Value-Update Rules for Real-Time Search

Sven Koenig
College of Computing
Georgia Institute of Technology
skoenig@cc.gatech.edu

Boleslaw Szymanski
Department of Computer Science
Rensselaer Polytechnic Institute
szymansk@cs.rpi.edu

Abstract

Real-time search methods have successfully been used to solve a large variety of search problems but their properties are largely unknown. In this paper, we study how existing real-time search methods scale up. We compare two real-time search methods that have been used successfully in the literature and differ only in the update rules of their values: Node Counting, a real-time search method that always moves to the successor state that it has visited the least number of times so far, and Learning Real-Time A*, a similar real-time search method. Both real-time search methods seemed to perform equally well in many standard domains from artificial intelligence. Our formal analysis is therefore surprising. We show that the performance of Node Counting can be exponential in the number of states even in undirected domains. This solves an open problem and shows that the two real-time search methods do not always perform similarly in undirected domains since the performance of Learning Real-Time A* is known to be polynomial in the number of states at worst.

Traditional search methods from artificial intelligence, such as the A* method (Nilsson 1971), first plan and then execute the resulting plan. Real-time (heuristic) search methods (Korf 1987), on the other hand, interleave planning and plan execution, and allow for fine-grained control over how much planning to perform between plan executions. Planning is done via local searches, that is, searches that are restricted to the part of the domain around the current state of the agent. The idea behind this search methodology is not to attempt to find plans with minimal plan-execution time but rather to attempt to decrease the planning time or the sum of planning and plan-execution time over that of traditional search methods. (Ishida 1997) gives a good overview of real-time search methods. Experimental evidence indicates that real-time search methods are efficient domain-independent search methods that outperform traditional search methods on a variety of search problems. Real-time search methods have, for example, successfully been applied to traditional search problems (Korf 1990), moving-target search problems (Ishida and Korf 1991), STRIPS-type planning problems (Bonet *et al.* 1997), robot navigation and localization problems with initial pose uncertainty (Koenig and Simmons 1998), totally observable Markov decision process problems (Barto *et al.* 1995), and partially observable Markov decision process problems (Geffner and Bonet 1998), among others. Despite this success of real-time search methods, not much is known about their properties. They differ in this respect from traditional search methods, whose properties have been researched extensively. For example, real-time search methods associate values with the states that are updated as the search progresses and used to determine which actions to execute. Different real-time search methods update these values differently, and no consensus has been reached so far on which value-update rule is best. Both (Russell and Wefald 1991) and (Pemberton and Korf 1992), for example, studied several value-update rules experimentally but arrived at different conclusions about which one outperformed the others. This demonstrates the need to understand better how the value-update rules influence the behavior of real-time search methods. In this paper, we investigate how two value-update rules scale up in undirected domains: one that interprets the values as approximations of the goal distances of the states (resulting in a real-time search method called Learning Real-Time A*) and one that interprets the values as the number of times the states have been visited (resulting in a real-time search method called Node Counting). The principle behind Node Counting is simple: always move to the neighboring state that has been visited the least number of times. This appears to be an intuitive exploration principle since, when exploring unknown environments, one wants to get to states that one has visited smaller and smaller number of times with the goal to get as fast as possible to states that one has not visited yet. This explains why Node Counting has been used repeatedly in artificial intelligence. Experimental results indicate that both Node Counting and uninformed Learning Real-Time A* need about the same number of action executions on average to reach a goal state in many standard domains from artificial intelligence. However, to the best of our knowledge, our paper analyzes the performance of Node Counting in undirected domains for the first time, which is not surprising since the field of real-time search is a rather experimental one. We show that Node Counting reaches a goal state in undirected domains

```
Initially, the u-values $u(s)$ are zero for all $s \in S$.
1. $s := s_{start}$.
2. If $s \in G$, then stop successfully.
3. $a :=$ one-of $\arg\min_{a \in A(s)} u(succ(s,a))$.
4. Update $u(s)$ using the value-update rule.
5. Execute action $a$ and change the current state to $succ(s,a)$.
6. $s := succ(s,a)$.
7. Go to 2.
```

Figure 1: Real-Time Search

sometimes only after a number of action executions that is exponential in the number of states, whereas uninformed LRTA* is known to always reach a goal state after at most a polynomial number of action executions. Thus, although many standard domains from artificial intelligence (such as sliding-tile puzzles, blocksworlds, and gridworlds) are undirected, this property alone is not sufficient to explain why Node Counting performs well on them. This result solves an open problem described in (Koenig and Simmons 1996). We also describe a non-trivial domain property that guarantees a polynomial performance of Node Counting and study a probabilistic variant of Node Counting. In general, our results show that experimental comparisons of real-time search methods are often insufficient to evaluate how well they scale up because the performance of two similar real-time search methods can be very different even if experimental results seem to indicate otherwise. A formal analysis of real-time search methods can help to detect these problems and prevent surprises later on, as well as provide a solid theoretical foundation for interleaving planning and plan execution. We believe that it is important that more real-time search methods be analyzed similarly, especially since most work on real-time search has been of an experimental nature so far.

Notation

We use the following notation in this paper: S denotes the finite set of states of the domain, $s_{start} \in S$ the start state, and $\emptyset \neq G \subseteq S$ the set of goal states. The number of states is $n := |S|$. $A(s) \neq \emptyset$ is the finite, nonempty set of actions that can be executed in state $s \in S$. $succ(s,a)$ denotes the successor state that results from the execution of action $a \in A(s)$ in state $s \in S$. To simplify matters, we measure the plan-execution times and goal distances in action executions throughout this paper, which is justified if the execution times of all actions are roughly the same. We also use two operators with the following semantics: Given a set X, the expression "one-of X" returns an element of X according to an arbitrary rule. A subsequent invocation of "one-of X" can return the same or a different element. The expression "arg $\min_{x \in X} f(x)$" returns the elements $x \in X$ that minimize $f(x)$, that is, the set $\{x \in X | f(x) = \min_{x' \in X} f(x')\}$.

Node Counting and LRTA*

We study two similar real-time search methods, namely uninformed variants of Node Counting and Learning Real-Time A* that have a lookahead of only one action execution and fit the algorithmic skeleton shown in Figure 1. (Some researchers feel more comfortable referring to these methods as "agent-centered search methods" (Koenig 1995) and reserving the term "real-time search methods" for agent-centered search methods whose values approximate the goal-distances of the states.) We chose to study uninformed real-time search methods because one of the methods we study has traditionally been used for exploring unknown environments in the absence of heuristic information (often in the context of robot navigation). Both real-time search methods associate a *u-value* $u(s)$ with each state $s \in S$. The semantics of the u-values depend on the real-time search method but all u-values are initialized with zeroes, reflecting that the real-time search methods are initially uninformed and thus do not have any a-priori information as to where the goal states are. The search task is to find any path from the start state to a goal state, not necessarily a shortest one. Both real-time search methods first check whether they have already reached a goal state and thus can terminate successfully (Line 2). If not, they decide on which action to execute in the current state (Line 3). They look one action execution ahead and always greedily choose an action that leads to a successor state with a minimal u-value (ties are broken arbitrarily). Then, they update the u-value of their current state using a value-update rule that depends on the semantics of the u-values and thus the real-time search method (Line 4). Finally, they execute the selected action (Line 5), update the current state (Line 6), and iterate the procedure (Line 7). Many real-time search methods from the literature fit this algorithmic skeleton. We chose to compare Node Counting and Learning Real-Time A* because both of them implement simple, intuitive rules of thumb for how to interleave planning and plan execution. Node Counting can be described as follows:

Node Counting: A u-value $u(s)$ of Node Counting corresponds to the number of times Node Counting has already been in state s. Node Counting always moves to a successor state with a minimal u-value because it wants to get to states which it has visited a smaller number of times to eventually reach a state that it has not yet visited at all, that is, a potential goal state.

Value-Update Rule of Node Counting (Line 4 in Figure 1)
$u(s) := 1 + u(s)$.

To the best of our knowledge, the term "Node Counting" was first used in (Thrun 1992b). Variants of Node Counting have been used in the literature to explore unknown grid-worlds, either on their own (Pirzadeh and Snyder 1990) or to accelerate reinforcement-learning methods (Thrun 1992b). Node Counting is also the foundation of "Avoiding the Past: A Simple but Effective Strategy for Reactive [Robot] Navigation" (Balch and Arkin 1993), except that "Avoiding the Past" is part of a schemata-based navigation architecture and thus sums over vectors that point away from adjacent locations with a magnitude that depends on how often that location has been visited. Finally, it has been suggested that variants of Node Counting approximate the exploration behavior of ants, that use pheromone traces to guide their exploration (Wagner *et al.* 1997). We compare Node Counting to Learning Real-Time A* (Korf 1990), that can be described as follows:

> **Learning Real-Time A*** (LRTA*): A u-value $u(s)$ of LRTA* approximates the goal distance of state s. LRTA* always moves to a successor state with a minimal u-value because it wants to get to states with smaller goal distances to eventually reach a state with goal distance zero, that is, a goal state.

Value-Update Rule of LRTA* (Line 4 in Figure 1)
$u(s) := 1 + u(succ(s,a))$.

LRTA* is probably the most popular real-time search method, and thus provides a good baseline for evaluating the performance of other real-time search methods.

Assumptions

We assume that one can reach a goal state from every state that can be reached from the start state. Domains with this property are called safely explorable and guarantee that real-time search methods such as Node Counting or LRTA* reach a goal state eventually. This can be shown by assuming that they cycle infinitely without reaching a goal state. The u-values of all states in the cycle then increase beyond every bound since both methods increase the smallest u-value of the states in the cycle by at least one every time the cycle is traversed. But then the u-values of all states in the cycle increase above the u-values of all states that border the cycle. Such states exist since the domain is safely explorable and one can thus reach a goal state from every state in the cycle. Then, however, Node Counting and LRTA* are forced to move to this state and leave the cycle, which is a contradiction.

Performance of Node Counting

The performance of search methods for search tasks that are to be solved only once is best measured using the sum of planning and plan-execution time. We measure the performance of real-time search methods using the number of actions that they execute until they reach a goal state. This is motived by the fact that, for sufficiently fast moving agents, the sum of planning and plan execution time is determined by the planning time, which is roughly proportional to the number of action executions since the real-time search methods perform roughly a constant amount of computations between action executions. For sufficiently slowly moving agents, on the other hand, the sum of planning and plan-execution time is determined by the plan-execution time, which is roughly proportional to the number of action executions if every action can be executed in about the same amount of time.

The performance of LRTA* is known to be at worst quadratic in the number of states in both directed and undirected domains (Koenig and Simmons 1995). The performance of Node Counting is known to be at least exponential in the number of states in directed domains (Koenig and Simmons 1996) but many domains of artificial intelligence are undirected, including sliding-tile puzzles, blocksworlds, and gridworlds. To the best of our knowledge, the performance of Node Counting in undirected domains has not been analyzed so far. However, it has been speculated, based on the good experimental results reported by the researchers who used Node Counting, that Node Counting was efficient in undirected domains, see (Koenig and Simmons 1996) and the references contained therein for experimental results.

In the following, we present an example (our main result) that shows that the performance of Node Counting in undirected domains can be at least exponential in the number of states. Despite its elegance, this example proved to be rather difficult to construct. It refutes the hypothesis that the performance of Node Counting in undirected domains is at most polynomial in the number of states. Our examples are undirected trees (and thus planar graphs). Figure 2 shows an instance of this example. In general, the trees have $m + 1$ levels (for $m \geq 2$). The levels consist of vertices of three different kinds: g-subroots, r-subroots, and leafs that are connected to the subroots. g-subroots and r-subroots alternate. At level $i = 0$, there is one subroot, namely a g-subroot g_0. At levels $i = 1 \ldots m$, there are two subroots, namely an r-subroot r_i and a g-subroot g_i. Subroot g_i has $m + i$ leafs connected to it, and subroot r_i has one leaf connected to it. Finally, subroot g_m is connected to two additional vertices, namely the start vertex and the only goal vertex. The trees have $n = 3/2\ m^2 + 9/2\ m + 3$ vertices. Node Counting proceeds in a series of passes through the trees. Each pass traverses the subroots in the opposite order than the previous pass. We call a pass that traverses the subroots in descending order a down pass, and a pass that traverses them in ascending order an up pass. We number passes from zero on upward, so even passes are down passes and odd passes are up passes. A pass ends immediately before it switches directions. We break ties as follows: During pass zero, ties among successor states are broken in favor of leaf vertices of g-subroots (with highest priority) and then subroots. Pass zero ends when the leafs of subroot g_0 have been visited once each. As a result, at the end of pass zero the subroots g_i have been visited $m + i + 1$ times each, and their leafs have been visited once each. The subroots r_i have been visited once each, and their subroots have not been visited at all. During all subsequent passes, ties among successor states are broken in favor of subroots whenever possible. A tie between two r-subroots (when Node Counting is at a g-subroot) is resolved by continuing with the current pass. A tie between two g-subroots (when Node Counting is at an r-subroot) is resolved by terminating the current pass and starting a new one in the opposite direction. In the following, we provide a sketch of the proof that Node Counting executes (at least)

$$\Omega(n^{\sqrt{(\frac{1}{6}-\epsilon)n}})$$

actions in this case, where n is the number of states and $0 < \epsilon < 1/6$ is an arbitrarily small constant. The actual proofs are by induction, and are omitted here because they are long

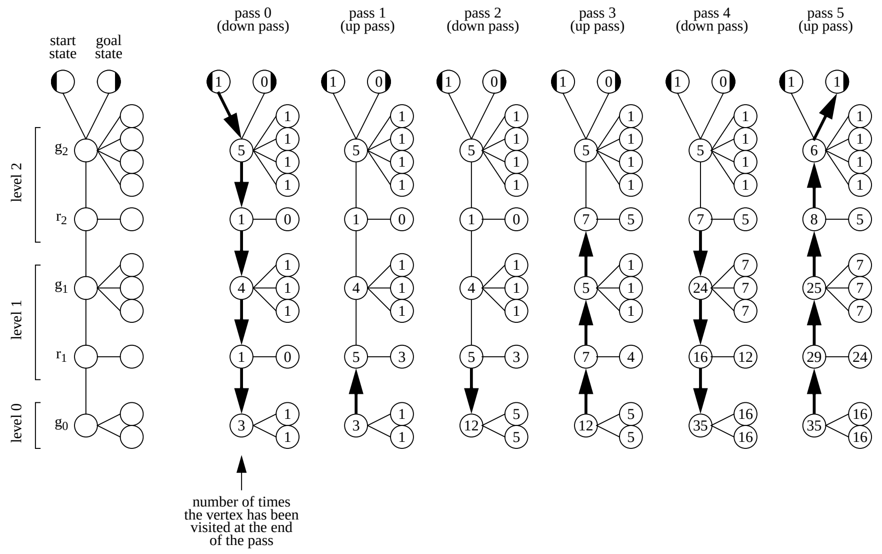

Figure 2: Node Counting has Exponential Runtime in Undirected Domains ($m = 2, n = 18$)

and tedious (Szymanski and Koenig 1998). To convince the reader of the correctness of our proofs, however, we provide experimental results from a simulation study that confirm our analytical results.

As part of the proof sketch, we study a tree with $m + 1$ levels. Let $u_{p,m}(s)$ denote the total number of times that subroot s has been entered at the end of pass p for a tree with $m+1$ levels. By definition, $u_{p,m}(s)$ is a nondecreasing function of p. Our tie breaking rules guarantee that all leafs of a subroot have been entered the same number of times at the end of each pass, so we let $w_{p,m}(s)$ denote the number of times each of the leafs of subroot s has been entered at the end of pass p for a tree with $m + 1$ levels. Finally, let $x_{p,m}(s)$ denote the total number of times subroot s has been entered from non-leafs at the end of pass p for a tree with $m + 1$ levels. These values relate as follows: The total number of times that a subroot was entered at the end of pass p is equal to the product of the number of its leafs and the total number of times that it was entered from each of its leafs at the end of pass p (which equals the total number of times that each of its leafs was entered at the end of pass p) plus the total number of times the subroot was entered from non-leafs at the end of pass p. For example, $u_{p,m}(g_i) = (m + i)w_{p,m}(g_i) + x_{p,m}(g_i)$.

Lemma 1 *Assume that Node Counting visits subroot s (with $s \neq g_m$) during pass p, where $0 < p < 2m + 2$. The values $u_{p,m}(s)$ can then be calculated as follows, for $i > 0$:*

$$\begin{aligned}
u_{2k+1,m}(g_0) &= m\, u_{2k,m}(r_1) + x_{2k,m}(g_0) \\
u_{2k,m}(g_0) &= m\, u_{2k,m}(r_1) + x_{2k,m}(g_0)
\end{aligned}$$

$$\begin{aligned}
u_{2k,m}(g_i) &= (m + i)\min(u_{2k-1,m}(r_i), u_{2k,m}(r_{i+1})) + x_{2k,m}(g_i) \\
u_{2k+1,m}(g_i) &= (m + i)\min(u_{2k,m}(r_{i+1}), u_{2k+1,m}(r_i)) + x_{2k+1,m}(g_i) \\
u_{2k,m}(r_i) &= \min(u_{2k-1,m}(g_{i-1}), u_{2k,m}(g_i)) + x_{2k,m}(r_i) \\
u_{2k+1,m}(r_i) &= \min(u_{2k,m}(g_i), u_{2k+1,m}(g_{i-1})) + x_{2k+1,m}(r_i)
\end{aligned}$$

Proof: by induction on the number of passes p. ■

Theorem 1 *If $p = 2k$ for $0 \leq k \leq m$, then the down pass ends at subroot g_0 and it holds that*

$$u_{2k,m}(g_i) = \begin{cases} m(u_{2k-2,m}(g_i) + 2k) + k + 1 & \text{for } i = 0 < k \\ (m + i)(u_{2k-2,m}(g_i) + 2k - 2i & \\ \quad +1) + 2k - 2i + 1 & \text{for } 0 < i < k \\ m + i + 1 & \text{otherwise} \end{cases}$$

$$u_{2k,m}(r_i) = \begin{cases} u_{2k-1,m}(g_{i-1}) + 2k - 2i + 2 & \text{for } 0 < i \leq k \\ 1 & \text{otherwise} \end{cases}$$

$$x_{2k,m}(g_i) = \begin{cases} k + 1 & \text{for } i = 0 \\ 2k - 2i + 1 & \text{for } 0 < i < k \\ 1 & \text{otherwise} \end{cases}$$

$$x_{2k,m}(r_i) = \begin{cases} 2k - 2i + 2 & \text{for } 0 < i \leq k \\ 1 & \text{otherwise} \end{cases}$$

$$\begin{aligned}
u_{2k,m}(r_i) &\geq u_{2k-1,m}(r_{i-1}) && \text{for } 1 < i \leq k \\
u_{2k,m}(g_i) &> u_{2k-1,m}(g_{i-1}) && \text{for } 0 < i < k
\end{aligned}$$

If $p = 2k + 1$ for $0 \leq k \leq m$, then the up pass ends at subroot r_{k+1} (with the exception of up pass $2m + 1$ that ends at the goal state) and it holds that

$$u_{2k+1,m}(g_i) = \begin{cases} u_{2k,m}(g_i) & \text{for } i = 0 \\ (m + i)(u_{2k-1,m}(g_i) + 2k & \\ \quad -2i) + 2k - 2i + 2 & \text{for } 0 < i < k \\ m + i + 2 & \text{for } 0 < i = k \\ m + i + 1 & \text{otherwise} \end{cases}$$

$$u_{2k+1,m}(r_i) = \begin{cases} u_{2k,m}(g_i) + 2k - 2i + 3 & \text{for } 0 < i \leq k \\ u_{2k+1,m}(g_{i-1}) + 2 & \text{for } i = k + 1 \leq m \\ 1 & \text{otherwise} \end{cases}$$

$$x_{2k+1,m}(g_i) = \begin{cases} k + 1 & \text{for } i = 0 \\ 2k - 2i + 2 & \text{for } 0 < i \leq k \\ 1 & \text{otherwise} \end{cases}$$

$$x_{2k+1,m}(r_i) = \begin{cases} 2k - 2i + 3 & \text{for } 0 < i \leq k \\ 2 & \text{for } i = k + 1 \leq m \\ 1 & \text{otherwise} \end{cases}$$

$$\begin{aligned}
u_{2k+1,m}(r_i) &\geq u_{2k,m}(r_{i+1}) && \text{for } 0 < i \leq k \\
u_{2k+1,m}(g_i) &> u_{2k,m}(g_{i+1}) && \text{for } 0 \leq i < k
\end{aligned}$$

Proof: by induction on the number of executed actions, using Lemma 1. ■

Thus, there are $2m+2$ passes before Node Counting reaches the goal vertex. Each down pass ends at subroot g_0. The final up pass ends at the goal state. All other up passes p end at subroot $r_{(p+1)/2}$. In the following, we need two inequalities. First, according Theorem 1, it holds for $0 \leq k \leq m$ that

$$u_{2k,m}(g_0) \quad = \quad \begin{cases} m+1 & \text{for } k=0 \\ m(u_{2k-2,m}(g_0)+2k)+k+1 & \text{otherwise.} \end{cases}$$

Solving the recursion yields

$$u_{2k,m}(g_0) \quad = \quad \frac{m^{k+3}+m^{k+2}+m^{k+1}-(2k+2)m^2+(k-2)m+k+1}{m^2-2m+1}.$$

Setting $k=m$ in this formula results in

$$u_{2m,m}(g_0) \quad = \quad \frac{m^{m+3}+m^{m+2}+m^{m+1}-2m^3-m^2-m+1}{m^2-2m+1}.$$

This implies that $u_{2m+1,m}(g_0) = u_{2m,m}(g_0) > m^m$.

Second, consider an arbitrary constant $0 < \epsilon < 1/6$ and assume that $m > \max\left(\frac{1}{\epsilon} - 4, \left(\frac{3}{2-8\epsilon}\right)^{1/\epsilon}\right) \geq 2$. Note that $n \geq m$ for our trees. Then,

$$\begin{array}{lll} n & = & \frac{3}{2}m^2 + \frac{9}{2}m + 3 \\ & < & \frac{3}{2}\left(1+\frac{4}{m}\right)m^2 \qquad \text{since } m > 2 \\ & < & \frac{3}{2}\left(1+\frac{4\epsilon}{1-4\epsilon}\right)m^2 \qquad \text{since } m > \frac{1}{\epsilon} - 4 \\ & = & \frac{3}{2}\frac{m^2}{1-4\epsilon} \end{array}$$

and thus $m > \sqrt{\frac{2}{3}n(1-4\epsilon)} > 1$ (A). In the following, we also utilize that $(an)^k > n^{(1-\epsilon)k}$ for $n > \left(\frac{1}{a}\right)^{1/\epsilon}$ and arbitrary constants $a > 0$ and $k > 0$ (B). Then,

$$\begin{array}{lll} m^m & > & \left(\sqrt{\frac{2}{3}n(1-4\epsilon)}\right)^{\sqrt{\frac{2}{3}n(1-4\epsilon)}} \qquad \text{since (A)} \\ & = & \left(\left(\frac{2}{3}-\frac{8}{3}\epsilon\right)n\right)^{\frac{1}{2}\sqrt{\frac{2}{3}n(1-4\epsilon)}} \\ & > & n^{(1-\epsilon)\frac{1}{2}\sqrt{\frac{2}{3}n(1-4\epsilon)}} \qquad \text{since (B)} \\ & = & n^{\sqrt{(1-\epsilon)^2\frac{1}{6}n(1-4\epsilon)}} \\ & > & n^{\sqrt{(1-2\epsilon)\frac{1}{6}n(1-4\epsilon)}} \\ & = & n^{\sqrt{\left(\frac{1}{6}-\epsilon+\frac{4}{3}\epsilon^2\right)n}} \\ & > & n^{\sqrt{\left(\frac{1}{6}-\epsilon\right)n}} \end{array}$$

Now, let $0 < \epsilon < 1/6$ be an arbitrarily small constant. Using the two inequalities above, it holds that $u_{2m+1,m}(g_0) > m^m > n^{\sqrt{(\frac{1}{6}-\epsilon)n}}$ for $m > \max\left(\frac{1}{\epsilon} - 4, \left(\frac{3}{2-8\epsilon}\right)^{1/\epsilon}\right)$ and thus also for sufficiently large n. $u_{2m+1,m}(g_0)$ is the u-value of g_0 after Node Counting terminates on a tree with $m+1$ levels. This value equals

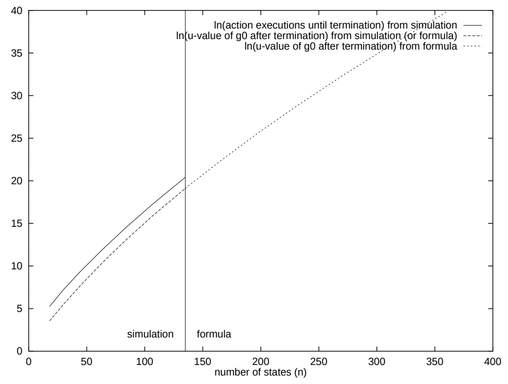

Figure 3: Results (Log Scale!)

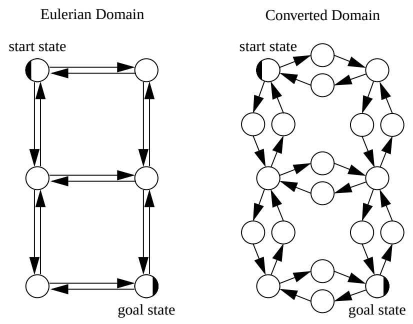

Figure 4: Sample Eulerian Domain and its Conversion

the number of times Node Counting has visited g_0, which is a lower bound on the number of actions it has executed. Consequently, Node Counting executes (at least)

$$\Omega(n^{\sqrt{(\frac{1}{6}-\epsilon)n}})$$

actions, and the performance of Node Counting can be exponential in the number of states even in undirected domains. To confirm this analytical result, we performed a simulation study of Node Counting on our trees, see Table 1. We stopped the simulation when m reached eight because the number of action executions and thus the simulation time became large. The simulation confirmed our formulas for how the number of states n, the number of passes until termination, and the u-value of g_0 after termination depend on m. The simulation also provided us with the number of action executions until termination, for which we do not have a formula in closed form, only a lower bound in form of the u-value of g_0 after termination. How the number of action executions and its lower bound relate is shown in Figure 3, that plots the *natural logarithms* of these values.

So far, we have shown that the performance of Node Counting is not guaranteed to be polynomial in the number of states in undirected domains. We are also able to describe a domain property that guarantees a polynomial performance of Node Counting. A Eulerian graph is a directed graph, each of whose vertices have an equal number of incoming and outgoing directed edges. (Undirected domains

m	number of states n	number of passes until termination	u-value of g_0 $u_{2m+1,m}(g_0)$ after termination	$\frac{\text{u-value of } g_0}{\text{number of states}}$ after termination	action executions until termination	$\frac{\text{action executions}}{\text{number of states}}$ until termination
2	18	6	35	1.9	190	10.6
3	30	8	247	8.2	1380	46.0
4	45	10	2373	52.7	12330	274.0
5	63	12	30256	480.3	142318	2259.0
6	84	14	481471	5731.8	2063734	24568.3
7	108	16	9127581	84514.6	36135760	334590.4
8	135	18	199957001	1481163.0	740474450	5484995.9

Table 1: Simulation Results for Node Counting

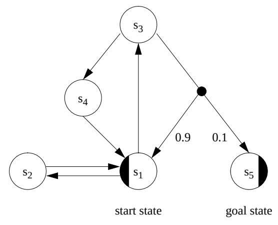

Figure 5: Node Counting does not Terminate

are a special case of Eulerian domains if one considers an undirected edge to be equivalent to a pair of directed edges with opposite directions.) Consider an arbitrary Eulerian graph whose number of edges is at most polynomial in the number of its states (for example, because no two edges connect the same states), and a graph that is derived from the Eulerian graph by replacing each of the directed edges of the Eulerian graph with two directed edges that are connected with a (unique) intermediate vertex. Figure 4 shows an example. Using results from (Koenig and Simmons 1996), we can show that the performance of Node Counting on the converted graph is guaranteed to be polynomial in its number of states. This is so because the behavior of Node Counting on the converted graph is the same as the behavior of Edge Counting on the original graph (Koenig and Simmons 1996) and the performance of Edge Counting on Eulerian graphs is polynomial in the product of the number of its edges and the number of its states (Koenig and Simmons 1996). To the best of our knowledge, this is currently the only known (non-trivial) domain property that guarantees a polynomial performance of Node Counting. We are currently investigating whether there are more realistic domain properties that also guarantee a polynomial performance of Node Counting.

Comparison of Node Counting with LRTA*

We have shown that the performance of Node Counting can be exponential in the number of states even in undirected domains. This is interesting because the value-update rule of LRTA* is similar to that of Node Counting but guarantees polynomial performance. There exist also other value-update rules that have this property in directed or undirected domains (perhaps with the restriction that every action has to result in a state change). The following table shows some of these variants of Node Counting. The real-time search methods by (Wagner *et al.* 1997) and (Thrun 1992a) resemble Node Counting even more closely than LRTA* resembles Node Counting since their value-update rules contain the term $1 + u(s)$ just like Node Counting. While their complexity was analyzed by their authors, it has been unknown so far how their performance compares to that of Node Counting in undirected domains. Our results demonstrate that modifications of Node Counting are necessary to guarantee polynomial performance and the resulting variants of Node Counting have a huge performance advantage over Node Counting itself.

Value-Update Rules WITHOUT Polynomial Performance Guarantee	
$u(s) := 1 + u(s)$.	Node Counting

Value-Update Rules WITH Polynomial Performance Guarantee	
$u(s) := 1 + u(succ(s,a))$.	LRTA*
if $u(s) \leq u(succ(s,a))$ then $u(s) := 1 + u(s)$.	(Wagner *et al.* 1997)
$u(s) := \max(1 + u(s), 1 + u(succ(s,a)))$.	(Thrun 1992a)

LRTA* has other advantages over Node Counting besides a better performance guarantee, for example that it can easily be generalized to probabilistic domains. This can be done by simply replacing every occurance of the u-value of the successor state, that is, $u(succ(s,a))$, with the *expected* u-value of the successor state. (Barto *et al.* 1995) have shown that this generalization of LRTA* to probabilistic domains reaches a goal state with probability one provided that one can reach a goal state with positive probability from every state that can be reached with positive probability from the start state. (This is the probabilistic equivalent of safely explorable domains.) We show, on the other hand, that Node Counting cannot be generalized to probabilistic domains by replacing every occurance of the u-value of the successor state with the expected u-value of the successor state. Figure 5 shows a (directed) domain that contains one action with a nondeterministic effect (the other actions are deterministic). In this case, Node Counting traverses the state sequence $s_1, s_2, s_1, s_3, s_4,\ s_1, s_2, s_1, s_3, s_4,\ s_1, s_2, s_1, s_3, s_4, \ldots$ (and so forth) if ties are broken in favor of successor states with smaller indices. In particular, whenever Node Counting is in state s_3, then $u(s_1) = 2u(s_4) + 2$ and consequently $u(s_4) < 0.9u(s_1) = 0.9u(s_1) + 0.1u(s_5)$. Node Counting then moves to s_4 and never reaches the goal state although the goal state can be reached with probability one by repeatedly executing the only nondeterministic action in state s_3.

Two other advantages of LRTA* over Node Counting are that it is easy to increase its lookahead (beyond looking at only the successor states of the current state) and to make use of heuristic knowledge to bias its search towards the goal.

Conclusion

This paper showed, for the first time, that similar value-update rules of real-time search methods can result in a big difference in their performance *in undirected domains*. Moreover, we obtained this result for two value-update rules for which several experimental results in standard domains from artificial intelligence indicated that both of them performed equally well. Our main result concerned the performance of Node Counting, a real-time search method that always moves to the neighboring state that it has visited the least number of times. We showed that its performance can be exponential in the number of states even in undirected domains. In particular, we constructed an undirected tree for which Node Counting executes (at least)

$$\Omega(n^{\sqrt{(\frac{1}{6}-\epsilon)n}})$$

actions, where n is the number of states and $0 < \epsilon < 1/6$ is an arbitrarily small constant. Thus, although many standard domains from artificial intelligence (such as sliding-tile puzzles, blocksworlds, and gridworlds) are undirected, this property is not sufficient to explain why Node Counting performs well on them. Our result solved an open problem described in (Koenig and Simmons 1996) and showed that a formal analysis of search methods can help one to detect problems and prevent surprises later on, reinforcing our belief that a formal analysis of real-time search methods can be as important as experimental work. Our result suggests to either abandon Node Counting in favor of real-time search methods whose performance is guaranteed to be polynomial in the number of states (especially since experimental results only indicated that Node Counting performed as well as other real-time search methods but but did not outperform them) or study in more detail which domain properties guarantee a polynomial performance of Node Counting.

References

Balch, T. and Arkin, R. 1993. Avoiding the past: A simple, but effective strategy for reactive navigation. In *International Conference on Robotics and Automation*. 678–685.

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

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism. In *Proceedings of the National Conference on Artificial Intelligence*. 714–719.

Geffner, H. and Bonet, B. 1998. Solving large POMDPs by real-time dynamic programming. Technical report, Departamento de Computación, Universidad Simón Bolivar, Caracas (Venezuela).

Ishida, T. and Korf, R. 1991. Moving target search. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 204–210.

Ishida, T. 1997. *Real-Time Search for Learning Autonomous Agents*. Kluwer Academic Publishers.

Koenig, S. and Simmons, R.G. 1995. Real-time search in non-deterministic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 1660–1667.

Koenig, S. and Simmons, R.G. 1996. Easy and hard testbeds for real-time search algorithms. In *Proceedings of the National Conference on Artificial Intelligence*. 279–285.

Koenig, S. and Simmons, R.G. 1998. Solving robot navigation problems with initial pose uncertainty using real-time heuristic search. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*. 154–153.

Koenig, S. 1995. Agent-centered search: Situated search with small look-ahead. Phd thesis proposal, School of Computer Science, Carnegie Mellon University, Pittsburgh (Pennsylvania).

Korf, R. 1987. Real-time heuristic search: First results. In *Proceedings of the National Conference on Artificial Intelligence*. 133–138.

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

Nilsson, N. 1971. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill.

Pemberton, J. and Korf, R. 1992. Making locally optimal decisions on graphs with cycles. Technical Report 920004, Computer Science Department, University of California at Los Angeles, Los Angeles (California).

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

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

Szymanski, B. and Koenig, S. 1998. The complexity of node counting on undirected graphs. Technical report, Computer Science Department, Rensselaer Polytechnic Institute, Troy (New York).

Thrun, S. 1992a. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, School of Computer Science, Carnegie Mellon University, Pittsburgh (Pennsylvania).

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

Wagner, I.; Lindenbaum, M.; and Bruckstein, A. 1997. On-line graph searching by a smell-oriented vertex process. In Koenig, S.; Blum, A.; Ishida, T.; and Korf, R., editors 1997, *Proceedings of the AAAI Workshop on On-Line Search*. 122–125.