Finding Optimal Satisficing Strategies for And-Or Trees*

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

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

September 9, 2005

Abstract

Many tasks require evaluating a specified Boolean expression φ over a set of probabilistic tests whose costs and success probabilities are each known. A strategy *specifies when to perform which test, towards determining the overall outcome of φ. We are interested in finding the strategy with the minimum expected cost.*

As this task is typically NP-hard — for example, when tests can occur many times within φ, or when there are probabilistic correlations between the test outcomes — we consider those cases in which the tests are probabilistically independent and each appears only once in φ. In such cases, 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. In this paper we investigate "probabilistic and-or tree resolution" (*PAOTR*), namely the problem of finding optimal strategies for and-or trees.*

We first consider a depth-first approach: evaluate each penultimate rooted subtree in isolation, replace each such subtree with a single "mega-test", and recurse on the resulting reduced tree. We show that the strategies produced by this approach are optimal for and-or trees with depth at most two but can be arbitrarily sub-optimal for deeper trees.

Each depth-first strategy can be described by giving the linear relative order in which tests are to be executed, with the understanding that any test whose outcome becomes irrelevant is skipped. The class of linear strategies is strictly larger than depth-first strategies. We show that even the best linear strategy can also be arbitrarily sub-optimal.

We next prove that an optimal strategy honours a natural partial order among tests with a common parent node ("leaf-sibling tests"), and use this to produce a dynamic programming algorithm that finds the optimal strategy in time $O(d^2\,(r+1)^d)$, where r is the maximum number of leaf-siblings and d is the number of leaf-parents; hence, for trees with a bounded number of internal nodes, this run-time is polynomial in the tree size. We also present another special class of and-or trees for which this task takes polynomial time.

We close by presenting a number of other plausible approaches to PAOTR*, together with counterexamples to show their limitations.*

Keywords: satisficing search, diagnosis, and-or tree, computational complexity

1 Introduction

A doctor needs to determine whether her current patient has a certain disease. She knows that a positive liver biopsy would conclusively show this disease, as would finding that the patient is jaundiced and has a

*This article significantly extends [GHM02].

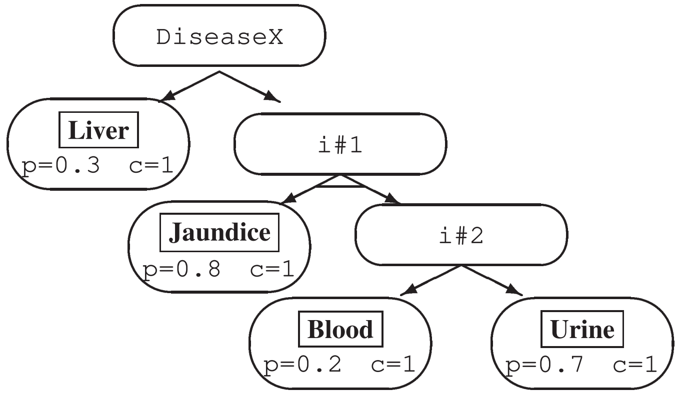

Figure 1: An and-or tree, T_1. Here and-nodes are indicated with a horizontal bar through the descending arcs; `i#1` is an and-node while `DiseaseX` and `i#2` are or-nodes.

certain compound in his blood or urine – that is,

$$\texttt{DiseaseX} \quad \Leftrightarrow \quad \texttt{Liver} \lor (\texttt{Jaundice} \land (\texttt{Blood} \lor \texttt{Urine}))$$

This situation corresponds to the and-or tree shown in Figure 1. We assume that each of the associated tests `L`, `J`, `B`, `U` has a known cost – say unit cost for this example. The outcome of one test may render other tests unnecessary; for example, if the liver test is positive, it does not matter whether the patient is jaundiced or not. Thus the cost of diagnosing this patient depends on the order in which tests are performed as well as on their outcomes.

A strategy[1] describes this testing order. For example, the strategy $\xi_{\langle LJUB \rangle}$ described in Figure 2(a) first performs the `L` test, returning `true`, namely the outcome +`DiseaseX`, if it succeeds; if it fails, $\xi_{\langle LJUB \rangle}$ performs the `J` test, returning `false`, namely −`DiseaseX`, if it fails. If `L` fails and `J` succeeds, $\xi_{\langle LJUB \rangle}$ performs the `U` test, returning `true` if it succeeds; if it fails, $\xi_{\langle LJUB \rangle}$ performs the `B` test, returning `true/false` if it succeeds/fails. There are other strategies for this situation, including $\xi_{\langle LJBU \rangle}$, which differs from $\xi_{\langle LJUB \rangle}$ only by testing `B` before `U`, and $\xi_{\langle LBUJ \rangle}$, which tests the `B-U` component before `J`. Notice that all these strategies correctly determine the patient's disease status. Moreover, each of these strategies typically performs only a subset of the tests before determining this status. Since, for a particular patient, different strategies might perform different tests, they could have different costs. If we know the distribution of patients considered and hence the likelihood that the various tests will succeed, we can then compute the *expected cost* of a strategy.

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 finding a best — namely, minimum expected cost — strategy, for various classes of these trees. We refer to this problem as "Probabilistic And-Or Tree Resolution" (PAOTR).

1.1 Overview

Each of the strategies discussed so far is *depth-first* in that, for each and-or rooted subtree, the tests on the leaves of the rooted subtree appear together in the strategy. We will also consider strategies that are not depth-first. For example, $\xi_{\langle ULJB \rangle}$ is not depth-first, since it starts with test `U` and then moves to test `L` before completing the evaluation of the `i#2`-rooted subtree.

Like the depth-first strategies, strategy $\xi_{\langle ULJB \rangle}$ is *linear*, as it can be described in a linear fashion: proceed through the tests in the given order, omitting any test that is logically unnecessary. The class of

[1] Formal definitions are presented in §2.

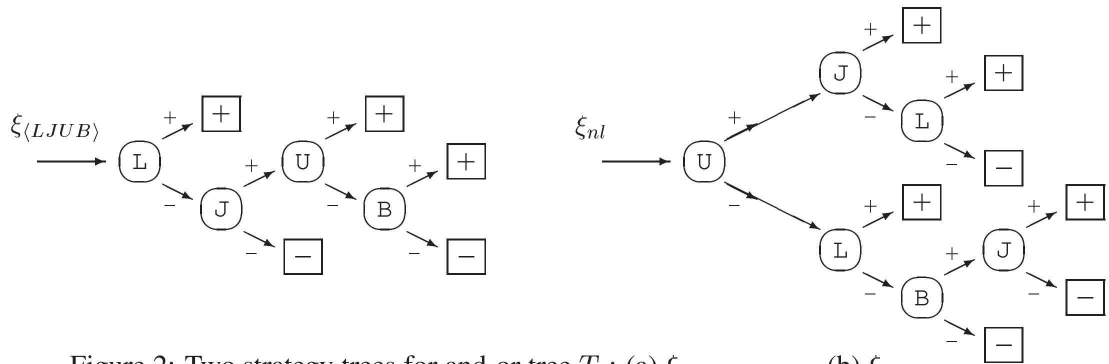

Figure 2: Two strategy trees for and-or tree T_1: (a) $\xi_{\langle LJUB \rangle}$ (b) ξ_{nl}.

linear strategies is so natural that it may be difficult to imagine strategies that do not have this property. Consider, however, the ξ_{nl} strategy, shown in Figure 2(b), which first tests `U` and, if positive, tests `J` and then if necessary `L`. However, if the `U` test is negative, ξ_{nl} then tests `L`, then if necessary `B`, and then if necessary `J`. No linear sequence can describe this strategy, as it tests `J` before `L` in some instances but `L` before `J` in others. As we will show, the best strategies are often non-linear, and in fact, the best linear strategy can in general be far from optimal.

We describe these and related notions more formally in §2. In §3 we discuss *depth-first strategies* in general and DFA [Nat86], an algorithm that produces a depth-first strategy, in particular. We show that DFA produces a strategy that, among depth-first strategies, has minimum expected cost, that DFA is optimal for and-or trees with depth at most two, but that DFA can be quite far from optimal in general. In §4 we discuss the larger class of *linear strategies* and show that the best linear strategy can be far from optimal.

In §5 we present a dynamic programing algorithm, DynProg, for finding an optimal strategy for *any* and-or tree. DynProg runs in time $O(d^2\,(r+1)^d)$, where r is the maximum number of tests with a common parent node and d is the number of leaf-parents. For trees with a bounded number of internal nodes, this run-time is polynomial. DynProg exploits the "Sibling Theorem", which shows that there is an optimal relative order for querying tests that are leaf-siblings. We also describe local conditions that guarantee that certain sibling tests can be performed together by an optimal strategy. We apply this result to produce poly-time solutions to PAOTR for other special and-or trees: trees with depth three whose tests are all identical.

PAOTR is surprisingly subtle. In §6 we present a number of plausible conjectures, followed in each case by a refuting counterexample. Finally, the appendices present the proofs of our theorems.

1.2 Related Work

We close this introduction by framing PAOTR and providing a brief literature survey. Table 1 summarizes previous work done on PAOTR. (Below, we extend PAOTR beyond just and-or trees, to apply to arbitrary Boolean formulae.)

The challenge of finding an optimal strategy for and-or trees is related to a large number of AI tasks. As our medical example suggests, it obviously connects to diagnosis, which has been a core AI topic since the first days. Many other application instances have been mentioned in the literature, including screening employment candidates [Gar73], competing in a quiz show [Gar73], mining for buried treasure [SK75], inferencing in an expert system [Smi89, GO91], and determining food preferences [GHM02].

Our goal is to compute a static strategy whose expected cost over a distribution of problems is minimum given the complete graph structure, namely the and-or tree representing the Boolean expression, together with the cost and probability information. This differs from the more familiar AI-style "heuristic search" algorithms such as A^* [Nil80, Pea84]. Such algorithms use only “local” structural information, namely nodes that are adjacent to the current node, and seek the heuristic cost function that is uniformly best, namely that expand the fewest number of nodes in every situation. These heuristic functions resemble strategies by implicitly specifying when and how to expand each node. Our strategies, however, are more fine-grained, as they can specify the proper action at each individual node, rather than just providing general “directions”.

We can view our task in the ”decision making under uncertainty” framework, as we are seeking a sequence of test-performing actions (aka a ”strategy”) that optimizes our ”expected utility” [RN95], based on a utility function that includes both the costs for performing the tests and an infinite penalty for making any mistakes (meaning we will only consider strategies that always return the correct value).

While an influence diagram (aka decision net [Sha86]) can be an effective tool for finding a *single* action or small fixed-length sequence of actions, there are challenges to scaling up to sequences that can be of variable length. This is true in our case, as a single strategy may require us to perform, say, a single test in one situation, four tests in another, and all of the tests in a third.

Note that our task is Markovian: after executing a subsequence of actions, we can encode the resulting state as an and-or tree, and know that this is sufficient for determining the optimal next action to take [Dre02]. Dynamic programming was designed to handle such problems; we explicitly use this technology in Section 5. Much of the field of Reinforcement Learning [SB98] involves providing clever tricks for obtaining relatively efficient approximations for complicated problems in this class. These would be overkill for our finite horizon ”simple evaluation” task.

The specific notion of PAOTR appears in Simon and Kadane [SK75], who use the term *satisficing search* in place of *strategy*. We motivate our particular approach by considering the complexity of PAOTR for various classes of probabilistic Boolean expressions.

Observation 1 PAOTR *is NP-hard, in the case of arbitrary Boolean formulae.*

This can be shown by reduction from satisfiability [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 in unnegated form. However, PAOTR remains NP-hard here, as we show in Theorem 25 (Appendix A). A further restriction is to consider “read-once” formulae, where each variable appears only one time. Observe that each and-or tree corresponds to a read-once formula, and each read-once formula with costs and success probabilities assigned for its variables corresponds to an and-or tree.[2] The complexity of PAOTR in this general case is not known.

Special cases of PAOTR have also been considered. Barnett [Bar84] investigated how the choice of optimal strategy depends on the probability values in the special case when there are two independent tests and so only two alternative search strategies. Geiger and Barnett [GB91] noted that optimal strategies for and-or trees cannot always be represented by a linear order of the nodes. Natarajan [Nat86] introduced the algorithm we call DFA for dealing with and-or trees, but did not investigate the question of when it is optimal. In this paper we show that DFA solves PAOTR for trees with depth at most two but can do poorly in general.

In considering PAOTR we assume that the tests are statistically independent of each other. For this reason, it suffices to consider individual, as opposed to conditional, probabilities when choosing the next test to perform. If we allow statistical dependencies, then the read-once restriction is not helpful, as we can

[2]This PAOTR problem also maps immediately to a “probabilistic series/parallel task”, where each arc in a graph corresponds to a probabilistic test, where success (respectively, failure) means that a flow is possible (not possible) from a specified source node through a sequence of arcs 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].

Table 1: Summary of Previous Results

Structure (with n nodes)	**Test Dependency**†	**Precondition?**†	**Results**
or tree and tree	independent	yes	$O(n \ln n)$ [Smi89]
or dag and dag	independent	yes	NP-hard [Gre91]
or tree and tree	dependent	yes	NP-hard [Observation 3]
and-or tree depth 2 ($\equiv$ read-once CNF, DNF)	independent	no	$O(n \ln n)$ §3
and-or tree with bounded d leaf-parents, each with at most r leaf-children	independent	no	$O(d^2\,(r+1)^d)$ §5
and-or tree $\equiv$ read-once Boolean formula	independent	no	?
and-or tree	dependent	no	NP-hard [Observation 2]
and-or dag $\equiv$ positive Boolean formula	independent	no	NP-hard [Theorem 25]
Boolean formula	independent	no	NP-hard [Observation 1]

And-or tree with n nodes, where r is the largest number of tests with the same parent node, and d is the number of leaf-parents, which is at most the number of internal nodes.

The contributions of this paper are boxed.

† Each NP-hardness result that holds for independent tests also holds for dependent tests, and each NP-hardness result that holds with no precondition also holds with preconditions.

convert any non-read-once but independent and-or tree to a read-once but correlated and-or tree by changing the j-th occurrence of the test "X" to a new test "X_j", and then insisting that X_j be equal to each other version of test X — i.e. $P(X_j = x | X = x) = 1$. This means...

Observation 2 *It is NP-hard to compute the optimal strategy for an and-or tree whose tests are correlated.*

In this paper we further assume that any test can be performed at any time. In a more general version, tests may have preconditions. For example, a blood test cannot be performed until the blood shipment has reached a laboratory; this shipment might have a non-zero cost and a non-unit success probability. This results in a more complicated cost structure on the and-or tree, as costs are now associated with internal nodes as well as leaves. This also means we should not collapse adjacent or-nodes (resp., and-nodes). Greiner [Gre91] shows that it is NP-hard to find the optimal strategy for such "preconditioned" or dags (resp., "preconditioned" and dags). We can use the same "DAG + independent-tests $\equiv$ tree + dependent-tests" reduction to show...

Observation 3 *It is NP-hard to compute the optimal strategy for an and-tree (resp., or-tree) with preconditions, when the tests are correlated.*

In [Jan03] Jankowska considers this more general "test precondition" version of PAOTR. She also shows how to reduce an arbitrary and-or tree to one in which all tests have the same cost and then shows that the expected cost of an optimal strategy for this tree is approximately the same as for the original tree. She also connects PAOTR to the theory of cographs, and explores ways to efficiently evaluate structures such as "and-or ladders", namely and-or trees such that each internal node is a parent of at most one internal node.

1.2.1 Deterministic And-Or Trees

Charikar et al. [CFG+02] proposed an alternative, deterministic way to evaluate the quality of a strategy for an and-or tree. Given a fixed boolean formula and a truth assignment for tests, a proof of the formula's value is a subset of the tests that is sufficient to establish the value of the formula. The cost of such a proof is the sum of costs of those tests. For a fixed assignment, the *performance ratio of a strategy* is the ratio of the cost of the strategy to the minimum cost over all proofs of the formula's value. The *competitive ratio* of a strategy is the maximum of the performance ratio over all assignments of tests. Here, a "cr-optimal" strategy is one that minimizes the competitive ratio.

Charikar et al. gave an efficient algorithm for finding a cr-optimal strategy for an and-or tree. Their algorithm relies on computed functions $f_0^T(c)$ and $f_1^T(c)$ that are lower bounds on the cost that any strategy for the and-or tree T, for any test assignment, has to pay in order to find a proof of cost c of the value `true` and `false` respectively. These functions are used by the algorithm to balance for each internal node the cost spent while performing the tests from each of the subtrees rooted at this node's children. The algorithm runs in time that is polynomial in the number of tree nodes and in the sum of all test costs.

In the *randomized model*, and-or trees are treated as fixed, non-stochastic structures but randomness is introduced into strategies. A *randomized strategy* is a strategy that can use coin flips to decide which test to perform. Formally, such a strategy is specified by a probabilistic distribution over a specified set of deterministic strategies. For a given assignment of tests, the cost of a randomized strategy is the expected cost of using the strategy under this assignment over all deterministic strategies. The *worst case cost of a randomized strategy* is the maximum cost of the strategy over all assignments of tests. A randomized strategy is optimal if it has the lowest worst case cost over all randomized strategies for a given and-or tree.

Saks and Wigderson [SW86] investigated *randomized depth-first strategies*, which differ from our deterministic depth-first strategy (§3) by selecting the next rooted subtree to evaluate at random, rather than in some fixed order. They prove that the randomized depth-first strategy is optimal for *uniform* and-or trees, namely those for which every test has the same cost and success probability, every root-to-leaf path has the same length, and all internal nodes have the same out-degree. Thus, while the worst case cost of any deterministic strategy for an and-or tree with n tests is n, for uniform and-or trees with n tests and with each internal node having exactly two children, this strategy has worst case cost $\Theta\left(n^{0.753\ldots}\right)$.

It has been conjectured that this is the largest gap between the worst case cost of a deterministic and a randomized strategy for a unit-cost and-or tree. Heiman and Wigderson [HW91] proved that the worst case cost of any randomized strategy for any unit-cost and-or tree with n tests is at least $n^{0.51}$.

The randomized strategies mentioned above are Las Vegas algorithms in that they are always correct. We can also consider Monte Carlo strategies, namely randomized strategies that may err with some probability, either on any input (two-sided error) or only on trees that evaluate to `true` (one-sided error). Using the best Monte Carlo strategy instead of a Las Vegas strategy does not increase the worst case cost. Santha [San95] proved that for any unit-cost Boolean expression a Las Vegas strategy can be transformed into a Monte Carlo strategy whose worst case cost is lower by a factor linear in the error probability, but that for unit-cost and-or trees Monte Carlo strategies cannot achieve any better improvement than this linear one.

2 Definitions

This paper focuses on read-once formulae. Each such formula corresponds to an *and-or tree* – namely, a rooted tree whose leaf nodes are each labeled with a probabilistic test (with a known positive cost[3] and success probability less than 1 and greater than 0; all tests of a tree are independent) and whose each internal node (namely non-leaf node) is labeled as either an or-node or an and-node.

A *test assignment* for an and-or tree with tests $\{X_1, X_2, \ldots, X_n\}$ is a vector $(V_1, V_2, \ldots, V_n)$, where for each i, V_i is the value — either `true` or `false` — of the test X_i. For a given test assignment, the *value of a leaf node* is the value of the associated test, while the *value of an or-node* (*value of an and-node*, respectively) is the value of the logical OR (AND, respectively) of its child nodes' values; the *value of a tree* is the value of its root node. With respect to a given assignment of tests, a node *resolves* its parent node if and only if this node value alone determines the value of the parent node. For example any node that has value `true` and is a child of an or-node resolves its parent.

A *rooted subtree* in an and-or tree is a subtree induced by a node and all its descendants.

For any variable X, "$+X$" refers to "$X = \mathtt{true}$" and "$-X$" refers to "$X = \mathtt{false}$" and so Pr($+X$) (Pr($-X$), respectively) refers to the probability that $X = \mathtt{true}$ ($X = \mathtt{false}$, respectively). For a test X, $c\,(X)$ denotes the cost of X.

A *strategy* for an and-or tree T is a decision tree for evaluating T — namely a tree whose internal nodes are labeled with probabilistic tests, whose arcs are labeled with the values of the parent's test, namely + or −, and whose leaf nodes are labeled either `true` or `false`, specifying the Boolean value of T, indicated by + and − respectively. For example, each of the strategies for T_1 in Figure 1 returns the Boolean value $\mathtt{L} \vee (\mathtt{J} \wedge (\mathtt{B} \vee \mathtt{U}))$ for any assignment to the variables. By convention, we will draw 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. Recall from our earlier discussion of ξ_{nl} that a strategy need not correspond to a linear sequence of tests.

With respect to a given and-or tree, for a test assignment γ we let $k(\xi, \gamma)$ be the cost of using strategy ξ to determine the value of the tree given this assignment. For example, for the preceding tree T_1 and for $\gamma = \{-\mathtt{L},\ +\mathtt{J},\ -\mathtt{B},\ +\mathtt{U}\}$, $k(\xi_{\langle LJUB \rangle}, \gamma) = c(\mathtt{L}) + c(\mathtt{J}) + c(\mathtt{U})$ (as we follow the path $\mathtt{L} - \mathtt{J} + \mathtt{U} + \boxed{+}$ of the strategy $\xi_{\langle LJUB \rangle}$) while $k(\xi_{nl}, \gamma) = c(\mathtt{U}) + c(\mathtt{J})$ (as we follow the path $\mathtt{U} + \mathtt{J} + \boxed{+}$ of the strategy ξ_{nl}).

[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 PAOTR whose tests all have strictly-positive costs.

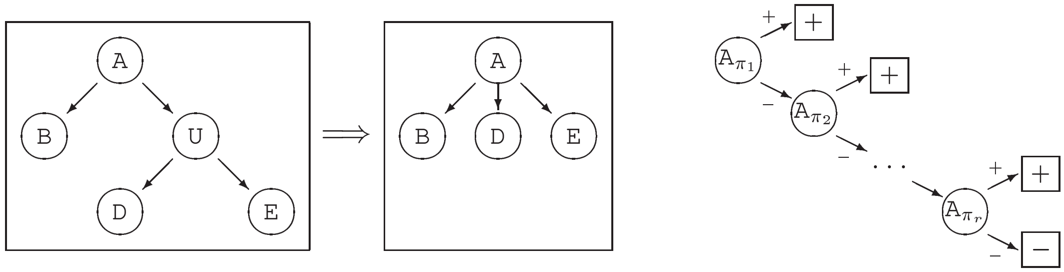

Figure 3: (a) Collapsing adjacent or-nodes into a single node. (b) A one-path strategy tree.

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

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

We call a strategy *nonredundant* if for every root-to-leaf path no test is performed more than once. Given the independence of the tests, there is a more efficient way to evaluate a nonredundant strategy than the algorithm implied by Equation 1. Extending the notation $C[\cdot]$ to apply to any strategy subtree, the expected cost of a leaf node is $C[\boxed{+}] = C[\boxed{-}] = 0$, and of a (sub)tree φ_χ rooted at a node χ labeled with a test x is

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

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

Definition 4 *A strategy ξ_T for an and-or tree T is* **optimal** *if and only if its expected cost is minimal, namely for any strategy ξ for T*

$$C[\xi_T] \quad \leq \quad C[\xi]\ . \qquad \square$$

Notice that for any and-or tree any optimal strategy is nonredundant because by removing a part of a redundant strategy we obtain a strategy with lower expected cost (assume that there is a root-to-leaf path of some strategy that contains two nodes v_1 and v_2 labeled by the same test; let v_1 be closer to the root of the strategy than v_2 and let A be the label (`true` or `false`) of the arc leaving v_1 on this path; then by removing the node v_2 together with the substrategy entered by the arc that leaves v_2 and is not labeled A we obtain a strategy that has lower expected cost). For this reason, in the rest of the paper we consider only nonredundant strategies.

The *depth* of a tree is the maximum number of internal nodes in any leaf-to-root path. Thus depth one and-or trees correspond to conjunctions or disjunctions while depth two and-or trees correspond to Boolean expressions in conjunctive normal form or disjunctive normal form.

We 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, since any and-or tree can be converted into an equivalent tree of this form by collapsing any or-node (and-node) child of an or-node (and-node) as shown in Figure 3(a). Similarly we obtain an equivalent tree by collapsing an internal node with only one child. For this reason we will assume that any internal node of and-or tree has out-degree at least two. Any strategy of the original tree is a strategy of the collapsed one, with identical expected cost.

2.1 Other Notation

In a rooted tree, a node with no child is a *leaf*, while a node with at least one child is *internal*. We also use the following less standard terms, each of which is defined with respect to a given rooted tree T:

Definition 5

- ***leaf-parent****: an internal node whose children include at least one leaf,*
- ***leaf-siblings*** *(aka* **siblings***): leaves with the same parent,*
- ***sibling class****: the set of all children of a leaf-parent that are leaves.* □

For the and-or tree T_1 in Figure 1, the leaves are `L`, `J`, `B` and `U`; the leaf-parents are `i#1`, `i#2` and `DiseaseX`; and the sibling classes are {`L`}, {`J`}, and { `B`, `U` }.

For any tree T, we let n refer to the number of nodes and $g(T)$ to the largest out-degree of any internal node. Notice that $g(T)$ bounds the number of siblings in any sibling class.

We will later define important notions like "R-ratio" (Definition 8), "contiguous" and "depth-first" (Definition 9), "linear strategies" (Definition 16), as well as some special types of and-or trees, such as "parameter-uniform" and "balanced" (Definition 14). The appendices also provide some additional notation, including the use of $\triangleleft$.

3 The depth-first algorithm DFA

To help define the depth-first algorithm DFA, we first consider depth one and-or trees. A rooted tree is *one-path* if every internal node has at most one internal child; a strategy is *one-path* if the associated strategy tree is one-path.

Observation 6 *[SK75] Let T_O be a depth one tree whose root is an or-node and 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 $A_{\pi_1}, \ldots, A_{\pi_r}$, shown in Figure 3(b) 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 $1 \leq j < r$.*

Proof: For a depth 1 or-rooted and-or tree, a successful test terminates a (nonredundant) strategy. Thus we may assume that every strategy has the form $A_{\rho(1)}, A_{\rho(2)}, \ldots, A_{\rho(r)}$, where $A_{\rho(j+1)}$ is the test performed if $A_{\rho(j)}$ fails.

Let ξ_{T_O} be an optimal strategy for T_O with tests relabeled so that $\xi_{T_O} = A_1, A_2, \ldots, A_r$. For each j, let $p_j = Pr(+A_j)$ and $c_j = c(A_j)$. Towards a contradiction, suppose that there exists $x < r$ such that $p_x/c_x < p_{x+1}/c_{x+1}$. Let ξ' be the strategy $A'_1, \ldots, A'_r$ obtained from ξ_{T_O} by interchanging the order of tests A_x and A_{x+1}, namely $A'_x = A_{x+1}$, $A'_{x+1} = A_x$, and $A'_j = A_j$ for $j \notin \{x, x+1\}$.

Let P_j be the probability that A_j is performed by ξ_{T_O}. Thus $P_1 = 1$, $P_t = \prod_{j=1}^{t-1}(1 - p_j)$, and the expected cost of ξ_{T_O} is $C[\xi_{T_O}] = \sum_{j=1}^{r} P_j c_j$. A straightforward computation shows that

$$C[\xi'] - C[\xi_{T_O}] = P_x\,(p_x c_{x+1} - p_{x+1} c_x) < 0\ ,$$

so ξ' has a lower expected cost than an optimal strategy, contradiction. □

An analogous proof shows ...

Observation 7 *Let T_A be a depth 1 tree whose root is an and-node, defined analogously to T_O in Observation 6. Then the optimal strategy for T_A is the one-path 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 $1 \leq j < r$.* □

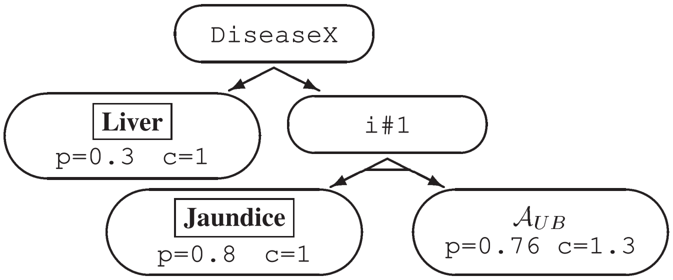

Figure 4: Intermediate results of DFA on T_1 (a) after 1 iteration (b) after 2 iterations.

As this relation holds in general, we will simplify our notation by defining ...

Definition 8 *For any test X of an and-or tree, define the **R-ratio** as:*

$$R(X) \;=\; \frac{p^r(X)}{c(X)}, \tag{3}$$

where $c(X)$ is the cost of X and $p^r(X)$ is the probability that X alone resolves its parent node, namely

$$p^r(X) = \begin{cases} Pr(+X) & \text{if the parent node of } X \text{ is or,} \\ Pr(-X) & \text{if the parent node of } X \text{ is and.} \end{cases}$$

□

Then for any depth-1 tree (either "and" or "or"), there is an optimal one-path strategy whose variables are in non-increasing R-ratio.

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 according to their R-ratios. Consider an or-node (the and-node case is analogous). For example, if dealing with Figure 1's T_1, DFA would compare $R(\mathtt{B}) = \Pr(+\mathtt{B})/c(\mathtt{B}) = 0.2/1$ with $R(\mathtt{U}) = \Pr(+\mathtt{U})/c(\mathtt{U}) = 0.7/1$, and order U 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 rooted 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 $\mathcal{A}_{UB}$-labeled node, with success probability $\Pr(+\mathcal{A}_{UB}) = 1 - (\Pr(-\mathtt{B}) \times \Pr(-\mathtt{U})) = 1 - 0.8 \times 0.3 = 0.76$, and cost $c(\mathcal{A}_{UB}) = c(\mathtt{U}) + \Pr(-\mathtt{U}) \times c(\mathtt{B}) = 1 + 0.3 \times 1 = 1.3$; see Figure 4(a).

Now recurse: 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 $R(\mathcal{A}) = \Pr(-\mathcal{A})/c(\mathcal{A})$ value, and so forth.

On T_1, DFA would then compare $R(\mathtt{J}) = \Pr(-\mathtt{J})/c(\mathtt{J}) = 0.2/1$ with $R(\mathcal{A}_{UB}) = \Pr(-\mathcal{A}_{UB})/c(\mathcal{A}_{UB}) = 0.24/1.3$ and so select the J-labeled node to go first. Hence, the substrategy associated with the i#1 rooted subtree will first perform J, and return $-$ if unsuccessful. Otherwise, it will then perform the $\mathcal{A}_{UB}$ mega-test: Here, it first performs U, and returns $+$ if U succeeds. Otherwise this substrategy will perform B, and return $+$ if it succeeds or $-$ otherwise.

DFA then creates a bigger mega-node, $\mathcal{A}_{JUB}$, with success probability $\Pr(+\mathcal{A}_{JUB}) = \Pr(+\mathtt{J}) \times \Pr(+\mathcal{A}_{UB}) = 0.8 \times 0.76 = 0.608$, and cost $c(\mathcal{A}_{JUB}) = c(\mathtt{J}) + \Pr(+\mathtt{J}) \times c(\mathcal{A}_{UB}) = 1 + 0.8 \times 1.3 = 2.04$; see Figure 4(b).

Finally, DFA compares L with $\mathcal{A}_{JUB}$, and selects L to go first as $R(\mathtt{L}) = \Pr(+\mathtt{L})/c(\mathtt{L}) = 0.3/1 > 0.608/2.04 = \Pr(+\mathcal{A}_{JUB})/c(\mathcal{A}_{JUB}) = R(\mathcal{A}_{JUB})$. This produces the $\xi_{\langle LJUB\rangle}$ strategy, shown in Figure 2. Figure 5 shows the DFA algorithm, in general.

```
DFA( and-or tree T ):  returns ⟨c_T, p_T, L_T⟩
        % c_T = C[ξ_T] is expected cost of executing the DFA-strategy for T, ξ_T
        % p_T = Pr(T) is probability that T evaluates to true
        % L_T is sequence of tests from T
        %    encoding the linear order of performing tests by ξ_T
    if T is single test x
        return ⟨c(x), Pr(+x), x⟩
    % Else
    For each immediate rooted subtree U_i of T,  i ≤ k
        ⟨c_i, p_i, L_i⟩ := DFA( U_i )
        p_i^r := { p_i        if root of T is "or"
                 { 1 - p_i    if root of T is "and"
    Set order π s.t.
        p^r_π(i) / c_π(i) ≥ p^r_π(i+1) / c_π(i+1)   for 1 ≤ i < k
    c_T := Σ_{i=1}^{k} c_π(i) Π_{j=1}^{i-1} (1 - p^r_π(j))
    p_T := { Π_{i=1}^{k} p_i             if root of T is "and"
           { 1 - Π_{i=1}^{k} (1 - p_i)   if root of T is "or"
    L_T := L_π(1) L_π(2) ... L_π(k)
    return ⟨c_T, p_T, L_T⟩
end DFA
```

Figure 5: DFA (Depth First Algorithm)

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 $O(\sum_v d^+(v) \ln d^+(v)) = O(n \ln g(T))$ 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 $g(T) < n$, 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 9

- *A strategy ξ_T is* ***contiguous*** *with respect to a set A of tests (of T) if and only if on any root-to-leaf path of ξ_T, whenever a test from A is performed, no test not in A will be performed until either the* *value of the least common predecessor of all tests in A has been determined or all tests in A have been performed.*

- *A strategy ξ_T is* **depth-first** *if and only if, for every rooted subtree f of T, ξ_T is contiguous with respect to the set of all tests from f.* □

The strategy $\xi_{\langle LJUB \rangle}$, shown in Figure 2(a), is depth-first as every time `U` appears it is next to its sibling `B` (so all of the children of `i#2` appear in a contiguous region); similarly, there is a contiguous region that contains all and only the tests under `i#1` — `J`, `B` and `U`. By contrast, the strategy $\xi_{\langle LUJB \rangle}$ is not depth-first, as there is a path where `U` is not next to its sibling `B`; 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 10 DFA *produces a strategy that has the lowest cost among all depth-first strategies.*

Proof: By induction on the depth of the tree. Observations 6 and 7 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 rooted subtree blocks associated with the depth-s and-or rooted subtrees; here we just re-use Observations 6 and 7 on the mega-blocks. □

Observations 6 and 7 show that DFA produces the best possible strategy, for the class of depth-1 trees. Moreover . . .

Theorem 11 DFA *produces the optimal strategies for depth-2 and-or trees.* □

Recall that depth-2 and-or trees are also known as read-once DNF or CNF formulae.

The proof (in Appendix B) shows that Theorem 11 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 12 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 $\xi_{\langle LJUB \rangle}$ strategy, whose expected cost (using Equation 2 with earlier results) is $C[\xi_{\langle LJUB \rangle}] = c(\mathtt{L}) + \Pr(-\mathtt{L}) \times c(\mathcal{A}_{JUB}) = 1 + 0.7 \times 2.04 = 2.428$. However, the ξ_{nl} strategy, which is *not* depth-first, has lower expected cost $C[\xi_{nl}] = 1 + 0.7[1 + 0.2 \times 1] + 0.3[1 + 0.7 \times (1 + 0.2 \times 1)] = 2.392$. 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 claim is far from being true.

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

Theorem 13 *There are unit-cost and-or trees with n nodes for which the best depth-first strategy costs $\Theta(n^{1-o(1)})$ times as much as the best strategy.* □

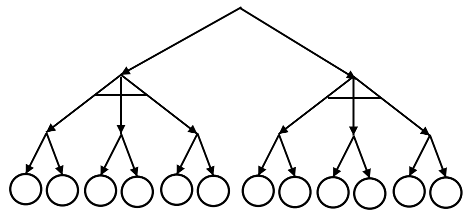

Figure 6: Balanced and-or tree, assuming each test has unit cost and success probability p.

There is one other interesting special case, dealing with arbitrary depth *balanced* trees. Here we need to define:

Definition 14 *A tree T is*

- **parameter-uniform***: if and only if every test has unit cost and same success probability*
- **balanced***: if and only if it is parameter-uniform and all nodes at each depth have same out-degree.* □

Figure 6 presents a balanced and-or tree.

Theorem 15 *[Tar83] For any balanced and-or tree, any depth-first strategy is optimal.* □

4 Linear Strategies

As noted above (Definition 9), we can write down each of these DFA-produced strategies in a linear fashion; for example, $\xi_{\langle LJUB\rangle}$ can be viewed as test `L`, then if necessary test `J`, then if necessary test `U` and if necessary test `B`. This motivates a large natural class of strategies: those that can be compactly written as a linear sequence of tests. Stated more precisely:

Definition 16 *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 resolve the tree, given what is already known.* □

Hence, $\xi_{\langle LJUB\rangle}$ will skip all of `J`, `U`, `B` if the `L` test succeeds; and it will skip the `U` and `B` tests if `J` fails, etc.

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. Moreover, the obvious algorithm can evaluate any such strategy on an instance in time linear in the number of tests. 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 strategies from this class. The answer here is "yes":

Observation 17 *The* DFA *algorithm produces a linear strategy.*

Proof: Argue by induction on the depth k. For $k = 1$, the result holds by Observations 6 and 7. For $k \geq 2$, use the inductive hypothesis to see that DFA will produce a linear ordering for each rooted subtree (as each rooted subtree is of depth $\leq k - 1$). DFA will then form a linear strategy by simply sequencing the linear strategies of the rooted subtrees. □

Using Theorem 11, this means the optimal strategy for depth 2 and-or trees is linear. Moreover, we can use Observation 10 to note there is always a linear strategy (perhaps that one produced by DFA) that is

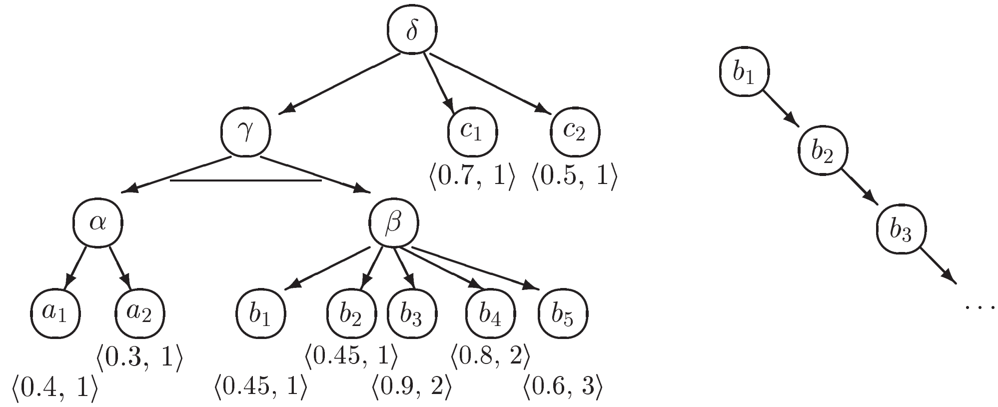

Figure 7: (a) A simple 3-level and-or tree, T_v. The $\langle p,\ c\rangle$ notation means the test's probability is p and cost c. We will later encode this tree as $(2, 5, 2)$. (b) Part of a strategy contiguous on $\{b_1,\ b_2,\ b_3\}$.

at least as good as any depth-first strategy. Unfortunately the converse is not true — the class of strategies considered in the proof of Theorem 13, showing the sub-optimality of depth-first strategies, 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 whether this class of linear strategies is effective in general. Is there always a linear strategy whose expected cost is near-optimal? Unfortunately ...

Theorem 18 *There are unit-cost and-or trees with n nodes for which the best linear strategy costs $\Theta(n^{1/3-o(1)})$ times as much as an optimal strategy.* □

5 The Dynamic Programming Algorithm DynProg

The most natural strategies to consider are depth-first strategies, but as shown above, they can be arbitrary bad for some and-or trees. This section presents an algorithm, DynProg, that is guaranteed to produce an optimal strategy for any and-or tree. DynProg resembles DFA in that it too builds strategies that respect an ordering on the leaf-sibling nodes — in fact, the same one, based on Observations 6 and 7. However, while DFA insisted that these siblings appear contiguously in the strategy (*i.e.*, the strategy always performs enough of these tests to resolve their common parent), DynProg allows these tests to be separated; moreover, DynProg only imposes this ordering of the *leaf nodes*, not on rooted subtrees higher in the tree.

Our DynProg computes an optimal strategy in time $O\left(d^2\,(r+1)^d\right)$ where r is the largest number of leaf-siblings (*i.e.*, tests under a common parent) and d is the number of leaf-parents. For trees with bounded number of internal nodes (which means d is bounded by a constant), it runs in time polynomial in r. It follows that, for example, if we are given a fixed structure of internal nodes (and- or or-nodes) than we can resolve quickly any and-or tree obtained by adding an arbitrary number of tests to this structure.

Subsection 5.1 presents the "Siblings and Twins Theorem" (Theorem 20) which leads to our main result: the DynProg algorithm, which appears in Section 5.2. Subsection 5.3 concludes this section by presenting other ramifications of the of the "Twins" part of Theorem 20, namely a way to simplify and-or trees in general and an algorithm for resolving *parameter-uniform* depth-3 and-or trees.

5.1 Siblings and Twins Theorems

Given any and-or tree, the DFA algorithm would begin by ordering the leaf-siblings under a common leaf-parent (Definition 5) — *e.g.*, given T_v (Figure 7(a)), it would order a_1 before a_2 and c_1 before c_2 (and not care about b_1 versus b_2). We know this ordering is optimal *in isolation* — *i.e.*, if the entire tree was just the leaf nodes under α (resp., δ). The fact that DFA is, in general, not optimal, may make us doubt whether these partial orderings are appropriate. The following theorem, however, partially refutes this concern, proving that there is an optimal strategy for T_v that respects this ordering, by always performing a_1 before a_2 and c_1 before c_2. We are not guaranteeing that they will appear contiguously; just that no part in this strategy will perform a_2 before a_1. Moreover, we see that identical tests — such as b_1 and b_2 — will always be performed *together* by an optimal strategy.

We now formalize these findings, using the notion of **R-ratio** defined in Definition 8.

Definition 19 *Tests x_1 and x_2 are **R-equivalent** if they are leaf-siblings and $R(\, x_1\,) = R(\, x_2\,)$.*
*An **R-class** is an equivalence class with respect to the relation of being R-equivalent.* □

In T_v (Figure 7(a)), the set $\{b_1,\, b_2,\, b_3\}$ forms an R-class. Figure 7(b) shows a portion of a possible strategy, that is contiguous with respect to this R-class (Definition 9).

If an optimal strategy ξ is contiguous with respect to some R-class W then the order of performing tests from W is arbitrary, in the sense that any strategy obtained from ξ by changing the order of performing the tests from W has the same expected cost (see Observation 26(ii) stated and proven in the Appendix B).

The following theorem specifies two conditions satisfied by an optimal strategy. The first one (1) deals with the best order of performing sibling tests; we will refer to it as to the Siblings Theorem. The second one (2), called the Twins Theorem, specifies the optimal way of performing sibling tests that are R-equivalent. The proof of the theorem, given in Appendix B, extends the approach taken in [Tar83].

Theorem 20 (The Siblings and Twins Theorem) *For any and-or tree T, there is an optimal strategy ξ_T that satisfies both of the following conditions:*

1. *for any sibling tests x and y such that $R(\, y\,) > R(\, x\,)$, x is not performed before y on any root-to-leaf path of ξ_T*
2. *for any R-class W, ξ_T is contiguous with respect to W.* □

5.2 Dynamic Programming Algorithm for PAOTR

The ordering of sibling-tests described by the Siblings Theorem (Theorem 20(1)) allows us to construct a dynamic programing algorithm for PAOTR that runs in time $O(d^2(r+1)^d)$, where r is the largest number of leaf-siblings and d is the number of leaf-parents in the input and-or tree; see Definition 5.

For an and-or tree T, let d be the number of leaf-parents in T and $\{L_1, L_2, \ldots, L_d\}$ be the sibling-classes of T. Assume that ξ_T is an optimal strategy for T that fulfills the conditions of Theorem 20. While evaluating T using ξ_T, we gradually reduce the tree (*viz.*, after performing any test, we obtain a new reduced and-or tree to evaluate) until we obtain the empty tree, at which point the evaluation of T is completed. Consider any reduced and-or tree I that we encounter while using ξ_T. Assume that I still contains m_i tests from the sibling-class L_i. If $m_i < |L_i|$, then we know the other $|L_i| - m_i$ tests from L_i have been already performed. Since we always query tests with higher R-ratio before sibling tests with lower R-ratio (and it does not matter in which order we query tests with the same R-ratios), the m_i tests still present in I must have the lowest R-ratios among all tests from L_i.

This means that, for any d-tuple $(m_1, m_2, \ldots, m_d)$, $0 \le m_i \le |L_i|$, there is only one (up to permutation of tests within one R-class; recall that the order of performing tests from one R-class is arbitrary) reduced tree that we may encounter that has exactly m_i tests from the set L_i, for any i: this tree contains the m_i tests with the lowest R-ratios among all tests from L_i. In this way we may identify a reduced and-or tree with such a d-tuple.

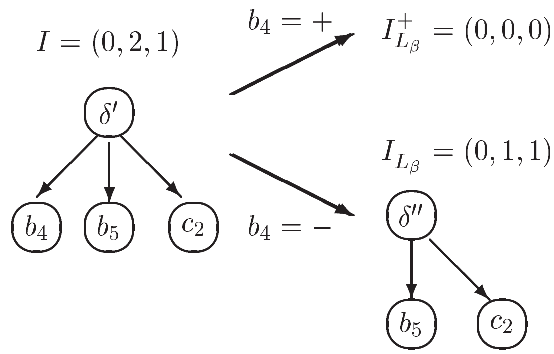

Figure 8: The reduced tree $I = (0, 2, 1)$ obtained from T_v, and the reduced trees obtained from I when b_4 succeeds and when b_4 fails.

For example, recall the T_v tree shown in Figure 7(a). We can represent this initial tree by the 3-tuple $(2, 5, 2)$, as there are 2 children of the leaf-parent α, 5 of leaf-parent β, and 2 of δ. (Note we are only considering the leaf-children of δ, and not its other branch, to γ.) Figure 8 shows the $(0, 2, 1)$ tree; here we know that the α tree has resolved successfully (*e.g.*, perhaps a_1 succeeded), and all three of b_1, b_2 and b_3 have been processed (and all failed), and that c_1 has been attempted and failed. (We know the α-rooted subtree was successful, as otherwise there is no reason to continue with its β sibling; moreover, all 3 of b_1, b_2 and b_3 must have failed for us to be considering b_4 and b_5.)

In general, there are $(|L_1| + 1) \times (|L_2| + 1) \times \ldots \times (|L_d| + 1)$ different reduced trees to consider, including the original tree. This number is at most $(r + 1)^d$ where $r = \max_{i=1}^{d} |L_i|$. Below we will identify each such tuple with the corresponding and-or tree.

Notice also that for any tree we need consider only d tests in order to find the first test to perform, namely a test with maximum R-ratio from each of the d sibling-classes.

We assume that the input tree T for the algorithm is strictly alternating, and that each internal node has out-degree at least two. Our algorithm needs a data structure that stores all internal nodes of the tree T, such that each internal node points to its parent and to all its internal node children. Additionally, for each sibling class (a set of test siblings) we need an array of the tests and their parameters; each sibling class has an associated index in d-tuples, and there is a link between a sibling class and its leaf parent.

Now assume that we are given a reduced tree I (obtained from T) encoded by a d-tuple. We now discuss how, for each sibling class L, we can calculate the d-tuples I^+_L (resp., I^-_L) corresponding to the reduced trees obtained from I when the test with maximum R-ratio from L in I succeeds (resp., fails).

For each sibling class L, let x_L be the test with maximum R-ratio from L in I.

If the sum of the numbers of tests in all sibling-classes of the d-tuple I is one, then x_L is the only test in the tree and both I^+_L and I^-_L will be the empty tree. So in this case we need time linear in d to find I^+_L or I^-_L.

Otherwise, we need to find the parent node of x_L in the collapsed I, using the structure of internal nodes of the original tree T. To do this, we first need to find the last internal node v on the path from the root of T to the parent of x_L, such that the sum of the number of tests in the sibling-classes inside the subtree rooted at v is greater than one.

The parent node of x_L is the last internal node y_L on the path from the root of T to v such that y_L has the same label ("or", "and") as v and y_L is the root of T, or y_L is a child node of the root of T, or the subtree rooted at the parent node of y_L contains at least one sibling-class with non-zero number of tests outside the subtree rooted at y_L.

Given the parent node y_L of x_L in I, we can easily modify the d-tuple I in order to obtain I^+_L or I^-_L. If x_L resolves its parent y_L, the required modification is setting to zero the number of tests for each sibling-class inside the subtree rooted at y_L. Otherwise, the modification is to decrement the number of tests associated with the sibling-class L by one.

Notice that the operation of finding the parent y_L as well as setting the tests' numbers of the corresponding sibling-classes to zero deal only with internal nodes and sibling-classes (not with particular tests) and can be performed in time linear in the number of internal nodes, and so also linear in d, since the number of all internal nodes is not greater than $2d$.

Thus the time required to find I_L^+ or I_L^- is in $O(d)$.

To illustrate this, consider again the $I = (0, 2, 1)$ reduced tree (Figure 8) obtained from the tree T_v from Figure 7(a). In I the test x_{L_β} (associated with the 2nd index, which are leaf children of β) with maximum R-ratio in L_β is b_4. We want to find $I_{L_\beta}^+$ and $I_{L_\beta}^-$. Using the algorithm described above, we first find the node v, which is β. Then we find the parent node y_L, which is δ, an or-node. If x_{L_β} succeeds, it resolves its parent node, thus we need to set to 0 the number of tests in all sibling-classes inside the subtree rooted at δ; this means we set $I_{L_\beta}^+ = (0, 0, 0)$. If x_{L_β} fails, we just need to decrement the number of tests in L_β by one: $I_{L_\beta}^- = (0, 1, 1)$. See Figure 8(b).

We now describe Dynamic Programing Algorithm (DynProg) for PAOTR. This algorithm enumerates all possible $(|L_1| + 1) \times (|L_2| + 1) \times \ldots \times (|L_d| + 1)$ reduced trees, identifying each with an associated d-tuple. That is, we identify each reduced tree with one entry in a d-dimensional matrix of size $(|L_1| + 1) \times (|L_2| + 1) \times \ldots \times (|L_d| + 1)$. The tree $(|L_1|, |L_2|, \ldots, |L_d|)$ is the *input tree*, containing all the tests, the tree $(0, 0, \ldots, 0)$ is the *empty tree* indicating that nothing remains to evaluate.

For each reduced tree I, we compute and store the following attributes:

`Cost`$[I]$: the expected cost of the optimal strategy for I,

`FirstTest`$[I]$: a first test performed by the optimal strategy for I,

`TrueArc` $[I]$: the pointer to the reduced tree obtained if the first test succeeds,

`FalseArc`$[I]$: the pointer to the reduced tree obtained if the first test fails.

These attributes, over the set of all reduced trees, encode an optimal strategy for the input tree T. The strategy starts with performing the test `FirstTest`$[T]$ and then depending on the value of this test, follows either `TrueArc`$[T]$ or `FalseArc`$[T]$; each points to a reduced tree, which is then evaluated. We follow the procedure until reaching the empty tree: if it is reached by a `TrueArc`, the value of the tree is `true`, otherwise its value is `false`.

Figure 9 presents the Dynamic Programing Algorithm for PAOTR. As shown, it incrementally deals with the set of reduced trees, in the order of the number of tests, starting with the empty tree.

Theorem 21 DynProg *produces an optimal strategy for and-or trees. The time complexity of the algorithm is in* $O\left(d^2(r+1)^d\right)$ *and the space complexity is in* $O\left((r+1)^d\right)$, *where* r *is the largest number of leaf-siblings of a tree and* d *is the number of leaf-parents in a tree.*

For the special case when d *is fixed, the time complexity is in* $O(r \ln r)$ *if* $d = 1$ *and in* $O\left(r^d\right)$ *for any fixed* $d \geq 2$, *while the space complexity is in* $O\left(r^d\right)$ *for any fixed* $d \geq 1$. □

The corollary below follows immediately from the previous theorem.

Corollary 22 *Probabilistic and-or tree resolution for and-or trees with a bounded number of internal nodes is in* P. □

```
DynProg( and-or tree T ):  returns optimal strategy for T
    % optimal strategy for T is encoded by the parameters
    %     ⟨FirstTest, TrueArc, FalseArc⟩ for all reduced trees
    % parameter Cost for all reduced trees gives expected costs
    %     of optimal strategies for all reduced trees, including T
    For each sibling-class L of T
       order tests of L by R ratio
    For each reduced tree d-tuple I
       Cost[I]:=∞
    Cost[empty tree]:=0
    FirstTest[empty tree]:=NIL
    For M = 1 to # of tests in T
       For each reduced tree d-tuple I with M tests
           For each sibling-class L of T that is not empty in I
               x_L:=test from L in I with maximum R
               I_L^+:=d-tuple of tree obtained from I if x_L succeeds
               I_L^-:=d-tuple of tree obtained from I if x_L fails
               C := c(x_L) + Pr(+x_L) ×Cost[I_L^+] + Pr(−x_L) ×Cost[I_L^-]
               If C <Cost[I]
                   Cost[I] := C
                   FirstTest[I] := x_L
                   TrueArc[I] is pointer to I_L^+
                   FalseArc[I] is pointer to I_L^-
    return ⟨Cost, FirstTest, TrueArc, FalseArc⟩ for all reduced trees
end DynProg
```

Figure 9: Dynamic Programming Algorithm (DynProg) for PAOTR

5.2.1 Example

As an example consider again the and-or tree T_v shown in Figure 7(a). Assume that we already processed all reduced trees with less than three tests. The calculated parameters for each of these trees are given in Table 2.

We now want to calculate the optimal strategy for the reduced tree $I = (0, 2, 1)$ with three tests; see Figure 8. The sibling-class L_α is empty in I. Now consider the sibling-class L_β. The test x_{L_β} with maximum R ratio from L_β in I is the test b_4 and $I^+_{L_\beta} = (0, 0, 0)$, $I^-_{L_\beta} = (0, 1, 1)$. Thus we now have

$$\begin{array}{rcccccccl} C & = & c\,(b_4) & + & \Pr(+b_4)\cdot\texttt{Cost}\left[I^+_{L_\beta}\right] & + & \Pr(-b_4)\cdot\texttt{Cost}\left[I^-_{L_\beta}\right] & = & \\ & = & 2 & + & 0.8\cdot 0 & + & 0.2\cdot 2.5 & = & 2.5\,. \end{array}$$

Thus we set `Cost`$[I]$ to 2.5 and `FirstTest`$[I]$ to b_4, we point `TrueArc` $[I]$ to $(0, 0, 0)$ and `FalseArc`$[I]$ to $(0, 1, 1)$. Now we proceed to the sibling-class L_γ. We have $x_{L_\gamma} = c_2$ and $I^+_{L_\gamma} = (0, 0, 0)$, $I^-_{L_\gamma} = (0, 2, 0)$. Thus

$$\begin{array}{rcccccccl} C & = & c\,(c_2) & + & \Pr(+c_2)\cdot\texttt{Cost}\left[I^+_{L_\gamma}\right] & + & \Pr(-c_2)\cdot\texttt{Cost}\left[I^-_{L_\gamma}\right] & = & \\ & = & 1 & + & 0.5\cdot 0 & + & 0.5\cdot 2.6 & = & 2.3\,. \end{array}$$

Since this cost is lower than current `Cost`$[I]$, we set `Cost`$[I]$ to 2.3 and `FirstTest`$[I]$ to c_2, we point `TrueArc` $[I]$ to $(0, 0, 0)$ and `FalseArc`$[I]$ to $(0, 2, 0)$. These parameters, together with the parameters

reduced tree I	`Cost`$[I]$	`FirstTest`$[I]$	`TrueArc`$[I]$ points to	`FalseArc`$[I]$ points to
$(0,0,0)$	0	NIL	NIL	NIL
$(0,0,1)$	1	c_2	$(0,0,0)$	$(0,0,0)$
$(0,1,0)$	3	b_5	$(0,0,0)$	$(0,0,0)$
$(1,0,0)$	1	a_2	$(0,0,0)$	$(0,0,0)$
$(0,0,2)$	1.3	c_1	$(0,0,0)$	$(0,0,1)$
$(0,1,1)$	2.5	c_2	$(0,0,0)$	$(0,1,0)$
$(0,2,0)$	2.6	b_4	$(0,0,0)$	$(0,1,0)$
$(1,0,1)$	1.5	c_2	$(0,0,0)$	$(1,0,0)$
$(1,1,0)$	1.9	a_2	$(0,1,0)$	$(0,0,0)$
$(2,0,0)$	1.6	a_1	$(0,0,0)$	$(1,0,0)$

Table 2: Parameters of reduced trees obtained from the and-or tree T_v from Figure 7(a) with less than three tests — see Figure 8.

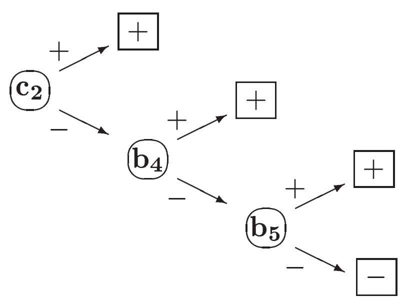

Figure 10: The optimal strategy for the tree I shown in Figure 8.

from Table 2, encode the optimal strategy for the reduced tree I. This strategy is presented as a binary tree in Figure 10.

5.3 Simplifying And-Or Trees Using the Twins Lemma

The Twins Theorem (Theorem 20(2)) provides a way of simplifying an and-or tree. Since all tests from an R-class are performed together by an optimal strategy, it only matters whether any of them resolves their common parent node. Thus we may replace each R-class containing more than one test, by a single meta-test with the effective cost and probability corresponding to performing all tests from the R-class.

By simple calculations we obtain the parameters of such a meta-test:

Observation 23 *Let W be an R-class and let R be the value of the R-ratio of the tests from W. In the search for an optimal strategy, we can replace W by a single meta-test w with the following parameters:*

$$Pr(+w) = \begin{cases} 1 - \prod_{x \in W} Pr(-x) & \textit{if the parent of } W \textit{ is or,} \\ \prod_{x \in W} Pr(+x) & \textit{if the parent of } W \textit{ is and,} \end{cases} \tag{4}$$

$$c(w) = \begin{cases} \frac{Pr(+w)}{R} & \textit{if the parent of } W \textit{ is or,} \\ \frac{1 - Pr(+w)}{R} & \textit{if the parent of } W \textit{ is and.} \end{cases} \tag{5}$$

□

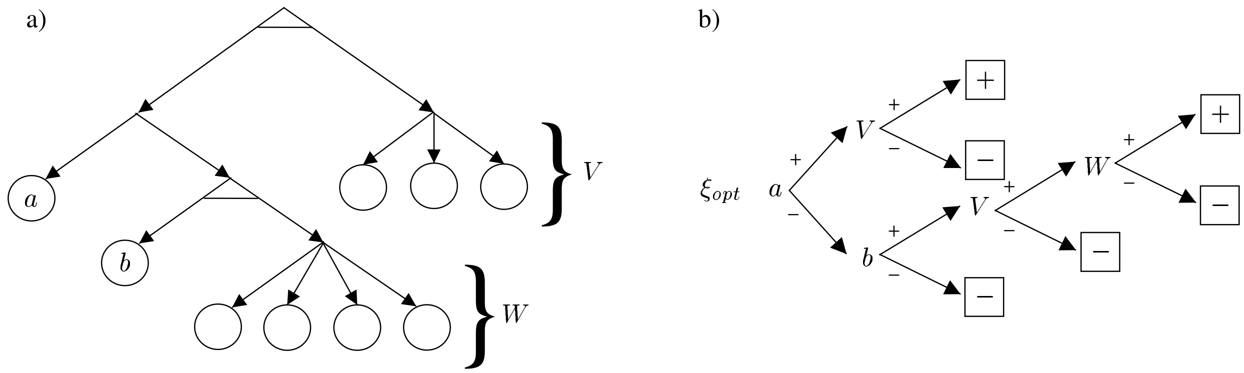

Figure 11: (a) A parameter-uniform and-or tree T_u; each test has unit cost and success probability 0.2; W and V denote entire depth 1 rooted subtrees. (b) The unique optimal strategy ξ_{opt} for the and-or tree T_u (nodes labeled by W and V denote evaluation of the corresponding rooted subtrees).

Observe that the R-ratio of a meta-test is the same as the R-ratio of each test from the class.

The simplification described above allows us to prove that the efficient DFA produces an optimal strategy for depth 3 parameter-uniform and-or trees. (Recall from Definition 14 that an and-or tree is parameter-uniform if and only if all tests have unit cost and the same success probability.)

Observation 24 DFA *produces an optimal strategy for depth three parameter-uniform and-or trees.*

Proof: Let T be a depth three parameter-uniform and-or tree. By Observation 10 DFA produces a strategy with minimum expected cost among all depth-first strategies. Thus it suffices to show that some optimal strategy for T is depth-first.

Let T' be the simplified tree obtained from T by replacing each R-class by a single meta-test. Observe that in T' each internal node at depth two has only one child: a single meta-test. Thus T' collapses to depth two. By Theorem 11 for any depth two and-or tree there is an optimal depth-first strategy. If we evaluate entire replaced subtrees in place of meta-tests, the strategy is depth-first for T and by the Twins Theorem (Theorem 20(2)) it is optimal for T. $\square$

Unfortunately, this property does not hold for deeper parameter-uniform and-or trees; there are depth-4 parameter-uniform and-or trees for which the best depth-first strategy is not optimal: The strategy ξ_{opt} in Figure 11(b) is the unique optimal strategy for the tree T_u shown in Figure 11(a), but ξ_{opt} is not depth-first.

6 Examples and Counterexamples

This section illustrates some of the subtleties of this PAOTR task, by presenting a number of seemingly plausible conjectures each followed by a counterexample. ([Jan03] presents several other plausible hypotheses and counterexamples.)

6.1 Best Test of a Rooted Subtree

The DFA algorithm uses only completely local properties to determine each part of the strategy; in particular, the substrategy ξ_f for the rooted subtree f is based only on f. While DFA does not produce the optimal strategy, one might ask whether this basic idea — that local properties are sufficient — applies. In

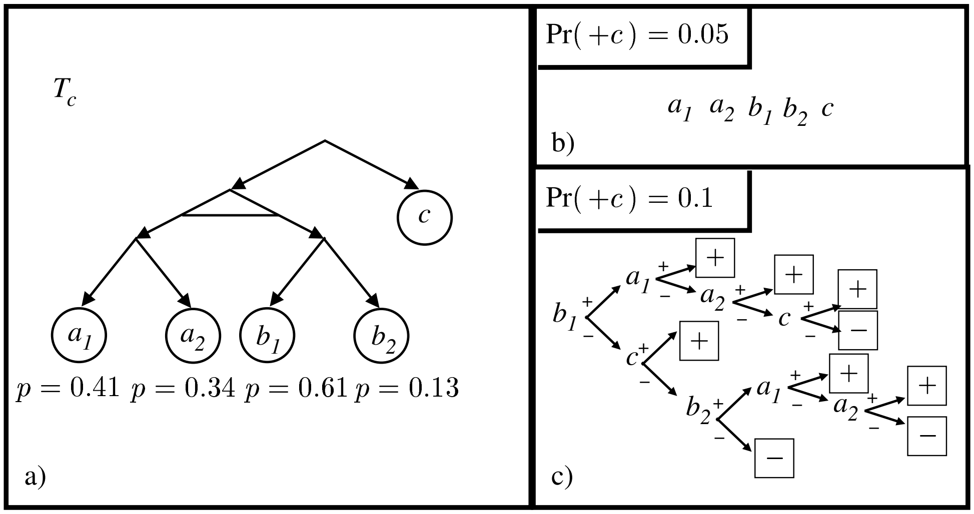

Figure 12: (a) An and-or tree T_c with all costs unit; p denotes success probability of a test. (b) The unique optimal strategy for T_c if Pr($+c$) $= 0.05$, encoded by the fixed order of tests, starting with a_1. (c) The unique optimal strategy for T_c if Pr($+c$) $= 0.1$, starting with b_1.

particular, is there a generalization of the Siblings Theorem (Theorem 20(1)) that allows the identification, for each rooted subtree f, of which of its tests should be performed first, *based only on properties of this subtree*?

Unfortunately, this is not possible in general. Consider the and-or tree T_c shown in Figure 12(a). Tests a_1, a_2, b_1 and b_2 are grand-children of the same and-node, but the relative order in which these tests are queried by an optimal strategy, varies with the success probability of a test *outside of that rooted subtree* c: If Pr($+c$) $= 0.05$, the unique optimal strategy starts by testing a_1 and then follows the linear strategy shown in Figure 12(b). However if Pr($+c$) $= 0.1$, the unique optimal strategy is the strategy shown in Figure 12(c), which starts by testing b_1; notice that this strategy is not linear.

Hence, given a rooted subtree f of T, the "first" test of the f-tests to perform in ξ_f depends on information that is *not* in f.

6.2 Resolving Rooted Subtrees

A depth-first strategy, which is optimal for depth-2 and-or trees, does not leave a given rooted subtree until determining its value. We saw that this approach is not necessary optimal for deeper trees; does some weaker property hold for optimal strategies?

After a test from an and-or tree is performed, let "the highest resolved node" in the tree be the root of the maximal rooted subtree whose value has been determined. Consider for example the x_1 test in the T_c tree of Figure 13(a).[4] If this test evaluates to `true`, then we have resolved the subtree rooted in γ; if it is `false`, then we have only resolved the singleton rooted subtree x_1.

Is it the case that, after performing a test x, an optimal strategy would at least in one case (when x is `true`, or if x is `false`) perform after x a test from the subtree rooted at the parent of the highest resolved node? If true, this would mean either

- if x_1 is `true`, perform some relevant test under β — *viz.*, x_0 — or
- if x_1 is `false`, perform some relevant test under γ — *viz.*, x_2,

[4] We gratefully acknowledge Jon Derryberry for contributing this counterexample.

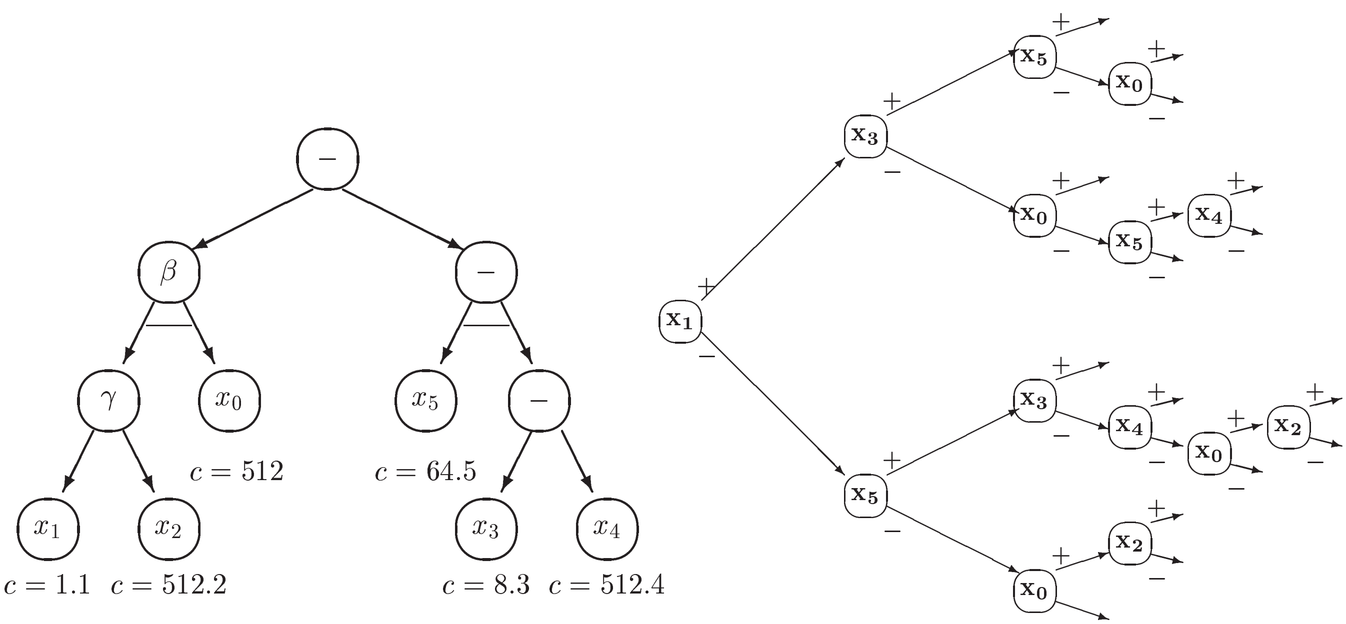

Figure 13: (a) and-or tree T_c; here each probability is $p = 0.5$. (b) Optimal strategy ξ_{T_c}.

or perhaps both. (Note that any depth-first-strategy necessarily does both.)

The answer is no. The only optimal strategy ξ_{T_c} for T_c appears in Figure 13(b); notice that the `true`-link does *not* point to x_0, and the `false`-link does *not* point to x_2.

6.3 Prime Implicants and Implicates

Recall that one can view an and-or tree as a boolean expression. In general, a set W of tests is a *prime implicant* of an and-or tree if W is a minimal set of tests with the property, that if all tests from W are `true`, the entire tree evaluates to `true`. ("Minimal" means that no proper subset has this property.) A *prime implicate* of an and-or tree is a minimal set of tests, such that if all tests from the set are `false`, the entire tree evaluates to `false`. A tree evaluates to `true` (resp., `false`) if and only if there is at least one prime implicant (resp., prime implicate) whose tests are all `true` (resp., `false`). (Proof: Assume that the value of T is `true`, but each prime implicant of T contains at least one `false` test. Note the union of all `true` tests in these implicants is sufficient to show the tree is `true`, which means it must include a prime implicant for T; contradiction.) That is, an and-or tree can only evaluate to `true` (for any strategy) after all tests of some prime implicant have been performed and succeeded. Moreover, the intersection of any prime implicant and any prime implicate is non-empty; in fact, by induction on depth of a tree, the intersection contains exactly one test.

The *purely-*`true` *path* (*purely-*`false` *path*, respectively) of a strategy is the root-to-leaf path of the strategy that contains only `true` (only `false`, respectively) arcs. Obviously the leaf node of the purely-`true` path is labeled `true`, the leaf of the purely-`false` path if labeled `false`.

Is it the case that, for any and-or tree, there is an optimal strategy such that either all tests performed on the purely-`true` path of the strategy are from exactly one prime implicant, or all tests performed on the purely-`false` path of the strategy are from exactly one prime implicate (or both)? If so, than either after the first test performed by an optimal strategy succeeds, the strategy performs tests from this prime implicant as long as they are `true`, or after the first test fails, the strategy performs tests from this prime implicate as long as they are `false`.

Again, the same T_c example (Figure 13) used above, also shows that this is not the case. The purely-`true` path $\{x_1, x_3, x_5\}$ is not a prime implicant (as $\{x_3, x_5\}$ is), and the purely-`false` path $\{x_1, x_5, x_0\}$ is not a prime implicate (as $\{x_0, x_5\}$ is).

7 Conclusions

Future Work: There are a few obvious extensions to our work. 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 $n^{1/3-o(1)}$ inferior to the optimal strategy, at least we know this strategy can be expressed succinctly, which is not true in general.

A second question 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 transform 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.

If there is such an efficient algorithm for arbitrary and-or trees — that is, 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 13) show we should not consider the DFA algorithm, there may be other algorithms, perhaps for special cases — for example, Greiner and Orponen [GO91] provide an efficient algorithm that produces strategies within $O(\ln n)$ of optimal, for a specified subclass of a related problem.

Contributions: This paper addresses the challenge of computing the optimal strategy for and-or trees, focussing on algorithms that exploit "local properties". Depth-first strategies are an extreme case of this, as they are formed by considering only each rooted subtree, in isolation. The obvious algorithm here, DFA, first finds the best substrategy for each penultimate rooted subtree then regards the resulting strategy as a "mega-node", and recurses up the tree. After confirming that DFA produces 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 possible strategies. We next consider the obvious class of "linear strategies" — strategies that can be described as a linear sequence of tests — and show that even the best such strategy can be also be considerably worse than the best possible strategy.

The DFA algorithm worked by providing an ordering for the leaf tests, then "gluing" them together in this order — in that it produced a strategy in which these nodes appeared contiguously, until resolving their common parent. We next investigated a weaker version of this constraint, which imposed the same ordering for the leaf-children of a common parent, but did not insist that they appear contiguously in the final strategy. Our main theorem here (Theorem 20) proves that an optimal strategy will honour this ordering; here each "x before y" ordering of leaf-siblings means only that y will never be tested before x in an optimal strategy; it does not mean that y will *immediately* follow x. We also determined the special cases when these sibling tests should be performed together. These findings led to the design of the **Dynamic Programming Algorithm**, DYNPROG, which is guaranteed to find an optimal strategy for and-or trees, and our proof that this algorithm runs in time $O(d^2(r+1)^d)$, where d is the number of internal nodes that are leaf-parents and r is the largest number of tests under a common parent. For and-or trees with a bounded number of internal nodes, this time is clearly polynomial in the tree's size.

We also used this theorem to show that the simple DFA algorithm produces an optimal strategy for depth-3 and-or trees whose tests are all identical (have the same cost and probability of success). We show that this claim does not hold for depth-4 and-or trees.

Acknowledgments

All authors gratefully acknowledge support from various NSERC grants; in addition, RG also acknowledges support from the Alberta Ingenuity Centre for Machine Learning, RH also acknowledges a University of Alberta Research Excellence Award; and MM also acknowledges a Sloan Research Fellowship. We gratefully acknowledge Jon Derryberry for contributing the example presented in Figure 13. We also gratefully acknowledge receiving helpful comments from many colleagues including Adnan Darwiche, Rob Holte, and Omid Madani, as well as the anonymous referees.

References

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

[CFG+02] Moses Charikar, Ronald Fagin, Venkatesan Guruswami, Jon Kleinberg, Prabhakar Raghavan, and Amit Sahai. Query strategies for priced information. *J. Computer and System Sciences*, 64:785–819, 2002.

[Dre02] Stuart Dreyfus. Richard Bellman on the birth of dynamic programming. *Operations Research*, 50:48–51, 2002.

[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.

[GHM02] R. Greiner, R. Hayward, and M. Malloy. Optimal depth-first strategies for And-Or trees. In *Proceedings of AAAI-02*, Edmonton, August 2002.

[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. http://www.cs.ualberta.ca/∼greiner/PAPERS/pao_kr91.ps.

[Gre91] Russell Greiner. Finding the optimal derivation strategy in a redundant knowledge base. *Artificial Intelligence*, 50(1):95–116, 1991. http://www.cs.ualberta.ca/∼greiner/PAPERS/redund.ps.

[HW91] Rafi Heiman and Avi Wigderson. Randomized vs. deterministic decision tree complexity for read-once boolean function. In *Proceedings of 6th IEEE Structure in Complexity Theory*, pages 172–179, 1991.

[Jan03] Magdalena Jankowska. Probabilistic and-or tree resolution. Master's thesis, Dept of Computing Science, University of Alberta, 2003.

[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.

[Nil80] Nils J. Nilsson. *Principles of Artifical Intelligence*. Tioga Press, Palo Alto, 1980.

[Pea84] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[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.

[San95] Miklos Santha. On the Monte Carlo boolean decision tree complexity of read-once formulae. *Random Structures and Algorithms*, 6:75–88, 1995.

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

[Sha86] Ross D. Shachter. Evaluating influence diagrams. *Operations Research*, 34:871–882, 1986.

[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.

[SW86] Michael Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *Proceedings of 27th IEEE FOCS*, pages 29–38, 1986.

[Tar83] Michael Tarsi. Optimal search on some game trees. *J. of ACM*, 30:389–396, 1983.

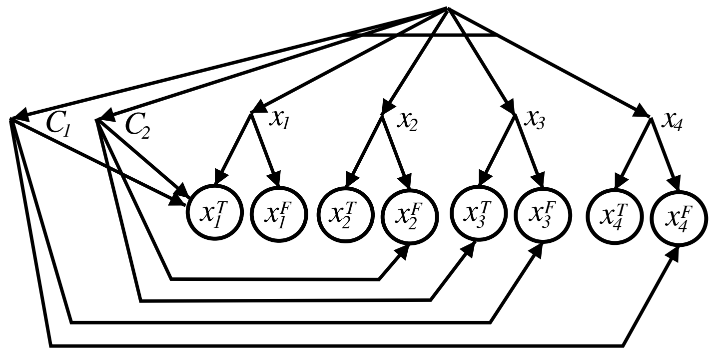

Figure 14: Illustration for the proof of Theorem 25. This dag corresponds to the formula $P = (x_1$ or $\neg x_3$ or $\neg x_4)$ and $(x_1$ or $\neg x_2$ or $x_3)$.

A Theorem 25

In our proofs we will need the notion of a probability of a strategy path. Let P be a path of strategy-tree nodes $v_0, v_1, \ldots, v_k$, with $k \geq 0$, of a nonredundant strategy. For any $i < k$ let x_{v_i} be the test that labels v_i. We define the probability $p\,(P)$ of the path P as the product of the probabilities of the corresponding values of the tests performed on P

$$p\,(P) = \begin{cases} 1 & \text{if } k = 0, \\ \prod_{i=0}^{k-1} p_{v_i} & \text{if } k \geq 1, \end{cases}$$

where

$$p_{v_i} = \begin{cases} \Pr(\,+x_{v_i}\,) & \text{if the arc } (v_i, v_{i+1}) \text{ is labeled } \texttt{true}, \\ \Pr(\,-x_{v_i}\,) & \text{if the arc } (v_i, v_{i+1}) \text{ is labeled } \texttt{false}. \end{cases}$$

Theorem 25 *It is NP-hard to find the optimal strategy for and-or dags, even if all test have unit costs.*

Our proof follows by introducing stochasticity into the construction presented in [Sah74] for a different and-or structure problem.

Proof of Theorem 25: Consider the 3-SAT problem:

> Given a boolean formula P that is the conjunction of m clauses, each of which is the disjunction of exactly 3 distinct literals (*i.e.*, variables or their negations), is P satisfiable?

which is known to be NP-complete [GJ79]. We will show that 3-SAT can be polynomially reduced to our task of computing the optimal strategy for and-or dags.

For a given instance of 3-SAT let $C_1, C_2, \ldots, C_m$ be the clauses in the formula P and let $x_1, x_2, \ldots, x_n$ be all variables from the formula P. Now construct the and-or dag D in the following way: The root of D is an and-node. It has $m+n$ child or-nodes: the nodes $C_1, C_2, ..., C_m$ correspond to the clauses of the formula P, the nodes $x_1, x_2, \ldots, x_n$ correspond to the variables from the formula P. Each or-node x_i has exactly two distinct child nodes: the tests x_i^T and x_i^F, corresponding to the respective values `true` and `false` of the variable x_i. Each test has cost 1 and the success probability $q = \left(1 - \frac{1}{2n}\right)^{\frac{1}{2n+1}}$. These are all of the nodes of D. Each or-node C_j has exactly 3 child nodes: if the clause C_j contains the literal x_i, the test x_i^T is a child of the node, if the clause C_j contains the literal $\neg x_i$, the test x_i^F is a child of the node. Figure 14 presents an example of such a construction.

We can construct such an and-or dag in polynomial time. Now we will show that P is satisfiable if and only if there is a strategy for D with expected cost at most $n+\frac{1}{2}$.

For any strategy the single root-to-leaf path of the strategy that includes only `true` arcs will be called the *purely-*`true`*-path*. Notice that the purely-`true`-path of any strategy for D has to include at least n internal nodes, because for each i we have to perform at least one of $\left\{x_i^T, x_i^F\right\}$ to conclude that the value of D is `true`.

Now observe

	P is satisfiable.
$\iff$	There is a truth assignment σ for P such that, for any clause C_j, there is at least one literal that σ assigns `true`.
$\iff$	There is a set W of tests from D, $\lvert W\rvert = n$, such that for any i, exactly one of x_i^T and x_i^F belongs to W and any node C_j has at least one child in W.
$\iff$	There is a strategy for D whose purely-`true`-path contains exactly n internal nodes.

To complete the proof, we need only show…

> A strategy ξ for D has the expected cost at most $n+\frac{1}{2}$ if and only if the purely-`true`-path of ξ contains exactly n internal nodes.

Proof: Let Q be the purely-`true`-path of a strategy ξ and let k be the number of the internal nodes of Q, $k \geq n$. The sum of the costs of all tests labeling nodes of Q is k and the probability of Q is q^k. Notice that for any other root-to-leaf path of ξ the sum of the costs of all tests labeling nodes of the path is at most $2n$ and at least 1.

Assume that $k = n$. Then we obtain the following upper bound on the expected cost of ξ:

$$\begin{aligned} C\left[\xi\right] \;\leq\; q^n n + (1-q^n)\,2n \;=\; n\,(2-q^n) \;=\; \\ = n\left[2-\left(1-\frac{1}{2n}\right)^{\frac{n}{2n+1}}\right] \;<\; n\left[2-\left(1-\frac{1}{2n}\right)\right] \;=\; n+\frac{1}{2}\,. \end{aligned}$$

Now assume that $k > n$. In this case we have the following lower bound on the expected cost of ξ:

$$\begin{aligned} C\left[\xi\right] \;\geq\; q^k k + \left(1-q^k\right)1 \;=\; q^k\,(k-1)+1 \;\geq\; q^k n + 1 \;=\; \\ = n\left(1-\frac{1}{2n}\right)^{\frac{k}{2n+1}} + 1 \;>\; n\left(1-\frac{1}{2n}\right)+1 \;=\; n+\frac{1}{2}. \end{aligned}$$

□

B Proofs

Theorem 11 DFA *produces the optimal strategies for depth-*2 *and-or trees.*

Proof of Theorem 11:

It is sufficient to prove the theorem for DNF formulae, say of the form

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

since the proof is virtually identical for CNF formulae. We let A^i refer to the term $A_1^i \wedge \cdots \wedge A_{n_i}^i$. For any term A^i the probability that the term evaluates to `true` is given by the formula $\Pr(\,+A^i\,) = \prod_{k=1}^{n_i} \Pr(\,+A_k^i\,)$.

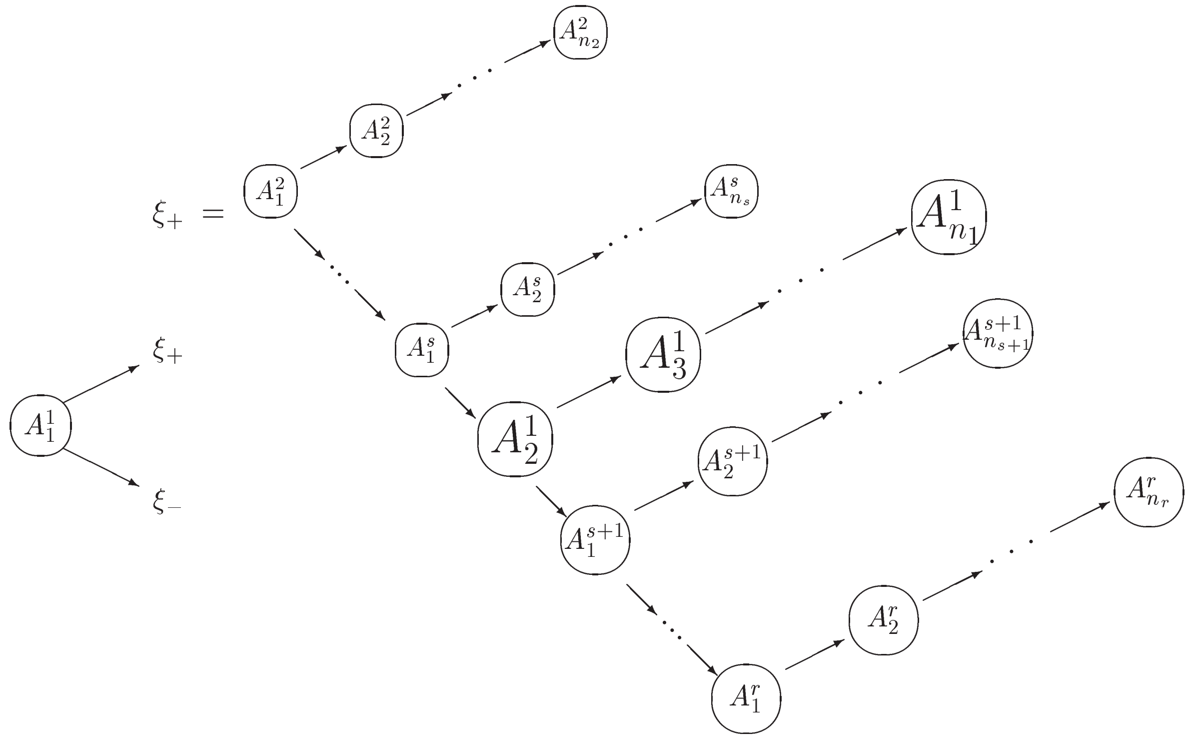

Figure 15: Form for of an optimal strategy ξ_ϕ for φ.

Let ξ_ϕ be an optimal strategy. By Observation 10, it suffices to show that ξ_ϕ is depth-first. Argue by induction on the number of variable occurrences $v = \sum_i n_i$, which is equal to the number of variables. In the base case $v = 1$ and the only possible strategy is depth-first, so the theorem holds. Suppose then that the theorem holds for $v = \ell \geq 1$ and consider a formula with $v = \ell + 1$. By relabelling variables if necessary we may assume that the first test in the strategy is A_1^1 as indicated on the left side of Figure 15.

Let ξ_- be the substrategy of ξ_ϕ that occurs when A_1^1 fails; since ξ_ϕ is optimal for the subexpression $\varphi - A^1$ of φ obtained by excluding the term A^1, and $\varphi - A^1$ has only $\ell + 1 - n_1 \leq \ell$ variables, ξ_- is depth-first by the inductive hypothesis.

Now consider what happens if A_1^1 succeeds. If $n_1 = 1$ then $A^1 \equiv A_1^1$ and ξ_ϕ simply returns success and again the overall strategy is clearly depth-first. Suppose then that $n_1 \geq 2$. Then the $A_1^1 = +$ branch leads to a substrategy, say ξ_+. Since the associated subexpression has ℓ variables, we may assume, by inductive hypothesis, that ξ_+ is depth-first; in particular, it will be of the form shown on the right side of Figure 15, namely first dealing with the $s-1$ terms $A^2, \ldots, A^s$, then dealing with $\tilde{A}^1 = A_2^1 \wedge \ldots \wedge A_{n1}^1$, the diminished version of A^1 that omits A_1^1, and then dealing with the remaining $r - s$ terms $A^{s+1}, \ldots, A^r$.[5]

By Observation 6, in ξ_+ these terms appear in descending order of $\Pr(+A^i)/c(A^i)$, namely

$$\frac{\Pr(+A^2)}{c(A^2)} \;\geq\; \ldots \;\geq\; \frac{\Pr(+A^s)}{c(A^s)} \;\geq\; \frac{\Pr\left(+\tilde{A}^1\right)}{c(\tilde{A}^1)} \;\geq\; \frac{\Pr(+A^{s+1})}{c(A^{s+1})} \;\geq\; \ldots \;\geq\; \frac{\Pr(+A^r)}{c(A^r)}$$

where $\Pr\left(+\tilde{A}^1\right) = \prod_{k=2}^{n_1} \Pr(+A_k^1)$, $c(\tilde{A}^1) = c(A_2^1) + \Pr(+A_2^1)\,[c(A_3^1) + \Pr(+A_3^1) \times \{\ldots\}]$, and for any $i \geq 2$ $c(A^i) = c(A_1^i) + \Pr(+A_1^i)\,[c(A_2^i) + \Pr(+A_2^i) \times \{\ldots\}]$. For each $i \in \{1, \ldots, r\}$, the variables

[5] In substrategy ξ_+, if $A_{n_k}^k$ succeeds then the substrategy returns success. Alternatively, suppose some A_m^k fails. If $k = s$ then ξ_+ continues with A_2^1; if $k \neq s$ and $k < r$ then ξ_+ continues with A_1^{k+1}; if $k = r$ then ξ_+ returns failure.

$\{A^i_k\}_k$ are ordered in non-increasing values of $\Pr(\,-A^i_k\,)\,/c(A^i_k)$; that is,

$$\frac{\Pr(\,-A^i_k\,)}{c(A^i_k)} \;\geq\; \frac{\Pr\left(\,-A^i_{k+1}\right)}{c(A^i_{k+1})}$$

for all $k \in \{2, \ldots n_1 - 1\}$ when $i = 1$, and all $k \in \{1, \ldots n_i - 1\}$ when $i \geq 2$.

Now observe that the other substrategy ξ_- will include essentially the same terms as ξ_+ and in the same order, differing only by not including $\tilde{A}^1$.

If $s = 1$ then $\tilde{A}^1 \equiv A^1_{2..n_1}$ is the first term of ξ_+, thus the overall strategy ξ_ϕ is depth-first and we are done. Then towards a contradiction, suppose that $s \geq 2$.

Now define

$$\begin{aligned}
\sigma_1 &= \sum_{i=2}^{s} c(A^i) \times \prod_{j=2}^{i-1} \Pr\left(\,-A^j\,\right), \\
\sigma_2 &= \sum_{i=s+1}^{r} c(A^i) \times \prod_{j=2}^{i-1} \Pr\left(\,-A^j\,\right), \\
\rho &= \prod_{j=2}^{s} \Pr\left(\,-A^j\,\right).
\end{aligned}$$

The expected cost of ξ_ϕ is

$$\begin{aligned}
C[\xi_\phi] &= c(A^1_1) + \Pr(\,+A^1_1\,)\,[\sigma_1 + \rho \times c(\tilde{A}^1) + \Pr\left(\,-\tilde{A}^1\,\right) \times \sigma_2] \;+\; \Pr(\,-A^1_1\,)\,[\sigma_1 + \sigma_2] \\
&= c(A^1_1) + \sigma_1 + [\Pr(\,+A^1_1\,) \times \Pr\left(\,-\tilde{A}^1\,\right) + \Pr(\,-A^1_1\,)] \times \sigma_2 + \Pr(\,+A^1_1\,) \times \rho \times c(\tilde{A}^1) \\
&= c(A^1_1) + \sigma_1 \;+\; \Pr(\,-A^1\,) \times \sigma_2 \;+\; \Pr(\,+A^1_1\,) \times \rho \times c(\tilde{A}^1)\,.
\end{aligned}$$

Now let ξ' be a different strategy for φ, namely 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 the strategy obtained from ξ_+ by inserting A^1_1 before A^1_2. Notice that the variables of A^1 appear in ξ' in the following order: $A^1_1, A^1_2, \ldots, A^1_{n_1}$, thus the cost of dealing with A^1 is given by the formula $c(A^1) = c(A^1_1) + \Pr(\,+A^1_1\,)\,c\left(\tilde{A}^1\right)$.

The cost of the ξ' strategy is

$$C[\xi'] \quad = \quad \sigma_1 \quad + \quad \rho \times c(A^1) \quad + \quad \Pr\left(\,-A^1\,\right) \times \sigma_2\,.$$

Now observe that

$$\begin{aligned}
&C[\xi_\phi] - C[\xi'] \\
&\quad = [c(A^1_1) + \sigma_1 + \;\Pr(\,-A^1\,) \times \sigma_2 \;+\; \Pr(\,+A^1_1\,) \times \rho \times c(\tilde{A}^1)] - [\sigma_1 \;+\; \rho \times c(A^1) \;+\; \Pr(\,-A^1\,) \times \sigma_2] \\
&\quad = c(A^1_1) \;+\; \rho\,[\Pr(\,+A^1_1\,) \times c(\tilde{A}^1) - c(A^1)] \\
&\quad = c(A^1_1) + \rho\,[-c(A^1_1)] \quad = \quad c(A^1_1)\,[1 - \rho] \quad > \quad 0
\end{aligned}$$

since ρ is a product of probability values less than 1 and $c(A^1_1)$ is nonnegative.

This implies that ξ' has a lower expected cost than the optimal strategy ξ_ϕ, contradiction. $\square$

Theorem 13 *There are unit-cost and-or trees with n nodes for which the best depth-first strategy costs $\Theta(n^{1-o(1)})$ times as much as the best strategy.*

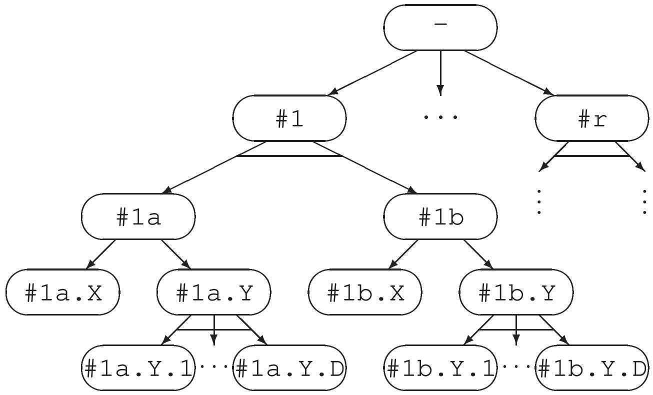

Figure 16: A problematic instance for DFA; Theorem 13

Proof of Theorem 13: To simplify the proof (here and for Theorem 18), 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 (e.g. $1 - 2^{-2^n}$), 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 with n nodes suggested by Figure 16: 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 that is `true` with probability $1 - q$ where $q = \frac{1}{2}(n/2r)^{-1/r}$. Each `*.Y` node has $D = (n - 7r - 1)/2r$ 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, ⟨`#1a.X`, `#1b.X`, ..., `#ra.X`, `#rb.X`⟩, and then, if necessary, carry out the children of two `*.Y` "cousins". If i is the index of the first, if any, pair (`#ia.X`, `#ib.X`) to both evaluate to `true`, then the cost is $2i$. The probability of this occurring is $\alpha^{i-1}(1-\alpha)$, where $\alpha = q + (1-q)q = 2q - q^2$ and $1 - \alpha = 1 - (2q - q^2) = (1-q)^2$. If none of these pairs both evaluate to `true`, then the total cost is at most $2r + 2D$.

We are free to choose relative values for r and D; for now, we assume $r \leq D$, so $2r + 2D \leq 4D$. Thus this strategy has an expected total cost of at most

$$\left(\sum_{i=1}^{r} 2i\alpha^{i-1}(1-\alpha)\right) + \alpha^r 4D = 2(1-\alpha)\left(\frac{1-\alpha^r(1+r(1-\alpha))}{(1-\alpha)^2}\right) + \alpha^r O(D) <$$
$$2/(1-\alpha) + (2q)^r O(D) < 2/(1-\alpha) + (n/2r)^{-1} O(n/2r) = O(1) .$$

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 rooted subtree `#1a` first. With probability at least q, it will have to evaluate rooted subtree `#1a.Y` at a cost of D. Setting $r = \log n$ yields the theorem. □

Theorem 18 *There are unit-cost and-or trees with n nodes for which the best linear strategy costs $\Theta(n^{1/3-o(1)})$ times as much as an optimal strategy.*

Proof of Theorem 18: We consider the following strictly alternating unit-cost tree, suggested by Figure 17. 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

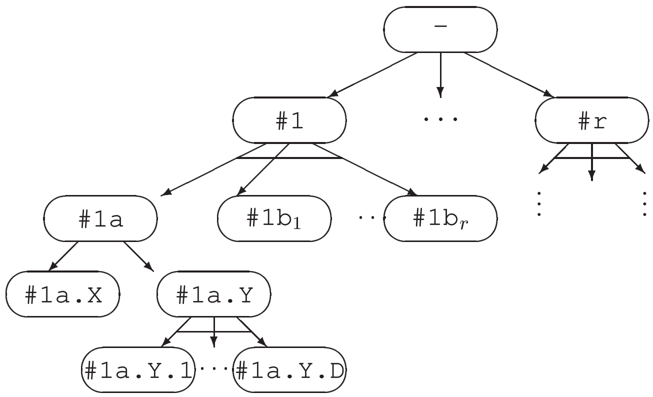

Figure 17: A problematic instance for Linear Strategies; Theorem 18

$r + 1$ children: `#ia`, `#ib`$_1, \dots$ `#ib`$_r$. Each `#ib`$_i$ is `true` with probability 1. The node `#ia` has 2 children: the leaf node `#ia.X` that is `true` with probability $q = \log^2 n/(n^{1/3})$; and the internal node `#ia.Y` that 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 $\min(D, r/q)$. Since $D \approx r/q$, the ratio between the expected cost of the strategy from the previous paragraph and the expected cost of the best linear strategy is $\approx (r/q)/(3r) = n^{1/3-o(1)}$

Consider an optimal linear strategy. Thus the strategy can be represented by an ordering of the leaves, which we denote by ξ. We test the leaves in this order, skipping a leaf if and only if by the time we get to it, it will not help resolve the tree, meaning that the value of one of its ancestor nodes in the and-or tree is already known.

We start by proving a few basic properties about ξ.

Claim 1: *For any i, we can assume that ξ is contiguous with respect to the set of nodes* `#ia.Y.1, ..., #ia.Y.D`.

Proof: Suppose that this is not a case, that is that `#ia.Y.1, ..., #ia.Y.D` do not form a consecutive subsequence in the linear ordering encoding ξ. 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 ξ'. It is easy to see that no leaf will be evaluated in ξ' but not in ξ. 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 ξ is contiguous with respect to the set of nodes* `#ib`$_1$`, ..., #ib`$_r$.

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 and 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` (respectively `#ia.X`) occurs later in ξ. Note that if it is the last choice, then `#ia.X` will always be redundant, and so its exact location in ξ is irrelevant.

Let ξ_i denote the particular consecutive subsequence described above ending with `#iB`. If `#i.Y` is in ξ_i, then the expected cost of carrying out ξ_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 $\xi_1, \xi_2, \ldots, \xi_r$ occur in that order in ξ. Let i^* be the first i such that either `#i.Y` is in ξ_i or some `#j.Y` precedes ξ_i. (Note that in the latter case, `#j.Y` comes after ξ_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 $r/q \approx D$, and so is at least $\min(D, r/q)$, as required. □

Theorem 20 — Siblings and Twins Theorem *For any and-or tree T, there is an optimal strategy ξ_T that satisfies both of the following conditions:*

1. *for any sibling tests x and y such that $R(\,y\,) > R(\,x\,)$, x is not performed before y on any root-to-leaf path of ξ_T*
2. *for any R-class W, ξ_T is contiguous with respect to W.*

Proof of Theorem 20: Our proof uses the following notation and observation:

We write $p(x)$ (resp., $\bar{p}(x)$) as a shorthand for Pr($+x$) (resp., Pr($-x$)).

For strategies ξ_1, ξ_2 and a test x

$$x : +(\xi_1)\,;-(\xi_2)$$

denotes the strategy whose root is labeled x and whose substrategies rooted at the root's children, entered by `true` and `false` arcs are respectively ξ_1 and ξ_2.

For a strategy ξ that has disjoint substrategies $\xi_1, \ldots, \xi_m$, $m \geq 1$, and for strategies $\xi'_1, \ldots, \xi'_m$

$$\xi\;(\xi_1 \triangleleft \xi'_1, \ldots, \xi_m \triangleleft \xi'_m)$$

denotes the tree that is obtained from the strategy ξ by replacing the rooted subtree ξ_k by the tree ξ'_k for each $k = 1, 2, \ldots, m$.

Observation 26 *Let tests x and y be child tests of the same or-node in an and-or tree T. Let ξ_{xy} be a nonredundant strategy for T of the following form: $x : +(\xi_+) ; -(y : +(\xi_+) ; -(\xi_-))$, for some strategies ξ_+ and ξ_-.*

By switching the labels y and x in ξ_{xy} we obtain another strategy ξ_{yx} that is nonredundant and

i) if $R(y) > R(x)$ then ξ_{yx} has the lower expected cost than ξ_{xy},

ii) if $R(y) = R(x)$ then ξ_{yx} has the same expected cost as ξ_{xy}.

Proof of Observation 26: The correctness and nonredundancy of ξ_{yx} are obvious. For the expected cost of ξ_{xy} we have:

$$C[\xi_{xy}] = c(x) + \bar{p}(x)\, c(y) + \bar{p}(x)\, \bar{p}(y)\, C[\xi_-] + [1 - \bar{p}(x)\, \bar{p}(y)]\, C[\xi_+].$$

Using a similar expression for $C[\xi_{yx}]$ we obtain

$$C[\xi_{yx}] - C[\xi_{xy}] = p(x)\, c(y) - p(y)\, c(x).$$

The observation follows immediately.

We prove Theorem 20 by induction on the number of tests in an and-or tree. The theorem holds for the base case of a tree with only one test. Now assume that it holds for any and-or tree that has fewer tests than the tree T has.

Let ξ be an optimal strategy for T. Thus ξ is nonredundant. Let x be the first test performed by ξ. Assume that x is a child of an or-node. (the proof for the "and" case is symmetric). Let ξ_{+x} (ξ_{-x}) be the substrategy of ξ that is followed when x is `true` (`false`). By induction, we may assume that ξ_{+x} and ξ_{-x} are contiguous on any R-class and preserve "the right order" of sibling tests (*i.e.*, never perform a sibling test before its sibling test with higher R-ratio).

Now assume that ξ does not fulfill the conditions of the theorem. That means that x has at least one sibling test with the same or higher R-ratio. We will show that, in this case, there is another optimal strategy that satisfies the conditions of the theorem. To construct such a strategy, we will use the technique of changing the order of parts of the original strategy.

Let Y be the set of all and only sibling tests of x with R-ratio higher than or equal to $R(x)$. Let y be the test with minimum R-ratio among all tests from Y. Observation 26ii implies that the order of performing tests from one R-class is arbitrary in a strategy that is contiguous on this class. Thus we may assume that y is always performed as the last test from Y by the substrategy ξ_{-x}.

Now let $M \geq 1$ be the number of nodes of ξ_{-x} labeled by test y, let $\xi_{y_1}, \xi_{y_2}, \ldots, \xi_{y_M}$ be the rooted subtrees of ξ_{-x} rooted at nodes labeled by y, and for $k = 1, 2, \ldots, M$, let ξ_{+y_k}, ξ_{-y_k} be the substrategies of ξ_{y_k} followed in the case when y is `true`, y is `false`, respectively. Also let ξ_r denote the (possibly empty) part of ξ_{-x} that contains all nodes outside $\xi_{y_1}, \xi_{y_2}, \ldots, \xi_{y_M}$ (see Figure 18(a)).

Consider the tree $\xi(x \to y) = \xi_{-x}(\xi_{y_1} \triangleleft \xi_{x_1}, \ldots, \xi_{y_M} \triangleleft \xi_{x_M})$, where for $k = 1, 2, \ldots, M$, $\xi_{x_k} = x : +(\xi_{+y_k}) ; -(\xi_{y_k})$, shown in Figure 18(c) (we query x just before y). To show that it is indeed a strategy, we need to check that for each leaf node L of $\xi(x \to y)$, the label of L (`true` or `false`) is the correct value of T for all assignments of tests that correspond to the path P_L from the root of $\xi(x \to y)$ to L. This obviously holds if P_L contains a node labeled by test y, since in ξ the corresponding root-to-leaf path differs from P_L only in the order of performing tests. Knowing that, we see that the label of L is correct if P_L contains a node labeled by x and the arc labeled `true` that leaves this node (because after x succeeds, we do exactly the same as what we do if x fails but its sibling test y succeeds). The only remaining case is when neither x nor y is performed on P_L. Let σ_L be any assignment of tests that correspond to P_L. In ξ we follow the path identical to P_L after x fails. And so for any σ_L in which x is `false` the label of the leaf L is correct. To see

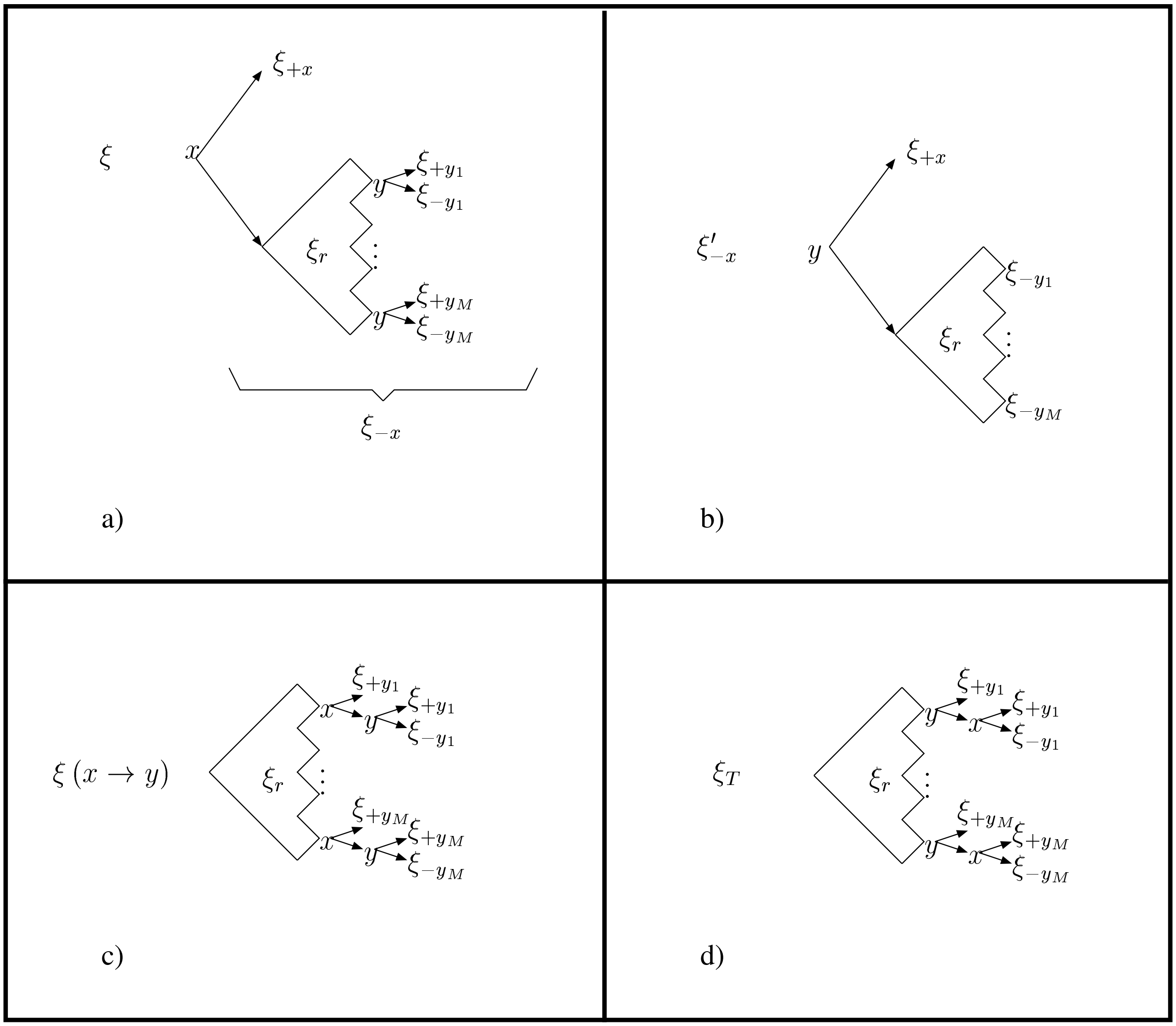

Figure 18: Illustration for the proof of Theorem 20; for any node the up (respectively down) arc denotes the `true` (resp. `false`) arc. (a) The optimal strategy ξ with substrategies ξ_{+x} and ξ_{-x}. (b) The strategy ξ'_{-x} that may replace the substrategy ξ_{-x}. (c) The optimal strategy $\xi\,(x \to y)$. (d) The optimal strategy ξ_T that fulfills the conditions of Theorem 20.

that it is also correct if x is `true`, consider any two assignments of tests σ_1 and σ_2, that may differ only in the values of x and y, assume that x is `true` in σ_1, x is `false` in σ_2, and that y is `true`, and then observe that the value of the tree T is the same for σ_1 and σ_2. And so the correctness of the label of the leaf node of P_L in this case follows from the fact that in ξ we do not test y on the corresponding root-to-leaf path. It is obvious that $\xi\,(x \to y)$ is nonredundant.

Now let ξ_T be obtained from $\xi\,(x \to y)$ by switching the labels x and y of its neighbour nodes (Figure 18(d)). By Observation 26 ξ_T is indeed a strategy for T (i.e. it evaluates T correctly).

If the R-class containing x also includes other tests, then it has to contain y, so ξ_T is contiguous on this class. Note that ξ_T is also contiguous on any R-class that does not include x; for the R-class including y (if $R(x) \neq R(y)$) this follows from the fact that y is performed as the last test from Y. Also, since x is tested just after y, when y is `false`, ξ_T preserves the right order of sibling tests of T.

Observation 26 implies that ξ_T does not have higher expected cost than $\xi\,(x \to y)$. Thus to complete the proof it is enough to show that $\xi\,(x \to y)$ is optimal.

Let C^r denote the expected cost of dealing with ξ_r, that is $C^r = \sum_{v\in\xi_r} p_v c\,(x_v)$, where the sum is over all nodes of ξ_r, p_v is the probability of the path from the root of ξ_{-x} to node v, and $c\,(x_v)$ is the cost of the test that labels node v . For any k, let p_{y_k} be the probability of the path from the root of ξ_{-x} to the labeled by y root node of ξ_{y_k}.

Then we can express the expected costs of ξ in the following way:

$$C\,[\xi] = c\,(x) + p\,(x)\,C\,[\xi_{+x}] + \bar{p}\,(x)\,C\,[\xi_{-x}]\,,$$

where

$$C\,[\xi_{-x}] = C^r + \sum_{k=1}^{M} p_{y_k}\left[c\,(y) + p\,(y)\,C\,[\xi_{+y_k}] + \bar{p}\,(y)\,C\,[\xi_{-y_k}]\right],$$

while for the expected cost of $\xi\,(x \to y)$ we have:

$$\begin{aligned} C\,[\xi\,(x \to y)] \quad = \quad & C^r + \sum_{k=1}^{M}\Big\{p_{y_k}\Big[c\,(x) + p\,(x)\,C\,[\xi_{+y_k}] + \\ & +\bar{p}\,(x)\left[c\,(y) + p\,(y)\,C\,[\xi_{+y_k}] + \bar{p}\,(y)\,C\,[\xi_{-y_k}]\right]\Big]\Big\}. \end{aligned}$$

Towards a contradiction, assume that the expected cost of $\xi\,(x \to y)$ is higher than ξ. Then using the notation $D = C^r + \sum_{k=1}^{M} p_{y_k} C\,[\xi_{+y_k}] - C\,[\xi_{+x}]$ and $P^r = 1 - \sum_{k=1}^{M} p_{y_k}$, we obtain

$$p\,(x)\,D > P^r c\,(x)\,.$$

Notice that $\sum_{k=1}^{M} p_{y_k}$ is the total probability of reaching any node labeled by y after entering the strategy ξ_{-x}, so $P^r \geq 0$. That implies that $D > 0$ and so

$$\frac{p\,(x)}{c\,(x)} > \frac{P^r}{D}. \tag{6}$$

We will show that it follows from (6) that we can replace the substrategy ξ_{-x} of the original strategy by a substrategy with strictly lower expected cost, which contradicts the optimality of ξ_{-x}.

Consider $\xi'_{-x} = y : +\,(\xi_{+x})\,; -\,(\xi_{-x}\,(\xi_{y_1} \triangleleft \xi_{-y_1}, \ldots, \xi_{y_M} \triangleleft \xi_{-y_M}))$; see Figure 18(b). It is not difficult to see that ξ'_{-x} is indeed a strategy for the reduced tree obtained from T when x is `false` and that it is nonredundant. We have the following expression for the expected cost of ξ'_{-x}:

$$C\,[\xi'_{-x}] = c\,(y) + p\,(y)\,C\,[\xi_{+x}] + \bar{p}\,(y)\left[C^r + \sum_{k=1}^{M} p_{y_k} C\,[\xi_{-y_k}]\right].$$

Using the same notation as before, we obtain

$$C\left[\xi_{-x}\right] - C\left[\xi'_{-x}\right] = p\left(y\right) D - P^{r} c\left(y\right).$$

But then from (6) and the fact that

$$\frac{p\left(y\right)}{c\left(y\right)} \geq \frac{p\left(x\right)}{c\left(x\right)},$$

it follows that $C\left[\xi_{-x}\right] - C\left[\xi'_{-x}\right] > 0$, contradiction. □

Theorem 21 DYNPROG *produces an optimal strategy for and-or trees. The time complexity of the algorithm is in* $O\left(d^2(r+1)^d\right)$ *and the space complexity is in* $O\left((r+1)^d\right)$*, where* r *is the largest number of leaf-siblings of a tree and* d *is the number of leaf-parents in a tree.*

For the special case when d *is fixed, the time complexity is in* $O(r \ln r)$ *if* $d = 1$ *and in* $O\left(r^d\right)$ *for any fixed* $d \geq 2$*, while the space complexity is in* $O\left(r^d\right)$ *for any fixed* $d \geq 1$.

Proof of Theorem 21:

The correctness of the algorithm follows from Theorem 20, as discussed in the description of the algorithm. We have also shown before that the total number of reduced trees is at most $(r+1)^d$.

Lines (1)–(2) order tests inside each sibling class, so the time required to perform them is in $O\left(dr \ln r\right)$, and for any fixed d this time is in $O\left(r \ln r\right)$.

Lines (3)–(4) take time $O\left((r+1)^d\right)$; for any fixed d this time is in $O\left(r^d\right)$.

The loop (9)–(18) has at most d iterations. Since each parent node of a sibling-class is associated with an array of leaf children, ordered by R-ratio, it takes constant time to find x_L (line 10). As we have shown before, we can calculate I_L^+ or I_L^- in time $O\left(d\right)$ (lines (11) and (12)). Also, since we identify trees with d-tuples, we may find data for trees I_L^+ and I_L^- (line 13) in $O\left(d\right)$ time, as entries of a d-dimensional matrix. And so the time required by the entire loop (9)–(18) is in $O\left(d^2\right)$.

For a given number M, the time required to calculate a next d-tuple I with the property that the sum of all elements of I is M, and to move to the corresponding entry in the d-dimensional matrix, is in $O\left(d\right)$. In total, the above mentioned operations, as well as the loop (9)–(18), are performed once for each reduced tree. Thus the time required by lines (7)–(18) is in $O\left(d^2(r+1)^d\right)$, and for any fixed d this time is in $O\left(r^d\right)$.

Thus the time complexity of the algorithm is in $O\left(dr \ln r + d^2(r+1)^d\right)$, that is in $O\left(d^2(r+1)^d\right)$. For any fixed d the time complexity is in $O\left(r \ln r + r^d\right)$. Thus if $d = 1$ the time complexity is in $O\left(r \ln r\right)$, for fixed $d \geq 2$ it is in $O\left(r^d\right)$.

The number of internal nodes of the input tree is at most $2d$ (as each internal node has out-degree at least 2), the number of leaves of the input tree is at most dr. Thus the size of the input tree is in $O\left(dr\right)$, and for any fixed d it is in $O\left(r\right)$. This is also the space required for ordering tests inside each sibling class (lines (1)–(2)). In the remaining of the algorithm, we store a constant amount of data for each reduced tree, and we need an additional $O\left(d\right)$ space for the calculations. Thus the space complexity is in $O\left((r+1)^d\right)$, and for any fixed d the space complexity is in $O\left(r^d\right)$.

□