Bounded Anytime Deflation

Thomas L. P. Allen
The Australian Centre for Field Robotics
The University of Sydney, NSW, Australia
t.allen@acfr.usyd.edu.au

Abstract

This paper rearranges a key property of anytime algorithms to derive a technique that determines an admissible lower bound on the minimum solution cost. In conjunction with a class of performance metric known as 'time-focussed metrics', the technique can be used to determine when to abort the deflation process within an anytime planning algorithm.

The aim of this paper is to exploit this technique to determine when to abort an anytime planning process, however the technique is shown to be unsuitable for dynamic state spaces. Despite this, the bounded anytime deflation technique is applicable to the wider field of anytime optimisation, and may be expected to have application outside the planning domain.

1 Introduction

Anytime algorithms are a class of algorithms which iteratively recompute an optimisation with decreasing upper bounds on the cost of the solution [Zilberstein and Russell, 1995]. They have been used in many applications in which the desired result requires an impractical amount of computational time, but where suboptimal solutions are acceptable. The most prevalent of these applications has been in planning systems such as [Belghith *et al.*, 2006; Hansen and Zhou, 2007; Allen *et al.*, 2009], since optimal planning is often intractable in complex scenarios.

This paper presents a novel contribution to the field of anytime algorithms, showing that a key property of heuristic graph search algorithms can be rearranged to provide a lower bound on the true optimal result. This is a significant result as there are many scenarios in which the true optimal result cannot even be computed in finite time. This result is then combined with a type of performance metric described in this paper as a 'time-focussed metric', which measures the performance of a system in terms of the time required to complete the system's objectives.

The aim of the paper is to derive a technique which combines these concepts to determine when continued anytime deflation is incapable of improving the overall system performance, for an agent-based path planning application. A negative result is obtained for this particular example, as the derived technique is shown to only be valid for static state spaces, rather than the dynamic spaces that feature in the application. Despite this result, the bounding process itself is applicable to the wider field of anytime processes, and may be expected to have application outside the planning domain.

The structure of this paper is as follows. Section 2 presents a brief description of discrete planning algorithms, of which heuristic graph search algorithms are a subset. Section 3 then describes anytime algorithms in detail, showing that a particular form known as 'interruptible algorithms' is required for the techniques presented in this paper, but also that any other form can be converted to an interruptible one. Section 4 describes the structure of metrics that are required to assess the performance of planning algorithms and systems, and introduces the class of 'time-focussed metrics' which are required for the techniques presented in this paper.

Section 5 presents these techniques, first showing that an admissible lower bound on the true result of an anytime deflation process can be determined while the process is in operation. By assessing system performance with a time-focussed metric, this bound can be used to determine when continued deflation is incapable of further improving the system performance. At this point the deflation process can be aborted, allowing its computational resources to be diverted to other tasks. Section 6 concludes the paper by describing example applications in which the bounding anytime deflation technique may be of value, and related work by the author which better serves the purpose of optimising system performance in dynamic state spaces.

2 Discrete Planning Algorithms

Discrete algorithms operate in a discrete state space, and find a plan by 'visiting' states in this domain and assessing their suitability to achieve a goal. A requirement for a discrete algorithm is that it be 'systematic', meaning that in the limit as the number of iterations tends to infinity, the algorithm will assess every achievable state. For a finite state space, this provides a guarantee of 'completeness', meaning that a solution will be found in finite time if one exists, which is desirable for many applications. For an infinite state space, a systematic algorithm will still find a solution if it exists, but there are no bounds on the search time, and it may continue searching forever if there is no solution. Although there are not necessarily any bounds on the time required to compute a solution, they can often be determined if the domain is finite, and the number of states is known. In practice, it is typical to enforce a time-limit on the search, and to use heuristically guided planning algorithms (see below) to reduce the volume of the state space explored.

[LaValle, 2006] provides a template (depicted in Algorithm 1) for discrete planning algorithms and describes several, including; Breadth First Search, Depth First Search, Dijkstra's Algorithm, Best First Search, and A^*. In this procedure, Q is a sorted container of states, initially populated with the starting state for the search, x_I. States are marked as 'visited' (usually implemented as a separate container, with direct and fast access to elements) when they are first added to Q. The algorithm proceeds by extracting the first state in Q, failing if there are none, and succeeding if this state is the goal state, x_G. In all other cases, every action that can be taken from the extracted state, x, is assessed. The neighbouring state, x', is marked as visited and added to Q, with the ability to perform additional operations if x' is already visited.

Particular forwards search algorithms can be instantiated by specifying the sorting criteria of Q, and sequences of actions from start to goal can be found by augmenting each newly assessed state, x', with a marker noting its parent state, x. For example; Breadth First Search implements Q as a First-In, First-Out queue; Depth First Search implements it as a Last-In, First-Out stack. Dijkstra's Algorithm sorts Q by the 'cost-to-come' – the sum of the action costs between each pair of states in the trail of states from a particular x in Q back to the x_G. The A^* algorithm sorts Q by the sum of the cost-to-come and the expected 'cost-to-go', as determined by a function known as a 'heuristic'.

In many cases, the cost of a solution using a discrete algorithm can be provably optimal with respect to the state space (which itself is discrete and thus suboptimal with respect to the continuous domain it discretises). This is achieved by noting the dynamic programming principle, namely that any subset of the set of actions in an optimal plan is also optimal. A technique called 'value iteration' then allows the algorithm to store the optimal sub-plan to any particular state, iteratively building up the plans until the goal is achieved. Since the plans are built out of provably optimal sub-plans, the full plan to the goal is optimal. Again, [LaValle, 2006] has a thorough description, derivation, and proof of these concepts.

Algorithm 1 A general template for forward search.

```
1: Q.insert(x_I) and mark x_I as visited
2: while Q ≠ ∅ do
3:    x ← Q.GetFirst()
4:    if x = x_G then
5:       return SUCCESS
6:    end if
7:    for all u ∈ U(x) do
8:       x' ← f(x, u)
9:       if x' ∉ visited then
10:         Mark x' as visited
11:         Q.insert(x')
12:      else
13:         Resolve duplicate x'
14:      end if
15:   end for
16: end while
17: return FAILURE
```

For the A^* algorithm, for example, it is shown in [Hart *et al.*, 1968; 1972] and [Pearl, 1984] that optimality can be guaranteed if the heuristic function used by the A^* algorithm is 'admissible'. An admissible heuristic is one which is guaranteed to equal or under-estimate the true cost of the optimal plan between two states. The A^* algorithm uses the heuristic to restrict the number of states that must be evaluated to find the optimum solution plan, by only evaluating those states whose cost-to-come plus heuristic cost-to-go are lower than their neighbouring states. The A^* algorithm in particular is guaranteed to expand an equal number or fewer states than any other algorithm using the same heuristic [Pearl, 1984], meaning it is provably the best choice of optimal heuristic-guided graph search algorithm, provided that a suitable heuristic function is available.

3 Anytime Algorithms

As stated above, anytime algorithms are a class of algorithms which iteratively recompute an optimisation with decreasing upper bounds on the cost of the solution [Zilberstein and Russell, 1995]. In the case of heuristic graph search anytime planning algorithms, the key operating principle derives from the fact that inflating the heuristic by a scalar value $\varepsilon \geq 1$ (and rendering it inadmissible) results in an upper bound on the solution's

cost of ε times the optimal cost [Pearl, 1984], and typically reduces the planning time significantly [Pohl, 1970; Thayer and Ruml, 2008]. For sampling-based planners, a similar effect can be achieved by varying the density of the sampling strategy, as described above, or by using an anytime heuristic algorithm to search the sampled graph [Belghith *et al.*, 2006]. For other systems it is often possible to vary the resolution of the underlying data structure, or adjust system parameters which affect the accuracy of the plans. The purpose of each of these strategies is to allow tuning of the trade-off between the accuracy of the solutions, and the computational time required to produce them.

Figure 1 shows the effect of varying the heuristic inflation factor, for a 2D path planning scenario in a static state space. The paths produced by the Weighted-A^* algorithm are visualised for three different values of ε; 1.0, 2.0, and 4.0. The background image represents the state space, with the colour scale from black to white indicating velocity limits from 5.0 to $0.5m/s$ respectively. Completely white regions are considered to be full obstacles, and have a velocity limit of $0m/s$. The heuristic used is the distance to the goal divided by the maximum velocity (that is, the minimum expected execution time), and is thus an admissible function when uninflated. The cost function used is similarly the distance across a grid cell (in the direction of travel) divided by the velocity limit in this cell, cumulatively added along the path in question. Each grid cell is a square with side length of $0.25m$.

In domains where the execution of a sub-optimal plan with this upper bound is acceptable, one can design a system which continues improving the solution whilst making use of the previous iteration. The Anytime A^* algorithm [Ferguson *et al.*, 2005] and Anytime Heuristic Search [Hansen and Zhou, 2007] operate in this manner, but have the significant deficiency that they are not incremental, and all prior information is discarded when deflating the heuristic and beginning a subsequent iteration. The Anytime Repairing A^* algorithm [Likhachev *et al.*, 2003] corrects for this by retaining the algorithm's state across iterations, and efficiently reordering the internal data store to allow subsequent plans to perform the minimum computation required.

Further improvement is provided by the Anytime Dynamic A^* algorithm [Likhachev *et al.*, 2005a; 2005b] (also known as Anytime D^*), which is a backwards incremental form of the previous algorithm that can account for changes to the weights of edges in the graph. The state associated with each node is extended, by labelling each node either consistent or inconsistent, referring to whether or not the cost-to-come function is correctly propagated to this node. The algorithm's internal data store is then sorted by a function which accounts for these labels. This algorithm can also account for the starting point of a plan changing in any iteration, since the backwards incremental structure focusses the search around the goal [LaValle, 2006]. This feature is particularly useful for scenarios such as route finding for autonomous agents, whose movements continually alter the starting point of subsequent plans.

Anytime algorithms can be constructed in two forms, with subtly different results. Contract algorithms are run for a specified length of planning time and return the best solution found in this time (i.e. they are given a contract to return a plan in a given time) [Russell and Zilberstein, 1991]. Interruptible algorithms are run until the optimum solution is found, but can be interrupted at any point prior to this, returning the best solution found thus far [Dean and Boddy, 1988; Boddy and Dean, 1989]. Contract algorithms are typically easier to construct, but there exists a method to convert any contract algorithm into an interruptible form, by running it repeatedly with increased contract lengths each time [Zilberstein, 1996; Hansen and Zilberstein, 2001]. At least in principle, each of the heuristic graph search anytime algorithms described above can be constructed in an interruptible form.

4 Performance Assessment Metrics

This paper distinguishes planning *algorithms* – specific procedures to determine a sequence of actions that achieves a goal, from planning *systems* – techniques which make use of planning algorithms to determine a plan of action, which is then executed by some agent. This section describes metric functions which are used to assess the performance of both planning algorithms and planning systems. This distinction is important, and the terms 'internal metric' and 'system metric' are used to refer to each type of performance assessment, respectively.

An internal metric is a function that takes two states and returns a real scalar value:

$$M : X \times X \to \mathbb{R} \tag{1}$$

Typical examples are the Euclidean distance between two states, or the cost accrued by taking an action to transition from one to the other. [LaValle, 2006] has a thorough definition of the axioms required for a function to be a true internal metric, which include non-negativity and the triangle inequality, among others.

Certain classes of algorithms, notably those which compute optimal plans, require an internal metric in order to evaluate actions while computing the plan. In the case of optimal graph search algorithms such as A^*, D^*, and their variations, an admissible heuristic function of the same dimensionality as the metric is also required.

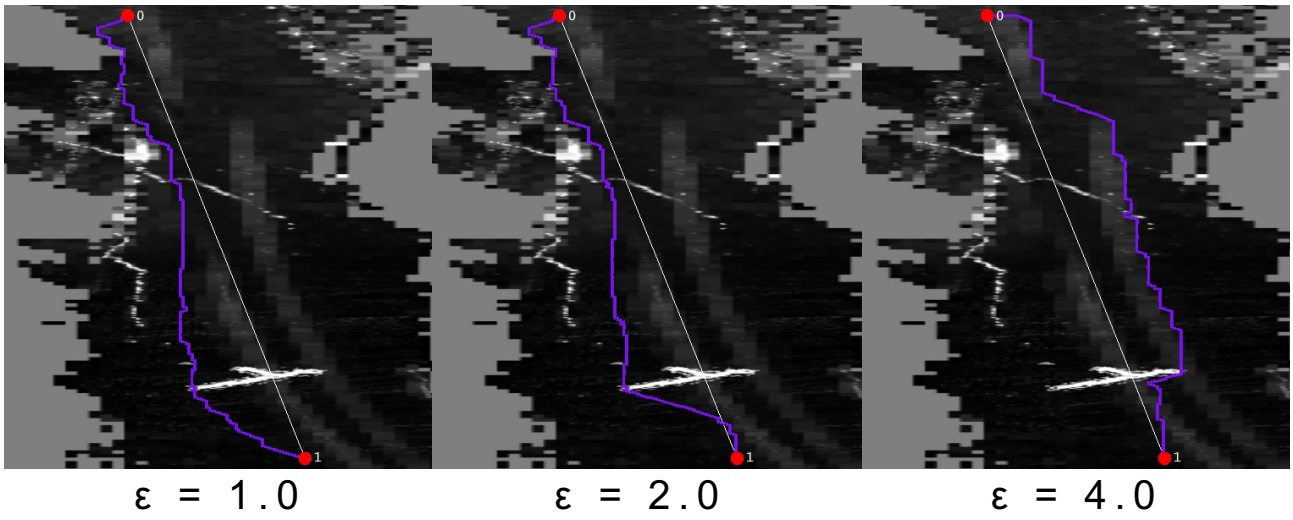

Figure 1: Visualisation of the effect of varying the heuristic inflation factor, for a 2D path planning scenario in a static state space. For high values of ε, the paths avoid obstacles (as these are unconnected vertices in the graph used by the planning algorithm), and have low planning times, but favour the heuristic approach of aiming directly for the goal too highly. As $\varepsilon \to 1$, the paths are closer to optimal, but require increasingly longer planning times.

As described above, an admissible heuristic is one which is guaranteed to equal or under-estimate the true cost of the optimal plan between two states. This is necessary because it is used to guide the search towards the goal, by indicating the best possible connection between two states, and only evaluating higher cost connections when all alternatives have been evaluated and found to be worse.

A system metric is a function that takes n variables, v_i, with associated weights, k_i, and returns a real scalar value that qualitatively describes the performance of the system:

$$m = M\left(k_1 \cdot v_1, \ldots, k_i \cdot v_i, \ldots, k_n \cdot v_n\right) \tag{2}$$

This definition is sufficient to cover system metrics that simply report the sum of the internal metric values for each action in a given plan, and also systems which assess values that an internal metric has no access to, such as the number of changes of direction required to execute a discrete 2D plan in a 4-connected grid, which depends on the sequence of actions rather than individuals.

While some planning scenarios will have no need of an internal metric, all must have a system metric in place. For example, a scenario with the objective of 'achieve a goal' can use any plan whose end state is a goal, without needing to determine how efficient or effective the plan was, and thus without an internal metric. However, a system metric is still required to specify whether or not a goal was achieved, even though the output is just success or failure. A system metric is a measure of how effectively the objectives of a scenario have been met. An internal metric assesses performance while the system is in use, and its results can provide a form of feedback that improves the final system metric value.

Time-focussed metrics are a special case of multi-parameter metrics, defined by the output of the metric being an 'uninflated real-time' quantity of time, where this phrase is used to indicate that the measure is not weighted by any other scalar factor. The most interesting form of time-focussed metrics for this paper are those for which the output is the expected execution time of the plan, as opposed to an inflated value that just happens to have units of time. This expected execution time is denoted $\hat{t_E}$, and is directly comparable with the required plan computation time, t_P, because they are both real-time measures.

The main benefit of such metrics is that they provide the ability to trade-off these two measures, to reduce the total time expended in achieving the goal, which is a function of both t_P and $\hat{t_E}$, determined by a time-focussed system metric. This is a significant difference compared to all other metrics which can only trade-off at best between t_P, and the expected cost of execution, $\hat{c_E}$. Systems without time-focussed internal metrics require a separate system metric to determine the value of any given distribution of planning time and execution cost. Systems employing a time-focussed metric can determine this trivially, since t_P and $\hat{t_E}$ have the same units, and their sum is a scalar value.

Figure 2 demonstrates this concept, showing the effect of varying the heuristic inflation factor, ε, for the same static path planning scenario shown in Figure 1, using

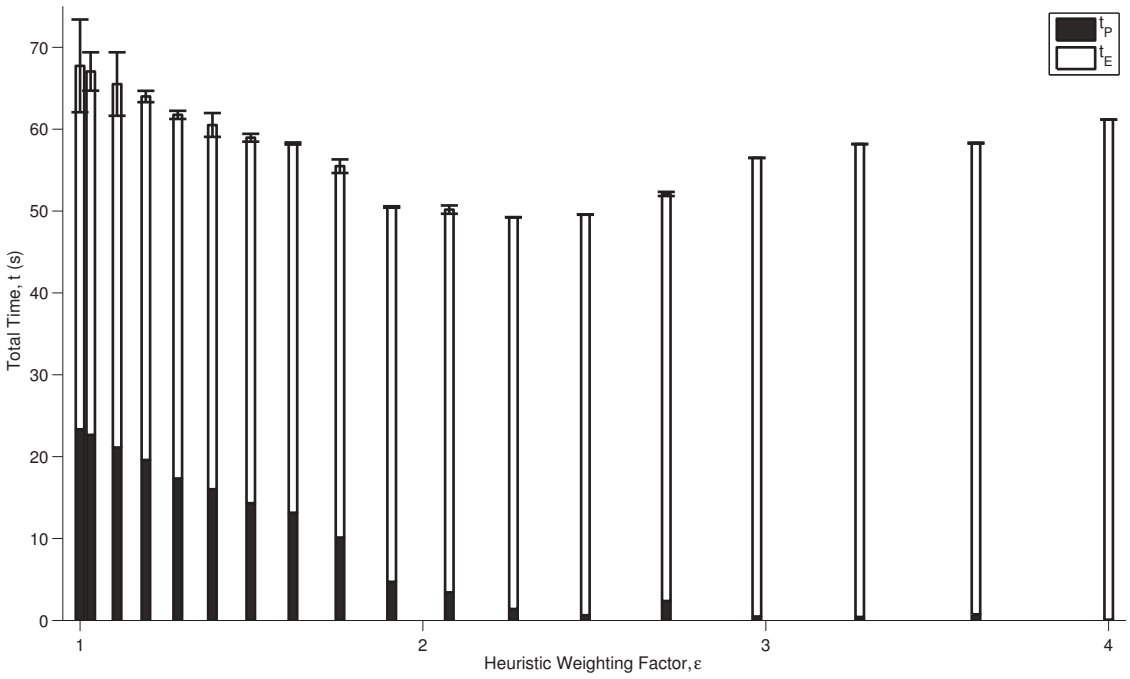

Figure 2: Total time required to achieve a goal in a static 2D path planning scenario versus the heuristic inflation factor, ε, using the Weighted-A^* algorithm. The error bars show $\pm 2\sigma$ from the mean t_{Total} after 10 repetitions. This variation is solely due to the variation in t_P, since the algorithm is deterministic and there is no variation in t_E.

the Weighted-A^* algorithm. The heuristic inflation factor values form a set ranging from 1 to 4 generated by starting from 4, raising the previous value to the power of 0.95, and including the uninflated heuristic where $\varepsilon = 1$. The results show that in this scenario, the values of ε that results in the lowest total expected time to achieve a goal (given this discrete set of options) is 2.264. The variation in t_P observed in these results is due to the planning algorithm being run on a standard desktop PC and multi-purpose operating system. Were it to be run on a dedicated processor it is expected that the variance would be minimal since the algorithm is deterministic.

Performing the same experiment on a computer with different computational power would result in a different set of results, but the use of a time-focussed metric allows analysis of the best performing ε value. The significance of this figure is not in the numeric values of the results, but rather in the demonstration that a time-focussed metric enables the selection of parameters (such as the heuristic inflation factor) that minimise the expected total time.

5 Bounded Anytime Deflation

This section describes a novel technique to determine when to abort the deflation process within an anytime planning algorithm. The technique relies on the fact that when using an anytime algorithm, the inflation factor, ε, provides an upper bound on the cost of the solution, c, as compared to the optimum (minimum) cost, c^*. However, this optimum cost is never known until the planner deflates to $\varepsilon = 1$, a value which may be computationally infeasible to achieve. This section demonstrates that there exists a simple solution to continually bound the optimum cost, and to do so in a manner which provides an admissible heuristic for this optimum cost, since it is shown to never overestimate.

5.1 Admissible Cost Estimation

After the i-th iteration of an anytime search algorithm in a static state space, a plan has been computed with cost c_i, given a heuristic inflation factor of ε_i. It is shown in [Pearl, 1984] that a weighted graph search plan with weighting factor ε results in a cost, $c \leq c^* \times \varepsilon$, and each iteration of an anytime algorithm can be considered as a single weighted graph search plan. Given this, it can be deduced that the optimum cost is always greater than or equal to the cost of a plan in any iteration, divided by the heuristic inflation factor in the same iteration. Thus, one can calculate a lower bound on the true optimum cost in any iteration, i, as:

$$c_i^* \geq \frac{c_i}{\varepsilon_i} \tag{3}$$

In subsequent iteration $i+1$, with $\varepsilon_{i+1} < \varepsilon_i$, the cost c_{i+1} could potentially be higher, lower, or equal to c_i

(since anytime planning only reduces the upper bound on the solution cost in each iteration, not necessarily the cost). As a result, the newly determined lower bound on optimal cost could be higher or lower than the previous, but is only maintained if it is higher. Thus the lower bound on optimum cost after iteration i is:

$$c_i^* \geq max \left\{ c_{i-1}^*, \frac{c_i}{\varepsilon_i} \right\} \tag{4}$$

with $c_0^* = -\infty$. This approach provides a non-decreasing estimate of the lower bound on solution cost. The following theorem shows that c_i^* will also never overestimate c^*.

Theorem 1. *For all i, $c_i^* \leq c^*$.*

Proof. After iteration n with $\varepsilon_n = 1$, $c_n = c^*$ by definition. Assume the theorem is incorrect, and that $c_n^* > c^*$. Since $c_n^* = \frac{c_n}{\varepsilon_n}$ this implies $c_n > c^* \times \varepsilon_n$. But $\varepsilon_n = 1$, and $c_n = c^*$, giving $c^* > c^*$. Thus the assumption is invalid and $c_n^* \leq c^*$. Since the sequence of c_i^* is non-decreasing for all i, and true for $i = n$, the theorem holds. □

Equation 4 provides an estimate of the lower bound on the optimum cost of the plan. The proof then shows that this estimate is admissible, meaning that it is guaranteed to never overestimate the optimum cost. The utility of this bound is that it provides a quantitative measure of the potential performance of the best possible plan, and the bound gets closer to the true cost as the anytime deflation proceeds.

Figure 3 illustrates one example of this bounding process using the same static planning scenario visualised in Figure 1. The planning system uses the Anytime-D^* algorithm, with a cost grid cell size of $0.25m$, and ε deflated from 4.0 to 1.0. The planning system also uses a time-focussed metric meaning that the cost is the expected execution time.

The figure shows the expected execution time as determined by each individual plan in blue, versus the cumulative planning time taken by the anytime deflation process. The dashed black line shows the lower bound on this expected execution time as determined by the bounding process, versus the cumulative planning time. The dotted horizontal line shows the true optimum expected execution time, which is only found once the plan with $\varepsilon = 1$ is completed, after a total delation process time of $56.1s$. All of these measures are against the left vertical axis, which is in units of expected execution time in seconds. The cost bound is seen to change to a new value, greater than or equal to the previous value, whenever a plan with reduced ε is computed. The time at which each subsequent plan is fully computed is shown by a black circle on the cost bound line. Again, the significance of this figure is not in the numeric values of the result, but rather the demonstration of the bounding process versus the cumulative planning time.

5.2 Aborted Anytime Planning

If a planning system uses a time-focussed metric however, as in the above example, the lower cost bound indicates the minimum expected execution time as determined in each iteration of the planning process. Since the planning time and expected execution time are comparable with a time-focussed metric, the bounding process can be used to determine when to abandon further anytime deflation, as follows.

After the i-th iteration of an anytime search algorithm, a plan has been computed in time t_{P_i} and with expected execution time $\hat{t_{E_i}}$, given suboptimality bound ε_i. As given by Equation 4, an admissible lower bound for the optimal expected execution time $\hat{t^*_{E_i}}$ is also available. Since the goal is expected to be acquired after a further $\hat{t_{E_i}}$ of time, anytime planning should be abandoned when the planning time in iteration $i+1$ exceeds $\hat{t_{E_i}} - \hat{t^*_{E_i}}$.

The difference between the expected execution time of the i-th plan, and the highest known lower bound on the optimal expected execution time, represents the possible benefit from continued planning. If computation requires more time than will be saved at execution, it is not worth continuing to plan. A planning system which incorporates this check can save up to $\hat{t^*_{E_i}}$ of planning time, which can potentially be reinvested into other tasks. Compared to a traditional anytime algorithm, this strategy yields the same total time to achieve a goal, but requires less planning time to do so if $\varepsilon_i > 1$ for all i.

This strategy is suited to static state spaces where planning is first performed to completion (or to a suitable quality ratio), and then execution is initiated. With one minor amendment, it is also suitable to replanning scenarios where planning and execution are performed simultaneously. At each iteration, i, the amount of time for which the previous plan was actually executed, $t_{e_{i-1}}$, is subtracted from the bound on expected optimal execution time. Equation 4 thus becomes:

$$\hat{t^*_{E_i}} \geq max \left\{ \hat{t^*_{E_{i-1}}} - t_{e_{i-1}}, \frac{\hat{t_{E_i}}}{\varepsilon_i} \right\} \tag{5}$$

with all c replaced by $\hat{t_E}$. Since each plan is executed for only the amount of time until the next plan is available, that is:

$$t_{e_{i-1}} = t_{P_i}, \tag{6}$$

this gives:

$$\hat{t^*_{E_i}} \geq max \left\{ \hat{t^*_{E_{i-1}}} - t_{P_i}, \frac{\hat{t_{E_i}}}{\varepsilon_i} \right\}. \tag{7}$$

This is valid even for agents that suffer imperfect execution – meaning they may be incapable of executing

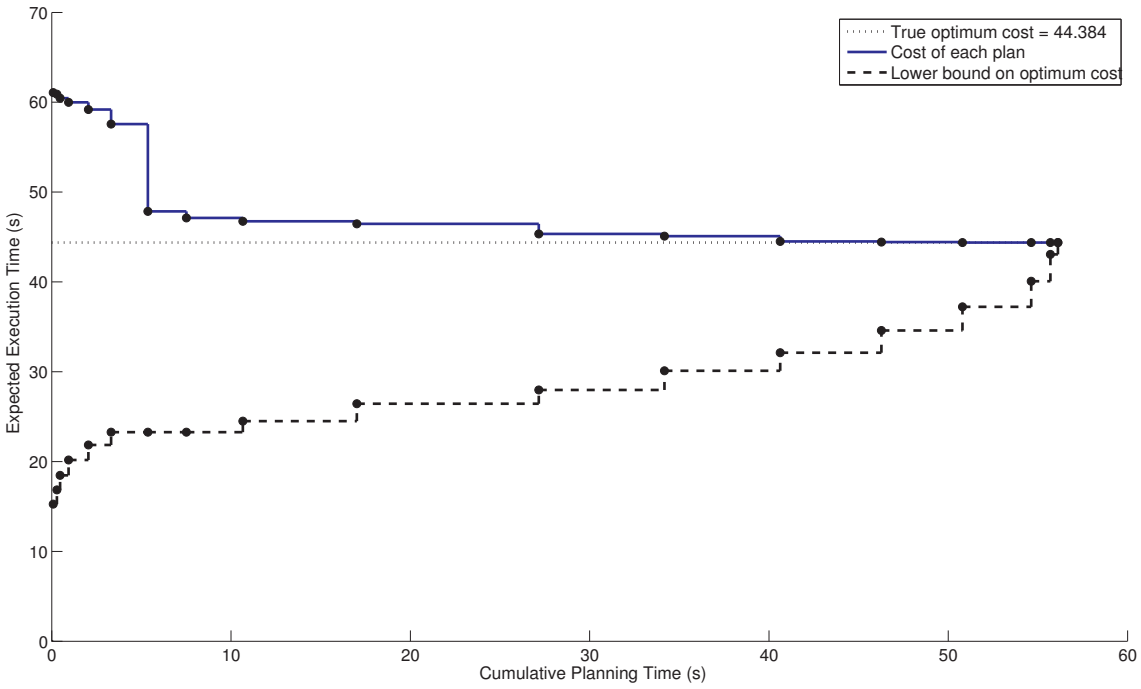

Figure 3: Illustration of the process of calculating an admissible bound on the expected optimal cost available from an anytime algorithm. The optimum expected execution time for this particular example process is $44.4s$, and is found after the plan with $\varepsilon = 1$ is completed, after a total delation process time of $56.1s$.

a plan exactly as predicted[1] – because the cost function is employed in the anytime algorithm by means of an admissible heuristic, which is then inflated. The optimum expected execution time, $\hat{t^*_E}$, is an underestimate, as required, and any imperfect execution can only result in an increase in execution time. Thus $\hat{t^*_{E_{i-1}}} - t_{P_i}$ is still guaranteed to be an admissible lower bound, and cannot overestimate.

Unfortunately, this strategy is not applicable to dynamic state spaces, for two reasons. Firstly, as information changes in the state space the optimum cost estimate must be discarded. This is due to the fact that the changes could result in the optimum path having higher or lower cost, and thus the sequence of c^*_i cannot be guaranteed to be non-increasing. Although the sequence could be made non-increasing by setting $c^*_i = -\infty$ every time information in the state space changes, it is likely that the rate at which information changes is higher than the planning rate, making the strategy redundant. Secondly, in a dynamic state space imperfect execution of a plan could have positive effects, and potentially reduce the time required to achieve a goal. Again, this means the sequence of c^*_i cannot be not guaranteed to be non-increasing, making the strategy unsuitable for dynamic state spaces.

[1]Formally, this is the case when $\hat{c_E} < c_E$, where $\hat{c_E}$ is the expected cost of execution as determined by the internal metric of the planning algorithm, and c_E is the actual cost of execution determined once a goal is achieved.

6 Conclusions

This paper presented a technique to determine a lower bound on the cost of the final plan in an anytime deflation process, while the algorithm is in operation. It was proved that this bound was admissible, and it was shown that in conjunction with a time-focussed metric, this bound could be used to determine when to abort the anytime process. This latter technique was found to be unsuitable for use in dynamic state spaces, making it less useful for applications such as the planning scenarios used as examples in this paper.

Despite this, the technique to calculate an admissible lower bound on the solution cost of an anytime plan is valid in both static and dynamic state spaces. It has use in applications such as operator displays that indicate the expected time remaining to achieve a goal, or estimation processes in which knowledge of the maximum benefit to be obtained may be relevant.

Related work by the author in [Allen, 2010] (currently under review) has developed a technique for online adjustment of system parameters using a time-focussed metric to continually optimise the total planning and execution time. The technique is suitable for dynamic state spaces and employs a similar method to that used to gen-

erate Figure 2, but for parameters other than ε, and for multiple parameters simultaneously. This process is performed at every iteration of the planning process (similar to bounding the optimum cost at every iteration of an anytime process), has been shown to be capable of significantly reducing the total time required to achieve a goal as compared to existing techniques.

In conclusion, although the aborted anytime planning technique is unsuited to dynamic state spaces, anytime algorithms themselves have applications in many fields. Since the anytime cost bounding technique is not specific to planning algorithms, it may have utility in applications outside the planning domain.

Acknowledgments

This work is supported in part by the Australian Research Council (ARC) Centre of Excellence programme, funded by the ARC and the New South Wales State Government.

References

[Allen *et al.*, 2009] T. Allen, A. Hill, J. Underwood, and S. Scheding. Dynamic path planning with multi-agent data fusion - the parallel hierarchical replanner. In *Proceedings of the International Conference on Robotics and Automation*, 2009.

[Allen, 2010] T. Allen. Time-optimal active decision making. Ph.D. Thesis (Submitted, under review), September 2010.

[Belghith *et al.*, 2006] K. Belghith, F. Kabanza, L. Hartman, and R. Nkambou. Anytime dynamic path-planning with flexible probabilistic roadmaps. In *IEEE International Conference on Robotics and Automation*, 2006.

[Boddy and Dean, 1989] M. Boddy and T. Dean. Solving time-dependent planning problems. In *International Joint Conference On Artificial Intelligence*, 1989.

[Dean and Boddy, 1988] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Association for the Advancement of Artificial Intelligence*, 1988.

[Ferguson *et al.*, 2005] D. Ferguson, M. Likhachev, and A. Stentz. A guide to heuristic-based path planning. In *Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS)*, June 2005.

[Hansen and Zhou, 2007] E. Hansen and R. Zhou. Anytime heuristic search. *Journal of Articial Intelligence Research*, 28:267–297, 2007.

[Hansen and Zilberstein, 2001] E. Hansen and S. Zilberstein. Montioring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126:139–157, 2001.

[Hart *et al.*, 1968] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 2:100–107, 1968.

[Hart *et al.*, 1972] P. Hart, N. Nilsson, and B. Raphael. Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *SIGART Newsletter*, 37:28–29, 1972.

[LaValle, 2006] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, UK, 2006. Available from `http://planning.cs.uiuc.edu/`.

[Likhachev *et al.*, 2003] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Neural Information Processing Systems*, 2003.

[Likhachev *et al.*, 2005a] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, June 2005.

[Likhachev *et al.*, 2005b] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic A*: The proofs. Technical Report CMU-RI-TR-05-12, Robotics Institute, Pittsburgh, PA, May 2005.

[Pearl, 1984] J. Pearl. *Heuristics*. Addison-Wesley, 1984.

[Pohl, 1970] I. Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1:193–204, 1970.

[Russell and Zilberstein, 1991] S. Russell and S. Zilberstein. Composing real-time systems. In *International Joint Conference on Artificial Intelligence*, pages 212–217, 1991.

[Thayer and Ruml, 2008] J. Thayer and W. Ruml. Faster than weighted A*: An optimistic approach to bounded suboptimal search. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, 2008.

[Zilberstein and Russell, 1995] S. Zilberstein and S. Russell. *Imprecise and Approximate Computation*, chapter Approximate Reasoning using Anytime Algorithms. Kluwer Academic Publishers, 1995.

[Zilberstein, 1996] S. Zilberstein. Using anytime algorithms in intelligent systems. *Artificial Intelligence Magazine*, 17:73–83, 1996.