Optimal Depth-First Strategies for And-Or Trees

Tracking Number: 253

Abstract

A probabilistic boolean expression *(PBE) consists of a boolean expression over a set of boolean variables, each with a corresponding cost and probability value that indicates respectively the cost of determining a variable's value and the probability that the value is true. Given a PBE, a* resolution strategy *is a sequential testing algorithm that determines the value of the expression, where each test is a query of the value of one variable. A strategy is* optimal *if its expected cost is minimum, over all possible strategies. The* minimum cost resolution strategy problem *(MRSP) is to find an optimal strategy of a given PBE.*

As MRSP is NP-hard in general, we consider the restricted case in which each variable occurs exactly once; the corresponding expressions are sometimes called `and-or` trees, *since they have a tree representation in which internal nodes correspond to (boolean) operators and leaf nodes correspond to variables. We further assume that variables are independent (for otherwise the problem remains NP-hard), and focus on a depth-first algorithm,* DFA*, that orders subexpressions within subtrees based on probability/cost ratios.*

We prove that DFA *produces optimal strategies for* `and-or` *trees with depth 1 or 2, but its results can be very bad for* `and-or` *trees with depth 3 or more. We then note that these results also apply if these tests can have preconditions. Finally, we also consider another natural subclass of strategies — those that can be expressed as a linear sequence of variables. We show that the best such linear strategy can also be very much worse than the optimal strategy in general.*

Keywords: Satisficing search, Diagnosis, Computational complexity

1 Introduction

Baby J is a fussy eater. J likes only 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 likes some new food, and that you can test whether the food satisfies 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 finds a food yummy depends on the order in which tests are performed as well as the outcomes.

A strategy[1] describes the order in which tests are to be performed. For example, this is strategy $\xi_{\langle SMFC\rangle}$: perform the Sweet test, returning `true` (aka `Yummy`) if it succeeds; if it fails, $\xi_{\langle SMFC\rangle}$ will perform the Milk test, returning `false` if it fails. Otherwise (if Sweet fails and Milk succeeds), $\xi_{\langle SMFC\rangle}$ will perform the Fruit test, returning `false` if it fails; if it succeeds, perform the Cereal test, returning `true`/`false` if it succeeds/fails. See Figure 2(a). Of course, there are other strategies for this situation, including $\xi_{\langle SMCF\rangle}$, which differs from $\xi_{\langle SMFC\rangle}$ only by testing Fruit before Cereal, and $\xi_{\langle SFCM\rangle}$, which tests the `Fruit-Cereal` component before `Milk`. Notice both strategies are "correct", in that they each correctly determine whether J will like a food or not. 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, all of which return the correct answer, but which vary in terms of their *expected cost*. This paper discusses the task of computing the best — that is, minimum cost and correct — such strategy, for various classes of problems.

1.1 Framework

Each of the strategies discussed so far can be described in a *linear* fashion: proceed through the tests in the given order, omitting tests only if logically unnecessary. We consider each such strategy to be *linear*.

In fact, each of these strategies is *depth-first* in that, for each `and-or` subtree, the tests on the leaves of the subtree appear together in the strategy. We can also consider a strategy such as $\xi_{\langle CSMF\rangle}$ which is not depth-first, since it starts with `Cereal` and then moves on to `Sweet` before completing the evaluation of the `i#2`-rooted subtree.

As suggested by the notation, this non-depth-first strategy is still linear. Not all strategies are linear. Consider, for example the ξ_{nl} strategy, which proceeds as follows: First test `Cereal`. If positive test `Milk` then (if neccessary) `Sweet`. If the `Cereal` test is negative then test `Sweet` then (if neccessary) `Fruit` then `Milk`. ξ_{nl} is depicted in Figure 2(b).

[1] Formal definitions are presented in §2.

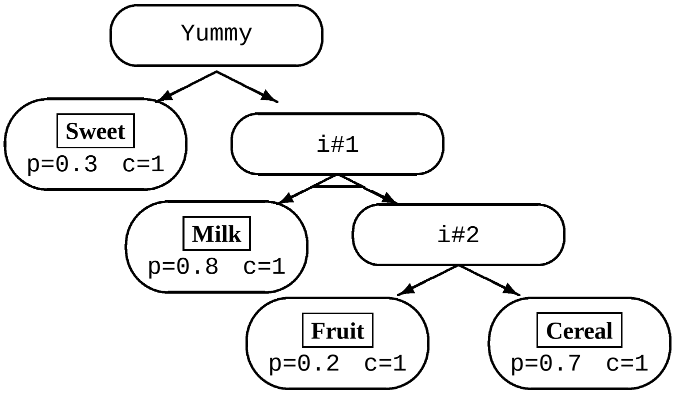

Figure 1: An `and-or` tree T_1. Here `and`-nodes are indicated with a horizontal bar through the descending arcs; `i#1` is an `and`-node while `Yummy` and `i#2` are `or`-nodes.

We consider only *correct* strategies, namely those which 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 that these tests are all independent of each other — *e.g.*, Figure 1 indicates that 30% of the foods sampled are Sweet, 80% are with Milk, etc. While each stategy returns the correct boolean value, they have different expected costs. For example, the expected cost of $\xi_{\langle SMFC\rangle}$ is

$$C[\xi_{\langle SMFC\rangle}] \quad = \quad c(\mathtt{S}) + \Pr(-\mathtt{S}) \times [c(\mathtt{M}) + \Pr(+\mathtt{M}) \times [c(\mathtt{F}) + \Pr(-\mathtt{F})\; c(\mathtt{C})]]$$

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) — *i.e.*, 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. The problem considered in this paper is, given such a formula, probabilities and costs, determine an optimal strategy; this problem is the minimum cost resolution strategy problem (MRSP).

In §2 we describe these and related notions more formally. §3 describes an algorithm, DFA [Nat86], that produces the optimal depth-first linear strategy, then proves several theorems about this algorithm; in particular, Theorem 4 shows that the DFA strategy is optimal for `and-or` trees with depth 1 or 2, while Theorem 6 proves it can be quite far from optimal in general. §4 then formally defines linear strategies, and proves (Theorem 8) that the best linear strategy can be far from optimal. §5 motivates and defines an extension to the PBE model, called "Precondition PBE", where each test can only performed in some context, and shows that the same results apply. The extended paper 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 *satisfi cing search* in place of *resolution 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 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 finding 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".[2] 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 efficient 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, which have depth 2 or 3, but can do very poorly in general.

We consider the tests to be statistically independent of each other; this is why we can succinctly write $\Pr(+X)$, as it does not depend on 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 Definitions

We focus on read-once formulae, which correspond to `and-or` trees — *i.e.*, a tree structure whose leaf nodes are each labeled with a probabilistic test (with a known positive cost[3]

[2]This problem also maps immediately to a "probabilistic series/parallel task", where each arc in a graph corresponds to a probabilistic test, where success (resp., failure) means there is a flow possible (resp., not possible) from a specified source node to a target. The challenge now is to determine the best arcs to test, to determine whether there will be flow in a given situation. [Colbourn, personal conversation, 1998].

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

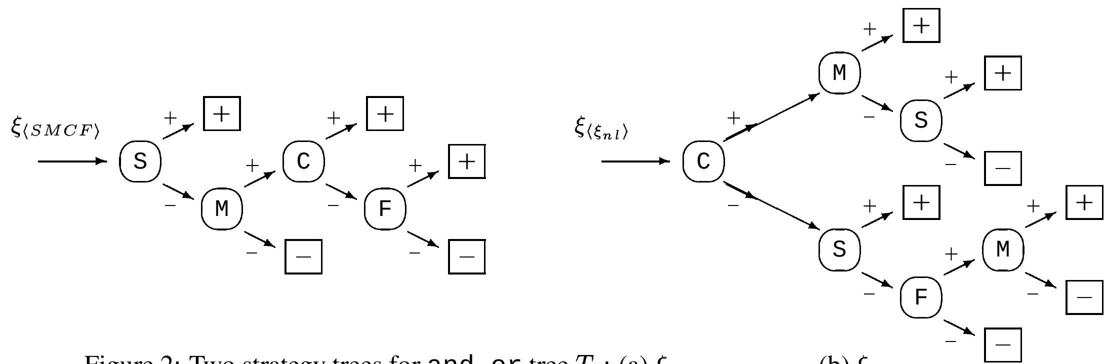

Figure 2: Two strategy trees for `and-or` tree T_1: (a) $\xi_{\langle SMCF \rangle}$ (b) ξ_{nl}.

and success probability) and whose internal nodes are each labeled as an `or`-node or `and`-node, with the understanding that the subtree rooted at an `and`-node (`or`-node) is satisfied if and only if all (at least one) of the subtrees are satisfied.

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. Notice that there is no unambiguous way to write ξ_{nl} as a simple linear sequence of experiments, as for some test assignments `M` precedes `S`, while in others `S` precedes `M`.

For any `and-or` tree T, we will only consider the set of *correct* strategies $\Xi(T)$, namely those which 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, over all assignments, namely

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

Given the indepedence of tests, there is a more efficient way to evaluate a strategy than the algorithm implied by Equation 1. Extending $C[\cdot]$ 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

$$C[\varphi_\chi] \quad = \quad c(\chi) \quad \begin{aligned} &+ \Pr(\,+\chi\,) \times C[\varphi_{+\chi}] \\ &+ \Pr(\,-\chi\,) \times C[\varphi_{-\chi}] \end{aligned} \tag{2}$$

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

To define our goal:

Definition 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 define 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-first algorithm DFA

To help define our DFA algorithm, we first 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 linear strategy $A_{\pi_1}, \ldots, A_{\pi_r}$, where π is defined so that $Pr\big(\,+A_{\pi_j}\,\big)\,/c(A_{\pi_j}) \;\geq\; Pr\big(\,+A_{\pi_{j+1}}\,\big)\,/c(A_{\pi_{j+1}})$ for $1 \leq j < r$. See Figure 3(c).*

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 linear 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 $\Pr(\,+A\,)\,/c(A) < \Pr(\,+B\,)\,/c(B)$. Now consider the strategy ξ_{BA} that reordered these tests; and observe that ξ_{BA}'s expected cost

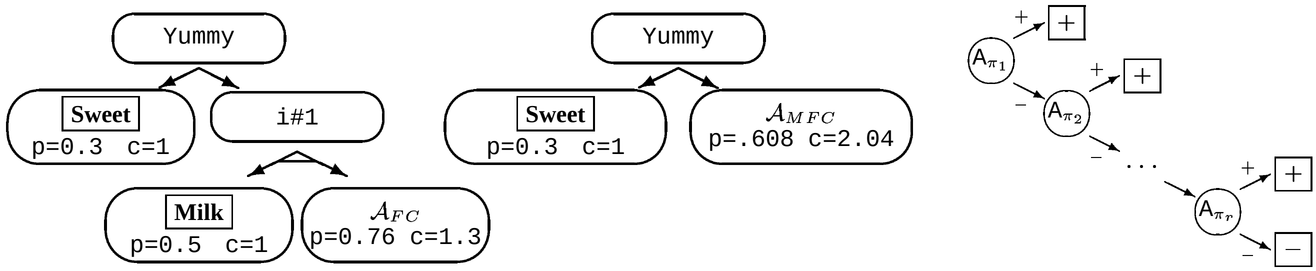

Figure 3: Intermediate results of DFA on T_1 (a) after 1 iteration (b) after 2 iterations. (c) A linear strategy tree.

is strictly less than ξ_{AB}'s, contradicting the claim that ξ_{BA} was optimal. □

An identical proof shows …

Observation 2 *Let T_A be a depth 1 tree whose root is an* and *node, defined analogously to T_O in Observation 1. Then the optimal strategy for T_A is the linear strategy $A_{\phi_1}, \ldots, A_{\phi_r}$, where ϕ is defined 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 first deal with the bottom tree layer, and order the children of each final 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 this or-node case; the and-node case is analogous.) For example, if dealing with Figure 1's T_1, DFA would compare $\Pr(+\mathtt{F})/c(\mathtt{F}) = 0.2/1$ with $\Pr(+\mathtt{C})/c(\mathtt{C}) = 0.7/1$, and order C first, 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 $\Pr(+\mathcal{A}_{FC}) = 1 - (\Pr(-\mathtt{F}) \times \Pr(-\mathtt{C})) = 1 - 0.8 \times 0.3 = 0.76$, and cost $c(\mathcal{A}_{FC}) = c(\mathtt{C}) + \Pr(-\mathtt{C}) \times c(\mathtt{F}) = 1 + 0.3 \times 1 = 1.3$; see Figure 3(a).

Now recur: consider the and-node that is the parent to various children, including this mega-node $\mathcal{A}$. This test gets inserted based on its $\Pr(-\mathcal{A})/c(\mathcal{A})$ value, and so forth.

On T_1, DFA would then compare $\Pr(-\mathtt{M})/c(\mathtt{M}) = 0.2/1$ with $\Pr(-\mathcal{A}_{FC})/c(\mathcal{A}_{FC}) = 0.24/1.3$ and so select the M-labeled node to go first. Hence, the substrategy associated with the i#1 subtree will first perform M, and return $-$ if unsuccessful. Otherwise, it will then perform the $\mathcal{A}_{FC}$ megatest: Here, it first performs C, and return $+$ if it 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 $\Pr(+\mathcal{A}_{MFC}) = \Pr(+\mathtt{M}) \times \Pr(+\mathcal{A}_{FC}) = 0.8 \times 0.76 = 0.608$, and cost $c(\mathcal{A}_{MFC}) = c(\mathtt{M}) + \Pr(+\mathtt{M}) \times c(\mathcal{A}_{FC}) = 1 + 0.8 \times 1.3 = 2.04$; see Figure 3(b).

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

Observe first that DFA is very efficient: indeed, 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 $d^+(v)$ is the out-degree of the node v, n is the total number of nodes in the and-or tree, and r is 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-first strategy*. To state this more precisely,

Definition 2 *A strategy ξ is* depth-first *if the tests associated with each* and-or *sub-tree appear together in the strategy ξ. Namely, for any* and-or *subtree T', let $v(T')$ be the tests labeling nodes within T'; then every place any test in T' appears in the ξ strategy, it appears in a contiguous subtree that includes every test of $v(T')$.*

The strategy $\xi_{\langle SMCF\rangle}$, shown in Figure 2(a), is depth-first, as every time C appears it is next to its sibling F (so all of the children of i#2 appear in a continguous region); similarly, there is a contiguous region that contains all-and-only the tests under i#1 — M, C and F. By contrast, the $\xi_{\langle CMSF\rangle}$ strategy is not depth-first, as there is a path where C is not next to its sibling F; similarly ξ_{nl} (Figure 2(b)) is not depth-first.

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-first 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-first 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. □

Moreover, 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 can 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 $C[\xi_{\langle SMCF \rangle}] = c(\mathtt{S}) + \Pr(\ -\mathtt{S}\ ) \times c(\mathcal{A}_{MCF}) = 1 + 0.7 \times 2.04 = 2.428$. However, the ξ_{nl} strategy, which is *not* depth-first, has lower expected cost $C[\xi_{nl}] \ = \ 1 \ + \ 0.7[1 + 0.2 * 1] \ + \ 0.3[1 + 0.7 * (1 + 0.2 * 1)] = 2.392$. In fact, the reader can verify that ξ_{nl} is the unique optimal strategy.

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 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 *There are unit-cost* `and-or` *trees T for which the best depth-first strategy costs $\Theta(n^{1-o(1)})$ times as much as the best strategy.*

4 Linear Strategies

As noted above (Definition 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 S, then if necessary M, then if necessary C and if necessary F. In general,

Definition 3 *A strategy is* linear *if it performs the tests in fixed linear order, with the understanding that the strategy will skip any test that will not help answer the question, 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.

While it is not clear that an optimal strategy can always be expressed in poly(n) space (let alone determined in poly(n) time), these linear strategies can always be expressed very efficiently. This section therefore considers this subclass of strategies.

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 *The* DFA *algorithm produces a linear strategy.*

Proof: Argue by induction on the depth k. For $k = 0$, the result holds by Observations 1 and 2. For $k \geq 1$, 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. □

Given Observation 3, this means there is always a linear strategy (perhaps that one produced by DFA) that is at least as good as any depth-first strategy. Unfortunately the converse is not true — the class of strategies considered in Theorem 6 are in fact linear. This shows that the best depth-first strategy can cost $O(n^{1-o(1)})$ times as much as the best *linear* strategy.

The next natural question is how effective this class of linear strategies is in general — *i.e.*, is there always a linear strategy whose expected cost is near-optimal. Our next result shows that this is not the case:

Theorem 8 *There are* `and-or` *trees for which the best linear strategy costs $\Theta(n^{1/3-o(1)})$ times as much as an optimal strategy.*

5 Precondition BPE Model

Many previous researchers have considered a generalization of our PBE model that identifies a precondition with each test — *e.g.*, test T can only be performed after test S has been performed and returns $+$. We show below that we get identical results 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 office will, with probability $1 - \Pr(\ \mathtt{i\#1}\ )$, 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 $c(\mathtt{i\#1})$, which is in addition to the cost associated with each of the specific tests (for Milk, or for Fruit, etc.). Hence if the first test performed is `Milk`, then its cost will be $c(\mathtt{i\#1}) + c(\mathtt{M})$. If we later perform, say, `Fruit`, the cost of this test is only $c(\mathtt{F})$ (and not $c(\mathtt{i\#1}) + c(\mathtt{F})$), as the sample is already at the lab.

In our current context of `and-or` trees, this model permits each internal node to have both a non-negative cost and a probability. In general, we refer to this as a "preconditioned probabilistic boolean expression", P-PBE. 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 $c(\mathtt{i\#2})$ into both F and C, as only the first of these tests, along any evaluation process, will have to pay this cost. And we cannot simply associate it with one of these tests as it will only be required by the first test, and we do not know which it will be; indeed, this first test could, conceivably, be different for different situations.[4])

This motivates us to define 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.

[4]This cannot happen in linear strategies, which includes all of the strategies considered in this paper. However, even for linear strategies, we still do not know which of the tests will be first.

5.1 Previous P-PBE Results

There are a number of prior results within this P-PBE framework. Garey [Gar73] gives an efficient algorithm for finding 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 efficient algorithm for finding 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 model is the source of these probability values. In the standard PBE framework, it is fairly easy to simultaneously estimate the values of all probabilities from a data sample. 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 difficult to collect sufficient instances to estimate the success probabilities of these "blockable" tests. However, Greiner and Orponen [GO96] show that it is always possible to collect the relevant information, for any P-PBE structure.

5.2 P-BPE Results

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

Our only positive result above is Theorem 4, which basically proved that an optimal strategy $\xi^*(T)$ for a 1-alternation `and-or` tree T 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-BPE model — but using the [Smi89] algorithm to deal with each "pure" subtree, rather than simply considering $\Pr(\pm X)/c(X)$. 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 the subtree associated with the parent of its root is not pure. Now let DFA* be the variant of DFA which forms a strategy from the bottom up by using [Smi89] to find a substrategy for each maximal pure subtree of the given and-or tree, with an associated success probability p and expected cost c. After replacing that subtree with a single mega-node with probability p and cost c, DFA* recurs 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 stategy.

As a corollary to Theorem 4,

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-first** 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-first 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.

References

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

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

[GB91] D. Geiger and J. Barnett. Optimal satisficing tree searches. In *Proc, AAAI-91*, 1991.

[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 satisficing strategies. *Artificial Intelligence*, 82(1–2):21–44, April 1996.

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

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

[Nat86] K. Natarajan. Optimizing depth-first search of AND-OR trees. Tech. report, RC-11842, IBM 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. *Artificial Intelligence*, 6:235–247, 1975.

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