Terminating Decision Algorithms Optimally

Tuomas Sandholm
sandholm@cs.cmu.edu
Computer Science Department
Carnegie Mellon University
Pittsburgh PA 15213

Abstract

Incomplete decision algorithms can often solve larger problem instances than complete ones. The drawback is that one does not know whether the algorithm will finish soon, later, or never. This paper presents a general decision-theoretic method for optimally terminating such algorithms. The stopping policy is computed based on a prior probability of the answer, a payoff model describing the value that different probability estimates would provide at different times, and the algorithm's run-time distribution. We present a linear-time algorithm for determining the optimal stopping policy given a finite cap on the number of algorithm steps. We exemplify this in a manufacturing scenario with a 3-satisfiability problem. To increase accuracy, the initial satisfiability probability and the run-time distribution are conditioned on features of the instance. The expectation of the result at each future time step is computed using Bayesian updating. We then extend the framework to settings where no exogenous cap is given on the number of algorithm steps. The method also provides a normative basis for algorithm selection. Finally, our method can be used to terminate/select complete algorithms optimally as well. [1,2]

1 Introduction

Decision problems are problems where the answer is either yes (Y) or no (N). Such problems are central to computer science and ubiquitous in the world. A *decision algorithm* is an algorithm that determines the answer to such a problem. A *complete decision algorithm* is a decision algorithm that always gives the answer in finite time. An *incomplete decision algorithm* is a decision algorithm that never finishes if the answer is N, and may or may not finish if the answer is Y. So, if such an algorithm finishes, the answer is Y.

[1]A short, very tentative version of this paper appeared in a workshop [22].

[2]This material is based upon work supported by the NSF under CAREER Award IRI-9703122, Grant IIS-9800994, ITR IIS-0081246, and ITR IIS-0121678.

Incomplete algorithms are important because they can often solve significantly larger problem instances than complete algorithms. Commonly the user of an incomplete algorithm initiates its execution, and after a while gets tired of waiting for a solution. She may be tempted to terminate the algorithm. At the same time she knows that the algorithm might finish, and that this might occur even in the very next step. Should she terminate the algorithm?

This paper presents a method for optimally determining when the algorithm should be terminated if it has not found a solution. Our decision-theoretic method computes the value of the information that different steps of running the incomplete algorithm are likely to provide, and uses that information to decide when to terminate the algorithm.

The first key observation is that incomplete algorithms are iterative refinement algorithms and approximation algorithms in that over time they implicitly refine a probability estimate that a solution exists. Let us define the following symbols:

SOL_t ="Solution found by time t" (so, if a solution is found at time t, then $SOL_{t'} = 1$ for all $t' \geq t$), and

$NOSOL_t$ ="No solution found by time t".

The iterative refinement algorithm emerges when we realize that the probability of the answer being Y decreases with the number of steps that the algorithm has executed (unless the algorithm halts which guarantees that the answer is Y). This probability, $p(Y|NOSOL_t)$, can be computed using a statistical performance profile, $p(SOL_t|Y)$, of the algorithm, i.e., the probability of finding a solution by time t given that a solution exists. The performance profile can be constructed from prior runs of the algorithm as we will describe.

We use 3-satisfiability (3SAT) as our example decision problem for the following reasons among others:

- Incomplete algorithms for 3SAT have recently been shown to solve problem instances that are reduced to 3SAT from other problems (such as planning [15]) more efficiently than algorithms that are tailored for solving those other problems directly.
- Incomplete algorithms for 3SAT [6, 24] scale up to significantly larger problem instances than even the best current complete algorithms [3, 23]. Therefore, incomplete algorithms for 3SAT are crucial from a practical perspective, and it is important to be able to terminate them efficiently.

- 3SAT is perhaps the most central decision problem in computer science. For one, it was the first problem to be proven $\mathcal{NP}$-complete.

3SAT is the decision problem of whether a satisfying truth assignment exists for variables in a 3CNF formula. A 3CNF formula is a conjunct of clauses where each clause is a disjunct of 3 literals. A literal is a negated or non-negated variable. The formula is satisfiable—corresponding to answer Y in our setting—if the variables can be assigned Boolean values so that the formula evaluates to true. For example, the formula $(v_1 \vee v_2 \vee v_3) \wedge (v_1 \vee v_3 \vee v_4) \wedge (v_1 \vee v_2 \vee v_4)$ is satisfiable by the truth assignment $v_1 = true, v_2 = true, v_3 = true, v_4 = false$ (among others).

The paper is organized as follows. Section 2 derives our method for optimally terminating incomplete decision algorithms given a finite cap on the number of possible algorithmic steps. Section 3 presents an example of how this method can be used. Section 4 generalizes the method to settings where no cap on the possible number of steps is exogenously given. Section 5 summarizes related research. Section 6 discusses pragmatics and alternative uses of the method.

2 Method for terminating decision algorithms

This section presents a method for optimally terminating an incomplete decision algorithm. The incomplete algorithm is used to update the probability estimate of the answer being Y. Based on a run-time distribution of the algorithm, an agent can anticipate how this estimate will change as more time is allocated to the algorithm. The agent can also anticipate its expected payoff in the real world given that it will act based on the probability estimate available at the time of action (the probability will be 1 if the algorithm happens to find a solution). With these two models, the agent can calculate the optimal time to terminate the algorithm.

Terminating optimally seems difficult because all of the following concerns have to be taken into account:

- Further computation adds value because it can cause the algorithm to find a solution. This is nontrivial to analyze because the probability of finding a solution at a given future time step changes based on how many unsuccessful steps the algorithm has executed. For example, at step 0, step 905 may look unprofitable while at step 708, step 905 may well look profitable. Alternatively, at step 0, step 905 may look profitable while at step 708, step 905 may look unprofitable.
- Further computation adds value because it refines the probability that a solution exists even if the algorithm does not terminate. The probability that a solution exists decreases as the algorithm takes unsuccessful steps.
- As this probability estimate gets refined, it can be used to make future termination/continuation decisions. Therefore, these decisions can be made with better information than what is available at the outset. The fact that such new information is valuable due to this reason is yet another motivation to execute the algorithm further.
- The payoff from a given probability estimate that a solution exists (this probability is 1 if a solution has been found) decreases with time because the agent misses the opportunities of using the answer earlier in the agent's choice of what to do in the world.
- Further computation adds to the computational cost.
- If the deliberation controller has let the algorithm execute past the optimal termination time, it can be optimal to let it execute even further since the losses incurred so far have become sunk cost.
- In some cases, the agent's expected payoff is maximized by never terminating the algorithm (unless the algorithm finishes, i.e., determines that the answer is Y).

It turns out that all of these factors can be soundly taken into account. The method that we present does this in a Bayesian framework and leads to an optimal termination decision. Specifically, the problem is that of finding an optimal policy for the deliberation controller, i.e., deciding what the agent should do in each of the $\max$ nodes in Figure 1.

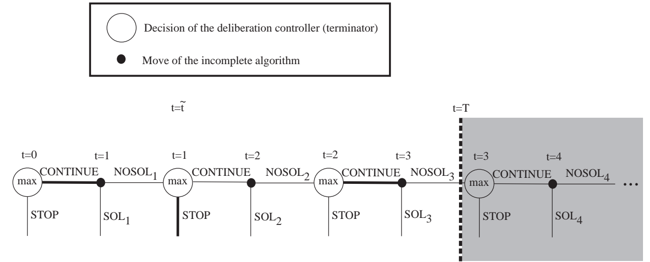

Figure 1: *Deliberation controller's decision tree. The bold lines show an example policy where the deliberation controller will terminate the algorithm at time $t = \tilde{t} = 1$ if the algorithm has not found a solution. The gray area and T will be discussed in Section 4.*

The rest of this section presents a method for determining an optimal termination policy in linear time. To act according to the policy, the deliberation controller simply remembers the first time when it is better to stop than to continue, and if the algorithm has not finished by that time, the deliberation controller terminates the algorithm. If the algorithm finishes before that, it is obviously not worth continuing.

2.1 Conditional performance profiles: Probability updates using a run-time distribution

To determine when to terminate, the deliberation controller needs to know how the probability of finding a solution by any given time changes based on how many steps the algorithm has executed so far without finding a solution. Let τ_1 and τ_2 be arbitrary times such that $\tau_1 \leq \tau_2$. We are interested in determining the quantity $p(SOL_{\tau_2}|NOSOL_{\tau_1})$. Trivially,

$$p(SOL_{\tau_2}|NOSOL_{\tau_1}) = 1 - p(NOSOL_{\tau_2}|NOSOL_{\tau_1}) \quad (1)$$

The right hand side can be solved using the definition of conditional probability to get

$$p(NOSOL_{\tau_2}|NOSOL_{\tau_1}) = \frac{p(NOSOL_{\tau_2} \wedge NOSOL_{\tau_1})}{p(NOSOL_{\tau_1})} \quad (2)$$

Because $p(NOSOL_{\tau_2} \wedge NOSOL_{\tau_1}) = p(NOSOL_{\tau_2})$, this can be simplified to

$$p(NOSOL_{\tau_2}|NOSOL_{\tau_1}) = \frac{p(NOSOL_{\tau_2})}{p(NOSOL_{\tau_1})} \quad (3)$$

which can be solved using

$$p(NOSOL_t) = p(Y)p(NOSOL_t|Y) + p(N)p(NOSOL_t|N) \quad (4)$$

Using the fact that $p(N) = 1 - p(Y)$ and the fact that the algorithm never finishes if no solution exists, i.e., $p(NOSOL_t|N) = 1$, the above equation can be rewritten:

$$p(NOSOL_t) = p(Y)p(NOSOL_t|Y) + 1 - p(Y) \quad (5)$$

The termination algorithm also needs to know the chance that the answer is Y given that no solution has been found by step t. This can be determined using Bayes rule:

$$\begin{aligned} p(Y|NOSOL_t) &= \frac{p(Y)p(NOSOL_t|Y)}{p(Y)p(NOSOL_t|Y) + p(N)p(NOSOL_t|N)} \\ &= \frac{p(Y)p(NOSOL_t|Y)}{p(Y)p(NOSOL_t|Y) + p(N)} \\ &= \frac{p(Y)p(NOSOL_t|Y)}{p(Y)p(NOSOL_t|Y) + 1 - p(Y)} \end{aligned} \quad (6)$$

where

$$p(NOSOL_t|Y) = 1 - p(SOL_t|Y) \quad (7)$$

So, the agent can compute both $p(SOL_{\tau_2}|NOSOL_{\tau_1})$ and $p(Y|NOSOL_t)$ in constant time if it knows $p(Y)$ and $p(SOL_t|Y)$. The quantity $p(Y)$ is simply the agent's prior probability that the answer is Y, i.e., the agent's belief before it has executed any steps of the algorithm. In Section 3 we present an example that demonstrates how $p(Y)$ can be obtained using features of the problem instance that are quick to measure. The quantity $p(SOL_t|Y)$ is obtained from the statistical run-time distribution of the algorithm. Specifically, it can be determined empirically off-line by running the algorithm on satisfiable instances (similar to the instance that needs to be solved in the on-line situation), and seeing on what fraction of them the algorithm has found a solution by time t. Alternatively, $p(SOL_t|Y)$ could be determined from an analytical model of the run-time distribution. The generation of $p(SOL_t|Y)$ will be discussed further in Sec. 3 and 6.

2.2 The payoff model

To determine when to terminate, the deliberation controller also needs to know how the agent would use the information that the algorithm provides in the real world. This depends on the application. However, for the purposes of the termination decision, this information can be represented in a domain independent way using a payoff function. Let us denote by $\pi_{world}(x, p_Y, t)$ the agents real-world payoff if the actual outcome is x, $x \in \{Y, N\}$, the agent's estimate—after running the algorithm for t steps—of the answer being Y is p_Y, and the agent acts according to this estimate at time t (or later if the agent finds that more beneficial).[3] The agent's choice of a real-world action depends on p_Y and t, but the real-world payoff, π_{world}, of that action depends on the true posteriori x and when the action is taken, t. In Section 3 we present an example application and illustrate how the π_{world} function can be constructed.

The agent's payoff, $\pi(x, p_Y, t)$, takes into account both the real-world payoff, $\pi_{world}(x, p_Y, t)$, and the computation cost. If they are independent, we can write

$$\pi(x, p_Y, t) = \pi_{world}(x, p_Y, t) - h(t) \quad (8)$$

where $h(t)$ is the computation cost. If there is a fixed unit cost of computation, c_{comp}, then $h(t) = c_{comp} \cdot t$. Our termination algorithm applies for general π, i.e., it does not assume that the computation cost is independent of the real-world payoff.

2.3 Algorithm for computing an optimal termination policy

Put together, the inputs to the algorithm that computes the optimal termination policy are

- the prior probability that a solution exists, $p(Y)$,
- the run-time distribution in the form of $p(SOL_t|Y)$, and
- the payoff model, $\pi(x, p_Y, t)$.

Conceptually, the stop/continue decisions are solved starting from the end of the decision tree (Figure 1), and moving step by step toward the root. For now, say that the tree ends at step T (we will relax this assumption in Section 4). This does not mean that the algorithm is terminated at step T. This section describes how the termination time is computed, and that time is usually before time T.

At every decision node of the deliberation controller, the expected payoff from stopping is computed, and so is the expected payoff from continuing. The expected payoff from continuing at decision node t depends on the solution that was acquired for decision node $t + 1$. At every decision node t, the deliberation controller should let the algorithm continue if and only if the expected payoff from continuing is higher than that of stopping. Algorithm 2.1 presents the pseudo-code for computing this optimal termination policy. The function $v(t)$ solves the expected value of the subtree rooted at the deliberation controller's decision node t. The policy can be solved by making the call $v(0)$. The optimal decision for each decision node, t, is stored in $decision[t]$,[4] and the time when the deliberation controller should first terminate the algorithm is stored in $\tilde{t}$.

Algorithm 2.1 (Compute an optimal termination policy)

```
function v(t)
  if t = 0
    p_SOL = 0 /* Chance that a solution was found in this step */
    π_SOL = 0 /* Payoff of that solution */
  else
    p_SOL = p(SOL_t | NOSOL_{t-1})
    π_SOL = π(1, 1, t)
  p_Y = p(Y | NOSOL_t)
  E[π|STOP] = p_SOL · π_SOL
              +(1 − p_SOL)(p_Y · π(1, p_Y, t) + (1 − p_Y) · π(0, p_Y, t))
  if t = T /* End of the tree */
    decision[t] = STOP
    t̃ = t
    return E[π|STOP] /* recursion bottoms here */
  else
    E[π|CONTINUE] = p_SOL · π_SOL + (1 − p_SOL) · v(t + 1)
    /* recursion occurs above */
    if E[π|STOP] ≥ E[π|CONTINUE]
      decision[t] = STOP
      t̃ = t
      return E[π|STOP]
```

[3] We measure real time in the same units as computation time. In other words, one unit of real time is the time it takes to execute one step of the algorithm. This is without loss of generality because the payoff function can be rescaled (on its t dimension) based on the speed of the machine on which the algorithm is executed.

[4] Note that in classical stopping problems, at every time step after the first optimal stopping point it is better to stop. Here that is not the case, and the algorithm determines a set of stopping points (those times t for which $decision[t] = STOP$) which need not be consecutive.

else
  $decision[t] = CONTINUE$
  return $E[\pi|CONTINUE]$

The algorithm runs in $O(T)$ time and space. The quantities $p(SOL_t|\ NOSOL_{t-1})$ and $p(Y|NOSOL_t)$ are computed in constant time from the inputs $p(Y)$ and $p(SOL_t|Y)$ using the formulas derived in Sec. 2.1.

The termination policy, stored in $decision[t]$, is optimal even if, for some reason, the algorithm has been executed past the first optimal termination point, and the losses so far have become sunk cost. At that point it may be optimal to execute further steps, or it may not. The termination policy gives the optimal stop/continue decisions from any step forward, even if the current step is past the first optimal stopping time.

3 Example of how to use the method

We demonstrate our method in the context of the following hypothetical problem. The question is whether a manufacturing project should be terminated before completion in order to avoid useless costs. A company is offered 60,000 if it produces an item by time 30,000. If the project does not meet that deadline, the contract becomes void: nothing is paid, and the project need not be finished. The cost of the project is 1 per time unit (again, the length of the time unit is the time it takes to execute one algorithm step). If the project can be completed, it can be completed exactly at time 30,000, but not before. Work on the project starts and the costs begin to cumulate.

The last phase of the project is machining, where the key question is: can the machining be scheduled to begin at time 15,000 and to finish by time 30,000? If not, obviously the sooner during the earlier phases the project is terminated, the better. Also, one can see that if the machining can be successfully scheduled, it is worth completing the project. If the project is prematurely terminated, it cannot be restarted. Once the machining phase starts, it cannot be terminated until time 30,000.

Let us call the possible answers to this decision problem Y = "Machining can be scheduled to begin at 15,000 and finish at 30,000" and N = "Machining cannot be scheduled to begin at 15,000 and finish at 30,000". We assume that the machining scheduling decision problem instance is reduced to 3SAT, so Y = "satisfiable" and N = "not satisfiable".

3.1 Payoff model π

Figure 2 illustrates the company's optimal decisions in the example described above. The π_{world}-values can be calculated from the company agent's situation in its environment. For example, if the true answer is Y and the probability estimate of the answer being Y is p_Y, the calculation is as follows. Since the answer is Y, the project can be completed, and the company will get a payoff (its payment minus its cost) of $60,000 - 30,000 = 30,000$ if the project is not terminated prematurely, and a payoff (cost so far) of $-t$ if the project is terminated prematurely. The project will not be prematurely terminated if it is already too late to terminate it by definition ($t > 15,000$) or the expected value of continuing is higher than that of terminating, i.e., $p_Y \cdot 60,000 > 30,000 - t$. So,

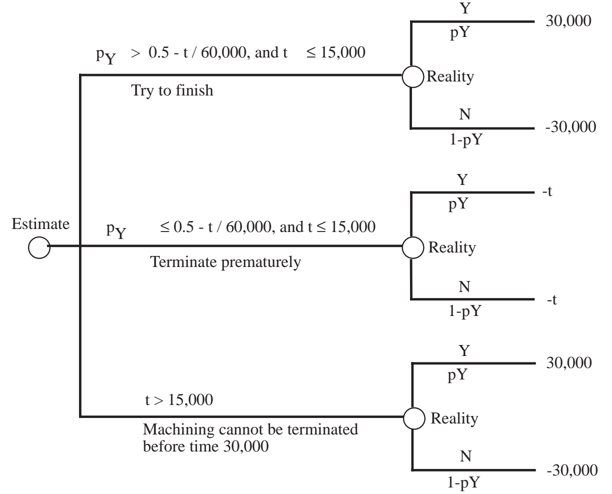

Figure 2: *The tree represents the company's optimal decisions and payoffs given an estimate p_Y. The real posterior event is Y or N.*

$$\pi_{world}(Y, p_Y, t) = \begin{cases} 60,000 - 30,000 = 30,000 & \text{if } p_Y > \frac{30,000-t}{60,000}, t \leq 15,000 \\ -t & \text{if } p_Y \leq \frac{30,000-t}{60,000}, t \leq 15,000 \\ 60,000 - 30,000 = 30,000 & \text{if } t > 15,000. \end{cases}$$

$$\pi_{world}(N, p_Y, t) = \begin{cases} 0 - 30,000 = -30,000 & \text{if } p_Y > \frac{30,000-t}{60,000}, t \leq 15,000 \\ -t & \text{if } p_Y \leq \frac{30,000-t}{60,000}, t \leq 15,000 \\ 0 - 30,000 = -30,000 & \text{if } t > 15,000. \end{cases}$$

Let the computation cost be $\frac{1}{2}$ per time unit, so

$$\pi(x, p_Y, t) = \pi_{world}(x, p_Y, t) - \frac{1}{2}t$$

This completes the payoff model part of the example. Next we will discuss how to construct $p(Y)$ and $p(SOL_t|Y)$ which are the other two inputs to the algorithm that computes the optimal termination policy.

3.2 Constructing the prior $p(Y)$

Often anytime algorithms have a mandatory phase during which a rough solution to the problem is generated. This rough solution is then iteratively improved in an anytime refinement phase. The mandatory phase can be viewed as a non-interruptible setup phase for the actual refinement algorithm. For example in the traveling salesman problem, the mandatory phase could be a greedy generation of an initial tour, and the refinement phase could be local search based on swapping two cities at a time in an ordered list of the cities to visit. In 3SAT, our mandatory phase consists of generating an initial estimate of the satisfiability probability. For this we use three features of the problem instance as predictors: the number of variables, v, the standard $\beta = \frac{c}{v}$ predictor [2, 19] (where c is the number of clauses), and a newer predictor, Δ, see [23].[5]

[5] $\Delta = \sum_{i \in variables} |pos_i - neg_i|$, where pos_i is the number of non-negated occurrences of variable i in the 3SAT instance, and neg_i is the number of negated occurrences of variable i in it.

We generated 3CNF formulas using the most common method [19] for constructing hard 3SAT instances: for every clause, pick three variables randomly disallowing duplicates, and then negate each variable separately with probability $\frac{1}{2}$.[6]

Figure 3 shows the contours of the initial satisfiability probability, $p(Y)$. Say that in our example the 3SAT instance happens to have the following features: $v = 150$, $c = 645$, (so $\beta = \frac{645}{50} \approx 4.3$), and $\Delta = 2.77$.[7] Thus the initial satisfiability probability $p(Y) = 0.454$, see Figure 3 right.[8]

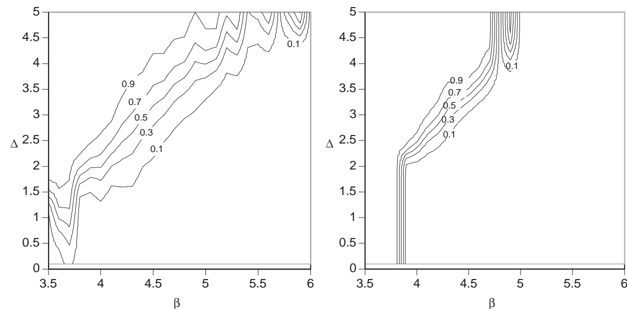

Figure 3: *Contours of the initial (prior) satisfiability probability $p(Y)$. Left: $v = 50$. Right: $v = 150$.*

Often the mandatory phase is fast compared to the refinement phase. The initial satisfiability probability, $p(Y)$, can be computed in $O(c + v)$ time. Each step of the refinement algorithm also takes $O(c + v)$ time. Thus the time of the mandatory phase is so negligible compared to the refinement phase that we ignore it.

3.3 Constructing the run-time distribution $p(SOL_t|Y)$

We used the incomplete BREAKOUT satisfiability determining algorithm [20] as the decision algorithm. If the formula is not satisfiable, the algorithm never finishes, but if the formula is satisfiable, the algorithm may finish proving satisfiability or it might not finish. (Our termination method could be used with any satisfiability determining algorithm.)

One way of representing the algorithm's run-time distribution is $p(SOL_t|Y)$. This probability is parameterized by the algorithm step, t, and by the problem instance features, v, β, and Δ. We gathered statistics for 50, 100, and 150 variables for $\beta = 0.1, 0.2, ..., 9.0$. At each such point, we generated 500 random formulas, determined their satisfiability using a complete algorithm [23] (which is similar to that in [3]), and measured the value of Δ. We ran the BREAKOUT algorithm on the satisfiable formulas and recorded its number of steps. If BREAKOUT had not found a solution by 20,000 steps, we aborted the run and recorded an unsuccessful result. Ideally one could fit a curve on the data for $p(SOL_t|Y)$ as a function of t, v, β, and Δ. This curve could be used for interpolation, but especially for extrapolation for large v, for which no current complete satisfiability determining algorithms run in reasonable time.

As mentioned above, the problem instance at hand happens to have $v = 150$, $\beta = 4.3$, and $\Delta = 2.77$. We constructed the performance profile by looking at prior data with the same v and c. Of these runs we chose the 20 that had their Δ closest to 2.77. From the recorded number of BREAKOUT algorithm steps needed for each of these runs, we constructed the performance profile $p(SOL_t|Y)$, Figure 4. This is merely an example. In real application one would like to use more than 20 training instances to construct $p(SOL_t|Y)$.

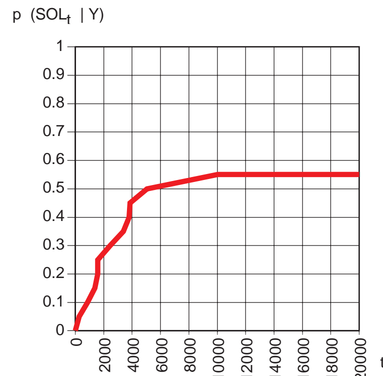

Figure 4: *Probability of finding a satisfying solution by algorithm step t given that the formula is satisfiable. Note that BREAKOUT is incomplete, so the curve may never reach 1.*

3.4 Computing the optimal termination policy

Now we have all the inputs in hand for computing the optimal termination policy. It is obvious that it is not worth computing beyond time 15,000 since after that the company can no longer stop the project. Therefore, we will set the end of the deliberation controller's decision tree to be at 15,000, i.e., $T = 15,000$. Algorithm 2.1 returns $\tilde{t} = 3,852$, and $v(0) = 1314.399350$. So, the deliberation controller should terminate the algorithm at time 3,852 if the algorithm has not finished by then. The company's overall expected payoff is 1314.399350 given that the project has been started. This ends the example.

4 Truncating the deliberation controller's decision tree

As presented, Algorithm 2.1 requires as input a time T beyond which it is certainly not worth computing. This time may be derivable from the payoff model (as in our example where there is a hard deadline at some point). In other settings, one might know, based on theoretical or statistical information, that the incomplete algorithm will never finish if it has not finished in some (large) number of steps T. In either case, our method of determining an optimal stopping time $\tilde{t} \leq T$ can be used.

4.1 A conservative terminator

In yet other settings, neither of these two approaches enables one to deduce a finite upper bound T. In such settings it is

[6] The presented estimation of satisfiability probability is based on a statistical analysis of 3SAT instances from this distribution. Therefore, it is not necessarily accurate for instances from a different distribution—e.g. reduced from a different problem.

[7] Given these feature values, one can predict that this problem instance is likely to be hard [23].

[8] A statistical method for computing this value and the contours of Figure 3 is presented in [23].

sometimes possible to *endogenously* determine T using the payoff model and run-time distribution together. Specifically, one can try increasing values of T, starting from 0. At each such choice of T, one can optimistically assume that from time T on, $p(SOL_t|Y)$ reaches its asymptote $p(SOL_\infty|Y)$ in a single step, i.e., $p(SOL_{T+1}|Y) = p(SOL_\infty|Y)$.[9] If, under this optimistic assumption, it seems better to stop at time T than to continue for one step (based on an expected payoff calculation), then it is certainly better to stop than to continue. Incorporating this test into Algorithm 2.1 is easy (the test "if $t = T$" is replaced with this test). Sometimes this optimistic test leads to a situation where it looks like the tree cannot be truncated at that step T while it really could.

4.2 An overeager terminator

Conversely, it is possible to construct a terminator that always recommends termination when the algorithm should be terminated, but sometimes also recommends termination when it should not. Let us denote by t^* the first step when higher expected payoff would be obtained by stopping the algorithm than by continuing (this definition is based on the untruncated (infinitely deep) decision tree (Fig. 1)). We showed that if one knows (e.g., by guessing) a finite T such that $T \geq t^*$, one can compute t^* by truncating the tree at T:

Theorem 4.1 *For any finite T such that $T \geq t^*$, Algorithm 2.1 returns t^*.*

Proof: Truncating the tree at T corresponds to removing the continuation option at step T. The quantity $v(T)$ is computed by maximizing over the expected values of the two branches—STOP and CONTINUE—so removing one of the branches cannot increase $v(T)$. Therefore, the expected value of the continuation option at step $T-1$ cannot have increased (and the value of the stopping option is the same), so $v(T-1)$ cannot have increased. The same inductive argument is applied to $v(T-2)$, and so on backward in the tree. Therefore, the expected payoff from continuing at step t^* can only decrease or stay the same. Thus, if Algorithm 2.1 would suggest terminating by time t^* originally ($\tilde{t} = t^*$), it certainly would not suggest continuing beyond t^* in this new setting where the tree was truncated at T.

What remains to be proven is that the truncation at T cannot cause Algorithm 2.1 to change its termination prescription to an earlier time than its original prescription, $\tilde{t} = t^*$. The following observation proves this: the value of the continuation branch at step t^* will not propagate earlier in the tree because at decision node t^* the algorithm chooses to stop rather than to continue (because stopping originally had higher expected payoff and truncation cannot increase the expected payoff of continuing). ■

The claim is not totally obvious because due to such truncation, the value of continuing the computation can seem lower than it really is. In fact, if one incorrectly guesses a T so that $T < t^*$, Algorithm 2.1 in some cases suggests stopping at some time before T although continuing would be the better choice. An interesting consequence of this is that any approach that simply guesses T (e.g., by a doubling scheme) can never be sure that the guess T is large enough—even if Algorithm 2.1 suggests stopping before T.

[9] Some incomplete algorithms have $p(SOL_\infty|Y) < 1$.

4.3 Never terminating the algorithm

The fact that in some cases a gap exists between the conservative and the overeager terminator might not be a facet of our artifacts, but inherent. The terminators attempt to find a stopping time, but if no finite optimal stopping time exists, one would not expect any algorithm to determine that in finite time (for one, it would have to read an infinite number of values from the run-time distribution). Indeed, in some settings it is best to never stop. For example, if the agent's payoff does not decrease with time, there is no reason to stop.

The following is a less trivial example. Let the agent get payoff $\pi_{NOSOL}(t) = -t$ if it decides to stop the algorithm at time t without having found a solution. Let the agent's payoff be $\pi_{SOL}(t) = 1000 - t$ if it finds a solution at time t. In other words, the payoff decreases linearly with time. (This is realistic, for example, if the agent's situation in the world does not change with time but the agent has to pay a constant amount for each CPU-cycle.) Let $p(SOL_{t+1}|NOSOL_t) = \alpha$. This corresponds to a setting where the run-time of the algorithm, treated as a random variable, is exponentially distributed. Because $p(SOL_{t+1}|NOSOL_t)$ is a constant and the payoff decreases by a constant at every step, the decision problem will look exactly the same to the deliberation controller in every subtree of the decision tree. Therefore, if α is high enough so that the deliberation controller receives higher expected payoff by choosing CONTINUE at time 0 than by choosing STOP, the deliberation controller will receive higher expected payoff by choosing CONTINUE at every time, t, than by choosing STOP.

5 Related research, briefly

Our approach has a rich intellectual background. Anytime algorithms have been studied extensively, albeit mainly in the more general setting of optimization problems (e.g., [1, 9, 13, 16, 25]). A variety of incomplete decision algorithms have been designed, especially for satisfiability (e.g., [3, 6, 24]), and the algorithms tend to scale orders of magnitude better than their complete counterparts. Considerable research has also been done on run-time distributions of algorithms (e.g., [4, 5, 8, 10, 12]).

Random restart algorithms are another family of incomplete algorithms that has received significant attention recently (e.g., [6, 7, 17]). Our method can be used to optimally terminate a decision algorithm, even if that algorithm internally uses random restarts.

An approach related to ours has been presented for theorem proving [11]. Finally, our method was inspired by seminal work on the value of information [14, 18, 21].

6 Conclusions and future research

Incomplete decision algorithms can often solve larger problem instances than complete algorithms. The difficulty is that if the algorithm has not finished, one does not know whether

it will finish soon, later, or never. We presented a decision-theoretic method for optimally terminating incomplete decision algorithms. With each execution step, such algorithms implicitly refine the probability that the answer is yes. Our terminator determines an optimal stopping policy based on a prior probability of the answer being yes, a payoff model describing the value that different probability estimates would provide at different times, and a statistical run-time distribution of the algorithm.

In many settings it is unrealistic to assume that run-time statistics exist for large numbers of problem instances of the scale of the real problem to be solved. If the real problem takes so long that sophisticated termination is needed, then collecting training data would most likely be prohibitive. Therefore, we advocate collecting training data on smaller problem instances, building an analytical model of the run-time distribution as a function of problem instance features (such as the order parameters used in this paper), and using the analytical extrapolated distribution as the run-time distribution in our termination method. This also allows terminating a run on an instance whose features are not identical, but similar, to the features of instances for which statistical run-time information has been collected.

In many incomplete decision algorithms, each computation step is short. In such settings, our (linear-time) deliberation control method might use more time than the incomplete algorithm itself. This can be avoided by considering a number of computing steps as an atomic step. (However, the coarser deliberation control might not lead to exactly accurate termination.) This approach also gives the conservative terminator a better chance to terminate the algorithm.

It is easy to incorporate risk attitudes into our method (assuming that an agent's utility is just a function of the agent's payoff, which itself depends on multiple factors as we discussed). This can be done by replacing the agent's payoff function π in the model with the agent's utility function $u(\pi)$.

Our method can be used to optimally select an algorithm. For each candidate algorithm, the expected payoff $v(0)$ is computed. This is the expected payoff under the optimal stopping policy. The algorithm with highest $v(0)$ is chosen.

Our method can also be used to optimally terminate complete algorithms. This is important when the complete algorithm can have a prohibitively long run time.

Future research includes applying our termination method to the control of decision algorithms in other domains, and extending our method beyond decision problems.

Acknowledgments

I thank Victor Lesser, Shlomo Zilberstein, and Neil Immerman for their helpful comments.

References

[1] M. Boddy and T. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67:245–285, 1994.

[2] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 331–337, Sydney, Australia, 1991.

[3] J. Crawford and L. Auton. Experimental results on the crossover point in satisfiability problems. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 21–27, Washington, D.C., 1993.

[4] J. Frank, P. Cheeseman, and J. Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997.

[5] D. Frost, I. Rish, and L. Vila. Summarizing CSP hardness with continuous probability distributions. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 327–333, Providence, RI, 1997.

[6] I. P. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 28–33, Washington, D.C., 1993.

[7] C. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001.

[8] C. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Madison, WI, 1998.

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

[10] H. H. Hoos and T. Stützle. Evaluating Las Vegas algorithms—pitfalls and remedies. In *Proceedings of the Uncertainty in Artificial Intelligence Conference (UAI)*, 1998.

[11] E. Horvitz and A. Klein. Reasoning, metareasoning, and mathematical truth: Studies of theorem proving under limited resources. In *Proceedings of the Uncertainty in Artificial Intelligence Conference (UAI)*, 1995.

[12] E. Horvitz, Y. Ruan, C. Gomez, H. Kautz, B. Selman, and M. Chickering. A Bayesian approach to tackling hard computational problems. In *Proceedings of the Uncertainty in Artificial Intelligence Conference (UAI)*, 2001.

[13] E. J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of Third Workshop on Uncertainty in Artificial Intelligence*, pages 429–444, Seattle, Washington, July 1987. American Association for Artificial Intelligence. Also in L. Kanal, T. Levitt, and J. Lemmer, ed., *Uncertainty in Artificial Intelligence 3,* Elsevier, 1989, pps. 301-324.

[14] R. Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, 2(1):22–26, 1966.

[15] H. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 359–363, 1992.

[16] K. Larson and T. Sandholm. Bargaining with limited computation: Deliberation equilibrium. *Artificial Intelligence*, 132(2):183–217, 2001. Short early version appeared in the Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 48–55, Austin, TX, 2000.

[17] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47:173–180, 1993.

[18] J. E. Matheson. The economic value of analysis and computation. *IEEE Transactions on Systems Science and Cybernetics*, 4(3):325–332, 1968.

[19] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 459–465, San Jose, CA, 1992.

[20] P. Morris. The breakout method for escaping from local minima. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 40–45, Washington, D.C., 1993.

[21] H. Raiffa. *Decision Analysis: Introductory Lectures on Choices under Uncertainty*. Addison-Wesley, 1968.

[22] T. Sandholm and V. R. Lesser. Utility-based termination of anytime algorithms. In *ECAI Workshop on Decision Theory for DAI Applications*, pages 88–99, Amsterdam, The Netherlands, 1994.

[23] T. W. Sandholm. A second order parameter for 3SAT. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 259–265, Portland, OR, 1996. Early version appeared at the AAAI Workshop on Experimental Evaluation of Reasoning and Search Methods, pages 57–63, Seattle, WA, 1994.

[24] B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 290–295, Chambery, France, 1993.

[25] S. Zilberstein and S. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1–2):181–213, 1996.