Optimal Depth-First Strategies for And-Or Trees

Russell Greiner **Ryan Hayward**
Dept of Computing Science
University of Alberta
{greiner, hayward}@cs.ualberta.ca

Michael Molloy
Dept of Computer Science
University of Toronto
molloy@cs.toronto.edu

Abstract

Many tasks require evaluating a speci£ed boolean expression φ over a set of probabilistic tests where we know the probability that each test will succeed, and also the cost of performing each test. A strategy *speci£es when to perform which test, towards determining the overall outcome of φ. This paper investigates the challenge of £nding the strategy with the minimum expected cost.*

We observe £rst that this task is typically NP-hard — e.g., when tests can occur many times within φ, or when there are probabilistic correlations between the test outcomes. We therefore focus on the situation where the tests are probabilistically independent and each appears only once in φ. Here, φ can be written as an `and-or` *tree, where each internal node corresponds to either the "And" or "Or" of its children, and each leaf node is a probabilistic test.*

There is an obvious depth-£rst approach to evaluating such `and-or` *trees: First evaluate each penultimate subtree in isolation; then reduce this subtree to a single "mega-test" with an appropriate cost and probability, and recur on the resulting reduced tree. After formally de£ning this approach, we prove £rst that it produces the optimal strategy for shallow (depth 1 or 2)* `and-or` *trees, then show it can be arbitrarily bad for deeper trees. We next consider a larger, natural subclass of strategies — those that can be expressed as a* linear *sequence of tests — and show that the best such "linear strategy" can also be very much worse than the optimal strategy in general. Finally, we show that our results hold in a more general model, where* internal *nodes can also be probabilistic tests.*

Keywords: Satis£cing search, Diagnosis, Computational complexity

1 Introduction

Baby J is a fussy eater, who only likes foods that are sweet, or contain milk and fruit, or contain milk and cereal; see Figure 1. Suppose that you want to determine whether J will like some new food, and that you can test whether the food satis£es J's basic food-properties (`Sweet`, `Milk`, `Fruit`, `Cereal`), where each test has a known cost (say unit cost for this example). Notice that the outcome of one test may render other test(s) unnecessary (if the food has `Fruit`, it does not matter whether it has `Cereal`), so the cost of determining whether J £nds a food yummy depends on the order in which tests are performed as well as their outcomes.

A strategy[1] describes the order in which tests are to be performed. For example, the strategy $\xi_{\langle SMCF\rangle}$ £rst performs the Sweet test, returning `true` (aka `Yummy`) if it succeeds; if it fails, $\xi_{\langle SMCF\rangle}$ will perform the Milk test, returning `false` if it fails. Otherwise (if Sweet fails and Milk succeeds), $\xi_{\langle SMCF\rangle}$ will perform the Cereal test, returning `false` if it fails; if it succeeds, $\xi_{\langle SMCF\rangle}$ will perform the Fruit test, returning `true`/`false` if it succeeds/fails. See Figure 2(a). Notice that $\xi_{\langle SMCF\rangle}$ will typically perform only a subset of the 4 tests; *e.g.*, it will skip all of the remaining tests if Sweet succeeds.

Of course, there are other strategies for this situation, including $\xi_{\langle SMFC\rangle}$, which differs from $\xi_{\langle SMCF\rangle}$ only by testing Fruit before Cereal, and $\xi_{\langle SCFM\rangle}$, which tests the `Cereal-Fruit` component before `Milk`. Each strategy is correct, in that it correctly determines whether J will like a food or not. However, for a given food, different strategies will perform different subsets of the tests. (*E.g.*, while $\xi_{\langle SMCF\rangle}$ will perform all of the tests for a non-sweet, milky non-cereal with fruit, $\xi_{\langle SMFC\rangle}$ will not need to perform the £nal Cereal test.) Hence, different strategies can have different costs. If we also know the likelihood that each test will succeed, we can then compute the *expected cost* of a strategy, over the distribution of foods considered.

In general, there can be an exponential number of strategies, each of which returns the correct answer, but which vary in terms of their expected costs. This paper discusses the task of computing the best — that is, minimum cost and correct — strategy, for various classes of problems.

1.1 Framework

Each of the strategies discussed so far is *depth-£rst* in that, for each `and-or` subtree, the tests on the leaves of the subtree appear together in the strategy. We will also consider strategies that are not depth-£rst; *e.g.*, $\xi_{\langle CSMF\rangle}$ is not depth-£rst, since it starts with `Cereal` and then moves on to `Sweet` before completing the evaluation of the `i#2`-rooted subtree.

This strategy, like the depth-£rst ones, is *linear*, as it can be described in a *linear* fashion: proceed through the tests in

[1]Formal de£nitions are presented in §2.

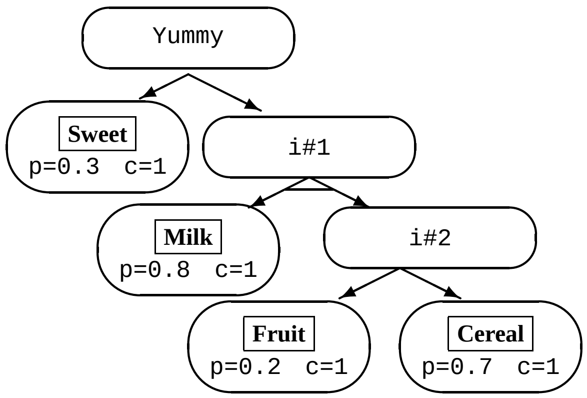

Figure 1: An `and-or` tree, T_1. Here `and`-nodes are indicated with a horizontal bar through the descending arcs; `i#1` is an `and`-node while `Yummy` and `i#2` are `or`-nodes.

the given order, omitting tests only if logically unnecessary.

There are also non-linear strategies. For example, the ξ_{nl} strategy, Figure 2(b), £rst tests `Cereal` and if positive, tests `Milk` then (if necessary) `Sweet`. However, if the `Cereal` test is negative, it then tests `Sweet` then (if necessary) `Fruit` then `Milk`. Notice no linear sequence can describe this strategy as, for some instances, it tests `Milk` before `Sweet`, but for others, it tests `Sweet` before `Milk`.

We consider only *correct* strategies, namely those that return the correct value for any assignment. We can measure the performance of a strategy by the expected cost of returning the answer. A standard simplifying assumption is to require that these tests be independent of each other — *e.g.*, Figure 1 indicates that 30% of the foods sampled are Sweet, 80% are with Milk, etc. While each strategy returns the correct boolean value, they have different expected costs. For example, the expected cost of $\xi_{\langle SMCF \rangle}$ is

$$\begin{aligned} C[\xi_{\langle SMCF \rangle}] \quad = \quad & c(\mathtt{S}) + \Pr(\,-\mathtt{S}\,) \times \\ & [c(\mathtt{M}) + \Pr(\,+\mathtt{M}\,) \times [c(\mathtt{C}) + \Pr(\,-\mathtt{C}\,)\ c(\mathtt{F})]\,] \end{aligned}$$

where $\Pr(\,+\mathtt{T}\,)$ (resp., $\Pr(\,-\mathtt{T}\,)$) is the probability that test `T` succeeds (resp., fails), and $c(\mathtt{T})$ is the cost of `T`.

Suppose that we are given a particular *probabilistic boolean expression* (PBE), which is a boolean formula, together with the success probabilities and costs for its variables. A strategy is *optimal* if it (always returns the correct value and) has minimum expected cost, over all such strategies. This paper considers the so-called minimum cost resolution strategy problem (MRSP): given such a formula, probabilities and costs, determine an optimal strategy.

§2 describes these and related notions more formally. §3 describes an algorithm, DFA [Nat86], that produces a depth-£rst strategy, then proves several theorems about this algorithm; in particular, that DFA produces the optimal *depth-£rst* strategy, that this DFA strategy is optimal for `and-or` trees with depth 1 or 2 (Theorem 4), but it can be quite far from optimal in general (Theorem 6). §4 then formally de£nes *linear strategies*, shows they are strict generalizations of depth-£rst strategies, and proves (Theorem 8) that the best linear strategy can be far from optimal. §5 motivates and de£nes an extension to the PBE model, called "Precondition PBE", where each test can only performed in some context (*i.e.*, if some boolean condition is satis£ed), and shows that the same results apply. The extended paper [GHM02] provides the proofs for the theorems.

1.2 Related Work

We close this section by framing our problem, and providing a brief literature survey (see also §5.1). The notion of MRSP appears in Simon and Kadane [SK75], who use the term *satis£cing search* in place of *strategy*. Application instances include screening employment candidates for a position [Gar73], competing for prizes at a quiz show [Gar73], mining for gold buried in Spanish treasure chests [SK75], and performing inference in a simple expert system [Smi89; GO91].

We motivate our particular approach by considering the complexity of the MRSProblem for various classes of probabilistic boolean expressions. First, in the case of arbitrary boolean formulae, MRSP is NP-hard. (Reduction from SAT [GJ79]: if there are no satisfying assignments to a formula, then there is no need to perform any tests, and so a 0-cost strategy is optimal.)

We can try to avoid this degeneracy by considering only "positive formulae", where every variable occurs only positively. However, the MRSP remains NP-hard here. (Proof: reduction from ExactCoverBy3Set, using the same construction that [HR76] used to show the hardness of £nding the smallest decision tree.)

A further restriction is to consider "read-once" formulae, where each variable appears only one time. As noted above, we can view each such formula as an "and-or tree". The MRSP complexity here is not known.

This paper considers some special cases. Barnett [Bar84] discusses how the choice of optimal strategy depends on the probability values in a special case when there are two independent tests (and so only two alternative search strategies). Geiger and Barnett [GB91] note that the optimal strategies for `and-or` trees cannot be represented by a linear order of the nodes. Natarajan [Nat86] introduced the ef£cient algorithm we call DFA for dealing with and-or trees, but did not investigate the question of when it is optimal. Our paper proves that this simple algorithm in fact solves the MRSP for very shallow-trees, of depth 1 or 2, but can do very poorly in general.

We consider the tests to be statistically independent of each other; this is why it suf£ces to use simply $\Pr(\,+X\,)$, as test X does not depend on the results of the other experiments that had been performed earlier. If we allow statistical dependencies, then the read-once restriction is not helpful, as we can convert any non-read-once but independent PBE to a read-once but correlated PBE by changing the i-th occurance of the test "X" to a new "X_i", but then insist that X_1 is equal to X_2, etc. We will therefore continue to consider the tests to be independent.

2 De£nitions

We focus on read-once formulae, which correspond to `and-or` trees — *i.e.*, a tree structure whose leaf nodes are each

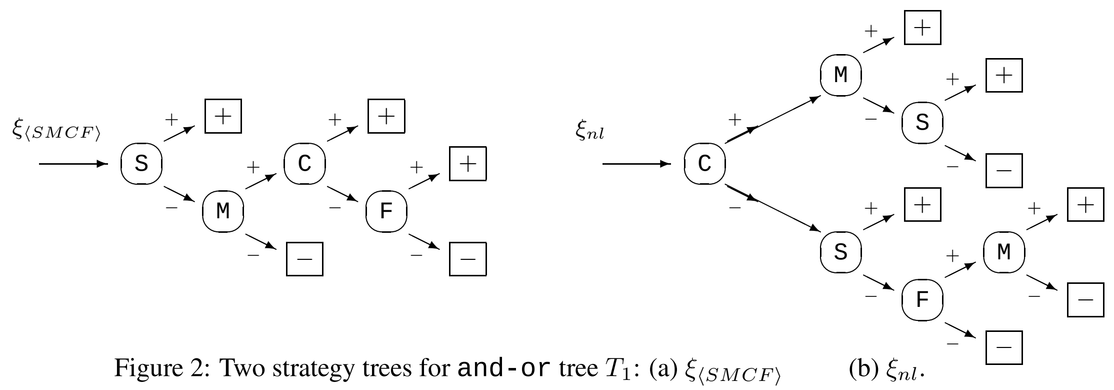

Figure 2: Two strategy trees for and-or tree T_1: (a) $\xi_{\langle SMCF\rangle}$ (b) ξ_{nl}.

labeled with a probabilistic test (with a known positive cost[2] and success probability) and whose internal nodes are each labeled as an or-node or an and-node, with the understanding that the subtree rooted at an and-node (or-node) is satis£ed if and only if all (at least one) of the subtrees are satis£ed.

Given any assignment of the probabilistic tests, for example $\{-\mathsf{S},\ +\mathsf{M},\ -\mathsf{F},\ +\mathsf{C}\}$, we can propagate the assignment from the leaf nodes up the tree, combining them appropriately at each internal node, until reaching the root node; the *value* is the tree's overall evaluation of the assignment.

A *strategy* ξ for an and-or tree T is a procedure for evaluating the tree, with respect to any assignment. In general, we present a strategy itself as a tree, whose internal nodes are labeled with probabilistic tests and whose leaf nodes are labeled either true + or false -, and whose arcs are labeled with the values of the parent's test (+ or -). By convention, we will draw these strategy trees sideways, from left-to-right, to avoid confusing them with top-to-bottom and-or trees. Figure 2 shows two such strategy trees for the T_1 and-or tree. Different nodes of a strategy tree may be labeled with the same test. Recall that the strategy need not corresponds to a simple linear sequence of experiments; see discussion of ξ_{nl}.

For any and-or tree T, we will only consider the set of *correct* strategies $\Xi(T)$, namely those that return the correct value for all test assignments. For T_1 in Figure 1, each of the strategies in $\Xi(T_1)$ returns the value $\mathsf{S} \vee (\mathsf{M} \wedge [\mathsf{F} \vee \mathsf{C}])$.

For any test assignment γ, we let $k(\xi, \gamma)$ be the cost of using the strategy ξ to determine the (correct) value. For example, for $\gamma = \{-\mathsf{S},\ +\mathsf{M},\ -\mathsf{F},\ +\mathsf{C}\}$, $k(\xi_{\langle SMFC\rangle}, \gamma) = c(\mathsf{S}) + c(\mathsf{M}) + c(\mathsf{F}) + c(\mathsf{C})$ while $k(\xi_{\langle CMSF\rangle}, \gamma) = c(\mathsf{C}) + c(\mathsf{M})$.

The *expected cost* of a strategy ξ is the average cost of evaluating an assignment, over all assignments, namely

$$C[\xi] \quad = \sum_{\gamma:\text{Assignment}} \Pr(\,\gamma\,) \times k(\xi, \gamma)\,. \qquad (1)$$

Given the independence of the tests, there is a more ef£cient way to evaluate a strategy than the algorithm implied by Equation 1. Extending the notation $C[\cdot]$ to apply to any strategy sub-tree, the expected cost of a leaf node is $C[\boxed{+}] = C[\boxed{-}] = 0$, and of a (sub)tree φ_χ rooted at a node labeled χ is just

$$C[\varphi_\chi] \quad = \quad c(\chi) \quad + \Pr(\,+\chi\,) \times C[\varphi_{+\chi}] \\ + \Pr(\,-\chi\,) \times C[\varphi_{-\chi}] \qquad (2)$$

where $\varphi_{+\chi}$ ($\varphi_{-\chi}$) is the subtree rooted at χ's $+$ branch ($-$ branch).

To de£ne our goal:

De£nition 1 *A correct strategy $\xi^* \in \Xi(T)$ is* optimal *for an* and-or *tree T if and only if its cost is minimal, namely*

$$\forall \xi \in \Xi(T), \quad C[\xi_T] \quad \leq \quad C[\xi]\,.$$

Depth, "Strictly Alternating": We de£ne the *depth* of a tree to be the maximum number of internal nodes in any leaf-to-root path. (Hence depth 1 corresponds to simple conjunctions or disjunctions, and depth 2 corresponds to CNF or DNF.)

For now (until §5), we will assume that an and-or tree is *strictly alternating*, namely that the parent of each internal and-node is an or-node, and vice versa. If not, we can obtain an equivalent tree by collapsing any or-node (and-node) child of an or-node (and-node). Any strategy of the original tree is a strategy of the collapsed one, with identical expected cost.

3 The depth-£rst algorithm DFA

To help de£ne our DFA algorithm, we £rst consider depth 1 trees.

Observation 1 *[SK75] Let T_O be a depth 1 tree whose root is an* or *node, whose children correspond to tests $A_1, \ldots, A_r$, with success probabilities $Pr(\,+A_i\,)$ and costs $c(A_i)$. Then the optimal strategy for T_O is the one-path strategy $A_{\pi_1}, \ldots, A_{\pi_r}$, (Figure 3(c)) where π is de£ned so that $Pr(\,+A_{\pi_j}\,)/c(A_{\pi_j}) \ \geq\ Pr(\,+A_{\pi_{j+1}}\,)/c(A_{\pi_{j+1}})$ for $1 \leq j < r$.*

Proof: As we can stop as soon as any test succeeds, we need only consider what action to perform after each initial sequence of tests has failed; hence we need only consider strategy trees with "one-path" structures. Towards a contradiction, suppose the optimal strategy ξ_{AB} did not satisfy this ordering, in that there was (at least one) pair of tests, A and B such that A came before B but $\Pr(\,+A\,)/c(A) <$

[2]We can also allow 0-cost tests, in which case we simply assume that a strategy will perform all such tests £rst, leaving us with the challenge of evaluating the reduced MRSP whose tests all have strictly-positive costs.

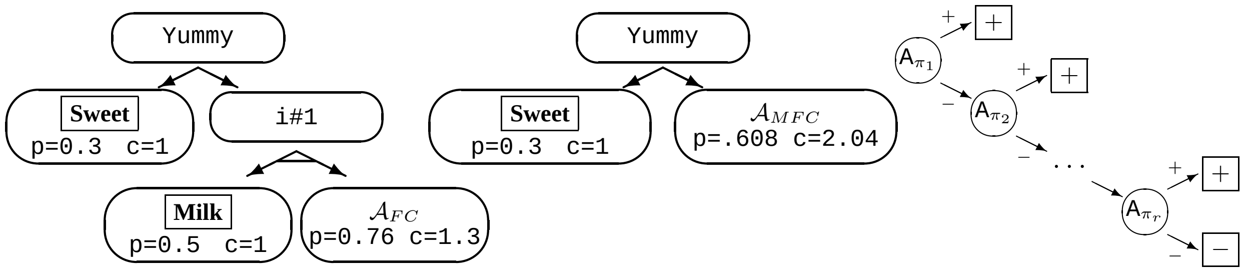

Figure 3: Intermediate results of DFA on T_1 (a) after 1 iteration (b) after 2 iterations. (c) A one-path strategy tree.

$\Pr(+B)/c(B)$. Now consider the strategy ξ_{BA} that re-ordered these tests; and observe that ξ_{BA}'s expected cost is strictly less than ξ_{AB}'s, contradicting the claim that ξ_{BA} was optimal. □

An isomorphic proof shows . . .

Observation 2 *Let T_A be a depth 1 tree whose root is an* and *node, de£ned analogously to T_O in Observation 1. Then the optimal strategy for T_A is the one-path strategy $A_{\phi_1}, \ldots, A_{\phi_r}$, where ϕ is de£ned so that $Pr(-A_{\phi_j})/c(A_{\phi_j}) \geq Pr(-A_{\phi_{j+1}})/c(A_{\phi_{j+1}})$ for $1 \leq j < r$.*

Now consider a depth-s alternating tree. The DFA algorithm will £rst deal with the bottom tree layer, and order the children of each £nal internal node as suggested here: in order of $\Pr(+A_i)/c(A_i)$ if the node is an or-node. (Here we focus on the or-node case; the and-node case is analogous.) For example, if dealing with Figure 1's T_1, DFA would compare $\Pr(+\mathsf{F})/c(\mathsf{F}) = 0.2/1$ with $\Pr(+\mathsf{C})/c(\mathsf{C}) = 0.7/1$, and order C £rst, as $0.7 > 0.2$.

DFA then replaces this penultimate node and its children with a single mega-node; call it $\mathcal{A}$, whose success probability is

$$\Pr(+\mathcal{A}) = 1 - \prod_i \Pr(-A_i)$$

and whose cost is the expected cost of dealing with this subtree:

$$c(\mathcal{A}) = c(A_{\pi_1}) + \Pr(-A_{\pi_1}) \times [c(A_{\pi_2}) + \Pr(-A_{\pi_2}) \times (\ldots c(A_{\pi_{r-1}}) + \Pr(-A_{\pi_{r-1}}) \times c(A_{\pi_r}))]$$

Returning to T_1, DFA would replace the $i\#2$-rooted subtree with the single $\mathcal{A}_{FC}$-labeled node, with success probability $\Pr(+\mathcal{A}_{FC}) = 1 - (\Pr(-\mathsf{F}) \times \Pr(-\mathsf{C})) = 1 - 0.8 \times 0.3 = 0.76$, and cost $c(\mathcal{A}_{FC}) = c(\mathsf{C}) + \Pr(-\mathsf{C}) \times c(\mathsf{F}) = 1 + 0.3 \times 1 = 1.3$; see Figure 2(a).

Now recur: consider the and-node that is the parent to this mega-node $\mathcal{A}$ and its siblings. DFA inserts this $\mathcal{A}$ test among these siblings based on its $\Pr(-\mathcal{A})/c(\mathcal{A})$ value, and so forth.

On T_1, DFA would then compare $\Pr(-\mathsf{M})/c(\mathsf{M}) = 0.2/1$ with $\Pr(-\mathcal{A}_{FC})/c(\mathcal{A}_{FC}) = 0.24/1.3$ and so select the M-labeled node to go £rst. Hence, the substrategy associated with the i#1 subtree will £rst perform M, and return − if unsuccessful. Otherwise, it will then perform the $\mathcal{A}_{FC}$ mega-test: Here, it £rst performs C, and returns + if C succeeds. Otherwise this substrategy will perform F, and return + if it succeeds or − otherwise.

DFA then creates a bigger mega-node, $\mathcal{A}_{MFC}$, with success probability $\Pr(+\mathcal{A}_{MFC}) = \Pr(+\mathsf{M}) \times \Pr(+\mathcal{A}_{FC}) = 0.8 \times 0.76 = 0.608$, and cost $c(\mathcal{A}_{MFC}) = c(\mathsf{M}) + \Pr(+\mathsf{M}) \times c(\mathcal{A}_{FC}) = 1 + 0.8 \times 1.3 = 2.04$; see Figure 2(b).

Finally, DFA compares S with $\mathcal{A}_{MFC}$, and selects S to go £rst as $\Pr(+\mathsf{S})/c(\mathsf{S}) = 0.3/1 > 0.608/2.04 = \Pr(+\mathcal{A}_{MFC})/c(\mathcal{A}_{MFC})$. This produces the $\xi_{\langle SMCF \rangle}$ strategy, shown in Figure 2.

This DFA algorithm is very ef£cient: As it examines each node only in the context of computing its position under its immediate parent (which requires sorting that node and its siblings), DFA requires only $O(\sum_v d^+(v) \log d^+(v)) = O(n \ln r)$ time, where n is the total number of nodes in the and-or tree, and $d^+(v)$ is the out-degree of the node v, which is bounded above by r, the largest out-degree of any internal node.

Notice also that DFA keeps together all of the tests under each internal node, which means it is producing a *depth-£rst strategy*. To state this more precisely, we £rst identify each (sub)tree S of a given and-or tree T with an associated boolean expression $\phi(S)$. (*E.g.*, the boolean expression associated with $S_{i\#1}$, the subtree of Figure 1's T_1 rooted in $i\#1$, is $\phi(S_{i\#1}) \equiv M\&(F \vee C)$.) During the evaluation of a strategy for T, an and-or subtree S is "determined" once we know the value of $\phi(S)$.

De£nition 2 *A strategy ξ for T is depth-£rst if, for each subtree S, whenever a leaf of S is tested, ξ will determine the boolean value of $\phi(S)$ before performing any test outside of S.*

To see that $\xi_{\langle SMCF \rangle}$ (Figure 2(a)) is depth-£rst, notice that every time C appears, it is followed (when necessary) by F; notice this $C - F$ block will determine the value of the $S_{i\#2}$ subtree. Similarly, the $M - C - F$ block in $\xi_{\langle SMCF \rangle}$ will always determine the value of the $S_{i\#1}$ subtree. By contrast, the $\xi_{\langle CMSF \rangle}$ strategy is not depth-£rst, as there is a path where C is performed but, before the $S_{i\#2}$ subtree is determined (by testing F) another test (here M) is performed. Similarly ξ_{nl} is not depth-£rst.

3.1 DFA Results

First observe that DFA is optimal over a particular subclass of strategies:

Observation 3 DFA *produces a strategy that has the lowest cost among all depth-£rst strategies.*

Proof: By induction on the depth of the tree. Observations 1 and 2 establish the base case, for depth-1 trees. Given the depth-£rst constraint, the only decision to make when considering depth-$s+1$ trees is how to order the strategy subtree blocks associated with the depth-s `and-or` subtrees; here we just re-use Observations 1 and 2 on the mega-blocks. □

Observations 1 and 2 show that DFA produces the best possible strategy, for the class of depth-1 trees. Moreover, an inductive proof shows …

Theorem 4 DFA *produces the optimal strategies for depth-*2 `and-or` *trees, i.e., read-once DNF or CNF formulae.*

Theorem 4 holds for arbitrary costs — *i.e.*, the proof does not require unit costs for the tests. It is tempting to believe that DFA works in all situations. However …

Observation 5 DFA *does not always produce the optimal strategy for depth* 3 `and-or` *trees, even in the unit cost case.*

We prove this by just considering T_1 from Figure 1. As noted above, DFA will produce the $\xi_{\langle SMCF\rangle}$ strategy, whose expected cost (using Equation 2 with earlier results) is $C[\xi_{\langle SMCF\rangle}] = c(\mathsf{S}) + \Pr(\,-\mathsf{S}\,) \times c(\mathcal{A}_{MCF}) = 1 + 0.7 \times 2.04 = 2.428$. However, the ξ_{nl} strategy, which is *not* depth-£rst, has lower expected cost $C[\xi_{nl}] \ = \ 1 \ + \ 0.7[1 + 0.2 \times 1] \ + \ 0.3[1 + 0.7 \times (1 + 0.2 \times 1)] = 2.392$.

Still, as this difference in cost is relatively small, and as ξ_{nl} is not linear, one might suspect that DFA returns a reasonably good strategy, or at least the best *linear* strategy. However, we show below that this claim is far from being true.

In the unit-cost situation, the minimum cost for any non-trivial n-node tree is 1, and the maximum possible is n; hence a ratio of $n/1 = n$ over the optimal score is the worst possible — *i.e.*, no algorithm can be off by a factor of more than n over the optimum.

Theorem 6 *For every* $\epsilon > 0$*, there is a unit-cost* `and-or` *tree* T *for which the best depth-£rst strategy costs* $n^{1-\epsilon}$ *times as much as the best strategy.*

4 Linear Strategies

As noted above (De£nition 2) we can write down each of these DFA-produced strategies in a linear fashion; *e.g.*, $\xi_{\langle SMCF\rangle}$ can be viewed as test S, then if necessary test M, then if necessary test C and if necessary test F. In general…

De£nition 3 *A strategy is* linear *if it performs the tests in £xed linear order, with the understanding that the strategy will skip any test that will not help determine the value of the tree, given what is already known.*

Hence, $\xi_{\langle SMCF\rangle}$ will skip all of M, C, F if the S test succeeds; and it will skip the C and F tests if M fails, and will skip F if C succeeds.

As any permutation of the tests corresponds to a linear strategy, there are of course $n!$ such strategies. One natural question is whether there are simple ways to produce "good" strategies from this class. The answer here is "yes":

Observation 7 DFA *algorithm produces linear strategies.*

Proof: Argue by induction on the depth k. For $k = 1$, the result holds by Observations 1 and 2. For $k \geq 2$, use the inductive hypothesis to see that DFA will produce a linear ordering for each subtree (as each subtree is of depth $\leq k-1$). DFA will then form a linear strategy by simply sequencing the linear strategies of the subtrees. □

Observation 3 implies there is always a linear strategy (perhaps that one produced by DFA) that is at least as good as any depth-£rst strategy. Unfortunately the converse is not true — the class of strategies considered in Theorem 6 are in fact linear, which means the best depth-£rst strategy can cost $n^{1-\epsilon}$ times as much as the best *linear* strategy, for any $\epsilon > 0$.

Is there always a linear strategy whose expected cost is near-optimal? Unfortunately, …

Theorem 8 *For every* $\epsilon > 0$*, there is an* `and-or` *tree* T *for which the best linear strategy costs* $n^{1/3-\epsilon}$ *times as much as an optimal strategy.*

5 Precondition BPE Model

Some previous researchers have considered a generalization of our PBE model that identi£es a precondition with each test — *e.g.*, test T can only be performed after test S has been performed and returned $+$. We show below that we get identical results even in this situation.

To motivate this model, suppose there is an external lab that can determine the constituent components of some unknown food, and in particular detect whether it contains milk, fruit or cereal. As the post of£ce will, with probability $1 - \Pr(\,\texttt{i\#1}\,)$, lose packages sent to the labs, we therefore view `i#1` as a probabilistic test. There is also a cost for mailing a food sample to this lab $c(\texttt{i\#1})$, which is in addition to the cost associated with each of the speci£c tests (for Milk, or for Fruit, etc.). Hence if the £rst test performed in some strategy is `Milk`, then its cost will be $c(\texttt{i\#1}) + c(\texttt{M})$. If we later perform, say, `Fruit`, the cost of this test is only $c(\texttt{F})$ (and not $c(\texttt{i\#1}) + c(\texttt{F})$), as the sample is already at the lab.

This motivates the notion of a "preconditioned probabilistic boolean expression" (P-PBE) which, in the context of `and-or` trees, allows each internal node to have both a cost and a probability. Notice this cost structure means that a pure or-tree will not collapse to a single level, but can be of arbitrary depth. (In particular, we cannot simply incorporate the cost $c(\texttt{i\#2})$ into both F and C, as only the £rst of these tests, within any evaluation process, will have to pay this cost. And we cannot simply associate this cost with one of these tests as it will only be required by the £rst test, and we do not know which it will be; indeed, this £rst test could, conceivably, be different for different situations.)

We de£ne the *alternation number* of an `and-or` tree to be the maximum number of alternations, between `and`-nodes and `or`-nodes, in any path from the root. Notice that the alternation number of a strictly alternating `and-or` tree is one less than its depth. For example, T_1 in Figure 1 has depth 3 and alternation number 2.

5.1 Previous P-PBE Results

There are a number of prior results within this P-PBE framework. Garey [Gar73] gives an ef£cient algorithm for £nding optimal resolution strategies when the precedence constraints can be represented as an `or` tree (that is, a tree with no conjunctive subgoals); Simon and Kadane [SK75] extend this to certain classes of `or` DAGs (directed-acyclic-graphs whose internal nodes are all `or`). Below we will use the Smith [Smi89] result that, if the P-PBE is read-once and involves only `or` connections, then there is an ef£cient algorithm for £nding the optimal strategy — essentially linear in the number of nodes [Smi89]. However, without the read-once property, the task becomes NP-hard; see [Gre91]. (Sahni [Sah74] similarly shows that the problem is NP-hard if there can be more than a single path connecting a test to the top level goal, even when all success probabilities are 1.)

One obvious concern with this P-PBE model is the source of these probability values. In the standard PBE framework, it is fairly easy to estimate the success probability of any test, by just performing that test as often as it was needed. This task is more complicated in the P-PBE situation, as some tests can only be performed when others (their preconditions) had succeeded, which may make it dif£cult to collect suf£cient instances to estimate the success probabilities of these "blockable" tests. However, [GO96] shows that it is always possible to collect the relevant information, for any P-PBE structure.

5.2 P-PBE Results

Note £rst that every standard PBE instance corresponds to a P-PBE where each internal node has cost 0 and success probability 1. This means every negative result about PBE (Theorems 6 and 8) applies to P-PBE.

Our only positive result above is Theorem 4, which proved that an optimal strategy $\xi^*(T)$ for a depth-2 ($\approx$ 1-alternation) `and-or` tree T is depth-£rst; *i.e.*, it should explore each sub-tree to completion before considering any other sub-tree. In the DNF case $T \equiv (X_1^1 \& \ldots \& X_{k_1}^1) \vee \cdots \vee (X_1^m \& \ldots \& X_{k_m}^m)$, once we evaluate any term — say $(X_1^i \& \ldots \& X_{k_i}^i)$ — $\xi^*(T)$ will sequentially consider each of these X_j^i until one fails, or until they all succeed, but it will never intersperse any X_ℓ^j $(j \neq i)$ within this sequence.

This basic idea also applies to the P-PBE model, but using the [Smi89] algorithm to deal with each "pure" subtree, rather than simply using the Pr$(\pm X)/c(X)$ ordering. To state this precisely: A subtree is "pure" if all of its internal nodes all have the same label — either all "Or" or all "And"; hence the `i#2`-rooted subtree in Figure 1 is a pure subtree (in fact, every penultimate node roots a pure subtree), but the subtree rooted in `i#1` is not. A pure subtree is "maximal" if it is the entire tree, or if the subtree associated with the parent of its root is not pure; *e.g.*, the `i#2`-rooted subtree is maximal. Now let DFA* be the variant of DFA that forms a strategy from the bottom up: use [Smi89] to £nd a substrategy for each maximal pure subtree of the given and-or tree, compute the success probability p and expected cost c of this substrategy, then replace that pure subtree with a single mega-node with probability p and cost c, and recur on the new reduced `and-or` tree. On the Figure 1 tree, this would produce the reduced trees shown in Figure 2. DFA* terminates when it produces a single node; it is then easy to join the substrategies into a single strategy.

As a corollary to Theorem 4 (using the obvious corollary to Observation 3),

Corollary 9 *In the P-PBE setting,* DFA* *produces the optimal strategies for* 1*-alternation* `and-or` *trees.*

6 Conclusions

This paper addresses the challenge of computing the optimal strategy for `and-or` trees. As such strategies can be exponentially larger than the original tree in general, we investigate the subclass of strategies produced by the DFA algorithm, which are guaranteed to be of reasonable size — in fact, linear in the number of tests. After observing that these DFA-produced strategies are the optimal **depth-£rst** strategies, we then prove that these strategies are in fact the optimal possible strategies for trees with depth 1 or 2. However, for deeper trees, we prove that these depth-£rst strategies can be arbitrarily worse than the best linear strategies. Moreover, we show that these best linear strategies can be considerably worse than the best possible strategy. We also show that these results also apply to the more general model where intermediate nodes are also probabilistic tests.

Acknowledgements

All authors gratefully acknowledge NSERC. RH also acknowledges a University of Alberta Research Excellence Award; and MM also acknowledges a Sloan Research Fellowship. We also gratefully acknowledge receiving helpful comments from Adnan Darwiche, Rob Holte, Omid Madani and the anonymous reviewers.

References

[Bar84] J. Barnett. How much is control knowledge worth?: A primitive example. *Arti£cial Intelligence*, 22:77–89, 1984.

[Gar73] M. Garey. Optimal task sequencing with precedence constraints. *Discrete Mathematics*, 4, 1973.

[GB91] D. Geiger and J. Barnett. Optimal satis£cing tree searches. In *Proc, AAAI-91*, pages 441–43, 1991.

[GHM02] R. Greiner, R. Hayward, and M. Malloy. Optimal depth-£rst strategies for And-Or trees. Tech. rep, U. Alberta, 2002.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* 1979.

[GO91] R. Greiner and P. Orponen. Probably approximately optimal derivation strategies. In *Proc, KR-91*, 1991.

[GO96] R. Greiner and P. Orponen. Probably approximately optimal satis£cing strategies. *Arti£cial Intelligence*, 82, 1996.

[Gre91] R. Greiner. Finding the optimal derivation strategy in a redundant knowledge base. *Arti£cial Intelligence*, 50, 1991.

[HR76] L. Hya£l and R. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 35(1):15–17, 1976.

[Nat86] K. Natarajan. Optimizing depth-£rst search of AND-OR trees. Report RC-11842, IBM Watson Research Center, 1986.

[Sah74] S. Sahni. Computationally related problems. *SIAM Journal on Computing*, 3(4):262–279, 1974.

[SK75] H. Simon and J. Kadane. Optimal problem-solving search: All-or-none solutions. *Arti£cial Intelligence*, 6:235–247, 1975.

[Smi89] D. Smith. Controlling backward inference. *Arti£cial Intelligence*, 39(2):145–208, June 1989.