Optimal Depth-First Strategies for And-Or Trees

Russell Greiner and **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

January 2, 2002

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, 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.*

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

[1] Formal definitions are presented in §2.

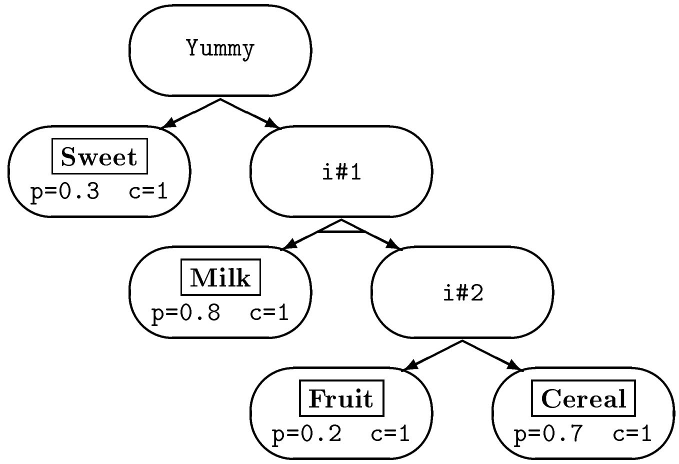

Figure 1: An `and-or` tree T_1. Here `and`-nodes are indicated with a horizontal bar through the descending arcs; `i#1` is an `and`-node while `Yummy` and `i#2` are `or`-nodes.

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 how 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).

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`.

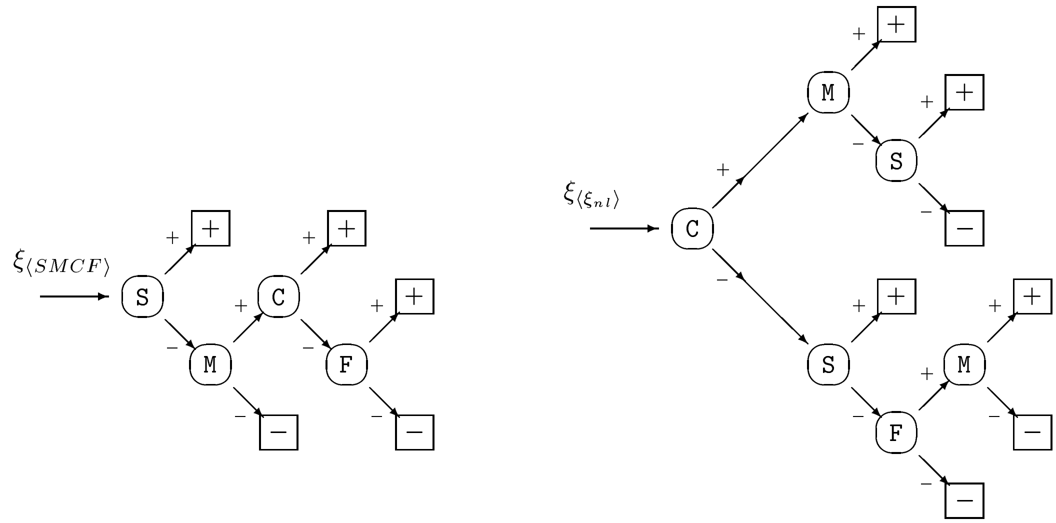

Figure 2: Two strategy trees for `and-or` tree T_1: (a) $\xi_{\langle SMCF \rangle}$ (b) ξ_{nl}.

Suppose that we are given a particular 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 appendix provides the proofs for the theorems. (We prove the observations in the text.)

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 *satisficing 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 to Exact-CoverBy3Set, 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 static 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 has same distribution as X_2, etc. We will therefore continue to consider the tests to be independent.

Note that we can view our model as a special class of Markov Decision Problem [RN95], where the state corresponds to the currently known set of assignments, and the actions are performing some test (*i.e.*, determining the value of some attribute), or simply returning either $+$ or $-$. Of course, the cost of test X is the $c(X)$, and there is an infinite cost for returning the wrong value. Hence, our results relate to (episodic) reinforcement learning [SB98], where our "strategies" correspond to reinforcement learning "policies". Our hardness results correspond to the "curse of dimensionality" [Bel61] (note that the state space is exponential in the number of variables), and our positive results show situations where the structure of the problem simplifies the task.

2 Definitions

As described above, an `and-or` tree is a tree structure whose leaf nodes are each labeled with a probabilistic test (with a known positive cost[3] 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 $\{-\mathtt{S},\ +\mathtt{M},\ -\mathtt{F},\ +\mathtt{C}\}$, 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`.

[2] This problem also maps immediately to a "probabilistic series/parallel task", where each arc in a graph is 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.

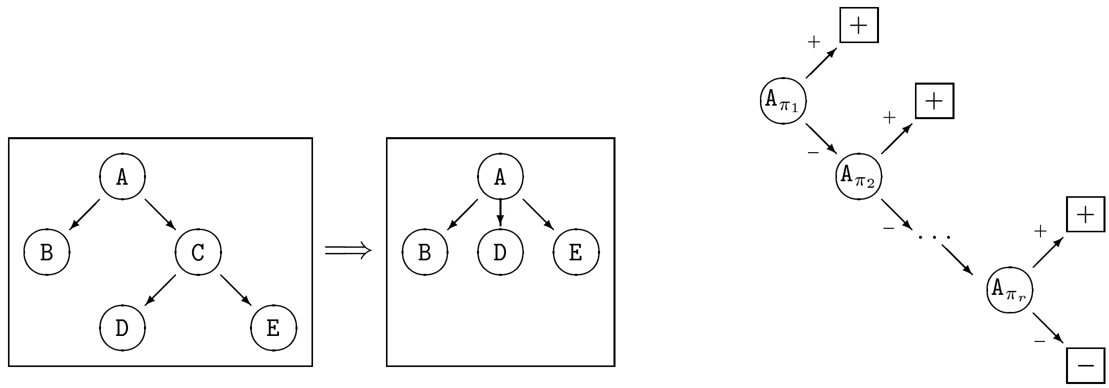

Figure 3: (a) Collapsing adjacent `or`-nodes into a single node. (b) A linear strategy tree.

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 $\mathtt{S} \vee (\mathtt{M} \wedge [\mathtt{F} \vee \mathtt{C}])$.

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 $\gamma = \{-\mathtt{S},\ +\mathtt{M},\ -\mathtt{F},\ +\mathtt{C}\}$, $k(\xi_{\langle SMFC \rangle}, \gamma) = c(\mathtt{S}) + c(\mathtt{M}) + c(\mathtt{F}) + c(\mathtt{C})$ while $k(\xi_{\langle CMSF \rangle}, \gamma) = c(\mathtt{C}) + c(\mathtt{M})$.

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

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

Given the indepedence of tests, there is a more efficient way to evaluate a strategy than the algorithm implied 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

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

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), as shown in Figure 3(a). 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.

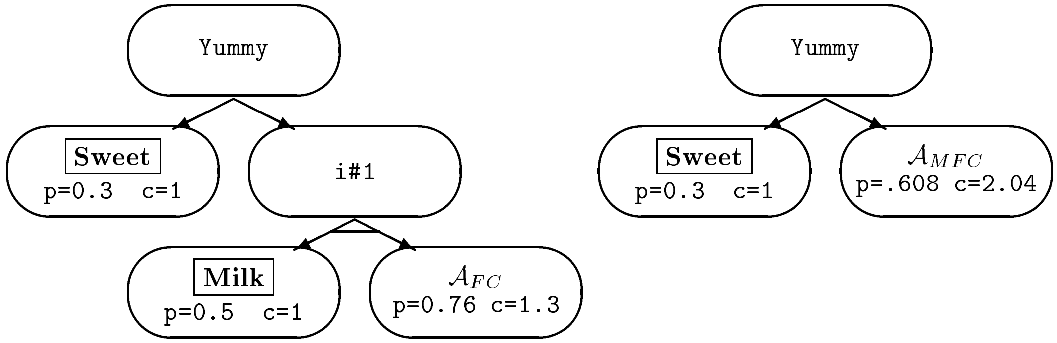

Figure 4: Intermediate results of DFA on T_1 (a) after 1 iteration (b) after 2 iterations.

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(b).*

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 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. $\square$

An identical prove 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 $Pr\Big(-A_{\phi_j}\Big)/c(A_{\phi_j}) \geq Pr\Big(-A_{\phi_{j+1}}\Big)/c(A_{\phi_{j+1}})$ 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.) If dealing with Figure 1's T_1, DFA would compare $\Pr(+\texttt{F})/c(\texttt{F}) = 0.2/1$ with $\Pr(+\texttt{C})/c(\texttt{C}) = 0.7/1$, 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 $\Pr(+\mathcal{A}_{FC}) = 1 - (\Pr(-\texttt{F}) \times \Pr(-\texttt{C})) = 1 - 0.8 \times 0.3 = 0.76$, and cost $c(\mathcal{A}_{FC}) = c(\texttt{C}) + \Pr(-\texttt{C}) \times c(\texttt{F}) = 1 + 0.3 \times 1 = 1.3$; see Figure 4(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 $\Pr(-\texttt{M})/c(\texttt{M}) = 0.2/1$ 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 $\mathcal{A}_{FC}$ mega-test: 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 4(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. Namely, for any* `and-or` *subtree T', let $v(T')$ be the tests labeling nodes within T'; then every time 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#2` — 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 (see Appendix) shows …

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

Notice this works for arbitrary costs — *i.e.*, the proof did 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 — just notice that 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 [GB91].

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 $1 - \Pr(\texttt{i\#1})$, 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 specific tests (for Milk, or for Fruit, etc.). Hence if the first test performed 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.

In our current context of `and-or` trees, we view this as having both a non-negative cost and a probability for each internal node, as well as each leaf node (test).

In general, we refer to this as an "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 $c(\texttt{i\#2})$ 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.

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 had succeeded, which may make it difficult to collect sufficient instances to estimate the success probabilities of these "blockable" tests. However, Greiner and Orponen [OG90, GO96] show that it is always possible to collect the relevant information, for any P-PBE structure.

[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.

5.2 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 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 consider all 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 subtree, rather than simply considering $\Pr(\pm X)/c(X)$ quantities. Hence, this means there is an efficient algorithm for finding the optimal strategy for the 1-alternation case in general.

6 Conclusions

Future Work: There are a number of remaining challenges. One question is whether there is an *efficient* algorithm for computing the optimal *linear* strategy for arbitrary `and-or` trees. Even though this can be $\theta(n^{1/3})$ inferior to the optimal strategy, at least we know this strategy can be expressed succinctly, which is not true in general.

A second question, suggested above, is whether there is an efficient algorithm for computing the optimal strategy, of any form, for arbitrary `and-or` trees. While this optimal strategy may be exponentially large, note that we never need to write it down; instead it is sufficient to simply determine, for any `and-or` tree, the *first* test to perform. Depending on its outcome, we can quickly reduce the given tree to the appropriate reduced tree, then run this "what to do now" algorithm on the result. If this first-test algorithm is efficient, we have an efficient algorithm for evaluating arbitrary `and-or` trees. (It is also interesting to ask whether this optimal strategy can always be described in poly space, if an oracle gives us the solution along with the shortest description of it for free.)

If there is such an efficient algorithm for arbitrary `and-or` trees. — *i.e.*, for arbitrary read-once formulae — we could also investigate whether there are efficient algorithms for read-k formulae, for $k < 5$. (Note our hardness proof of arbitrary boolean formulae required $k = 5$; see [GJ79].) If not, it would be interesting to determine if there are good approximation algorithms. While our results (Theorem 6) show we should not consider the DFA algorithm, there may be other algorithms — *e.g.*, Greiner and Orponen [GO91] provide an efficient algorithm that produces strategies within $O(\ln n)$ of optimal, for a specified subclass of P-BPE problems.

Contributions: 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 comment that these results also apply to the more general model where intermediate nodes are also probabilistic tests.

Acknowledgements

All authors gratefully acknowledge NSERC; in addition, 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 many colleagues including Adnan Darwiche and Rob Holte.

References

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

[Bel61] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

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

[GB91] Dan Geiger and Jeffrey A. Barnett. Optimal satisficing tree searches. In *Proc, AAAI-91*, pages 441–43, Anaheim, CA, 1991. Morgan Kaufmann.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[GO91] Russell Greiner and Pekka Orponen. Probably approximately optimal derivation strategies. In J.A. Allen, R. Fikes, and E. Sandewall, editors, *Proc, KR-91*, Rochester, NJ, April 1991. Morgan Kaufmann.

[GO96] Russell Greiner and Pekka Orponen. Probably approximately optimal satisficing strategies. *Artificial Intelligence*, 82(1–2):21–44, April 1996.

[Gre91] Russell Greiner. Finding the optimal derivation strategy in a redundant knowledge base. *Artificial Intelligence*, 50(1):95–116, 1991.

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

[Nat86] K. S. Natarajan. Optimizing depth-first search of AND-OR trees. Technical report, Research report RC-11842, IBM T. J. Watson Research Center, January 1986.

[OG90] Pekka Orponen and Russell Greiner. On the sample complexity of finding good search strategies. In *Proceedings of COLT-90*, pages 352–358, Rochester, August 1990.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

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

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press (A Bradford Book), Cambridge, MA, 1998.

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

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

A Proofs

Proof of Theorem 4: We will consider CNF formulae,

$$F \quad \equiv \quad (A_1^1 \vee \cdots \vee A_{n_1}^1) \;\wedge\; (A_1^2 \vee \cdots \vee A_{n_2}^2) \;\wedge\; \cdots \;\wedge\; (A_1^r \vee \cdots \vee A_{n_r}^r)$$

the proof for DNF formulae is identical.

Let ξ^* be any optimal strategy. By Observation 3, it suffices to show that ξ^* is depth-first. Here we use induction on the number of variables, $\ell = \sum_i n_i$. The base case is trivial. Now suppose the claim is true when there are $\leq \ell$ variable, and consider a formula with $\ell + 1$ variables. WLOG, the strategy will be of the form shown on left side of Figure 5, renumbering if necessary.

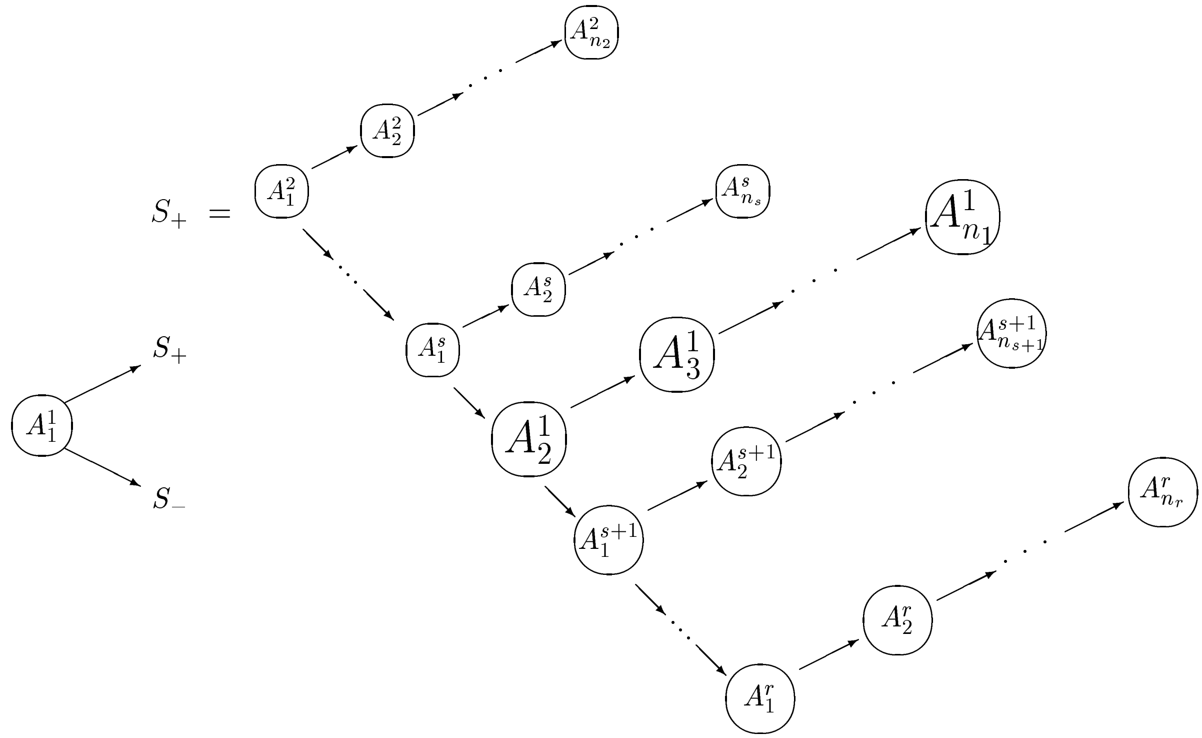

Figure 5: Form for Optimal Strategy

If this A_1^1 test fails, we pursue the optimal substrategy S_- over the remaining expression that excludes the term A^1. As this involves only $\ell + 1 - n_1 \leq \ell$ variables, S_- will be depth-first by the inductive hypothesis. Alternatively if A_1^1 succeeds, we then consider a substrategy S_+ over ℓ variables, which means it will also be depth-first;

in particular, it will be of the form shown on right side of Figure 5 — first dealing with the $s-1$ terms $A^2, \ldots, A^s$, then a diminished version of the term A^1 that omits A_1^1 (call it $B = \langle A_2^1, \ldots, A_{n1}^1 \rangle$), then the remaining $r-s$ terms $A^{s+1}, \ldots, A^r$.[5] Notice that we allow $s = 1$; here S_+ will begin with A_2^1, and hence the overall strategy ξ^* will be depth-first.

We can use Observation 1 to see that these terms will appear in descending order of $\Pr\big(\, A^k \,\big) / c(A^k)$,

$$\frac{\Pr(\, A^2 \,)}{c(A^2)} \;\geq\; \ldots \;\geq\; \frac{\Pr(\, A^s \,)}{c(A^s)} \;\geq\; \frac{\Pr(\, B \,)}{c(B)} \;\geq\; \frac{\Pr(\, A^{s+1} \,)}{c(A^{s+1})} \;\geq\; \ldots \;\geq\; \frac{\Pr(\, A_n \,)}{c(A^n)}$$

where $\Pr(\, A \,) = \prod_i \Pr(\, A_i \,)$ is the product of $\Pr(\, A_i \,)$ over the tests in that term, and for each term A in F, $c(A) = c(A_1) + \Pr(\, A_1 \,)\, [c(A_2) + \Pr(\, A_2 \,) \times \{\ldots\}]$ where the A_i's are ordered in descending values of $\Pr(\, \neg A_i \,) / c(A_i)$; *i.e.*,

$$\frac{\Pr(\, \neg A_1 \,)}{c(A_1)} \;\geq\; \frac{\Pr(\, \neg A_2 \,)}{c(A_2)} \;\geq\; \ldots \;\geq\; \frac{\Pr(\, \neg A_n \,)}{c(A_n)}$$

Now observe that the other substrategy S_- will include essentially the same terms, in the same order, differing only by not including B.

[5] If the substrategy S_+ ever reaches $A_{n_k}^k$ which succeeds, we return success. Alternatively, if any A_m^k fails, S_+ continues with A_1^{k+1} (or A_2^1 for A_m^s tests), unless $k = r$; if A_m^r fails, S_+ will fail.

As noted earlier, if $s = 1$ — *i.e.*, $B \equiv A^1_{2..n_1}$ is the first term — then we are done. So, towards a contradiction, suppose this is not the case. Call this purportedly-optimal strategy S_{Bad}. Here, notice the $+$-link from A^1_1 points to A^2_1, as does its $-$-link.

Using

$$\begin{aligned}
\sigma_1 &= \sum_{i=1}^{s} c(A^i) \times \prod_{j=1}^{i-1} \Pr\left(\neg A^j\right) \\
\sigma_2 &= \sum_{i=s+1}^{n} c(A^i) \times \prod_{j=1}^{i-1} \Pr\left(\neg A^j\right) \\
\rho &= \prod_{j=1}^{s} \Pr\left(\neg A^j\right)
\end{aligned}$$

its cost is

$$\begin{aligned}
C[S_{Bad}] &= c(A^1_1) + \Pr(A^1_1)\,[\sigma_1 + \rho \times [c(B) + \Pr(\neg B) \times \sigma_2]] \;+\; \Pr(\neg A^1_1)\,[\sigma_1 + \sigma_2] \\
&= c(A^1_1) + \sigma_1 + [\Pr(A^1_1) \times \Pr(\neg B) + \Pr(\neg A^1_1)] \times \sigma_2 + \Pr(A^1_1) \times \rho \times c(B) \\
&= c(A^1_1) + \sigma_1 \;+\; [\Pr(\neg A^1) \times \sigma_2] \;+\; [\Pr(A^1_1) \times \rho \times c(B)]
\end{aligned}$$

Now let S_{Good} be a different strategy: the depth-first strategy that deals with the terms in the order $A^2, \ldots, A^s,\ A^1,\ A^{s+1}, ..., A^n$. (That is, this complete strategy corresponds to S_+, but has A^1_1 inserted before A^1_2.)

The cost of the S_{Good} strategy is

$$C[S_{Good}] \quad = \quad \sigma_1 \quad + \quad \rho \times c(A^1) \quad + \quad \Pr\left(\neg A^1\right) \times \sigma_2$$

Now observe that $C[S_{Good}] \leq C[S_{Bad}]$:

$$\begin{aligned}
&C[S_{Bad}] - C[S_{Good}] \\
&\quad = [c(A^1_1) + \sigma_1 + \Pr(\neg A^1) \times \sigma_2 \;+\; \Pr(A^1_1) \times \rho \times c(B)] - [\sigma_1 \;+\; \rho \times c(A^1) \;+\; \Pr(\neg A^1) \times \sigma_2] \\
&\quad = c(A^1_1) \;+\; \rho\,[\Pr(A^1_1) \times c(B) - c(A^1)] \\
&\quad \geq c(A^1_1) + \rho\,[-c(A^1)] \quad = \quad c(A^1)\,[1 - \rho] \quad \geq \quad 0
\end{aligned}$$

as $c(A^1) \geq 0$ and ρ is a product of numbers ≥ 0.

(This used the observation that $c(A^1) \;\leq\; c(A^1_1) + p(A^1_1) \times c(B)$ where equality holds only if A^1_1 is the "best" of A^1 — *i.e.*, maximizes $\Pr(A^1_i)\,/c(A^1_i)$.)

This means the alleged optimal strategy, S_{Bad}, is not optimal, and hence we must have $s = 1$. Hence, there is an optimal strategy that is depth-first. □

Proof of Theorem 6: To simplify the proof (here and for Theorem 8), we will allow some tests to be true with probability 1. However, a strategy must still perform the test, to confirm that it is true, before using the fact that it is true to evaluate the tree. This convention allows for a more easily presentable construction. If the reader is uncomfortable with this rule, then by replacing each probability 1 with a value that is a negligible distance away from 1 (eg $1 - 2^{-n}$), it is easy to prove the theorem using a construction in which all probabilities are strictly between 0 and 1.

We use the unit-cost strictly alternating `and-or` tree suggested by Figure 6: The root is an OR node with r identical children, `#1` through `#r`, each of which has 2 children (`#i.a` and `#i.b`), each of which has 2 children (`#i.*.X` and `#i.*.Y`, where `*` is one of {`a`,`b`}). Each `*.X` is a leaf node

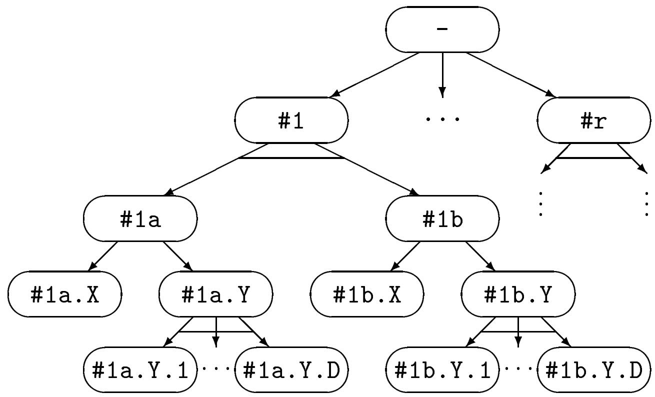

Figure 6: A problematic instance for DFA; Theorem 6

that is `true` with probability $1-q$ where $q = \frac{1}{2}(n/2r)^{-1/r}$. Each `*.Y` node has $D = (n-1)/2r - 4$ children, each of which is `true` with probability 1. Thus the total number of nodes is n and the tree evaluates to true, although it may be expensive to perform the evaluation.

A good strategy is to carry out all the `*.X` nodes first, $\langle$`#1a.X`, `#1b.X`, $\ldots$, `#ra.X`, `#rb.X`$\rangle$, and then, if neccessary, carry out the children of some of the `*.Y` nodes. Letting i be the index of the first, if any, pair (`#ia.X`,`#ib.X`) to both evaluate to `true`, this has an expected total cost of less than $\sum_{i=1}^{r}(2q)^{i-1}(1-2q) + (2q)^r \times O(D) < (1-2q)^{-1} + O(1) = O(1)$.

However, the optimal depth-first strategy (produced by DFA) has expected cost greater than $qD = \Theta((n/2r)^{1-1/r})$. To see this, assume by symmetry that the strategy evaluates subtree `#1a` first. With probability at least q, it will have to evaluate subtree `#1a.Y` at a cost of D. Setting $r = \log n$ yields the theorem. $\Box$

Proof of Theorem 7: We consider the following strictly alternating unit-cost tree, suggested by Figure 7. The root is an Or node which has $r = \lfloor n^{1/3}/\log n \rfloor$ identical children. Each level 1 node is an and-node with $r+1$ children – `#ia`, `#ib`$_1$, $\ldots$ `#ib`$_r$. Each `#ib`$_i$ is true with probability 1. `#ia` has 2 children: `#ia.X` which is a leaf that is True with probability $q = \log^2 n/(n^{1/3})$; and `#ia.Y`, which has D children, each of which is True with probability 1, where $D \approx r/q - r \approx n^{2/3}/\log^3 n - r$ is the solution to $r(r+D+4)+1 = n$. Thus the total number of nodes is n and the tree evaluates to true, although it may be expensive to perform the evaluation.

A good strategy is to start by evaluating all the `#ia.X` nodes, until one of them turns out to be true. Then evaluate the `#ib`$_i$ nodes that are the uncles of the true `#ia.X` node. In the unlikely chance that all the `#ia.X` nodes are false, evaluate a `#ia.Y` node and then the corresponding `#ib`$_i$ nodes. The probability that one has to do this is $(1-q)^r < \frac{1}{n} < q$. The expected cost of this strategy is at most $r + r + (1-q)^r \times D < r + r + qD \approx 3r$.

We will prove that every linear strategy has expected cost at least $\Theta(D)$. Since $D \approx r/q$, and $1/q = n^{1/3-o(1)}$, this proves the theorem.

Consider an optimal linear strategy. Thus the strategy can be represented by an an ordering of the leave, which we denote by S. We evaluate the leaves in this order, skipping a leaf iff by the time we get to it, it is *redundant*, meaning that one of its ancestor nodes has already been evaluated.

We start by proving a few basic properties about S.

Claim 1: *For any i, we can assume that* `#ia.Y.1, ..., #ia.Y.D` *occur as a consecutive subsequence within S.*

Proof: Suppose that they do not form a consecutive subsequence. Then there is some `#ia.Y.j1` occurring before some `#ia.Y.j2` such that there is a non-empty sequence of leaves between them, none of which are equal to some `#ia.Y.j3`. Consider moving `#ia.Y.j1` ahead in the sequence to the spot just before `#ia.Y.j2`, thus creating a new sequence S'. It is easy to see that no leaf will

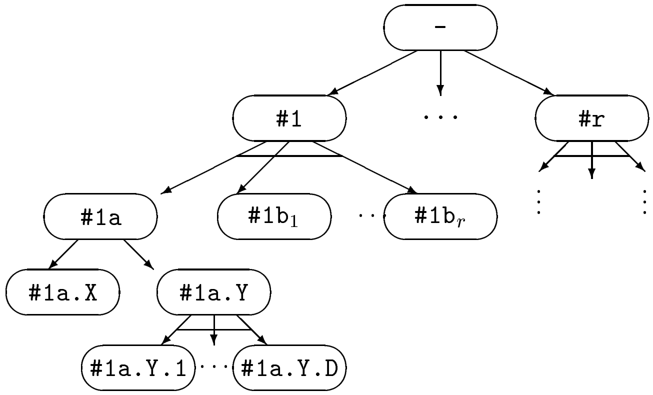

Figure 7: A problematic instance for Linear Strategies; Theorem 8

be evaluated in S' but not in S. This is because evaluating `#ia.Y.j1` cannot cause any ancestor to be evaluated until all the `#ia.Y.j` leaves are evaluated. Therefore, this operation does not increase the expected cost of the strategy. Repeating this operation enough times will produce a strategy in which all `#ia.Y.j` leaves occur as a consecutive subsequence. □

The same reasoning proves:

Claim 2: *For any i, we can assume that* `#i`b_1, ..., `#i`b_r *occur as a consecutive subsequence within S.*

Thus, we can collapse these two sets of leaves into the "superleaves" `#ia.Y` and `#iB` which are each true with probability 1 and which have costs r,D respectively.

Claim 3: *For any i, we can assume that* `#iB` *occurs immediately after either* `#ia.X` *or* `#ia.Y`*.*

Proof: Carrying out `#iB` cannot cause any ancestor to be evaluated unless we also know the result of either `#ia.X` or `#ia.Y`. Furthermore, carrying out `#ia.X` or `#ia.Y` cannot cause `#i` to be evaluated unless we also know the result of `#iB`. The rest of the argument is similar to the proof of Claim 1. □

Claim 4: *For any i, we can assume that* `#ia.X` *either occurs immediately before either* `#iB` *or* `#ia.Y`*, or occurs after both of them.*

Proof: Carrying out `#ia.X` cannot cause any ancestor to be evaluated unless we also know the result of either `#ia.B` or `#ia.Y`. The rest of the argument is similar to the proof of Claim 1. □

The same argument yields:

Claim 5: *For any i, we can assume that* `#ia.Y` *either occurs immediately before either* `#iB` *or* `#ia.X`*, or occurs after both of them.*

This establishes that each `#iB` is the end of one of the following consecutive subsequences: `#ia.X, #ia.Y, #iB`; `#ia.Y, #ia.X, #iB`; `#ia.X, #iB`; or `#ia.Y, #iB`. If it is one of the last two, then `#ia.Y` (resp. `#ia.X`) occurs later in S. Note that if it is the last choice, then `#ia.X` will always be redundant, and so its exact location in S is irrelevant.

Let S_i denote the particular consecutive subsequence described above ending with `#iB`. If `#i.Y` is in S_i, then the expected cost of carrying out S_i (if it is carried out) is at least $(1-q)D \approx D$, and with probability 1, this will successfully evaluate the root. Otherwise, the expected cost is at least r and with probability q this will successfully evaluate the root.

By symmetry, we can assume that $S_1, S_2, ..., S_r$ occur in that order in S. Let i^* be the first i such that either `#i.Y` is in S_i or some `#j.Y` precedes S_i. (Note that in the latter case, `#j.Y` comes after S_j.) If there is no such i, then we set $i^* = r+1$. Then the expected total cost is at least

$$\left(\sum_{i=1}^{i^*-1} r \times (1-q)^{i-1} \right) + (1-q)^{i^*-1} \times D = \frac{1-(1-q)^{i^*-1}}{q} \times r + (1-q)^{i^*-1} \times D.$$

This is a linear combination of D and $r/q \approx D$, thus proving the lemma. $\square$