edge counting executes $3 \times 2^{n-2} - 2$ actions before it reaches the goal state (for $n \geq 2$) if ties are broken in favor of successor states with smaller numbers.

- **Quicksand State Spaces**: In every state of a quicksand state space, there are more actions that move the agent away from the goal than move it towards it. Quicksand state spaces differ from reset state spaces in the effort that is necessary to recover from mistakes: It is possible to recover in only one step in

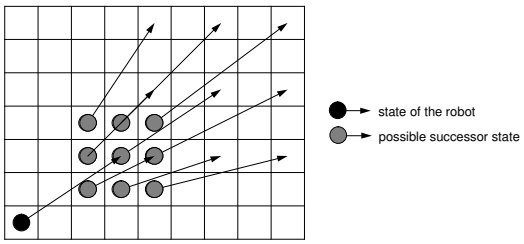


Figure 4: Racetrack domain

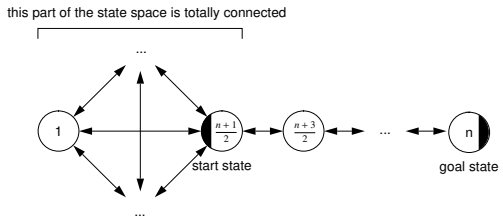


Figure 5: An undirected state space

quicksand state spaces. Nevertheless, quicksand state spaces can be hard to search. For the quicksand state space in Figure 3, edge counting executes $2^{n+1}-3n-1$ actions before it reaches the goal state (for $n \geq 1$) if ties are broken in favor of successor states with smaller numbers.

## Undirected and Eulerian State Spaces

In this section, we consider the complexity of real-time search algorithms in both undirected and Eulerian state spaces and show that they are all of Type 1.[2]

**Definition 1** *A state space is* **Eulerian** *iff* $|A(s)| = |\{(s', a') : s' \in S \wedge a' \in A(s') \wedge succ(s', a') = s\}|$ *for all* $s \in S$*, i.e. there are as many actions that leave a state as there are actions that enter the (same) state.*

Since an undirected edge is equivalent to one incoming and one outgoing edge, all undirected state spaces are Eulerian. Many domains typically used to test AI search algorithms are undirected (and thus Eulerian). Examples include sliding tile puzzles and gridworlds, where space is discretized into squares and movement is restricted to immediately adjacent squares. There also exist domains that are Eulerian, but not undirected, for example racetrack domains (Gardner 1973). They correspond to gridworlds, but are a bit more realistic robot navigation domains (Figure 4). A state of the state space is characterized not only by the X-Y square that the robot occupies, but also by its speed in both the X and Y directions. Actions correspond to adjusting both X and Y speed components by -1, 0, or 1 (within bounds). Given an action (speed change) the successor state is determined by computing the new speed components and determining the location of the robot by adding each speed component to its corresponding location component. Racetrack domains are Eulerian except around obstacles or at boundaries. In particular, an obstacle free racetrack domain on a torus is truly Eulerian. Race track domains have been used as testbeds for real-time search algorithms by (Barto, Bradtke, & Singh 1995).

We now show that Eulerian state spaces are easier to search with real-time search algorithms than state spaces in general, but undirected state spaces do not simplify the search any further.

**LRTA*-Type Search** The complexity of min-LRTA* does not decrease in undirected or Eulerian state spaces.

**Theorem 4** *Min-LRTA* has a tight complexity of* $O(n^3)$ *action executions in undirected or Eulerian state spaces.*

Figure 5 shows an example of an undirected (and thus Eulerian) state space for which min-LRTA* needs at least $O(n^3)$ action executions in the worst case in order to reach the goal state – it executes $(n^3+6n^2-3n-4)/16$ actions before it reaches the goal state (for $n \geq 1$ with $n \bmod 4 = 1$) if ties are broken in favor of successor states with smaller numbers (Koenig & Simmons 1992).

**An Efficient Search Algorithm** For Eulerian state spaces, real-time search algorithms do exist with lower complexity. One example, called BETA[3] ("Building a Eulerian Tour" Algorithm), informally acts as follows: "Take unexplored edges whenever possible. If all actions in the current state have been executed at least once, retrace the closed walk of unexplored edges just completed, stopping at nodes that have unexplored edges, and apply this algorithm recursively from each such node." This algorithm is similar to depth-first search, with the following difference: Since chronological back-

[2]Eulerian state spaces correspond to directed Euler(ian) graphs as defined by the Swiss mathematician Leonhard Euler when he considered whether the seven Königsberg bridges could be traversed without recrossing any of them (Newman 1953).

[3]The exact origin of the algorithm is unclear. (Deng & Papadimitriou 1990) and (Korach, Kutten, & Moran 1990) stated it explicitly as a search algorithm, but it has been used earlier as part of proofs about Eulerian tours (Hierholzer 1873).

tracking is not always possible in directed graphs, BETA repeats its first actions when it gets stuck instead of backtracking its latest actions.

BETA fits our real-time search skeleton if we interpret each integer value $V(s,a)$ as a triple: the first component of the triple (the "cycle number") corresponds to the level of recursion. The second component counts the number of times the action has already been executed, and the third component remembers when the action was executed first (using a counter that is incremented after every action execution). The variable *memory* is also treated as a triple: its first two components remember the first two components of the previously executed action and its third component is the counter. All values are initialized to $(0,0,0)$.

BETA

```
line 3   a                                          :=
         one-of arg min_{a'∈X} V(s,a')[3]
         where
         X = arg max_{a'∈Y} V(s,a')[1]
         and
         Y = arg min_{a'∈A(s)} V(s,a')[2]
line 4   if V(s,a)[2] = 0 then
           V(s,a)[3] := memory[3]+1
           if memory[2] = 1 then
             V(s,a)[1] := memory[1]
           else then
             V(s,a)[1] := memory[1]+1
         V(s,a)[2] := V(s,a)[2] + 1
         memory[1] := V(s,a)[1]
         memory[2] := V(s,a)[2]
         memory[3] := memory[3] + 1
```

BETA always reaches a goal state with a finite number of action executions and, moreover, executes every action at most twice. The following theorem follows.

**Theorem 5** *BETA has a tight complexity of $O(n^2)$ action executions in undirected or Eulerian state spaces.*

Furthermore, no real-time search algorithm that fits our real-time search skeleton can do better in Eulerian or undirected state spaces in the worst case (Koenig & Smirnov 1996).

**An Inefficient Search Algorithm** Although edge-counting is exponential, in general, its worst-case complexity decreases in undirected and Eulerian state spaces.

**Theorem 6** *Edge counting has a tight complexity of $O(n^3)$ action executions in undirected or Eulerian state spaces.*

To be precise: We can prove that the complexity of edge counting is tight at $e \times gd(s_{start}) - gd(s_{start})^2$ action executions in undirected or Eulerian state spaces. Figure 5 shows an example of an undirected (and thus Eulerian) state space for which edge counting needs at least $O(n^3)$ action executions in the worst case in order to reach the goal state – it executes $e \times gd(s_{start}) - gd(s_{start})^2 = (n^3 + n^2 - 5n + 3)/8$ actions before it reaches the goal state (for odd $n \geq 1$) if ties are broken in favor of successor states with smaller numbers.

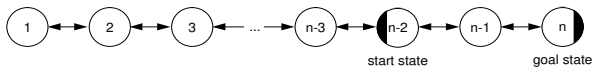


Figure 6: A linear state space

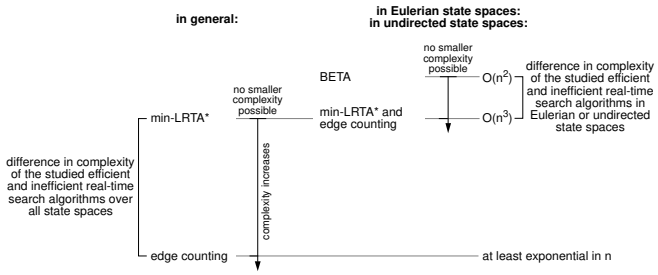


Figure 7: Diagram of worst-case performance results

Edge counting can have a better worst-case performance for a given search problem than min-LRTA*. An example is shown in Figure 6. Min-LRTA* executes $n^2 - 3n + 4$ actions in this undirected state space before it reaches the goal state (for $n \geq 3$) if ties are broken in favor of successor states with smaller numbers except for the first action execution in which the tie is broken in the opposite way. On the other hand, we have shown that edge counting is guaranteed not to need more than $e \times gd(s_{start}) - gd(s_{start})^2 = 4n - 8$ action executions in order to reach a goal state, which beats min-LRTA* for $n > 4$.

**Summary** When comparing the complexity of min-LRTA* with the complexities of efficient and inefficient real-time search algorithms, we derived the following results (Figure 7). In general, no real-time search algorithm can beat the complexity of min-LRTA*, which is a small polynomial in $n$. In contrast, the deterministic real-time search algorithm (edge counting) that we derived from random walks has a complexity that is at least exponential in $n$. The picture changes in Eulerian state spaces. The complexity of edge counting decreases dramatically and equals the complexity of min-LRTA*, which remains unchanged (it even beats min-LRTA* in certain specific domains). In addition, there exists a dedicated real-time search algorithm for Eulerian state spaces (BETA) that has a smaller complexity. All complexities remain the same in undirected state spaces, a subset of Eulerian state spaces.

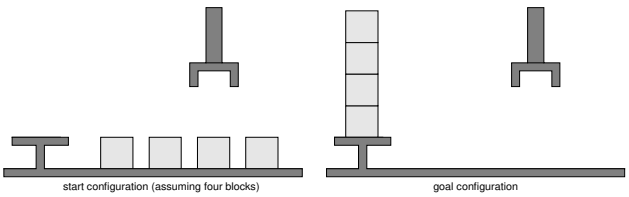


Figure 8: A simple blocksworld problem

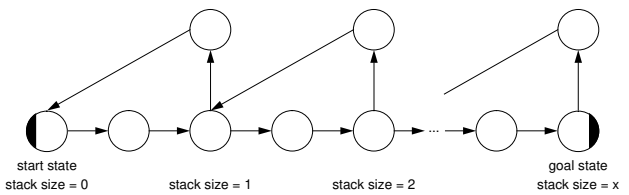


Figure 9: Domain 1

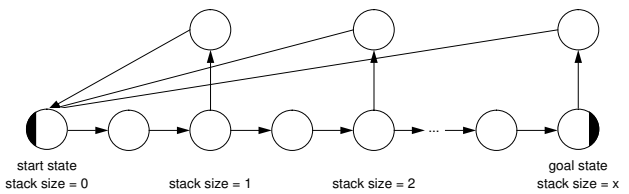


Figure 10: Domain 2

## Experimental Analysis

Although the theoretical analyses provide worst-case performance guarantees, they do not necessarily reflect average case performance. To show that the average-case performance follows a similar trend, we ran trials in two blocksworld domains, in which the start state consists of a set of $x$ indistinguishable blocks on a table, and the goal state has all the blocks stacked on top of one another on a platform (Figure 8). Domain 1 has four operators: "pickup block from table," "put block on stack," "pickup block from stack," and "put block on table." A block picked up from the table is always followed by a "put on stack," and a block picked up from the stack is always subsequently placed on the table. Domain 1 is Eulerian (Figure 9). Domain 2 has the same two pickup operators and the same "put block on stack" operator, but the "put block on table" operator (which always follows a "pickup block from stack" operator) knocks down the whole stack onto the table.

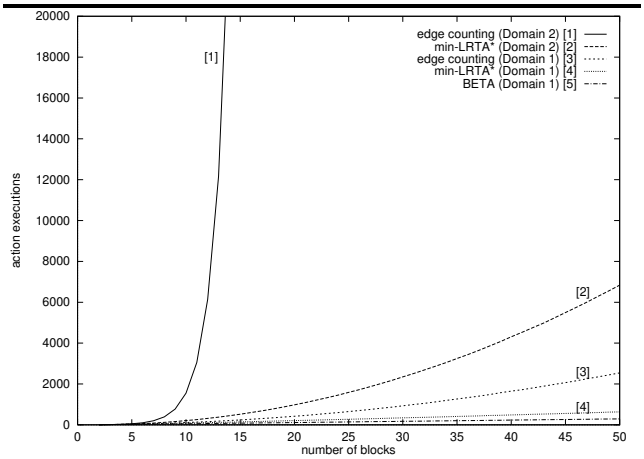


Figure 11: Performance results (blocksworld problem)

Domain 2 is a reset state space (Figure 10).

The experiments show that the relationship of the average-case performances are similar to those in the worst case. Figure 11 shows how many actions the real-time search algorithms execute in the two blocksworld domains. Note that the search algorithms are uninformed – in particular, they initially have no knowledge that putting blocks on the stack is the best way to achieve the goal state. The horizontal axis shows the size of the state space (measured by the number of blocks) and the vertical axis measures the number of actions executed until a goal state is reached from the start state. We averaged this over 5000 runs with randomly broken ties.

Every algorithm does better in Domain 1 than in Domain 2. Edge counting quickly becomes intractable in Domain 2. With 50 blocks, for example, edge counting needs about $1.7 \times 10^{15}$ (estimated) action executions, on average, in order to reach the goal state and thus performs about 250 billion times worse than min-LRTA*. On the other hand, all algorithms do quite well in Domain 1. With 50 blocks, for example, min-LRTA* performs 2.2 times worse than BETA and edge counting performs only 8.7 times worse. Thus, the interval spanned by the average-case complexity of efficient and inefficient real-time search algorithms is much smaller in Domain 1 than in Domain 2. This difference is to be expected, since Domain 1 is Eulerian (and thus of Type 1), whereas Domain 2 resembles a reset state space of Type 2.

If we change the start state in both domains so that all but four blocks are already stacked initially, then both domains become easier to solve. However, the performance relationships in Domain 2 remain simi-

lar, whereas the performance relationships in Domain 1 change dramatically. With 50 blocks, for example, min-LRTA* now performs 1.3 times worse than BETA, but edge counting performs 3.8 times *better* than BETA. Thus, for this search problem in a Eulerian state space, edge-counting (a real-time search algorithm that can be intractable) outperforms min-LRTA* (a real-time search algorithm that is always efficient).

## Larger Lookaheads

Some of our results also transfer to real-time search algorithms with larger lookaheads. In the following, we discuss node counting, a variant of edge counting, and the original 1-step LRTA* algorithm, a variant of min-LRTA*. Both algorithms have been used in the literature and have a larger lookahead than their relatives.

Node counting differs from edge counting in that it looks at the successor states of its current state when choosing actions.

| Node Counting | |
|---|---|
| line 3 | $a :=$ one-of $\arg\min_{a' \in A(s)} \sum_{a'' \in A(succ(s,a'))} V(succ(s,a'),a'')$ |
| line 4 | $V(s,a) := 1 + V(s,a)$ |

The action selection step always executes the action that leads to the successor state that has been visited the least number of times. Note that, in an actual implementation, one would maintain only one value $V(s)$ for each state $s$ with $V(s) = \sum_{a \in A(s)} V(s,a)$. Initially, $V(s) = 0$ for all $s \in S$.

| Node Counting | |
|---|---|
| line 3 | $a :=$ one-of $\arg\min_{a' \in A(s)} V(succ(s,a'))$ |
| line 4 | $V(s) := 1 + V(s)$ |

Korf's original LRTA* algorithm with lookahead one (1-step LRTA*) is similar to node-counting in that it looks at the successor states of its current state when choosing actions, but it has a different value update step (line 4).

| 1-Step LRTA* | |
|---|---|
| line 3 | $a :=$ one-of $\arg\min_{a' \in A(s)} V(succ(s,a'))$ |
| line 4 | $V(s) := 1 + V(succ(s,a))$ |

Korf showed that 1-step LRTA* always reaches a goal state with a finite number of action executions. (Koenig & Simmons 1995) showed that its complexity is tight at $n^2 - n$ and remains tight at $O(n^2)$ for undirected or Eulerian state spaces.

We can show that node counting is similar to edge counting in that there are state spaces for which its complexity is at least exponential in $n$. In particular, in our blocksworld domains, the appearance of the intermediate "pickup" operators makes it so that a 1-step lookahead is insufficient to avoid the reset traps. Furthermore, in these domains node counting and edge counting behave identically: they are efficient in Domain 1, but are both exponential in Domain 2, if ties are broken appropriately. Although we are not aware of any complexity analysis for node counting in undirected or Eulerian state spaces, variations of node counting have been used independently in (Pirzadeh & Snyder 1990) and (Thrun 1992) for exploring unknown gridworlds, in both cases with great success. Our experiments confirm these results. In one experiment, we compared node counting and 1-step LRTA* on an empty gridworld of size 50 times 50. We averaged their run-times (measured in action executions needed to get from the start state to the upper left square) over 25000 runs with randomly broken ties. The same 25000 randomly selected start states were used in both cases. Node counting needed, on average, 2874 action executions to reach the goal state, compared to 2830 action executions needed by 1-step LRTA*. Out of the 25000 runs, node counting outperformed 1-step LRTA* 12345 times, was beaten 12621 times, and tied 34 times. Nearly similar results were obtained in experiments with the eight-puzzle – the average performance of both algorithms was nearly identical, and each beat the other about the same number of times.

Thus, 1-step LRTA* and node counting were almost equally efficient on both gridworlds and sliding tile puzzles, but reset and quicksand state spaces are able to differentiate between them. Similar reset and quicksand state spaces can also be constructed for real-time search algorithms with even larger look-aheads.

## Conclusion

This paper presented properties of state spaces that make them easy, or hard, to search with real-time search algorithms. The goal was to separate the inherent complexity of a given search problem from the performance of individual real-time search algorithms. Our approach was to compare several uninformed real-time search algorithms with minimal lookahead that solve suboptimal search problems – all algorithms had previously been used by different researchers in different contexts. More precisely, we compared versions of LRTA* to efficient real-time search algorithms (such as BETA) and – equally importantly – inefficient real-time search algorithms (such as edge counting). We demonstrated, both theoretically and experimentally, that the performance characteristics of the studied real-time search algorithms can differ significantly in Eulerian and non-Eulerian state spaces (real-time search algorithms differ in this respect from traditional search algorithms such as A*). We have shown that real-time search algorithms that can be intractable in non-Eulerian state spaces (such as edge counting) have a small complexity in Eulerian and undirected state spaces. This result helps explain why the reported performance of real-time search algorithms have been so good: They tended to

be tested in Eulerian (usually undirected) domains.

Many state spaces, however, are not undirected or Eulerian. One way to avoid uncritical generalizations of performance figures for real-time search algorithms by non-experts is to report experimental results not only for Eulerian state spaces (such as sliding tile puzzles and gridworlds), but also for non-Eulerian state spaces. In particular, one has to use non-Eulerian state spaces to show the superiority of a particular real-time search algorithm across a wide range of domains. To this end, we presented two classes of hard-to-search state spaces ("reset" and "quicksand" state spaces) that do not suffer from (all of) the problems of the standard test domains. Minor variations of these state spaces are also applicable in distinguishing real-time search algorithms that have larger lookahead. We therefore suggest that variations of these two state spaces be included in test suites for real-time search algorithms.

Our study provides a first step in the direction of understanding what makes domains easy to solve with real-time search algorithms. In this paper, we reported results for one particular property: being Eulerian. Our current work concentrates on identifying and studying additional properties that occur in more realistic applications, such as real-time control.

## References


Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 73(1):81–138.

Deng, X., and Papadimitriou, C. 1990. Exploring an unknown graph. In *Proceedings of the Symposium on Foundations of Computer Science*, 355–361.

Gardner, M. 1973. Mathematical games. *Scientific American* 228(1):108–115.

Hierholzer, C. 1873. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen* 6:30–32.

Ishida, T., and Korf, R. 1991. Moving target search. In *Proceedings of the IJCAI*, 204–210.

Ishida, T. 1992. Moving target search with intelligence. In *Proceedings of the AAAI*, 525–532.

Ishida, T. 1995. Two is not always better than one: Experiences in real-time bidirectional search. In *Proceedings of the International Conference on Multi-Agent Systems*, 185–192.

Knight, K. 1993. Are many reactive agents better than a few deliberative ones? In *Proceedings of the IJCAI*, 432–437.

Koenig, S., and Simmons, R. 1992. Complexity analysis of real-time reinforcement learning applied to finding shortest paths in deterministic domains. Technical Report CMU–CS–93–106, School of Computer Science, Carnegie Mellon University.

Koenig, S., and Simmons, R. 1995. Real-time search in non-deterministic domains. In *Proceedings of the IJCAI*, 1660–1667.

Koenig, S., and Simmons, R. 1996a. The effect of representation and knowledge on goal-directed exploration with reinforcement learning algorithms. *Machine Learning Journal* 22:227–250.

Koenig, S., and Simmons, R. 1996b. The influence of domain properties on the performance of real-time search algorithms. Technical Report CMU-CS-96-115, School of Computer Science, Carnegie Mellon University.

Koenig, S., and Smirnov, Y. 1996. Graph learning with a nearest neighbor approach. In *Proceedings of the Conference on Computational Learning Theory*.

Korach, E.; Kutten, S.; and Moran, S. 1990. A modular technique for the design of efficient distributed leader finding algorithms. *ACM Transactions on Programming Languages and Systems* 12(1):84–101.

Korf, R. 1987. Real-time heuristic search: First results. In *Proceedings of the AAAI*, 133–138.

Korf, R. 1988. Real-time heuristic search: New results. In *Proceedings of the AAAI*, 139–144.

Korf, R. 1990. Real-time heuristic search. *Artificial Intelligence* 42(2-3):189–211.

Korf, R. 1993. Linear-space best-first search. *Artificial Intelligence* 62(1):41–78.

Matsubara, S., and Ishida, T. 1994. Real-time planning by interleaving real-time search with subgoaling. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, 122–127.

Newman, J. 1953. Leonhard Euler and the Königsberg bridges. *Scientific American* 188(6):66–70.

Pearl, J. 1985. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Menlo Park, California: Addison-Wesley.

Pemberton, J., and Korf, R. 1992. Incremental path planning on graphs with cycles. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, 179–188.

Pirzadeh, A., and Snyder, W. 1990. A unified solution to coverage and search in explored and unexplored terrains using indirect control. In *International Conference on Robotics and Automation*, volume 3, 2113–2119.

Russell, S., and Wefald, E. 1991. *Do the Right Thing – Studies in Limited Rationality*. Cambridge, Massachusetts: The MIT Press.

Stentz, A. 1995. The focussed D* algorithm for real-time replanning. In *Proceedings of the IJCAI*, 1652–1659.

Sutherland, I. 1969. A method for solving arbitrary-wall mazes by computer. *IEEE Transactions on Computers* C–18(12):1092–1097.

Thrun, S. 1992. The role of exploration in learning control with neural networks. In White, D., and Sofge, D., eds., *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Florence, Kentucky: Van Nostrand Reinhold. 527–559.