Terminating Decision Algorithms Optimally

Tuomas Sandholm

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

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. 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 and/or 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* 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.

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. The key observation is that incomplete algorithms are iterative refinement algorithms and approximation

[1] A short, very tentative version of this paper appeared in a workshop [3].

[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.

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.

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). Using this information, 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.

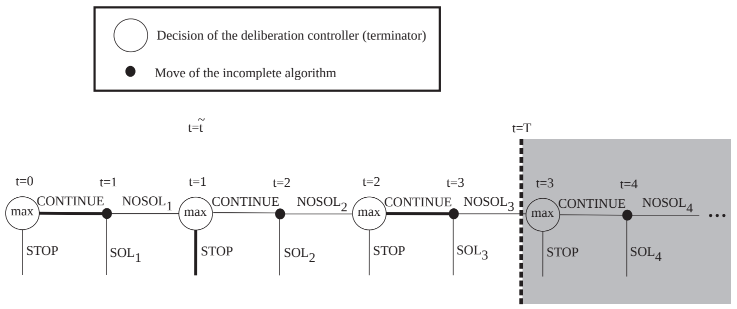

Fig. 1. *Deliberation controller's decision tree. The bold lines show an example policy where the deliberation controller will terminate the algorithm at time $t = \tilde{t} = 1$ if the algorithm has not found a solution.*

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,

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

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

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

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

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

which can be solved using

$$p(NOSOL_t) = p(Y)p(NOSOL_t|Y) + p(N)p(NOSOL_t|N) \tag{4}$$

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:

$$p(NOSOL_t) = p(Y)p(NOSOL_t|Y) + 1 - p(Y) \tag{5}$$

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:

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

where

$$p(NOSOL_t|Y) = 1 - p(SOL_t|Y) \tag{7}$$

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 the extended version at `www.cs.cmu.edu/~sandholm/util_term.extended.pdf` 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 $p(SOL_t|Y)$ can be determined empirically off-line by running the algorithm on instances (similar to the instance that needs to be solved in the on-line situation) whose answer is Y, 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.

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). 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 the extended version of this paper 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

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

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 method applies to general π: 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 1) the prior probability that a solution exists, $p(Y)$, 2) the run-time distribution in the form of $p(SOL_t|Y)$, and 3) the payoff model, $\pi(x, p_Y, t)$.

Conceptually, the stop/continue decisions are solved starting from the end of the decision tree (Fig. 1), and moving toward the root. For now, say that the tree ends at step T (we relax this assumption of an exogenous upper bound on the optimal termination time in the extended version of this paper). This does not mean that the algorithm is terminated at step T. This section describes how the termination time, $\tilde{t}$, is computed ($\tilde{t} \leq T$).

For every decision node of the tree, the expected payoff from stopping is computed, and so is the expected payoff from continuing. The expected payoff from continuing at node t depends on the solution that was acquired for node $t+1$. At a node, the deliberation controller should terminate the algorithm if and only if the expected payoff from stopping is higher than that of continuing. The pseudocode below computes 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 $decision[t]$, and the time when the deliberation controller should first terminate the algorithm is stored in $\tilde{t}$.[3]

Algorithm 1 (Compute an optimal termination policy)

```
function $v(t)$
    if $t = 0$
        $p_{SOL} = 0$ /* Chance that a solution was found in this step */
        $\pi_{SOL} = 0$ /* Payoff of that solution */
    else
        $p_{SOL} = p(SOL_t|NOSOL_{t-1})$
        $\pi_{SOL} = \pi(1, 1, t)$
    $p_Y = p(Y|NOSOL_t)$
    $E[\pi|STOP] = p_{SOL} \cdot \pi_{SOL} + (1 - p_{SOL})(p_Y \cdot \pi(1, p_Y, t) + (1 - p_Y) \cdot \pi(0, p_Y, t))$
    if $t = T$ /* End of the tree */
        $decision[t] = STOP$; $\tilde{t} = t$; return $E[\pi|STOP]$ /* recursion bottoms here */
    else
        $E[\pi|CONTINUE] = p_{SOL} \cdot \pi_{SOL} + (1 - p_{SOL}) \cdot v(t + 1)$ /* recursion */
        if $E[\pi|STOP] \geq E[\pi|CONTINUE]$
            $decision[t] = STOP$; $\tilde{t} = t$; return $E[\pi|STOP]$
        else
            $decision[t] = CONTINUE$; return $E[\pi|CONTINUE]$
```

Algorithm 1 runs in $O(T)$ time and space. The values $p(SOL_t|\ NOSOL_{t-1})$ 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 Section 2.1.

3 Other results

An extended version of this paper is available at `www.cs.cmu.edu/~sandholm/util_term.extended.pdf`. It presents an example application of how the method can be used (in the context of a manufacturing planning problem converted to 3SAT), how the payoff model $\pi(x, p_Y, t)$ can be derived, how the prior $p(Y)$ can be constructed (using statistical information and features of the problem instance), and how the run-time distribution $p(SOL_t|Y)$ can be constructed from runs of the algorithm. It also discusses ways how the method can be used when an exogenous upper bound T on the optimal run-time is not given, and presents example settings where it is best to never terminate the algorithm. Finally, it discusses related research (e.g., [2, 1]), and presents more elaborate conclusions and directions for future research.

References

1. H. Hoos and T. Stützle. Evaluating Las Vegas algorithms—pitfalls and remedies. *Proceedings of the Uncertainty in Artificial Intelligence Conference (UAI)*, 1998.
2. E. Horvitz and A. Klein. Reasoning, metareasoning, and mathematical truth: Studies of theorem proving under limited resources. *UAI*, 1995.
3. T. Sandholm and V. Lesser. Utility-based termination of anytime algorithms. *ECAI Workshop on Decision Theory for DAI Applications*, p. 88–99, Amsterdam, 1994.

[3] 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. The algorithm determines a set of stopping points (t s.t. $decision[t] = STOP$) which need not be consecutive. These points state when to terminate the incomplete algorithm even if, for some reason, it was not terminated at the first optimal stopping point.