Deadline-Aware Search Using On-line Measures of Behavior

Austin J. Dionne and **Jordan T. Thayer** and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
austin.dionne at gmail.com, jtd7, ruml at cs.unh.edu

Abstract

In many applications of heuristic search, insufficient time is available to find provably optimal solutions. We consider the contract search problem: finding the best solution possible within a given time limit. The conventional approach to this problem is to use an interruptible anytime algorithm. Such algorithms return a sequence of improving solutions until interuppted and do not consider the approaching deadline during the course of the search. We propose a new approach, Deadline Aware Search, that explicitly takes the deadline into account and attempts to use all available time to find a single high-quality solution. This algorithm is simple and fully general: it modifies best-first search with on-line pruning. Empirical results on variants of gridworld navigation, the sliding tile puzzle, and dynamic robot navigation show that our method can surpass the leading anytime algorithms across a wide variety of deadlines.

Introduction

Heuristic search is an oft employed technique for automated problem solving. Given an admissible and consistent heuristic, A* search (Hart, Nilsson, and Raphael (1968)) finds an optimal solution using the smallest possible number of expansions, up to tie-breaking, of any similarly informed algorithm (Dechter and Pearl (1988)). Unfortunately for many problems of practical interest finding an optimal solution still requires an impractical amount of time. In this paper, we address one attractive approach to this dilemma, *contract search*, in which the objective is to find the cheapest solution possible within a given deadline. Note that this problem is distinct from real-time search (Korf 1990), in which one merely wishes to find the single next action of an agent within bounded time.

Despite the importance of contract search in real applications, we are aware of only two contract search algorithms in the literature. Neither performs particularly well in the following evaluation. This may be why the prevailing approach to solving such search problems is to use an interruptible anytime algorithm. While anytime algorithms are applicable to the problem of contract search, they are designed for use in problems where the deadline is unknown. The deadline has no impact on the search order of these algorithms, save for what node will be the last to be expanded. We propose that knowledge of the time remaining in the search can be used to alter the search order productively by allocating all search effort towards optimizing a single solution, rather than discarding all but the last one found.

In this paper we propose a new algorithm called Deadline Aware Search (DAS) that is based directly on the objective of contract search: finding the best single solution possible within the deadline. At each iteration the search expands the state that appears to lead to the best solution deemed reachable within the time remaining. Our empirical analysis shows that DAS can compete with and often surpasses previous contract approaches and the leading anytime algorithms on variants of gridworld navigation, the sliding tiles puzzle, and dynamic robot navigation without relying on off-line learning or parameter optimization as previous proposals do.

Previous Work

We will first review the anytime approach to search under a deadline. We then the two previous proposals for contract algorithms before presenting Deadline Aware Search, a new approach to the problem of contract search.

Anytime Algorithms

Interruptible anytime algorithms are a class of algorithms that are designed to quickly return a highly suboptimal solution followed by a sequence of solutions of improving quality, eventually converging on optimal. These algorithms are often applied to the problem of search under a deadline because they can be configured to find the first solution very quickly, guaranteeing that some solution will be present at the deadline, and as the deadline is extended the cost of the solutions returned decreases, eventually to optimal.

Many anytime algorithms have been proposed. Depth-first branch-and-bound is a well-known anytime algorithm, but it applies only to problems in which duplicate states are rare and a useful bound is known on the depth of solutions, criteria which exclude many domains, including the benchmarks considered here. Thayer and Ruml (2010) performed an evaluation of several anytime methods across a wide variety of domains. The repairing framework of Anytime Repairing A* (Likhachev, Gordon, and Thrun (2003)) had the best general performance across all domains and

search strategies, making it the prime anytime competitor for deadline search.

Anytime Repairing A* (ARA*) performs weighted A* (Pohl 1973) search to find a starting incumbent solution and then continues searching to find a sequence of improved solutions, eventually converging to the optimal. After each new solution is found the weight used in the search is reduced by some predefined amount, the open list is resorted, and search continues. ARA* also uses a novel method of postponing the expansion of duplicate states at each round of the search, improving performance in domains with tight cycles and a large numbers of duplicates.

Richter, Thayer, and Ruml (2009) proposed a modification to the weighted anytime approach which, while possibly appearing counter-intuitive at first, often results in better anytime performance. Their algorithm, Restarting Weighted A* (RWA*) performs a series of Weighted A* searches, each essentially from scratch. Heuristics and $g(s)$ values are cached between iterations, while the open list is cleared. While RWA* is generally slower than ARA* (Thayer and Ruml 2010), in some search domains, particularly domain independent planning, restarting the search at each iteration has a surprisingly positive effect on the results. We include results for this algorithm as well as ARA* in our evaluation.

While the interruptible anytime approaches are applicable to the problem of contract search, they do not directly address it, as they are deadline agnostic. Only the best solution found thus far is returned when the deadline is reached. The previous incumbent solutions provide some benefit to the search, as they are used for pruning the search space while searching for the next solution. However, there is some amount of effort wasted in finding a sequence of solutions that, in the end, are not returned.

Another problem with the current anytime approaches is that the best performing algorithms are based on bounded suboptimal search, which requires that the bound be set prior to execution. While in some domains there is a single initial bound that performs well over the range of deadlines, there are others in which one setting will perform better for shorter deadlines and another for longer. There is currently no clear way to select a bound based on anything other than training on similar problems and deadlines, or intuition.

Time Constrained Search

Hiraishi, Ohwada, and Mizoguchi (1998) proposed Time Constrained Search, a contract algorithm based on weighted A*. It attempts to measure search behavior in order to adjust the weight used in weighted A* in order meet the deadline while optimizing solution quality. They perform a standard weighted A* search on $f'(s) = g(s) + w \cdot h(s)$, where $g(s)$ represents the cost of the path explored thus far, $h(s)$ is the heuristic estimate of cost-to-go, and w is a weight factor that they adjust dynamically. They take advantage of the fact that increasing the weight w generally has the effect of biasing search effort towards states that are closer to goals, reducing solving time.

Search behavior is adjusted using search velocity, defined as follows: They first define the distance D as the estimate of cost-to-go at the starting state: $D = h(s_{start})$. They then use the time bound T assigned to the search to calculate a desired "average velocity" $V = D/T$. During the search they use the time elapsed t and the minimum $h(s)$ value for any state generated thus far h_{min} in order to calculate an "effective velocity" $v = (D - h_{min})/t$. If the effective velocity falls outside a selected tolerance from the desired average velocity, the current weight applied to $h(s)$ when calculating $f'(s)$ is adjusted accordingly by subtracting/adding a predetermined value, Δw.

The upper and lower bounds on average search velocity and the value of Δw have a significant impact on algorithm performance. Unfortunately, these and other critical portions of the algorithm, such as how often (if ever) to resort the open list and whether or not there is a minimum delay between weight adjustments, are not specified. Attempts to contact the authors to resolve these issues were unsuccessful. While their empirical analysis illustrates the quality of solutions found over a range of real-time deadlines (with the contract specified in seconds of computation time), no comparisons were made to previously proposed algorithms. Despite our best efforts to implement and optimize this algorithm we were unable to create a real-time version that was comparable to existing approaches.

Contract Search

Contract Search (Aine, Chakrabarti, and Kumar 2010) attempts to meet the specified deadline by limiting the number of state expansions that can be performed at each depth in the search tree. The algorithm is based around the following insight into search on trees: for an algorithm to expand the optimal goal, it need only expand a single state along an optimal path at each depth. The idea behind Contract Search is to expand only as many states as needed at each depth in order to encounter the optimal solution. We can obviously not know this information a priori. We can, however, assume that the more states we expand at a given depth, the more likely we are to have expanded the state on the optimal path. Contract search attempts to maximize the likelihood that an optimal solution is found within the deadline by maximizing the likelihood that this optimal state is expanded at each depth, subject to an expansion budget.

The algorithm has two phases: determining off-line how many states should be considered at a given depth, $k(depth)$, and then efficiently using this information on-line in an A*-like search. Rather than a single open list sorted on $f(n)$, contract search maintains a separate open list for each depth. Each of these per-depth open lists tracks the number of states it has expanded and becomes disabled when this reaches its $k(depth)$ limit. At every iteration of the search, the state with the smallest $f(n)$ across all open lists that have not yet exhausted their expansion budget is expanded. For very large contracts, this will behave exactly like A*, and for smaller contracts it approximates a beam search on $f(n)$.

Approximating $k(depth)$, the number of states to expand at each depth which maximizes the probability of success can be done off-line, before search. To do so, we need to know a probability distribution over the depths at which goals are likely to appear, the average branching factor for the domain, and the likelihood that the optimal state at a

Deadline Aware Search(*starting state*, *deadline*)
1. *open* $\leftarrow$ {*starting state*}
2. *pruned* $\leftarrow$ {}
3. *incumbent* $\leftarrow$ *NULL*
4. while *(time)* $<$ *(deadline)*
5. if *open* is non-empty
6. $d_{max} \leftarrow$ *calculate_d_bound()*
7. $s \leftarrow$ remove state from *open* with minimal $f(s)$
8. if s is a goal and is better than *incumbent*
9. *incumbent* $\leftarrow s$
10. else if $\widehat{d}(s) < d_{max}$
11. for each child s' of state s
12. add s' to *open*
13. else
14. add s to *pruned*
15. else (* *open* is empty *)
16. *recover_pruned_states(open, pruned)*
17. return *incumbent*

Recover Pruned States(*open*, *pruned*)
18. *exp* $\leftarrow$ *estimated expansions remaining*
19. while *exp* > 0 and *pruned* is non-empty loop
20. $s \leftarrow$ remove state from *pruned* with minimal $f(s)$
21. add s to *open*
23. *exp* = *exp* $-\widehat{d}(s)$

Figure 1: Pseudo-code sketch of Deadline Aware Search

given depth will be expanded within a fixed number of states. These can be approximated by analyzing sample problems from the domain. Given these values, the number of states to expand at each depth for a given contract can be solved for using dynamic programming. These values are then stored for future use.

The largest contract considered by (Aine, Chakrabarti, and Kumar 2010) is 50,000 expansions. In our evaluation, we will be considering search deadlines of up to a minute, which for our benchmark domains could mean more than five million states. This is problematic because the time and space complexity of computing these values grows quadratically in the size of the contract. Aine (2011) suggested approximating the table by only considering states in chunks, rather than a single state at a time. This cuts down on the size of the table and the number of computations we need to perform to compute it. In the results presented below, the resolution was selected so that the tables needed could be computed within 8 GB of memory. Computing the tables typically took less than eight hours per domain, although a new table must be computed for each considered deadline.

Deadline Aware Search

We now present a new approach for the contract search problem, the called Deadline Aware Search (DAS). Unlike anytime search algorithms that do not alter their search strategy in reaction to the approaching deadline, DAS reacts to the approaching deadline during search. We begin by presenting a general overview of the algorithm and its behavior. We then discuss the estimation of two quantities needed by the algorithm: the maximum achievable search depth d_{max} and distance to the cheapest solution beneath a node $\hat{d}_{cheapest}(s)$. Finally, we discuss DAS's technique for recovering from situations in which it estimates that no goal is reachable given the current search behavior.

DAS is a simple approach, derived directly from the objective of contract search. It expands, among all the states leading to solutions deemed reachable within the time remaining, the one with the minimum $f(s)$ value. Pseudocode of the algorithm is presented in Figure 1. The open list is first initialized with the starting state and then the search proceeds to expand nodes from the open list until either the search time expires or the open list is empty (indicating that there is no solution deemed reachable). At each iteration of the algorithm, the state with minimal $f(s)$ is selected for expansion and the current maximum reachable distance, d_{max}, is calculated. If the distance to this state's best goal $d_{cheapest}(s)$ is less than d_{max}, it is expanded and its children are added to the open list. Otherwise, it is added to the pruned list and the search will select the next best node for expansion. What drives the search to find a solution before the deadline is that at each iteration d_{max} is calculated, and any state s such that $\hat{d}_{cheapest}(s) > d_{max}$ is deemed "unreachable", added to a separate pruned list, and not expanded. Thus the best reachable state is expanded at each iteration.

In the case that all nodes have been deemed "unreachable" and there is search time remaining, the Recover Pruned States method is invoked, retrieving a subset of the states from the pruned list and resetting the on-line estimates of search behavior. This subset of states will contain at least one state, and therefore the search will never terminate before the deadline unless all possible solutions have been considered. Note that the methods of measuring the search behavior on-line are not present in the pseudocode. These details will be discussed further in later sections.

Reachability, as estimated by DAS, is a function of a state's distance from its best possible solution, $d_{cheapest}(s)$. When there is not enough time to explore all interesting paths in the search space, it makes sense to favor those paths that are closer to solutions. One result of using an admissible heuristic $h(s)$ is that often the best f value of the states under a particular state s will be higher than the value of $f(s)$. Assuming this heuristic error is distributed across a search space, the states that are farther from solutions have the potential of experiencing this increase in f value more often before reaching their respective solutions than states that are closer. For this reason, when selecting states for expansion ties on $f(s)$ are broken in favor of smaller $h(s)$. Also, because search spaces generally increase exponentially, states that are farther from solutions also generally have larger subtrees lying beneath them, generally requiring more search effort to find their respective best solutions.

Calculating $\mathbf{d_{max}}$

One could argue that all remaining search effort should be put towards finding the best possible solution under the sin-

gle state in the search space estimated to be best and therefore the d_{max} value used to prune states should be equal to the estimated number of expansions possible in the time remaining. In practice, however, this approach renders the value of d_{max} meaningless for most of the search in which the number of expansions will often be far larger than the estimated length of the path to any particular solution. If the number of expansions allowed were close to the length of the path to a solution then one could hardly consider performing any type of search other than depth-first! Pilot empirical studies have confirmed that such an interpretation of d_{max} produces unreasonable behavior.

To understand the true number of expansions it will take to find a solution under a particular state, one must consider the behavior of the search itself. In a best-first search, it is not typical that a single path will be followed from the starting state to the optimal solution. Often, due to error in the heuristic, when a state is expanded the f value of its best child state s_{bc} is greater than $f(s)$. When this occurs it is possible that there exists some other state in the open list s' such that $f(s') < f(s_{bc})$. If the search continues unhindered, it will stop exploring the path leading to s_{bc} and start examining the path currently leading to s'. This behavior is a phenomenon that we call *search vacillation*, where multiple partial solutions are being explored during the search.

One way to measure this vacillation, is a concept we call *expansion delay*. During the search, the number of expansions performed is tracked, e_{curr}. At each expansion, each child state s generated is marked with the current expansion number $e(s)$. When a state comes up for expansion during the search, the current expansion number e_{curr} can be used to calculate the expansion delay using Δe:

$$\Delta e = e_{curr} - e(s) \tag{1}$$

In the case that there is no heuristic error, that is $h(s) = h^*(s)$, the expansion delay will always be 1 and the search will expand directly along the optimal path given that ties in $f(s)$ are broken in favor of higher $g(s)$ value as is done in DAS. On the other end of the spectrum, in the case of uniform cost search, the expansion delay can be expected to grow exponentially over the course of the search along with the size of the search frontier.

We can use this measure of search behavior in order to calculate a value for d_{max} that more accurately reflects the average distance along any particular path that will be explored given the current behavior of the search and the time remaining. During the course of the search, the average expansion delay $\overline{\Delta e}$ is tracked. In order to react to the changing behavior of the search, a sliding window average rather than a global average is used. The current expansion rate r is estimated by taking a sliding window average of the delta times between expansions. Using the time remaining t the number of expansions remaining in the search *exp* can be estimated using Equation 2.

$$\text{exp} = t \cdot r \tag{2}$$

Using this estimate, the average distance that could be explored along any particular path in the search d_{max} can be calculated using Equation 3.

$$d_{max} = \frac{\text{exp}}{\overline{\Delta e}} \tag{3}$$

While we seed the value of exp based on some reasonable default expansion rate $r_{default}$ until the online estimate is initialized, this has no significant impact on the performance of DAS, as no pruning typically occurs during the small window of time before the on-line estimate is formed.

The very act of pruning states from the search space will have an effect on the average expansion delay experienced. It represents a direct measure of search behavior including the effects that the measure itself has introduced, and learning the value on-line during the search is most appropriate.

It should be noted that a window of time at the beginning of the search must be allotted for the average expansion delay estimate to settle. Until enough samples are taken, the average could produce unrealistic estimates of d_{max}. In the experiments reported below we used a settling time of 200 expansions. In pilot experiments the effects of modifying this parameter were insignificant.

Pruning On $\hat{\mathbf{d}}(\mathbf{s})$

DAS makes use of a heuristic $\hat{d}_{cheapest}(s)$ that estimates the length of the path to the cheapest goal state under a particular state s. For unit-cost domains this heuristic is the same as the standard cost-to-go $h(s)$. For non-unit cost domains it can often be constructed similarly to $h(s)$ by estimating the same cheapest path while replacing all actions costs with 1. We do not require that $d_{cheapest}(s)$ be admissible or even consistent, in fact, it is preferable for $d_{cheapest}(s)$ to act as a differentiator between different paths by more accurately accounting for heuristic error. We employ the path-based correction model of Thayer, Dionne, and Ruml (2011) to calculate a corrected heuristic $\hat{d}_{cheapest}(s)$. Briefly, this correction model uses error experienced at each expansion: $\epsilon_d = d_{cheapest}(s) - d_{cheapest}(p(s)) + 1$, where $p(s)$ represents the parent of state s. The mean single step error encountered so far $\overline{\epsilon_d}(s)$ is tracked for each partial solution and is used as an estimator of the average single step error remaining from the end of that partial solution to the corresponding goal state. The true distance-to-go $d^*_{cheapest}(s)$ can be estimated by $\hat{d}_{cheapest}(s) = \frac{d_{cheapest}(s)}{(1-\overline{\epsilon_d}(s))}$, when $\overline{\epsilon_d}(s) < 1$. In the case that $\overline{\epsilon_d}(s) \geq 1$, the value of $\hat{d}_{cheapest}(s)$ is set to infinity.

Search Recovery

It can be the case that the effects of pruning are not enough to keep the $\hat{d}(s)$ of at least one state in the open list below the value of d_{max}. In these cases, DAS will prune all states from the open list as "unreachable". This is an indication that despite the best efforts of the algorithm, the amount of vacillation remaining in the search due to competing states on the open list will result in no further solutions being reached.

To recover from this, we first estimate the number of expansions possible in the time remaining exp. States are then selected from the pruned list in order of minimal $f(s)$ value

such that the sum of $\hat{d}(s)$ for all states inserted is approximately equal to exp. In order to guarantee that at least one state is placed back into open at each recovery we insert states into open until the sum of $\hat{d}(s)$ first exceeds exp (see Figure 1). The intention is that only the best set of states that would be reachable regardless of the search behavior are kept. Because the open list has changed so drastically, the previous estimation of average expansion delay $\overline{\Delta e}$ is no longer relevant and the running average is reset. This allows the search to continue and measure the new local behavior after the recovery. The same settling time used at the start of the search must be applied, during which d_{max} is not calculated or used.

Optimality for Large Deadlines

One of the desired features for a contract search algorithm is that, given a long enough deadline, the algorithm will devolve to A* and return a guaranteed optimal solution (given an admissible heuristic). DAS has this property.

Theorem 1 *Given a sufficiently long deadline D and assuming that the one-step error model does not result in any states with $\widehat{d}(s) = \infty$, DAS will perform an equivalent search to A* and return a guaranteed optimal solution.*

Proof: Given that DAS performs a best-first search on an admissible $f(s)$ and returns a solution only when the state is selected for expansion, it behaves exactly like A* with the exception of pruning states due to reachability. However, given a long enough deadline and the use of any of the proposed methods for calculating d_{max} there will never be a state selected for expansion such that $\widehat{d}(s) > d_{max}$. Therefore, the algorithm will behave exactly as A* as it will never prune. The proof that A* returns an optimal solution can then be directly applied. □

The minimum value of the deadline D necessary for this fact to hold depends on the method of calculating d_{max} used as well as the characteristics of the search space. This means that, unfortunately, D can not be easily determined a priori.

Empirical Analysis

We performed an empirical analysis over several benchmark domains in order to evaluate the performance of Deadline Aware Search in comparison to Anytime Repairing A*, Restarting Weighted A*, and Contract Search. In each domain we tested over a range of deadlines covering around four orders of magnitude. All algorithms were implemented in Objective Caml using similar data structures and all tests were performed on the same type of machine.

In an attempt to judge algorithms fairly in the case that no solution is found within the deadline, all algorithms are required to run an initial search algorithm we call "Speedier". Speedier search is a greedy search on $d_{cheapest}(s)$ in which duplicate states are detected and, if already expanded, are ignored. This search completes very quickly, returning what is typically a highly suboptimal solution that all algorithms use as an incumbent. Therefore any algorithm that fails to find an improved solution within the deadline will return this sub-optimal Speedier solution. This both simplifies our analysis by assigning a meaningful cost to the null solution and is a realistic implementation in the case of a any setting in which returning no solution is absolutely unacceptable.

For ARA* and RWA* we evaluated the following range of initial weight settings: 1.2, 1.5, 3.0, 6.0, 10.0 and a weight decrement of 0.2. The optimal initial weight setting found for the 15-Puzzle, Weighted 15-Puzzle, Unit-Cost Grid-World, Life-Cost Grid-World, and Dynamic Robot Navigation were 3.0, 3.0, 3.0, 6.0, and 1.5, respectively. In each plot the results for the top two weight settings are illustrated, as there were settings which did not produce the best results overall but performed better for a specific range of deadlines.

Contract Search uses the number of expanded states for its deadline, as originally proposed, rather than a time cutoff like DAS. Similar to all other methods evaluated, Speedier is run initially and the time elapsed is subtracted from the deadline. At this point the time remaining is multiplied by the average state expansion rate, measured off-line for each domain, in order to estimate the state contract to apply. The deadline is still measured in seconds and when it arrives the algorithm returns the best solution found thus far.

15-Puzzle

Experiments were performed on the 100 instances of the 15-Puzzle presented by Korf (1985) using the Manhattan distance heuristic. We evaluted both a uniform cost model as well as a model in which the cost of moving each tile was the inverse of the numeric value of the tile (1-15). Results are shown in Figure 2. The X-axis of the plots represents the deadline in seconds and is displayed on a logarithmic scale. The Y-axis of the plot represents solution quality, being defined as cost of the best solution found by any algorithm for the particular instance over the achieved solution cost. Solution quality is used rather than raw solution cost to reduce because we have many domains in which individual instances may have very different optimal solution costs. It is a standard metric used in the satisficing track of the International Planning Competition.

In both cost models of the 15-Puzzle domain Deadline Aware Search is a clear improvement over both ARA* and Contract Search for the full range of deadlines. Although RWA* outperforms DAS for the very shortest deadlines, its performance does not scale well to the larger deadlines.

Contract Search exhibited the anticipated behavior of resorting to the Speedier solution for short deadlines up the point where optimal or near optimal solutions are found for some of the instances. Because of the way that Contract Search defines the goal probability distribution it is less likely to come across highly suboptimal solutions, as the goal probabilities at those depths will be zero, or close to it. Another interesting behavior of Contract Search is that after a certain point the results for larger deadlines start to decrease in quality. This is at least in part due to the imprecise conversion from search time remaining to state expansions remaining, and may also partially be the result of using a relaxation to compute $k(depth)$. For larger deadlines, the total error will be larger and Contract Search could end up

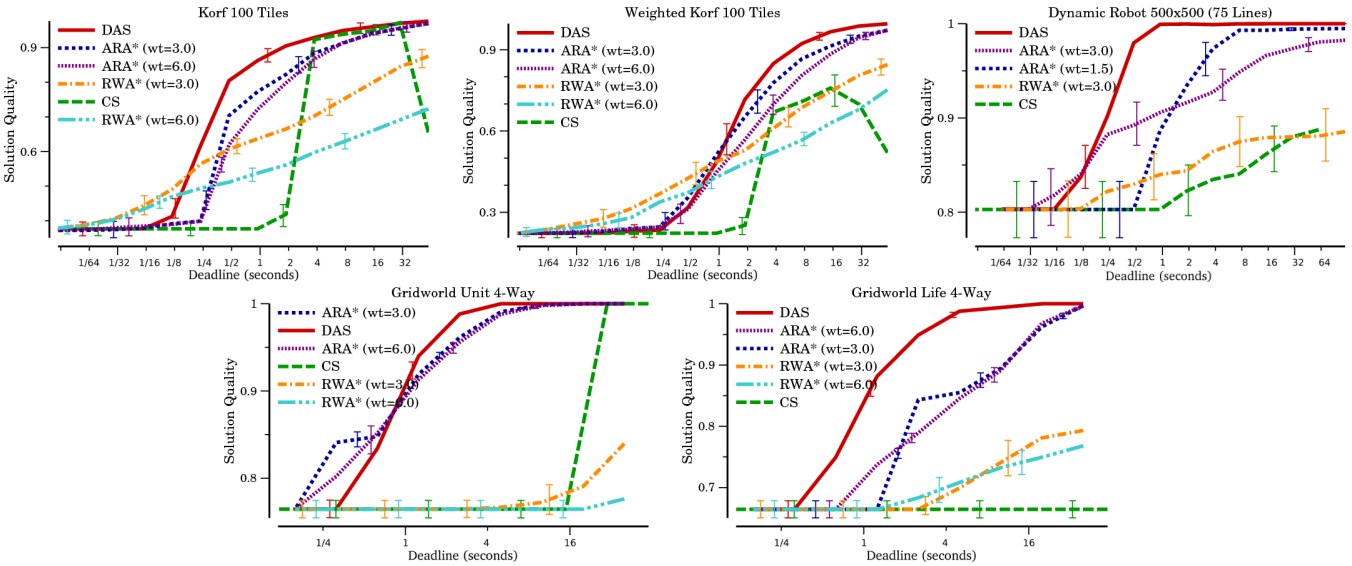

Figure 2: Solution quality (best known cost / achieved cost) for a given solving time deadline.

initializing the levels of the tree closest to the root with too many possible expansions, not leaving enough for the lower levels to be fully explored before the deadline arrives.

Some important insight can be drawn by looking horizontally through the plots. For example, Figure 2 shows that with the short deadline of only around 0.5 seconds, DAS was able to find, on average, the same quality of solutions that took ARA* with an optimal weight setting would find with more than twice that much time. For the standard tiles domain, contract search is competitive with deadline aware search for large deadlines, where both algorithms solve the problem optimally, and for small deadlines, where both algorithms return the Speedier solution.

Dynamic Robot Navigation

Experiments were performed in the domain of dynamic robot navigation, similar to that used by Likhachev, Gordon, and Thrun (2003) to evaluate ARA*. The objective in these problems is to find the fastest path from the starting location of the robot to some goal location and heading, taking motion dynamics such as momentum into account. The instances used in our experiments were 500 by 500 cells in size. We scatter 75 lines, up to 70 cells in length, with random orientations across the domain and present results averaged over 100 instances. Results are shown in Figure 2.

Results in this domain show Deadline Aware Search as a clear improvement over ARA*, RWA*, and Contract Search for the full range of deadlines. Contract Search performed particularly weakly in this domain. We believe this is in part attributed to the fact that the domain has a fairly accurate, albeit inadmissible, $d_{cheapest}(s)$ heuristic. Taking advantage of this heuristic allows Deadline Aware Search to more accurately decide the reachability of states and may have contributed to its success. Outside of tie breaking, it is not obvious how the other algorithms could make use of distance estimates.

Grid-World Navigation

Experiments were performed on two sets of four-way movement grid-world navigation problems; unit-cost and life-cost. In both domains the starting state is in the lower-left corner of a 2000x1200 map with the goal state in the lower-right corner. Obstacles are distributed randomly and uniformly with a probability of 0.35. The life-cost grid-world, first proposed by Ruml and Do (2007), varies the cost of movement in different layers within the grid creating a clear distinction between shortest path and cheapest path. The cost of traversing each square in the map is equal to the Y coordinate of that location, with (0,0) being the bottom left corner. This implies that the cheapest path through the map would be to traverse to the top of the map, across, then back down to the solution.

Results are shown in Figure 2. In the case of life-cost grid-world Deadline Aware Search was competitive with the best of the anytime methods at the optimal parameter settings for shorter deadlines and provided improved results for larger deadlines. In the case of unit-cost grid-world Deadline Aware Search is surpassed by ARA* for the shorter deadlines but is competitive for larger deadlines. RWA* did not perform competitively in this domain and Contract Search did not manage to return results for almost any of the deadlines used. In the grid-world domain the solutions lie very deep in the search space, there is a lot of variation between solution depths, and there are a lot of duplicate states. In order to generate the $k(depth)$ tables for Contract Search in a reasonable amount of time an approximation method was used. We believe that all of these could be contributing factors to the failure of Contract Search in these domains.

DAS Behavior

The purpose of d_{max} in DAS is to act as a decreasing upper bound on the distance between any state expanded and

its respective best solution. This bound is intended to force the search to progress forwards to meet a particular deadline when it would normally have spent more time exploring different partial solutions sorting out the increasing $f(s)$ values. While it is difficult to justify what the "correct" value of d_{max} should be over the course of the search, we can impose a few reasonable limitations. The value should represent a smooth function, as a large variability would cause unstable behavior in DAS. Depending on the given deadline, the value should typically fall somewhere within the range of current $\widehat{d}(s)$ values such that some pruning will occur when necessary and not all states will be pruned unnecessarily.

In order to evaluate the behavior of d_{max} relative to the $\widehat{d}(s)$ of states expanded during a Deadline Aware Search, we implemented a version which uses a limit on the number of state expansions as a deadline and records relevant information during the search. This way the overhead of recording could be factored out of the analysis. Figure 3 contains the results of the analysis on a single instance of unit-cost four-way gridworld navigation for a range of deadlines.

The plots illustrate the current value of d_{max} and the $\widehat{d}(s)$ value of the expanded states over the course of the search. The plots are shown for deadlines of 400, 200, 100, and 50 thousand expansions. DAS returned solutions of cost 2967, 3043, 3159, and 3525, respectively. As a reference point, A* completes for this instance after 402,220 expansions returning a solution cost of 2967. Speedier solves in 15,327 expansions returning a solution cost of 4041. The expansions taken by the initial Speedier search are subtracted from the remaining deadline. In examining these plots, one should note that the red line representing d_{max} appears to spike at various locations. With the exception of the start of the search, these points represent when the reachability model is reset due to all states being pruned as "unreachable". One can identify the points when solutions were returned: wherever $\widehat{d}(s) = 0$.

From the plots, one can see that for longer deadlines, the values of d_{max} effectively push the search towards finding a solution with a somewhat significant amount of time (or in this case, state expansions) remaining. After the first solution is found the search will have much more strict bounds for pruning and one can see that the vacillation increases such that the search repeatedly stagnates resulting in all states being pruned and the estimates of d_{max} resetting. Despite the effort wasted in pruning and reinserting states, this does not need to be interpreted as negative behavior. For example, looking at the plot for 50 thousand expansions one can see that during the search there are several points at which the model estimates that no solution will be reachable given the current search behavior and the recover pruned states method is invoked. It is thanks to this recovery method's trimming of the search space that the search can complete at all in such a short deadline with a solution significantly better than the original Speedier incumbent solution provided. This illustrates the versatility of the approach between large (A* sized) and short (Speedier search sized) deadlines.

Discussion

The choice to include a Speedier search first in our empirical analysis for all algorithms could lead to biasing results such that algorithms which do not return any solution before the deadline are rated closer to those which return sub-optimal solutions which are only marginally better than the Speedier solution. This is not a significant issue in our results, as most solutions found after the Speedier solution are substantially improved. We recorded the number of times each algorithm succeeded in improving on the Speedier solution and show results in Figure 4. One can see that in some domains such as the sliding tiles puzzle, DAS fails to improve on the speedy solution more often than RWA* for short deadlines on the unweighted model and both ARA* and RWA* for short deadlines on the weighted model. Despite this fact, the average solution quality for DAS on these domains was generally higher than the anytime approaches. One can also see that part of the success of DAS on Dynamic Robots and the Life-Cost Gridworld Navigation problems comes from the fact that it returns significantly more improved solutions for shorter deadlines than the other approaches.

Overall, the experimental results indicate that Deadline Aware Search can lead to a significant improvement over ARA* and RWA* in some domains while remaining competitive in others. In no domain did the time-based version of Contract Search perform particularly well.

It should be noted that in the results for ARA* there were several cases (Figure 2, dynamic robots and life grids) in which different parameter settings resulted in the best performance for certain ranges of deadlines. In both Sliding Tiles domains RWA* had the best performance for a range of very short deadlines but did not scale well up to the longer deadlines. It may not be possible to determine the appropriate algorithm or parameter setting in advance for a given problem and deadline and selecting an incorrect configuration may result in poor results. In contrast, the same configuration of Deadline Aware Search was used in all domains and remained competitive or outperformed the optimized parameter settings for ARA* and RWA*. DAS is a parameterless and general approach.

Conclusion

We have proposed a new method of measuring search behavior, expansion delay, that can be used as an indicator of the level of vacillation present in the search due to heuristic error leading to competition between different paths on the open list. Using this measure we have constructed a very simple and general approach to the problem of heuristic search under deadlines: Deadline Aware Search. DAS appears to be the first effective contract heuristic search algorithm, showing improvements over ARA* and RWA* in several domains using real time as deadlines. Our approach also has the benefit of being parameterless, learning necessary information on-line, while previous approaches required either parameter optimization or off-line training and pre-computation.

The problem of heuristic search under real-time deadlines is of great importance in practice and yet few algorithms have been proposed for that setting. While anytime meth-

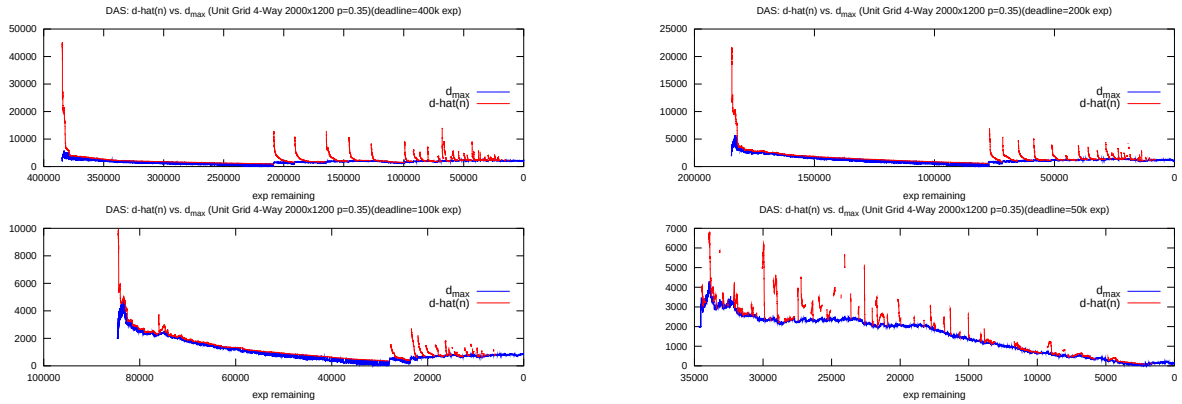

Figure 3: DAS Analysis of d_{max} vs. $\widehat{d}(s)$

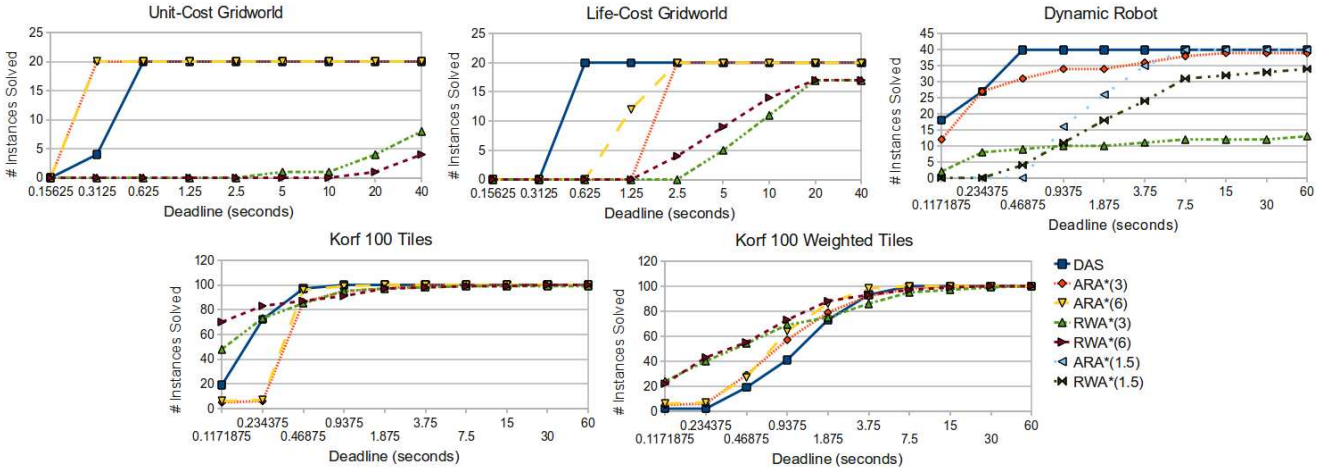

Figure 4: Number of instances in which initial Speedier solution was improved

ods are certainly applicable, they are really designed to address the problem of search when the deadline unknown. While simple, our approach illustrates that knowledge of the termination deadline can improve performance for contract search.

Acknowledgements

We gratefully acknowledge support from NSF (grant IIS-0812141) and the DARPA CSSG program (grant N10AP20029). We also thank Sandip Aine for his responses to our inquiries about Contract Search.

References

Aine, S.; Chakrabarti, P.; and Kumar, R. 2010. Heuristic search under contract. *Computational Intelligence* 26(4):386–419.

Aine, S. 2011. Personal communication.

Dechter, R., and Pearl, J. 1988. The optimality of A*. In Kanal, L., and Kumar, V., eds., *Search in Artificial Intelligence*. Springer-Verlag. 166–199.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics* SSC-4(2):100–107.

Hiraishi, H.; Ohwada, H.; and Mizoguchi, F. 1998. Time-constrained heuristic search for practical route finding. In *Pacific Rim International Conferences on Artificial Intelligence*.

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

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *Proceedings of the Seventeenth Annual Conference on Neural Information Porcessing Systems (NIPS-03)*.

Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computation issues in heuristic problem solving. In *Proceedings of IJCAI-73*, 12–17.

Richter, S.; Thayer, J. T.; and Ruml, W. 2009. The joy of forgetting: Faster anytime search via restarting. In *Symposium on Combinatorial Search*.

Ruml, W., and Do, M. B. 2007. Best-first utility-guided search. In *Proceedings of IJCAI-07*, 2378–2384.

Thayer, J. T., and Ruml, W. 2010. Anytime heuristic search: Frameworks and algorithms. In *SoCS 2010*.

Thayer, J. T.; Dionne, A.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS-11)*.