

# Dynamic Step Size Adjustment in Iterative Deepening Search

Daniel Sheridan

University of Edinburgh, Kings Buildings, Edinburgh, UK  
djsheridan@sms.ed.ac.uk

**Abstract.** If an iterative procedure has the property that solutions at a given iteration are also found at later iterations, it is correctness-preserving to skip iterations. We will look at the conditions required for skipping to be worthwhile and discuss methods for approximating the optimal step size while the procedure progresses.

## 1 Introduction

The iterative deepening search algorithm (IDS) forms the basis of many useful systems, including planning and bounded model checking [1]. The process of iteratively trying to solve a problem of increasing size can be time consuming, and the ideal algorithm would choose just the iteration at which the solution is first found. While this is, of course, impossible, we can consider methods of reducing the number of iterations taken to reach the goal.

The approach taken in the present work is to compute which iterations may be omitted without a negative impact on the run-time. For example, an algorithm which tries only problem depths 10, 20, 30, . . . will be quicker at finding a solution which is first seen at depth 100 than an algorithm which tests all depths. On the other hand, if the solution lay at depth 1, this algorithm would require solving a potentially much harder problem (at depth 10) than would otherwise be required. We will discuss heuristics for dealing with this issue, and minimising the amount of wasted time.

### 1.1 Background

Consider the problem  $f$ , which has solution  $\pi$ , written  $\pi \models f$ . We will attempt to solve the problem with an iterative procedure which restricts the amount of the search space under consideration. We write the depth  $i$  as  $\pi \models_i f$ . We will refer to the time taken to solve a problem  $f$  at depth  $i$  as  $T(f, i)$ . We now make explicit the assumptions under which we will construct the framework for skipping steps: we assume that IDS is a valid procedure for the problem, that skipping steps may be done without loss of generality, and that the hardness of the problem continues to increase as the depth of solving is increased.

*Assumption 1.* If  $f$  has a solution, this solution may be found by iterative deepening search to some depth  $k$ :

$$\pi \models f \rightarrow \exists k \cdot \pi \models_k f$$

*Assumption 2.* If  $f$  has a solution at depth  $i$  then it is solvable at all greater depths:

$$\pi \models_i f \rightarrow \forall j \geq i, \pi \models_j f$$

*Assumption 3.*  $T(f, i)$  is monotonically increasing with  $i$ :

$$\forall j \geq i, T(f, j) \geq T(f, i)$$

## 2 Conditions for step size increases

In order to decide on the size of a step to be taken during search, we must consider the circumstances under which a particular step size will save time overall.

Suppose we are currently at depth  $i$  during the IDS. It is preferable to solve  $\pi \models_{i+\Delta} f$  next rather than the sequence  $\forall_{j=i..n} \pi \models_j f$  iff

$$i + \Delta > n \tag{1}$$

and

$$T(f, i + \Delta) < \sum_{j=i+1}^n T(f, j) \tag{2}$$

That is, when skipping to some deeper position is greater than computing all of the steps required to reach some intermediate position. We wish first to solve this inequation, and eventually to maximise the time difference. In order to do this, we must make certain parts of the inequality more concrete.

### 2.1 Intermediate Position

Suppose that we know that the first solution of  $f$  lies at depth  $k$ . The best algorithm would minimise the number of iterations taken in order to reach this depth. If the algorithm is currently at depth  $i < k$  then making the next iteration at depth  $k + \Delta$  is likely to be worse than simply computing all of the intermediate steps up to  $k$ . In this case, the ‘intermediate position’  $n$  in Condition 2 is  $k$ : skipping steps is only worthwhile if it is quicker than performing the iterations up to this point.

Obviously, we cannot, in general, know what value to pick for  $n$  — otherwise we could simply make that the first iteration. For the remainder of this paper, we will use the following heuristic to choose the intermediate point:

The first solution of  $f$  is equally likely to lie at any depth  $k, 0 \leq k < \infty$ .

Thus we take  $n$  in Condition 2 as

$$n = i + \left\lceil \frac{\Delta}{2} \right\rceil \quad (3)$$

We could conceive of other possibly more effective heuristics, considering for example the increase in the number of nodes in the search space with increasing depth, and we shall see later how a heuristic may need to take into account extra time taken to refine a solution if the *smallest* solution is required..

## 2.2 Time at Depth

To be able to solve the inequalities we need a more concrete idea of the time taken at a given iteration depth. By observation of the original target domain of the present method (bounded model checking) we choose to approximate  $T(f, i)$  as the exponential

$$T(f, i) \approx ba^i \quad (4)$$

This approximation will obviously depend on the problem domain, however exponential growth is common in the type of problems that are solved with IDS. We can also make an intuitive observation about the applicability of the method with this approximation: it is clear that, whatever the value  $n$ , if  $a = 2$ , there is no value of  $\Delta$  which satisfies Condition 2. This follows from the observation that at depth  $i + \Delta$ , the problem will take twice as long as at the largest value of  $n$  allowed by Condition 1,  $i + \Delta - 1$ , and as the intermediate iterations up to  $n - 1$  will take strictly less time than this, the total time taken to reach depth  $n$  iteratively must be less than the time taken if jumping to depth  $i + \Delta$ .

## 2.3 Solving the Inequations

We can now proceed to a solution to the conditions. By substituting the approximation of Equation 4 and the heuristic of Equation 3 into Condition 1 we obtain the new inequation

$$ba^{i+\Delta} < \sum_{j=i+1}^{i+\lceil \frac{\Delta}{2} \rceil} ba^j \quad \text{or} \quad a^\Delta < \frac{a^{\lceil \frac{\Delta}{2} \rceil + 1} - a}{a - 1} \quad (5)$$

by forming the sum of the geometric progression. In order to determine the circumstances under which a particular jump may be made, we must solve the inequality for given values of  $\Delta$ . Numerical methods give us the results in Table 1.

## 3 Dynamic Step Size Adjustment

While we can use the approximation of  $T(f, i)$  given in Equation 4 to predict future behaviour of the algorithm, we can also use the past behaviour of the algorithm to determine values for the constants in the approximation — so they

**Table 1.** Numerical solutions to the conditions of skipping

Step size ( $\Delta$ )	Intermediate point ( $n$ )	Exponential ( $a$ )
2	1	1.618
4	2	1.466
6	3	1.380
8	4	1.325
10	5	1.285

do not need to be determined in advance. This, in effect, makes the algorithm ‘self-analysing’, adapting its behaviour to the current problem.

The ideal next step is chosen by maximising the difference between the left and right sides of the inequality in Equation 5. We propose the following algorithm for IDS with dynamic step size adjustment.

- Initialise constants  $a, b \leftarrow \infty$
- Initialise the list of past behaviour  $B \leftarrow []$
- Initialise the current solving depth  $i \leftarrow 0$
- Until a solution is found, loop:
  - Solve  $\pi \models_i f$ , recording the time taken in  $t$
  - Append the pair  $\langle i, t \rangle$  to  $B$
  - Use best-fit on  $B$  to determine  $a$  and  $b$
  - Using Equation 5, determining new  $i$

## 4 Finding the Smallest Example

In some of the problem domains for IDS, we would like to be able to find the *smallest* depth at which the problem is solvable: the shortest plan in a planning problem, or the shortest counterexample in a model checking problem. Indeed, this may be one reason for choosing to use IDS in the first place. More formally, if the latest iteration is the first we have seen which is solvable, we set  $k^+$ , the position of the smallest known solution, to the current depth, and  $k^-$ , the position of the largest known non-solution, to the depth at the previous iteration. We now want the smallest  $i$ ,

$$k^- < i \leq k^+ \text{ such that } \pi \models_i f$$

While a binary search at this point is known to take worst-case  $\log_2(k^+ - k^-)$  steps, the time for each step depends on what decision is taken. A step which moves deeper in the search space is more expensive than one which moves shallower; a balanced, minimum-time search algorithm must therefore be biased towards the deeper end of the search space. We propose a weighted binary search: given current  $k^-$  and  $k^+$ , we wish to choose the next point  $i, k^- < i \leq k^+$  based on the predicted time taken for to complete the search. If  $\pi \models_i f$  then the search will proceed to the shallower part of the search space, and if  $\pi \not\models_i f$  the search will proceed to the deeper part of the search space. We would like the time in

both cases to be the same. If we approximate the remainder of the search with a standard binary search, the cases become

$$\sum_{j=1}^{\log_2(i-k^-)-1} ba^{k^-+2^j} \quad \text{and} \quad \sum_{j=1}^{\log_2(k^+-i)-1} ba^{k^+-2^j}$$

By rearranging, we obtain the following equation, which is easily solvable during search:

$$a^{\frac{1}{2}(i-k^-)} = \frac{1 + a^{k^+-k^- - 2}}{1 + a^{\frac{1}{2}(k^+-k^-)}}$$

#### 4.1 Interaction with Heuristics

The heuristic in Equation 3 balances the cost of skipping to a deeper, hence harder, problem instance against the possibility of finding a at a shallower point. However, the heuristic does not take into account the extra time taken to find the shortest solution by weighted binary search. This could be used to refine the heuristic.

## 5 Summary and Further Work

We have presented a method for speeding up iterative deepening search under certain specific circumstances, which are known to occur in real problems such as bounded model checking. The method involves dynamically measuring the behaviour of the solving algorithm on each iteration in order to predict the future behaviour, together with a heuristic to decide, on this basis, whether iterations can be omitted from the search. The dynamic behaviour has the side effect of adapting the algorithm to the problem: if the conditions set out in this work are not met, there is no performance degradation: the search proceeds as normal.

A preliminary experimental evaluation demonstrates the efficacy of this method on bounded model checking problems; however, other iterative-deepening-style problems must be tried. While the simple proposed heuristic is effective in this case, it may be insufficiently accurate for other problem classes.

Finally, we note that this work provides a method for leveraging small improvements in the solving algorithms. A small reduction in the exponential growth rate of solver can translate into bigger steps during search, and hence yet bigger savings of time.

## References

1. Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In W.R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems. 5th International Conference, TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag Inc., July 1999.