University of Alberta

Library Release Form

Name of Author: Magdalena Jankowska

Title of Thesis: Probabilistic And-Or Tree Resolution

Degree: Master of Science

Year this Degree Granted: 2004

Magdalena Jankowska
203 Vanier House, Michener Park
Edmonton, Alberta
Canada, T6H 4N1

Date: ______________

University of Alberta

Probabilistic And-Or Tree Resolution

by

Magdalena Jankowska

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Spring 2004

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Probabilistic And-Or Tree Resolution** submitted by Magdalena Jankowska in partial fulfillment of the requirements for the degree of **Master of Science**.

Dr. Ryan Hayward (Supervisor)

Dr. Russell Greiner

Dr. Erhan Erkut

Date: ____________________

Abstract

An *and-or tree* is a Boolean expression over a set of independent probabilistic tests, each with an associated performance cost and truth probability, such that no test appears more than once. It can be represented by a tree in which leaf nodes are tests and non-leaf nodes are either `and` or `or`. *Probabilistic and-or tree resolution* (PAOTR) is the problem of finding an algorithm for evaluating an `and-or` tree with smallest expected cost. The complexity of PAOTR is unknown.

Our main result is that a natural partial order of sibling tests in an `and-or` tree yields a dynamic programming algorithm for PAOTR which for trees with a bounded number of non-leaf nodes runs in time polynomial in the input size. We also study some generalizations of PAOTR and present some special classes of `and-or` trees for which PAOTR is in P.

Acknowledgements

I am especially grateful to my supervisor Ryan Hayward for his assistance, encouragement, and support. I thank him, Russell Greiner, and Omid Madani for providing several ideas and conjectures throughout my work. I am very grateful to Ryan Hayward, Russell Greiner, and Erhan Erkut for the comments on the thesis. I thank Leah Hackman and Martha Lednicky, WISEST 2003 participants, whose experimentation with instances of `and-or` trees resulted in helpful observations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Related Work

1.1 Introduction

Consider the following recruitment situation at a hypothetical company. Potential employees must meet certain psychological standards and either be in good physical shape or pass both an Intelligence Quotient (IQ) test and a knowledge test. Four evaluation tests are prepared: a psychological test P, a fitness test F, an IQ test I and a knowledge test K. For each test the only possible output is either pass or fail. A candidate will be hired if the Boolean expression $e = P$ `and` $[F$ `or` $(I$ `and` $K)]$ evaluates to `true`; see Figure 1.1. Each test has a performance cost. For each test, the company knows from its recruiting history the probability that a candidate will pass that test. There are several algorithms (strategies) the company may use to

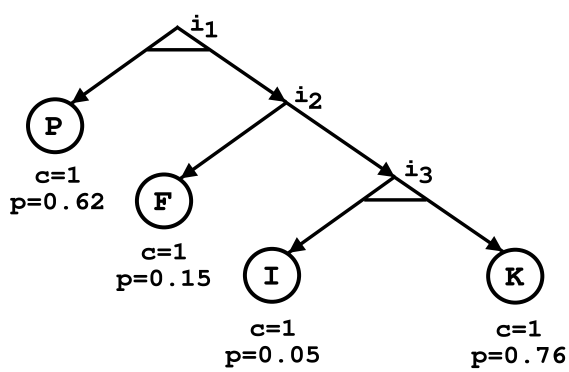

Figure 1.1: An `and-or` tree T_r. An internal node with a horizontal bar (respectively no bar) indicates an `and`-node (respectively `or`-node). For each test, the cost and probability values are denoted respectively by c and p.

decide whether to accept a candidate. One such strategy is the strategy S_1 shown in Figure 1.2a: first administer the psychological test; if the candidate fails this test then return `false` (the candidate is rejected), otherwise next administer the fitness test; if the candidate passes this test then return `true` (the candidate is accepted), otherwise administer the IQ test; if the candidate fails this test then return `false`, otherwise administer the knowledge test; if the candidate passes this test then return `true`, otherwise return `false`. Another possible strategy the company might

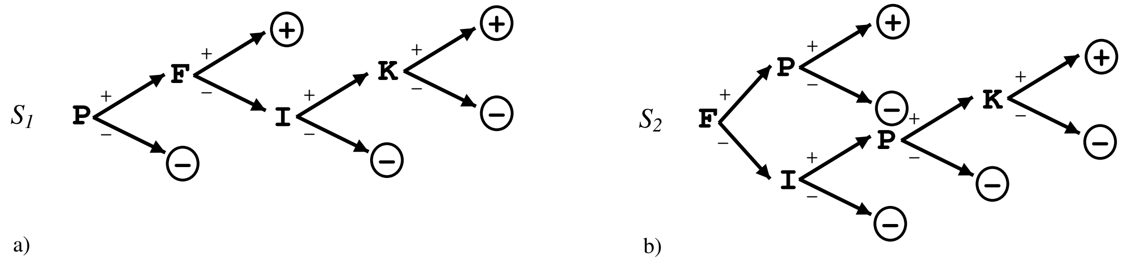

Figure 1.2: Two strategies for the `and-or` tree T_r.

use is the strategy S_2 shown in Figure 1.2b. The company wants to use a strategy which has the minimum expected cost.

The aforementioned expression e is an example of a *probabilistic Boolean expression*, namely a Boolean expression with a cost and probability for each variable. A probabilistic Boolean expression such as e that can be represented by a tree in which each internal node corresponds to a Boolean operator, either `and` or `or`, and each leaf node corresponds to a test with `true/false` output is called an `and-or` tree. We will provide more formal definitions in Section 1.2. For a given probabilistic Boolean expression (respectively `and-or` tree), the problem of finding a strategy with the minimum expected cost over all strategies that evaluate the expression correctly is called *probabilistic Boolean expression resolution* (PBER) (respectively *probabilistic and-or tree resolution* (PAOTR)).

As discussed above, the problem of deciding the value of a probabilistic Boolean expression may arise in practical applications. For many Boolean expressions, including all `and-or` trees, each (deterministic) strategy has to query all variables in the worst case. For this reason the expected cost of a strategy is a reasonable performance measure. The objective is to find an algorithm that will *on average* cost as little as possible.

In this thesis we present the results of our research on PAOTR and some of its generalizations. PAOTR is a natural restriction of PBER which is also interesting from the complexity point of view. PBER is NP-hard for general probabilistic Boolean expressions; it is also NP-hard for the class of probabilistic Boolean expressions in which no variable is negated. `And-or` trees constitute a subclass of the latter class, containing expressions in which each variable appears exactly once. The complexity of finding an optimal strategy for `and-or` trees is unknown.

Previous to our work, a polynomial-time algorithms were known for depth one or two `and-or` trees (see Section 1.3.3) and balanced `and-or` trees (see Section 1.3.4). We present a dynamic programing algorithm to find an optimal strategy for `and-or` trees which runs in time $O(d^2n^d)$, where n is the number of leaves and d is the number of internal nodes that are leaf-parents in a tree. Thus for trees with a bounded number of internal nodes, the running time of our algorithm is polynomial in the input size. The algorithm relies on the discovered optimal relative order of querying tests (variables) that are siblings in a tree. We also show that PAOTR for some special `and-or` trees is in P as well. We describe a way in which PAOTR for trees with all costs unit can be used as an approximation for general PAOTR. We also consider a generalization of PBER, in which there are precondition constraints for tests. Among such expressions, preconditioned `and-or` trees generalize `and-or` trees. We show that an extension of a previously known algorithm (Smith's Algorithm) produces an optimal strategy for a subclass of preconditioned `and-or` trees.

In this chapter we provide formal definitions (Section 1.2) and present a survey of the previous results related to PBER (Section 1.3).

We state the key lemma describing the optimal relative order of performing sibling tests in Section 2.1. In Section 2.2 we present the dynamic programming algorithm to find an optimal strategy for `and-or` trees. We show that PAOTR for special classes of `and-or` trees is in P in Sections 2.3 and 2.4. In Section 2.5 we describe the reduction that allows to approximate any `and-or` tree by a tree whose all tests have the same cost.

In Chapter 3 we discuss several natural conjectures with respect to optimal strategies for `and-or` trees. For some, we present counterexamples; others remain open. In Chapter 4 we study preconditioned `and-or` trees and describe a generalization of Smith's Algorithm for finding an optimal strategy for some such trees. In Chapter 5 we summarize our findings and discuss open problems.

1.2 Definitions and Notations

Since we use directed graphs to represent both Boolean expressions and algorithms for their evaluation, we begin with the related definitions. We follow the definitions from [5].

A directed graph G is an ordered pair (V, A), where V is a finite set and A is a set of ordered pairs of elements of V. Elements of V are called *nodes* and elements of A are called *arcs*.

Let $G = (V, A)$. For an arc $a = (v, w)$ we say that a leaves v and enters w. The *out-degree* of a node v is the number of the arcs that leave v; the *in-degree* of v is the number of the arcs that enter v. A *subgraph of G induced by a set $V' \subseteq V$* is the directed graph $G' = (V', A')$, where $A' = \{(v, w) \in A : v, w \in V'\}$. A *path* of *length* k from v to w in G is a sequence of nodes $(v_0, v_1, \ldots, v_k)$, $k \geq 0$ such that $v_0 = v$, $v_k = w$ and for $i = 1, 2, \ldots, k$ $(v_{i-1}, v_i) \in A$. The arcs of the path are the arcs (v_{i-1}, v_i), $i = 1, 2, \ldots, k$. A node w is *reachable* from v if there is a path from v to w. A path $(v_0, v_1, \ldots, v_k)$ is a *cycle* if $v_0 = v_k$ and $k \geq 1$. If there is no cycle in G, G is *acyclic* and is called a *DAG* (directed acyclic graph). In a DAG, for each arc (v, w) we call w a *child node* of v and v a *parent node* of w. A node of a DAG that has in-degree zero (respectively out-degree zero) is called a *source* (respectively *sink*).

Let us now define formally a special kind of directed acyclic graph called a *directed rooted tree*. This concept is often referred to elsewhere as simply a rooted tree; we add the term "directed" to emphasize that this is in fact a directed graph.

Definition 1 *A directed graph $T = (V, A)$ is a* ***directed tree rooted at a node*** *r if* *i)* $V = \{r\}$ *and* $A = \emptyset$*, or*

ii) $V = \{r\} \cup V_1 \cup V_2 \cup \ldots \cup V_k$*,* $A = \{(r, r_1), (r, r_2), \ldots, (r, r_k)\} \cup A_1 \cup A_2 \cup \ldots \cup A_k$*,* $k \geq 1$*, where* $\{r\}, V_1, V_2, \ldots, V_k$ *are disjoint sets and* $T_1 = (V_1, A_1), T_2 = (V_2, A_2), \ldots, T_k = (V_k, A_k)$ *are directed trees rooted respectively at nodes* $r_1, r_2, \ldots, r_k$*.*

A graph G is a *directed rooted tree* if there is a node r such that G is a directed tree rooted at r. The graphs shown in Figures 1.1 and 1.2 are examples of directed rooted trees.

Let $T = (V, A)$ be a directed tree rooted at r. We call r the *root* of T. Two nodes with the same parent node are *siblings*. A node with out-degree zero is a *leaf*; a node with positive out-degree is *internal*. A (full) *binary tree* is a directed rooted tree in which each internal node has out-degree two. The *depth of a node* v is the length of the path from the root r to v. The *depth* of the tree T is the maximum depth of a node, over all nodes of T.

Each node reachable from v is called a *successor* of v; each node from which v is reachable is called a *predecessor* of v. Notice that each node is both a successor and a predecessor of itself. Observe that the subgraph of T induced by all successors of some node v is a directed tree rooted at v.

Definition 2 *A* ***subtree*** *of a directed rooted tree* T *is a subgraph of* T *induced by all successors of some node of* T*.*

A subtree rooted at a child node of r is called an *immediate subtree* of T and a *child subtree* of r; the node r is called the *parent node* of its child subtrees.

For a directed tree $T = (V, A)$ rooted at r with $|V| > 1$, there is a unique set of directed rooted trees $T_1, T_2, \ldots, T_k$ that fulfills condition ii) of Definition 1, namely the set of immediate subtrees of T. Together with the root, this set specifies T. For a node r and a finite set of directed rooted trees Ψ, we use

$$\langle r, \Psi \rangle \tag{1.1}$$

to denote the directed tree rooted at r and with the set of immediate subtrees Ψ.

A *probabilistic Boolean expression* is a Boolean expression over a set of probabilistic tests, each of which has an associated cost and truth probability; we assume that costs are non-negative.

We say that a test *succeeds* if it has the value (the outcome) `true`, otherwise we say that the test *fails*. By *performing* a test we mean determining its value.

For any test x of a probabilistic Boolean expression, $c(x)$ denotes the cost of performing x, $p(x)$ denotes the probability of success of x, and $\bar{p}(x) = 1 - p(x)$ denotes the probability of failure of x. We will on occasion use the symbols c_x, p_x, $\bar{p}_x$ to denote $c(x)$, $p(x)$, $\bar{p}(x)$ respectively.

Consider a set of tests $D = \{x_1, x_2, \ldots, x_k\}$. A *setting* of tests $x_1, x_2, \ldots, x_k$ is an assignment of Boolean values (`true` or `false`) for the tests, that is a vector $\sigma = (v(x_1), v(x_2), \ldots, v(x_k))$ where, for $i \leq k$, $v(x_i)$ is a Boolean value of the test x_i. The probability $p(\sigma)$ of a setting σ is the probability that for each $i \leq k$ the test x_i has the value $v(x_i)$.

The tests $x_1, x_2, \ldots, x_k$ are *independent* if for any subset $D' = \{y_1, y_2, \ldots, y_l\} \subseteq D$ and any setting $\sigma = (v(y_1), v(y_2), \ldots, v(y_l))$ of the tests from D'

$$p(\sigma) = \prod_{i=1}^{l} p_{v(y_i)}, \tag{1.2}$$

where

$$p_{v(y_i)} = \begin{cases} p(y_i) \text{ if } v(y_i) \text{ is } \texttt{true}, \\ \bar{p}(y_i) \text{ if } v(y_i) \text{ is } \texttt{false}. \end{cases} \tag{1.3}$$

Definition 3 *A **strategy** for a probabilistic Boolean expression e is a binary tree S such that each internal node is labeled by a test from e, each leaf node is labeled either* `true` *or* `false`*, and for each internal node v of S, one of the arcs leaving v is labeled* `true` *and the other is labeled* `false`*.*

Whenever we represent a strategy graphically, we draw the tree so that arcs point to the right, and we use the symbols $+$ and $-$ to denote respectively the labels `true` and `false`. See Figure 1.2.

As defined above, a strategy represents an algorithm for calculating (not necessary correctly) the value of the expression, for any setting of tests; the algorithm performs tests sequentially, and the selection of each subsequent test is based on the values of the previous tests. A leaf node of S represents the return by the algorithm of the value equal to the leaf label as the value of the input expression. The first test performed by the algorithm is the test that labels the root r of S. Depending on the value of this test, one of arcs leaving r (the one labeled by this value) is followed; the child subtree of r entered by this arc represents the further actions of the algorithm. For example the tree shown in Figure 1.2a represents an algorithm that starts with performing the test P. If the value of P is `false`, the algorithm returns the value `false`, otherwise the algorithm performs the test F.

Whenever it does not cause any confusion, we use the word "strategy" to denote both a tree and its associated algorithm. For example, we use alternatively the equivalent phrases "the root of the strategy S is labeled by the test x" and "the strategy S starts with performing the test x".

If in a strategy (tree) a path P contains a node labeled by a test x then we say that *x is performed on P*. If P contains also a node labeled by a test y and this node has smaller depth than the node labeled by x, we say that *y is performed before x* on P.

Let $P = (v_0, v_1, \ldots, v_k)$ with $k \geq 0$ be a path from the root v_0 of S to a leaf node v_k. A setting σ corresponds to P if for any $i < k$, the value in σ of the test x_i that labels the node v_i is equal to the label of the arc (v_i, v_{i+1}). Observe that any setting corresponds to exactly one root-to-leaf path, and several settings may correspond to one root-to-leaf path. For example, the settings $\sigma_1 = (F-, I+, P-, K+)$ and $\sigma_2 = (F-, I+, P-, K-)$, correspond to the same root-to-leaf path of the strategy S_2 shown in Figure 1.2b.

A strategy S for a probabilistic expression e is *correct* if for any root-to-leaf path P of S, the value of e under any setting that corresponds to P is the same as the label of the leaf of P.

Considering a strategy as an algorithm, correctness of the strategy means just that the algorithm calculates correctly the value of a probabilistic Boolean expression for any setting of tests.

The *cost of a strategy* S *under a setting* σ is the sum of costs of all tests performed on the root-to-leaf path of S to which σ corresponds. For example, the cost of the strategy S_2 shown in Figure 1.2b under both the above given settings σ_1 and σ_2 is $c(F) + c(I) + c(P)$. We denote by $c_\sigma(S)$ the cost of S under a setting σ.

The *expected cost* $C(S)$ of a strategy S for an expression e is the average cost of the strategy S over all settings of the tests of e:

$$C(S) = \sum_{\sigma \in Settings(e)} p(\sigma)\, c_\sigma(S), \tag{1.4}$$

where the sum is taken over all settings of the tests of e and $p(\sigma)$ is the probability of the setting σ.

The expected cost is the measure of the performance of a strategy we are concerned with in this thesis.

Definition 4 *A correct strategy S for a probabilistic Boolean expression e is **optimal for e** if S has the minimum expected cost among all correct strategies for e, that is if for any correct strategy S' for e it holds that $C(S) \leq C(S')$.*

Probabilistic Boolean expression resolution (PBER) is the problem of finding an optimal strategy for a given probabilistic Boolean expression. For a given probabilistic Boolean expression e, the *optimal resolution cost* is the expected cost of an optimal strategy for e.

Several internal nodes of a strategy can be labeled with the same test; see for example strategy S_2 of Figure 1.2b. Notice though that for any setting of tests, performing the same test more than once always yields the same value, while increasing unnecessarily the cost of the strategy under this setting (unless the cost of the test is zero, in which case performing the test does not influence the cost of the strategy under this setting). More formally, if two nodes of one root-to-leaf path P of the strategy are labeled by the same test x, then either there is no setting that corresponds to P (because two arcs of P that leave the nodes labeled by x are labeled by different values) or removing one node labeled by x does not change the set of the settings that correspond to the path without increasing the cost of the strategy under any of these settings. We call a strategy *nonredundant* if on any root-to-leaf path of the strategy no two nodes are labeled with the same test. We can restrict our attention to such strategies, because for any probabilistic Boolean expression there is an optimal nonredundant strategy.

In this thesis we will concentrate on a special case of a probabilistic Boolean expression, which we define formally in terms of labeled directed rooted trees with special features.

Definition 5 *A rooted directed tree $T = \langle r, \Psi \rangle$ with labeled nodes and a Boolean value is an **and-or tree** if one of the following conditions is fulfilled:*

- *Ψ is empty, r is labeled by a probabilistic test with given non-negative cost and success probability and the Boolean value of T is equal to the value of the test, or*
- *any tree $T' \in \Psi$ has labeled nodes and a Boolean value so that it is an* `and-or` *tree, the tests that label all leaves of all trees in Ψ are distinct and independent, and*
  - *either r is labeled* `and` *and the value of T is* `true` *if and only if the value of each tree from Ψ is* `true`*, or*
  - *r is labeled* `or` *and the value of T is* `true` *if and only if the value of at least one tree from Ψ is* `true`*.*

We draw `and-or` trees so that arcs point downward. The `and`-labeled nodes (respectively the `or`-labeled nodes) are denoted by a horizontal bar (respectively no horizontal bar) below the nodes. See for example the `and-or` tree in Figure 1.1.

Obviously, an `and-or` tree T defines a probabilistic Boolean expression (obtained recursively by joining the expressions for subtrees by the logic operators `and` or `or` depending on the label of the root), with the same Boolean value as the value of T. We identify an `and-or` tree with the expression it represents. The strategy for an `and-or` tree is the strategy for the associated probabilistic Boolean expression. *Probabilistic and-or tree resolution* (PAOTR) is the problem of finding an optimal strategy for an `and-or` tree.

A Boolean function that corresponds to an `and-or` tree (that is the function that for any setting of tests returns the Boolean value of the `and-or` tree) is called a *read-once function*, because it can be expressed by a formula in which each variable appears exactly once. Observe that any function that can be expressed by such a formula, corresponds to an `and-or` tree.

Since all leaf nodes of an `and-or` tree are labeled by distinct tests, we identify tests with leaf nodes. Therefore we say that a test x is a leaf of T meaning that x labels a leaf of T.

We require that tests of an `and-or` tree are independent, otherwise we could convert an arbitrary probabilistic Boolean expression (not necessary with read-once property) to an `and-or` tree with dependent tests by replacing any two occurrences of the same variable (or occurrences of the variable and its negation) by two separate variables that in any setting must have the same value (or must have opposite values).

A strategy to evaluate an `and-or` tree can be viewed as a search through the tree in order to find any subset of leaf nodes with required value that suffices to determine the value of the entire tree. There may be several such subsets. This model of search is called *satisficing search* by Simon and Kadane in [25], where the notion of PAOTR first appears.

An `and-or` tree is *strictly alternating* if there is no arc (v, w) in T such that v and w are both labeled `and` or both labeled `or`. For any `and-or` tree there exist an equivalent strictly alternating one. This is because for any $T_1 = \langle r_1, \Psi_1 \rangle$, $T_2 = \langle r_2, \Psi_2 \rangle$ such that $T_2 \in \Psi_1$ and the labels of r_1 and r_2 are identical (both `and` or both `or`), T_1 is `true` if and only if $T_1' = \langle r_1, \Psi_1 - \{T_2\} \cup \Psi_2 \rangle$ is `true`. Therefore, without loss of generality, we may assume that `and-or` trees are strictly alternating.

Observe also that for any `and-or` tree we can find an equivalent one in which all internal nodes have out-degree at least two. If $T_1 = \langle r_1, \{T_2\} \rangle$, then T_1 is `true` if and only if T_2 is `true`.

By *collapsing* we mean replacing an `and-or` tree by an equivalent one that is strictly alternating and in which all internal nodes have out-degree at least two.

We say that a subtree U of an `and-or` tree *resolves* its parent node if the value of U alone determines the value of the tree rooted at the parent node of U, namely if either U has value `true` and its parent node is `or` or U has value `false` and its parent node is `and`.

For a given tree T the *reduced* tree T' obtained by performing some tests from T is the tree obtained from T by removing all subtrees whose values have been determined by the values of the performed tests. The empty reduced tree is obtained when the value of the entire tree T has been determined.

We now present a few formulae to calculate the expected cost of a nonredundant strategy. For an internal node v of a strategy, let x_v denote the test that labels the node v. For a leaf node v, let x_v denote its label (`true` or `false`) and let $c(x_v) = 0$.

Let $P = (v_0, v_1, \ldots, v_k)$ with $k \geq 0$, be a path of a nonredundant strategy. We define the cost $c\,(P)$ of a path P as the sum of costs of the tests performed on P

$$c\,(P) = \sum_{i=0}^{k} c\,(x_{v_i}) \tag{1.5}$$

and the probability $p\,(P)$ of a path P as the product of the probabilities of the corresponding values of the tests performed on P

$$p\,(P) = \begin{cases} 1 & \text{if } k = 0, \\ \prod_{i=0}^{k-1} p_{v_i} & \text{if } k \geq 1, \end{cases} \tag{1.6}$$

where

$$p_{v_i} = \begin{cases} p(x_{v_i}) & \text{if the arc } (v_i, v_{i+1}) \text{ is labeled } \texttt{true}, \\ \bar{p}(x_{v_i}) & \text{if the arc } (v_i, v_{i+1}) \text{ is labeled } \texttt{false}. \end{cases} \tag{1.7}$$

The sum of the probabilities of all settings that correspond to one root-to-leaf path of a nonredundant strategy is equal to the probability of this path. Therefore we have the following expression for the expected cost of a nonredundant strategy S:

$$C(S) = \sum_{P \in RootToLeafPaths(S)} p(P)\,c(P), \tag{1.8}$$

where the sum is taken over all root-to-leaf paths of S.

It follows from (1.8) by induction on the number of nodes in a strategy that the following expression holds for a nonredundant strategy S:

$$C(S) = \sum_{v \in InternalNodes(S)} p(P_v) c(x_v), \tag{1.9}$$

where the sum is taken over all internal nodes of S and P_v is the path from the root of S to the node v.

Again by induction on the number of nodes in a strategy it follows that the following recursive formula computes the expected cost $C(\cdot)$:

For any leaf node v (labeled `true` or `false`) of a strategy,

$$C(v) = 0. \tag{1.10}$$

For any (sub)strategy S_w rooted at an internal node w,

$$C(S_w) = c(x_w) + p(x_w) \times C(S_{w+}) + \bar{p}(x_w) \times C(S_{w-}), \tag{1.11}$$

where x_w is the test that labels the root node w and S_{w+} and S_{w-} are the child subtrees of w whose root nodes the respective `true` and `false` arcs enter.

We will use the following notation to describe strategies: For strategies S_1, S_2 and a test x

$$x : +(S_1); -(S_2) \tag{1.12}$$

denotes the strategy whose root is labeled x and whose child subtrees of the root, entered by `true` and `false` arcs are respectively S_1 and S_2.

For a strategy S that has disjoint substrategies $S_1, \ldots, S_m$, $m \geq 1$, and for strategies $S'_1, \ldots, S'_m$

$$S(S_1 \triangleleft S'_1, \ldots, S_m \triangleleft S'_m) \tag{1.13}$$

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

If an `and-or` tree contains tests that have cost zero, then there is an optimal strategy for the tree that first performs in an arbitrary order all 0-cost tests. This means that the problem of calculating the optimal strategy for a tree that has some tests with the cost zero reduces to the problem of finding the optimal strategy for the `and-or` tree obtained after performing all 0-cost tests. For this reason, we assume from now on that all `and-or` tree tests have strictly positive costs.

For any `and-or` tree T and any test x from T there is a Boolean value v such that after determining that x has the value v the reduced tree still contains all tests other than x. From this it follows that that every correct strategy for an `and-or` tree performs in the worst case all the tests from the tree (that is it contains a root-to-leaf path whose nodes are labeled by all tests).

Decision problems are problems for which the only possible solutions are the answers "yes" or "no". The decision version of PBER is the following problem: Given a probabilistic Boolean expression e and a nonnegative real number B, is there a correct strategy for e with the expected cost at most B? The class P is the class of decision problems that are solvable in time polynomial in the size of their inputs. The class NP is the class of decision problems for which there exist a "certificate" that allows to verify the answer "yes" for problem's inputs, in time polynomial in the input's size. A problem from NP is NP-complete if any problem from NP can be reduced to it in polynomial time, which means that if an NP-complete problem can be solved in polynomial time, any problem from NP can. A problem Π is NP-hard, if there is an NP-complete problem that can be reduced to Π in polynomial time. See [7, 18] for introduction to the complexity theory.

1.3 Previous Work

1.3.1 NP-Hard Results

Our interest in PAOTR, the restriction of PBER to `and-or` trees, is motivated by the fact that for some more general classes of probabilistic Boolean expressions the problem is NP-hard.

First consider PBER for arbitrary probabilistic Boolean expressions. Let P be a Boolean formula and let e_P be the associated probabilistic Boolean expression, constructed by assigning an arbitrary probability and an arbitrary positive cost to each variable. Now consider the probabilistic Boolean expression $e = (x \texttt{ and } e_P)$, where x is a single test with positive cost. If P is unsatisfiable, then so is e and the optimal strategy for e has expected cost zero, since it suffices to return `false` without performing any test. If P is satisfiable, then any correct strategy for e has the expected cost greater than zero, since we will have to perform at least one test (e is satisfiable but may evaluate to `false` for example if x fails). It follows from this observation and from the NP-completeness of the satisfiability problem [7] that PBER is NP-hard.

A natural restriction of PBER is to "positive" expressions, namely those that do not include negated variables (thus are always satisfiable). Such expressions can be represented as `and-or` directed acyclic graphs; an `and-or` DAG is a DAG with only one source which is called the root, and such that each sink is a probabilistic test, each node that is not a sink is an `or`-node or an `and`-node. The value of an `or`-node (respectively `and`-node) is the value of logical `or` (respectively `and`) of its child nodes' values. The value of the root is the value of the entire expression.

PBER for `and-or` DAGs is still NP-hard as we now show. Our proof follows by introducing stochasticity into the construction presented in [22] for a different `and-or` structure problem.

The decision version of PBER for `and-or` DAG is the following problem: Given an `and-or` DAG D and a positive real number B, is there a strategy to evaluate D with expected cost at most B?

Theorem 1 *PBER for* `and-or` *DAGs is NP-hard, even if all test have unit costs.*

Proof: Consider the 3-SAT problem: Given a Boolean formula P that is the `and` of m clauses, each of which is the `or` of exactly three distinct literals (that is variables or their negations), is P satisfiable?

3-SAT is NP-complete [7]. We will show that 3-SAT can be polynomially reduced to PBER for `and-or` DAG.

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 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: nodes $C_1, C_2, ..., C_m$ correspond to the clauses of P while nodes $x_1, x_2, \ldots, x_n$ correspond to the variables from 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 the cost 1 and the success probability $q = \left(1 - \frac{1}{2n}\right)^{\frac{1}{2n+1}}$. These are all 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 1.3 presents an example of such a construction.

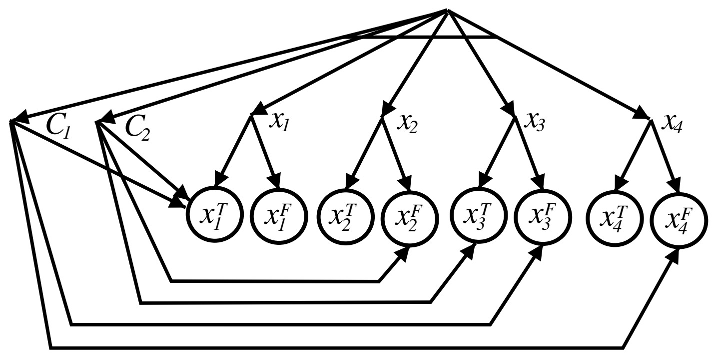

Figure 1.3: Illustration for the proof of Theorem 1. The DAG corresponding to the formula $P = (x_1 \texttt{ or } \neg x_3 \texttt{ or } \neg x_4) \texttt{ and } (x_1 \texttt{ or } \neg x_2 \texttt{ or } x_3)$.

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 correct strategy the single root-to-leaf path of the strategy that includes only `true` arcs will be called the *true path*. Notice that the true path of any correct strategy has to include at least n internal nodes, because we have to perform at least one child test of any x_i node to conclude that the value of D is `true`.

We will first prove the following claim:

Claim: A correct strategy S for D has the expected cost at most $n + \frac{1}{2}$ if and only if the true path of S contains exactly n internal nodes.

Proof of Claim: Let Q be the true path of a strategy S and let k be the number of the internal nodes of Q, $k \geq n$. Thus the cost of the path Q is k and the probability of Q is q^k. Notice that the cost of any other root-to-leaf path of S is at most $2n$ and at least 1.

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

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

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

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

$\square$

Now we can finish the proof, by means of the following equivalent statements, for which it is not difficult to see that $(i) \Leftrightarrow (ii)$, $(ii) \Leftrightarrow (iii)$, $(iii) \Leftrightarrow (iv)$, and, by the above Claim, $(iv) \Leftrightarrow (v)$:

(i) P is satisfiable.

(ii) There is a truth assignment σ for P such that for any clause C_j there is at least one literal that has the value `true` under σ.

(iii) There is a set W of tests from D, $|W| = n$, such that for any i one and only one of x_i^T and x_i^F belongs to W and any node C_j has at least one child in W.

(iv) There is a correct strategy for D whose true path contains exactly n internal nodes.

(v) There is a correct strategy for D with the expected cost at most $n + \frac{1}{2}$. $\square$

`And-or` trees constitute a proper subclass of `and-or` DAGs. It is unknown whether PBER for `and-or` trees (namely PAOTR) is in P. Previous to our work, polynomial-time algorithms have been known for special classes of `and-or` trees, namely for `and-or` trees with depth one or two (see Section 1.3.3) and for balanced `and-or` trees (see Section 1.3.4).

In this work we prove that PAOTR for the class of `and-or` trees with bounded number of internal nodes, as well as for some special classes of trees whose all tests have identical cost and identical probability, is in P.

1.3.2 Applications of And-Or Trees

Previously, various applications of PAOTR have been presented. They include: screening candidates for a certain position [6], selecting categories at a quiz show [6], gold mining [6, 8, 25], selecting problem-solving methods by simple control systems [1], performing inferences in expert systems [10, 26], selecting tests for medical diagnosis [11], and food testing [12].

`And-or` trees have also been studied as game-trees [14, 19, 20, 27]. A `min-max` tree is a directed rooted tree whose leaf nodes are assigned some values and internal nodes either are labeled `max` and evaluate to the maximum value of their child nodes, or are labeled `min` and evaluate to the minimum value of their child nodes. Observe that `and-or` trees are special cases of `min-max` trees restricted to values 0 (`false`) and 1 (`true`). In this context an `and-or` tree represents all possible plays of a two-player game. Consider an `or`-rooted tree. Call the first player OR, the second player AND. Any leaf node represents a terminal position in the game, that is win (`true`) or loss (`false`) of OR player. Any `or`-node (respectively `and`-node) represents a position in which it is the OR player's (respectively AND player's) turn to move. Each root-to-leaf path corresponds to one complete play of the game. Observe that the tree evaluates to `true` (win) if and only if the OR player can force a win.

1.3.3 Depth-First Strategies

Natural strategies to consider for evaluating `and-or` trees are those that evaluate child subtrees of a node until the value of the node is determined. We call such strategies *depth-first.*

Definition 6 *A strategy S for an* `and-or` *tree T is* ***depth-first*** *if for any subtree U of T and any root-to-leaf path P of S, whenever a test x from U is performed on P, no test from outside U is performed on P until the value of U is determined.*

```
Depth-First Strategy(and-or tree T)

If T is a single test x
    Perform x
    Return value of x
Else
    Take some order U_π1, U_π2, ..., U_πk of immediate subtrees of T
    i := 0
    Repeat
        i := i + 1
        ChildValue := Depth-First Strategy(U_πi)
     Until value of T is determined
     Return value of T
End Else
```

Figure 1.4: Depth-First Strategy.

Notice that in Figure 1.2 the strategy S_1 is depth-first, but the strategy S_2 is not: after test I is performed, S_2 "jumps" to test P before the value of the node i_3 is determined.

The pseudo-code of the algorithm represented by a depth-first strategy is presented in Figure 1.4. If the root of an `and-or` tree T is `or`, the value of T is determined when any of its immediate subtrees evaluates to `true` or when all of them evaluate to `false`. For a tree T rooted at an `and`-node, the strategy stops evaluating subtrees if any of them has value `false`.

In other words, for any internal node a depth-first strategy recursively evaluates child subtrees until one of them resolves its parent node or all are evaluated. It is now only necessary to find for each internal node the best order of evaluating its child subtrees.

For depth one `and-or` trees the best such order was described by Simon and Kadane [25]. Notice that for such trees any correct strategy is depth-first: it is sufficient to perform one test after another until any of them resolves its parent node or all are performed.

Theorem 2 *[25] Let T be a depth one* `and-or` *tree and let $x_1, x_2, \ldots, x_k$ be leaf nodes (tests) of T. An optimal strategy for T performs one test after another, until the value of T is determined, in the order $x_{\pi_1}, x_{\pi_2}, \ldots, x_{\pi_k}$ such that for $i < k$, $\frac{p^r(x_{\pi_i})}{c(x_{\pi_i})} \geq \frac{p^r(x_{\pi_{i+1}})}{c(x_{\pi_{i+1}})}$, where $p^r(x_i)$ is the probability that x_i resolves its parent node, namely*

$$p^r(x_i) = \begin{cases} p(x_i) & \text{if the root of } T \text{ is } \texttt{or}, \\ \bar{p}(x_i) & \text{if the root of } T \text{ is } \texttt{and}. \end{cases}$$

Proof Let $x_{\pi_1}, x_{\pi_2}, \ldots, x_{\pi_k}$ be the order of performing the tests by an optimal strategy S. Assume that this order violates the condition of the theorem, that is there is at least one index $l < k$ such that $\frac{p^r(x_{\pi_l})}{c(x_{\pi_l})} < \frac{p^r(x_{\pi_{l+1}})}{c(x_{\pi_{l+1}})}$. We will show that if we switch the tests x_{π_l} and $x_{\pi_{l+1}}$, the expected cost of the resulting strategy is not greater than the expected cost of S. Thus we can continue, if necessary, to switch tests until the resulting strategy, still optimal, fulfills the condition of the theorem.

Let P_i be the probability of a path from the root of S to the node labeled by x_i, that is $P_1 = 1$, and $P_i = \prod_{j=1}^{i-1} \left(1 - p^r\left(x_{\pi_j}\right)\right)$ for $i > 1$. The following expression holds for the expected cost of S:

$$C(S) = \sum_{i=1}^{k} P_i \, c\left(x_{\pi_i}\right).$$

Now let S' be the strategy obtained from S by switching the tests x_{π_l} and $x_{\pi_{l+1}}$. Then

$$C(S') - C(S) = P_l \left[p^r\left(x_{\pi_l}\right) c\left(x_{\pi_{l+1}}\right) - p^r\left(x_{\pi_{l+1}}\right) c\left(x_{\pi_l}\right)\right] \leq 0.$$

□

Natarajan [17] proposed an algorithm to find the best depth-first strategy for deeper trees. The algorithm, called DFA (Depth First Algorithm), calculates the best order of child subtrees for any internal node of a tree. It is described in Figure 1.5.

Notice that in any depth-first strategy, once one decides how to evaluate each child subtree of a given internal node, one can treat each such subtree as a single meta-test, whose effective cost is equal to the expected cost of evaluating the subtree and whose probability of success is equal to the probability that the subtree evaluates to `true`. For a (sub)tree T, DFA calculates the best order of immediate subtrees of T (using the rule given in Theorem 2), the expected cost $c_{DFA}(T)$ of the resulting depth-first strategy for T (as described in the proof of Theorem 2) and the probability $p(T)$ that T evaluates to `true`.

The time complexity of DFA is $O(N \ln b)$, where N is the total number of nodes in a tree and b is the maximum out-degree of internal nodes (because the time spent at a node is the time required to order that node's children).

The next theorem follows by induction on the depth of an `and-or` tree, with the base case provided by Theorem 2.

Theorem 3 *[12] For any* `and-or` *tree T, the depth-first strategy produced by DFA has minimum expected cost among all depth-first strategies for T.*

Greiner, Hayward and Molloy [12] proved that algorithm DFA produces an optimal strategy for depth-two `and-or` trees. Combined with Theorem 2 that gives us the following theorem.

Theorem 4 *[12] DFA produces an optimal strategy for* `and-or` *trees with depth one or two.*

DFA(and-or tree T)

If T is a single test x
    Return $(c(x),\ p(x))$
Else
    For each immediate subtree U_i of T, $i \le k$
        $(c_{DFA}(U_i),\ p(U_i)) :=$ DFA(U_i)
        $p^r(U_i) := \begin{cases} p(U_i) & \text{if the root of } T \text{ is or} \\ 1 - p(U_i) & \text{if the root of } T \text{ is and} \end{cases}$
    End For
    Find an order $U_{\pi_1}, U_{\pi_2}, \ldots, U_{\pi_k}$ of immediate subtrees of T
      such that $\frac{p^r(U_{\pi_i})}{c_{DFA}(U_{\pi_i})} \ge \frac{p^r(U_{\pi_{i+1}})}{c_{DFA}(U_{\pi_{i+1}})}$ for $i < k$
    $c_{DFA}(T) := c_{DFA}(U_{\pi_1}) + \sum_{i=2}^{k} c_{DFA}(U_{\pi_i}) \prod_{j=1}^{i-1} \left(1 - p^r\left(U_{\pi_j}\right)\right)$
    $p(T) := \begin{cases} 1 - \prod_{i=1}^{k} (1 - p(U_i)) & \text{if the root of } T \text{ is or} \\ \prod_{i=1}^{k} p(U_i) & \text{if the root of } T \text{ is and} \end{cases}$
    Return $(c_{DFA}(T),\ p(T))$
End Else

Figure 1.5: DFA (Depth First Algorithm).

However, algorithm DFA does not always produce an optimal strategy for deeper trees. The best depth-first strategy may be suboptimal for and-or trees with depth three, even if all tests have unit costs. For example, the unique optimal strategy for the tree T_r from Figure 1.1 is the strategy S_2 shown in Figure 1.2b, which is not depth-first.

Notice that for a unit-cost and-or tree with n tests, assuming that probabilities of tests are greater than zero and less than one, the minimum possible expected cost of a strategy is 1 and the maximum is n. Thus no strategy can have expected cost more than n times higher than the expected cost of the optimal strategy. Greiner, Hayward and Molloy showed that DFA may produce an extremely costly strategy:

Theorem 5 *[12] There are unit-cost* and-or *trees with n tests for which the best depth-first strategy has expected cost $\Theta\left(n^{1-o(1)}\right)$ times higher than the optimal resolution cost.*

1.3.4 Balanced And-Or Trees

And-or trees with a uniform structure have been studied extensively in the context of game-trees.

Definition 7

- *An* and-or *tree T is* ***parameter-uniform*** *if all tests of T have unit costs and the same probability of success.*

- *An* `and-or` *tree T is* ***balanced*** *if T is parameter-uniform, and all nodes of T with the same depth have equal out-degree.*

- *An* `and-or` *tree T is* ***uniform*** *if T is balanced, and all internal nodes of T have equal out-degree.*

Figure 1.6 presents two balanced `and-or` trees, among which the `and-or` tree T_1 is not uniform whereas the `and-or` tree T_2 is uniform.

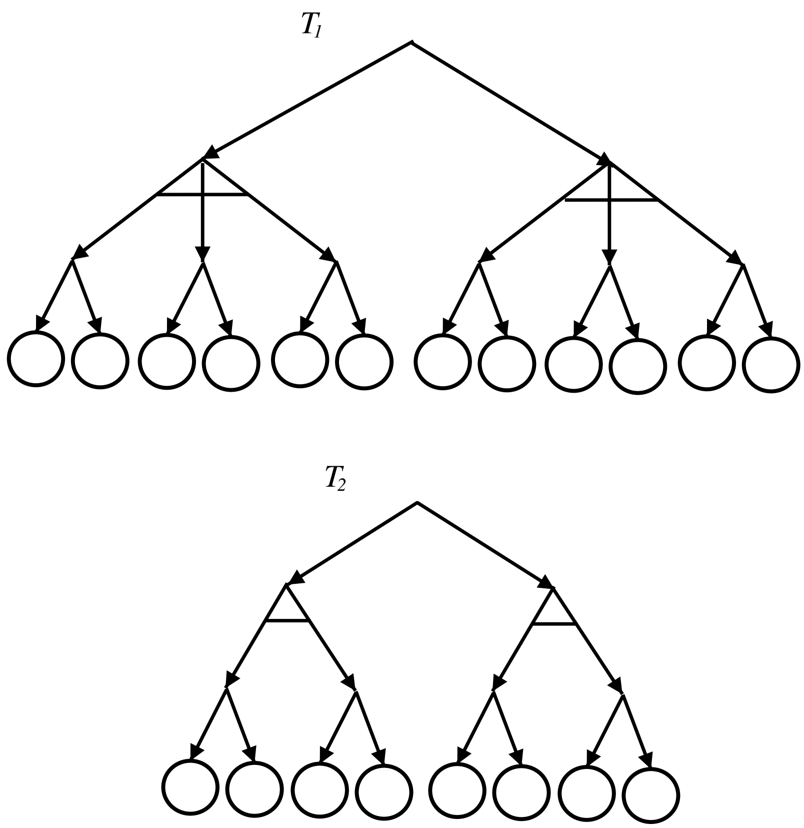

Figure 1.6: Balanced `and-or` trees T_1 and T_2. Each test has unit cost and success probability 0.7. T_1 is not uniform. T_2 is uniform.

Notice that for an internal node of a balanced tree, all child subtrees are indistinguishable. Thus any depth-first strategy for a balanced tree has the same expected cost.

Pearl studied asymptotic properties of uniform trees. The following are his results [19, 20] related to a specific value of success probability of tests that constitutes a significant threshold.

Let $U(d, b, p)$ denote an `or`-rooted uniform `and-or` tree, with depth equal to $d = 2k$, for integer $k \geq 0$, out-degree of the internal nodes equal to b and the success probability of tests equal to p.

Let ξ_b be the positive root of the equation $x^b + x - 1 = 0$. If the probability of success of tests is ξ_b, the tree $U(d, b, \xi_b)$ evaluates to `true` with the probability ξ_b, for any depth d. Moreover, in the limit as d goes to ∞, the probability that $U(d, b, p)$ has value `true` is 0 for all $p < \xi_b$ and 1 for all $p > \xi_b$.

The threshold value ξ_b is also significant with respect to the expected cost of a depth-first strategy. Let $c_{DFA}\left(U(d,b,p)\right)$ denote the expected cost of any depth-first strategy for $U(d,b,p)$. For test probabilities ξ_b and any depth d:

$$c_{DFA}\left(U(d,b,\xi_b)\right) = \left(\frac{\xi_b}{1-\xi_b}\right)^d > b^{\frac{d}{2}}.$$

Also, for any $p \neq \xi_v$ it holds that

$$\lim_{d\to\infty} \left[c_{DFA}\left(U(d,b,p)\right)\right]^{\frac{1}{d}} = b^{\frac{1}{2}}.$$

Notice that one always has to perform at least $b^{\frac{d}{2}}$ tests to determine the value of a tree $U(d,b,p)$. Thus the above result indicates that for test success probability different than ξ_b the depth-first strategy is asymptotically optimal for deep uniform trees.

Tarsi [27] proved the optimality of the depth-first strategy for finite depth, arbitrary probability and more general class of trees.

Theorem 6 *[27] For any balanced* `and-or` *tree, any depth-first strategy is optimal.*

In a sense, a depth-first strategy for an `and-or` tree is a special variant of the commonly used α-β pruning search for evaluating `min-max` game trees (the description of the search can be found in [21], Section 5.4). In fact, the optimality of a depth-first strategy for balanced `and-or` trees allows to establish the asymptotic optimality of α-β algorithm for continuous-valued `min-max` trees as the tree depth approaches infinity [27, 20].

Karp and Zhang [14] studied uniform trees $U(d,b,p)$ for the case when $p = \xi_b$. They showed that the cost of a depth-first strategy for such trees is likely to concentrate around the expected cost $\left(\frac{\xi_b}{1-\xi_b}\right)^d$ and that the standard deviation for the cost of the strategy is of the same order as the expected cost with respect to the depth d of a tree.

1.3.5 Linear Strategies

Notice that the strategy S_1 from Figure 1.2 always performs the tests in the relative order: P, F, I, K, where a test is skipped if its value is not needed after performing previous tests. We call such strategies *linear.*

Definition 8 *A strategy S for an* `and-or` *tree T is* ***linear*** *if there is a total order $<$ on the set of all tests from T such that for any two different tests x and y such that $y < x$, x is not performed before y on any root-to-leaf path of S.*

Linear strategies are of interest because they can be expressed very efficiently; it is not known whether optimal strategies can be expressed using space polynomial in the size of an `and-or` tree.

Depth-first strategies are linear. Greiner, Hayward and Molloy [12] showed that the best depth-first strategy for some trees may be arbitrarily worse then the best linear strategy; the ratio is as bad as the one given in Theorem 5. Moreover, the best linear strategy can be significantly worse than the optimal one:

Theorem 7 *[12] There are unit-cost* `and-or` *trees with n tests for which the best linear strategy has the expected cost $\Theta\left(n^{\frac{1}{3}-o(1)}\right)$ times higher than the optimal resolution cost.*

1.3.6 Preconditioned Probabilistic Boolean Expressions

There is a more general framework in which expressions over probabilistic tests and their resolution have been studied by many researchers. In this framework, for each probabilistic test we are given not only non-negative cost and probability of success but also preconditions in terms of other tests that need to be performed and return required values before the given test can be queried.

Some such expressions can be regarded as `and-or` trees with costs and probabilities associated with internal nodes as well as leaf nodes. For example, consider the situation in the recruiting process described in Section 1.1 when the fitness test, the IQ test and the knowledge test may be administrated only if some additional single screening test has been performed and passed by the candidate. In this case we may consider the node i_2 of Figure 1.1 as a probabilistic test, and all tests from the subtree rooted at the node i_2 may be performed only after this screening test has succeeded.

By a *preconditioned and-or directed acyclic graph* we mean a directed acyclic graph with only one source, called the root node, and such that each node of the graph is a distinct, independent probabilistic test with a given non-negative cost and success probability. Each node that is not a sink is either an `or`-node or an `and`-node. Any `or`-node and `and`-node is associated with a "required value" (`true` or `false`): its child tests can be performed only after the test itself has been queried and returned this required value. The value of a node that has out-degree zero is just the output of the test. If the output of the test that is an `or`-node (respectively `and`-node) is its required value, then the node evaluates to the logical `or` (respectively `and`) of its child nodes' values, otherwise the value of the node is the output of the test. The value of a preconditioned `and-or` DAG is the value of its root node.

In our example with the single screening test associated with the node i_2 in the tree T_r in Figure 1.1, the required value of that node is `true`. If the screening test is passed by the candidate, the value of i_2 is the logical `or` of the values of the nodes i_3 and F. If the candidate fails this test, the value of i_2 is `false`.

A preconditioned `and-or` DAG that is a directed rooted tree is called a *preconditioned and-or tree.*

The optimal strategy for a preconditioned DAG is defined in the same way as for a probabilistic Boolean expression, with the restriction that on any root-to-leaf path of a correct strategy, the precedence constraints of tests must be fulfilled.

Notice that a directed path in a preconditioned `and-or` DAG does not necessarily alternate between `or`-nodes and `and`-nodes: we cannot collapse a preconditioned `and-or` DAG that contains an arc whose ends are both `or` or `and`. By a *preconditioned* `or`*-DAG* (respectively *preconditioned* `or`*-tree*) we mean a preconditioned `and-or` DAG (respectively tree) with no `and`-node.

Positive Boolean expressions (described in Section 1.3.1) are special cases of preconditioned `and-or` DAGs, and `and-or` trees are special cases of preconditioned `and-or` trees, namely such that all tests associated with `or`-nodes and `and`-nodes have the cost zero, the required value `true`, and the probability of success one (thus one can "reach" any sink node on zero cost and with probability one). Thus the negative results we stated so far hold also for preconditioned `and-or` DAGs/trees. In particular, it is NP-hard to find an optimal strategy for general preconditioned `and-or` DAGs, and the best depth-first or linear strategy may perform very badly on some preconditioned and-or trees.

Garey [6] and Simon and Kadane [25] studied preconditioned or-DAGs with required values false for any or-node (that means that whenever any test succeeds, the entire expression evaluates to true). Garey [6] proposed an $O(n^2)$ algorithm to find an optimal strategy for trees of this type, where n is the number of tests. Simon and Kadane [25] extended this approach to deal with directed acyclic graphs. The algorithm identifies so-called "maximal indivisible blocks", that is sets of tests that are performed together by an optimal strategy.

This approach was also used by Smith [26], who provided a polynomial-time algorithm to find an optimal strategy for preconditioned or-trees with required value true for any or-node. The algorithm is described in detail in Section 4.1.

Greiner [10] showed that finding an optimal strategy for a preconditioned or-DAG with required values true for any or-node is NP-complete, even if the probability of success of any or-node is one.

1.3.7 Estimations of Success Probabilities

An important difficulty related to the success probabilities of tests arises in real life problems. Though the assumption that one knows the costs of the tests is a reasonable one, the exact probability distribution of tests is usually not known. The recruiting company from our example in Section 1.1 may know exactly how much it has to pay for each test, but the probabilities of success can only be estimated using data from previous recruiting. Thus the optimal strategy calculated for the estimated probabilities may not be optimal for the actual, unknown probability distribution.

Barnett [1] considered and-or trees with only two tests. For such trees he studied how the expected cost of a strategy is sensitive to the approximation of tests' parameters. His work indicates that the increase in the cost caused by a suboptimal strategy, selected for the estimates instead of the exact values of the parameters, is bounded by the accuracy of these estimates. Thus reasonable approximations will lead to reasonably good strategies.

Greiner and Orponen [11] addressed the problem of finding sufficiently good probability estimations for tests of preconditioned and-or trees within the probably approximately correct (PAC) model [28]. If the success probabilities of tests are to be estimated using the outputs of the tests for several trials, the authors showed how many trials are required to get such estimates that the optimal strategy calculated for them is with high confidence approximately optimal for actual probability values.

First let us concentrate on and-or trees. Assume that a tree contains n tests and the total cost of all tests is C. If for each test the success probability has been estimated using $\left\lceil 2\left(\frac{nC}{\varepsilon}\right)^2 \ln\frac{2n}{\delta}\right\rceil$ results of performing the test, then the optimal strategy for these estimations has with probability $1-\delta$ the expected cost within ε of the expected cost of the optimal strategy for the actual probability distribution. This required number of results can be collected while performing some (suboptimal) strategies; since we always can use a strategy starting with any given test, we are able to obtain the required number of results in time polynomial in n, C, $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$.

The situation is more complicated for preconditioned and-or trees, because some tests can only be performed if other tests have already returned required values, so no strategy can assure performing some tests. Authors proposed for such trees a polynomial-time algorithm to find estimations of tests' probabilities accurate enough so that the resulting best strategy is probably approximately optimal.

1.3.8 Non-Stochastic And-Or Trees

`And-or` tree resolution has also been studied in frameworks in which values of tests are not *probabilistic* variables. Since each deterministic strategy has to perform in the worst case all tests from the tree, so the simple cost of a strategy under the worst setting of tests is of no use as a performance measure.

A purely non-stochastic model is proposed by Charikar et al. [2]. For any setting of tests of a given `and-or` tree there exists the "proof" of the value of the tree, that is a set of tests such that the partial setting restricted to this set decides about the value of the tree. The cost of such a proof is the sum of costs of the tests from the set. The *performance ratio* of a strategy under a given setting is the ratio of the cost of the strategy under this setting to the minimum cost of a proof of the tree's value under this setting. For example, consider the `and-or` tree shown in Figure 1.1. The value of the tree for the setting $\sigma = (F-, I+, P-, K-)$ is `false`. The sets $\{P\}$ and $\{F, K\}$ are proofs of this value. The minimum cost proof is the set $\{P\}$ with the cost equal to 1. The cost of the strategy S_2, shown in Figure 1.2, under setting σ is equal to 3. Thus the performance ratio of S_2 under setting σ is equal to $\frac{3}{1} = 3$. The *competitive ratio* of a strategy is the maximum of the performance ratio over all settings of tests. The optimal strategy for an `and-or` tree is the one that minimizes the competitive ratio.

The authors provide an efficient algorithm to find an optimal strategy for `and-or` trees. The algorithm relies on discovered functions $f_0^T(c)$ and $f_1^T(c)$ which are lower bounds on the cost that any strategy for an `and-or` tree T has to pay in the worst case in order to find a proof with cost c of the value `true` and `false` respectively. These lower bound functions are used by the algorithm to balance for each internal node the cost spent while performing the tests from each of its child subtrees. The algorithm runs in time that is polynomial in the number of the nodes in the tree and in the sum of the costs of all tests.

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 is allowed to perform a random experiment and use the output of the experiment to decide about a test to perform. Formally, it is specified by a set of deterministic strategies and a probability distribution on this set. For a given setting of tests of an `and-or` tree, the cost of a randomized strategy is the expected cost of using the strategy under this setting (over all deterministic strategies). The worst case cost of a randomized strategy is the maximum cost of the strategy over all settings of tests. A correct randomized strategy is optimal if it has the lowest worst case cost over all correct randomized strategies for a given `and-or` tree.

In *randomized depth-first strategy* the deterministic strategies with non-zero probability are all depth-first: to evaluate a (sub)tree T the strategy evaluates immediate subtrees of T until the value of T is determined, and the next subtree to evaluate is selected at random.

Saks and Wigderson [23] showed that the randomized depth-first strategy is optimal for uniform `and-or` trees. For uniform binary `and-or` trees with n tests this strategy has the worst case cost $\Theta\left(n^{0.753\ldots}\right)$; recall that the worst case cost of any deterministic strategy is n.

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 [13] 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 always correct (randomized algorithms that never err are called *Las Vegas algorithms*). We can also consider Monte Carlo strategies (*Monte Carlo algorithms* are randomized algorithms that may err with some non-zero probability). By using the best Monte Carlo strategy instead of a Las Vegas strategy one can never increase the worst case cost. Santha [24] proved that for any unit-cost Boolean expression one can transform a Las Vegas strategy into a Monte Carlo strategy with a slightly lower (by a factor linear in the error probability) worst case cost, however, for unit-cost `and-or` trees Monte Carlo strategies cannot achieve any better improvement than this linear one.

Chapter 2

Optimal Order of Sibling Tests

2.1 Siblings and Twins Lemma

In the search for an optimal strategy one is led to the problem of efficiently selecting the first test to perform. A polynomial-time algorithm to find the first test of an optimal strategy suffices to construct a polynomial-time algorithm to find an optimal strategy: one just needs to perform the first test, reduce the tree accordingly to its output and recurse. A simple approach to solving PAOTR is to try each test as the first one by recursive calculation of the best strategy starting with each test and then selecting the one with the minimum expected cost. For an `and-or` tree with n tests the running time of such an algorithm is in $O(n^n)$. In the search for properties that lead to efficient first test selection, we investigated local structures of `and-or` trees. This approach led us to discovering an optimal a-priori relative order of sibling test. The order yields a dynamic programming algorithm for finding an optimal strategy for `and-or` trees which we describe in Section 2.2. We also showed that some siblings tests can be performed together by an optimal strategy. We now present these results.

Definition 9 *For any test x of an* `and-or` *tree define the* ***R-ratio*** *as:*

$$R(x) = \frac{p^r(x)}{c(x)}, \tag{2.1}$$

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

$$p^r(x) = \begin{cases} p(x) & \textit{if the parent node of } x \textit{ is } \texttt{or}, \\ \bar{p}(x) & \textit{if the parent node of } x \textit{ is } \texttt{and}. \end{cases}$$

Notice that the R-ratio is the same ratio that defines the best order of sibling tests in a depth one `and-or` tree (Theorem 2).

Definition 10 *Tests x_1 and x_2 are* ***R-equivalent*** *if the parent node of x_1 and x_2 is the same and $R(x_1) = R(x_2)$. An* ***R-class*** *is an equivalence class with respect to the relation of being R-equivalent.*

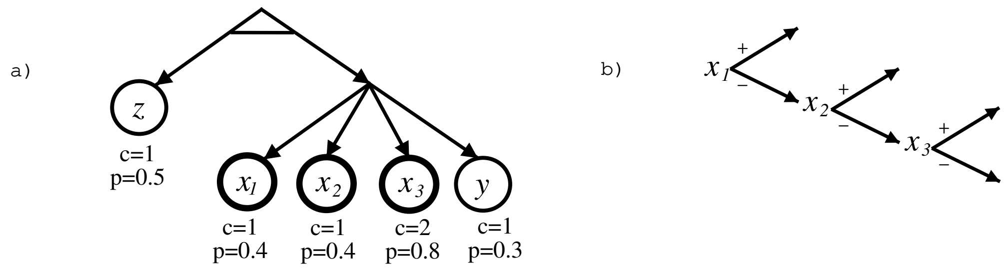

Figure 2.1: a) Sibling tests x_1, x_2 and x_3 have R-ratio 0.4. The set $W = \{x_1, x_2, x_3\}$ is an R-class. b) Part of a strategy contiguous on W performing the tests from W.

Definition 11 *For an R-class W a strategy S is* ***contiguous*** *on W if on any root-to-leaf path of S, whenever a test from W is performed, no test from outside W is performed until a test from W resolves its parent node or all tests from W are performed.*

Figure 2.1 shows an example of an R-class and illustrates the way of performing tests from one R-class by a strategy that is contiguous on this class.

Observation 8 *Let tests x and y be siblings in an* `and-or` *tree T. Let S_{xy} be a correct nonredundant strategy for T. If the parent node of x and y is* `or` *(respectively* `and`*) and $S_{xy} = x : +(S_1); -\Big(y : +(S_1); -(S_2)\Big)$ (respectively $S_{xy} = x : +\Big(y : +(S_2); -(S_1)\Big); -(S_1)$) for some substrategies S_1, S_2, and the strategy S_{yx} is obtained from S_{xy} by switching the labels x and y, then S_{yx} is nonredundant, correct for T, and*

i) if $R(y) > R(x)$ then S_{yx} has lower expected cost than S_{xy},
ii) if $R(y) = R(x)$ then S_{yx} has the same expected cost as S_{xy}.

Proof: The correctness and nonredundancy of S_{yx} follows from the correctness and nonredundancy of S_{xy}. We will prove the relations (i) and (ii) assuming that the parent node of x and y is an `or`-node. The proof for the other case is symmetric. For the expected cost of S_{xy} we have:

$$C(S_{xy}) = c(x) + \bar{p}(x)\,c(y) + \bar{p}(x)\,\bar{p}(y)\,C(S_2) + [1 - \bar{p}(x)\,\bar{p}(y)]\,C(S_1).$$

Using a similar expression for $C(S_{yx})$ we obtain

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

The observation follows immediately from the definition of $R(x)$ and $R(y)$. $\square$

The following observation follows from Observation 8ii).

Observation 9 *Let W be an R-class in an* `and-or` *tree T and let S be a correct, nonredundant strategy for T that is contiguous on W. Then any strategy obtain from S by changing order of performing tests from W has the same expected cost as S has.*

The following theorem specifies two conditions satisfied by an optimal strategy. The first condition deals with the best order of performing sibling tests. The second condition specifies the optimal way of performing sibling tests that are R-equivalent.

Theorem 10 *For any* `and-or` *tree there is an optimal strategy S such that both of the following conditions are satisfied:*

i) **(Siblings Lemma)** *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 S,*

ii) **(Twins Lemma)** *for any R-class W, S is contiguous on W.*

Proof: We prove Theorem 10 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 S be an optimal strategy for T. We may assume that it is nonredundant and that all substrategies of S are optimal for the corresponding reduced trees (because if it is not, we can replace it by such an optimal strategy). Let x be the first test performed by S. Assume that x is a child of an `or`-node (the proof for the other case is symmetric). Let S_{+x}, S_{-x} be the substrategies of S that are followed when respectively x is `true`, x is `false`. By induction, we may assume that S_{+x} and S_{-x} are contiguous on any R-class and preserve "the right order" of sibling tests (that is never perform a sibling test with lower R-ratio before its sibling test with higher R-ratio) of the corresponding reduced trees.

Now assume that S 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 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. By Observation 9 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 S_{-x}.

Now let $M \geq 1$ be the number of nodes of S_{-x} labeled by test y, let $S_{y_1}, S_{y_2}, \ldots,$ S_{y_M} be the subtrees of S_{-x} rooted at nodes labeled by y, and for $k = 1, 2, \ldots, M$, let S_{+y_k}, S_{-y_k} be the substrategies of S_{y_k} followed in the case when respectively y is `true`, y is `false`. Also let S^r denote the (possibly empty) part of S_{-x} that contains all nodes outside $S_{y_1}, S_{y_2}, \ldots, S_{y_M}$. See Figure 2.2a.

Consider the strategy $S(x \to y) = S_{-x}(S_{y_1} \triangleleft S_{x_1}, \ldots, S_{y_M} \triangleleft S_{x_M})$, where for $k = 1, 2, \ldots, M$, $S_{x_k} = x : +(S_{+y_k}) ; -(S_{y_k})$, shown in Figure 2.2c. In this strategy we query x just before y. This strategy is obviously nonredundant. To show that it is also correct, we need to check that for each leaf node L of $S(x \to y)$, the label of L (`true` or `false`) is the correct value of T for all setting of tests that correspond to the path P_L from the root of $S(x \to y)$ to L. This obviously holds if P_L contains a node labeled by test y, since in S there is the root-to-leaf path that 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 node labeled by x and the arc labeled `true` that leaves this node (because after x succeeds, we do exactly the same that we do if x fails

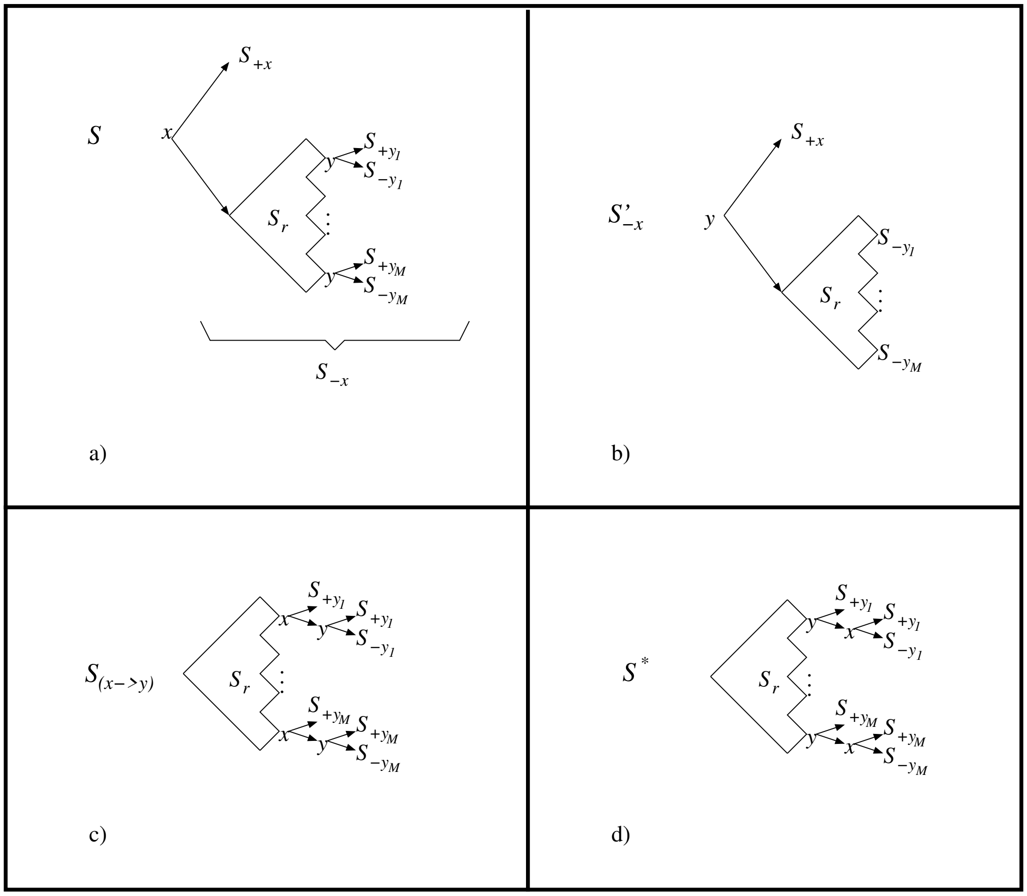

Figure 2.2: Illustration for the proof of Theorem 10. For any node the up (respectively down) arc denotes the `true` (respectively `false`) arc. (a) The optimal strategy S with substrategies S_{+x} and S_{-x}. (b) The strategy S'_{-x} that may replace the substrategy S_{-x}. (c) The optimal strategy $S\,(x \to y)$. (d) The optimal strategy S^* that fulfills the conditions of Theorem 10.

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 setting of tests that correspond to P_L. In S we follow the path identical to P_L after x fails. Thus for any σ_L in which x is `false` the label of the leaf L is correct. To see that it is also correct if x is `true`, consider any two setting of tests σ_1 and σ_2, that may differ only in the values of x and y, assume that in σ_1 x is `true`, in σ_2 x is `false`, y is `true`, and observe that the value of the tree T is the same for σ_1 and σ_2. Thus the correctness of the label of the leaf node of P_L in this case follows from the fact that in S we do not test y on the corresponding root-to-leaf path.

Now let S^* be the strategy obtained by switching the labels x and y of the neighbour nodes of the strategy $S\,(x \to y)$. See Figure 2.2d.

If the R-class containing x includes also other tests, then it has to contain y, so S^* is contiguous on this class. And S^* is contiguous on any R-class that does not include x; for the R-class including y (if $R(x) \neq R(y)$) it 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`, S^* preserves the right order of sibling tests of T.

Observation 8 implies that S^* is correct and does not have higher expected cost than $S(x \to y)$. Thus to complete the proof it is enough to show that $S(x \to y)$ is optimal.

Let $C(S_r)$ denote the expected cost of performing S_r, that is the sum of costs of tests labeling nodes of S_r, factored by the probabilities of paths from the root of S_{-x} to a given node (if S_r is empty, then $C(S_r) = 0$). For any k, let p_{y_k} be the probability of the path from the root of S_{-x} to the labeled by y root node of S_{y_k}.

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

$$C(S) = c(x) + p(x)\,C(S_{+x}) + \bar{p}(x)\,C(S_{-x}), \tag{2.2}$$

where

$$C(S_{-x}) = C(S_r) + \sum_{k=1}^{M} p_{y_k}\left[c(y) + p(y)\,C(S_{+y_k}) + \bar{p}(y)\,C(S_{-y_k})\right], \tag{2.3}$$

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

$$\begin{aligned} C(S(x \to y)) = C(S_r) + \sum_{k=1}^{M} \Big\{ p_{y_k} \Big[c(x) + p(x)\,C(S_{+y_k}) + \\ + \bar{p}(x)\left[c(y) + p(y)\,C(S_{+y_k}) + \bar{p}(y)\,C(S_{-y_k})\right]\Big]\Big\}. \end{aligned} \tag{2.4}$$

Now assume, by way of contradiction, that $S(x \to y)$ has the higher expected cost than S. Then using the notation $D = C(S_r) + \sum_{k=1}^{M} p_{y_k} C(S_{+y_k}) - C(S_{+x})$ 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 S_{-x}, so $P^r \geq 0$. That implies that $D > 0$, thus

$$\frac{p(x)}{c(x)} > \frac{P^r}{D}. \tag{2.5}$$

We will show that it follows from (2.5) that we can replace the substrategy S_{-x} of the original strategy by a substrategy with strictly lower expected cost, which contradicts the optimality of S_{-x}.

Consider the strategy $S'_{-x} = y : +(S_{+x}); -(S_{-x}(S_{y_1} \triangleleft S_{-y_1}, \ldots, S_{y_M} \triangleleft S_{-y_M}))$ shown in Figure 2.2b. Observe that S'_{-x} is nonredundant and correct for the reduced tree obtained from T when x is `false`. We have the following expression for the expected cost of S'_{-x}:

$$C(S'_{-x}) = c(y) + p(y)\,C(S_{+x}) + \bar{p}(y)\left[C(S_r) + \sum_{k=1}^{M} p_{y_k} C(S_{-y_k})\right]. \tag{2.6}$$

Using the same notation as previously we obtain

$$C(S_{-x}) - C(S'_{-x}) = p(y)\,D - P^r c(y). \tag{2.7}$$

But then from (2.5) and the fact that

$$\frac{p(y)}{c(y)} \geq \frac{p(x)}{c(x)}, \tag{2.8}$$

it follows that $C(S_{-x}) - C(S'_{-x}) > 0$, contradiction.

□

2.2 Dynamic Programming Algorithm for PAOTR

The ordering of sibling-tests described by the Siblings Lemma allows us to construct a dynamic programing algorithm for PAOTR that runs in time $O(d^2 n^d)$, where n is the number of leaves (tests) and d is the number of leaf-parents (that is nodes that are parents of tests) in the input `and-or` tree. Thus the running time of the algorithm is polynomial if the number of leaf-parents is bounded.

As shown in Section 1.1, for any `and-or` tree there is an equivalent one whose all internal nodes have out-degree at least two. For such trees the number of internal nodes is of the same order as the number of leaf-parents, as we show in the following observation.

Observation 11 *Let T be an* `and-or` *tree whose all internal nodes have out-degree at least two. Let d be the number of leaf-parents in T and let N be the number of all internal nodes in T. Then $N \geq d > \frac{N}{2}$.*

Proof: Let m be the number of all arcs leaving internal nodes that are not leaf-parents. Since out-degree of each internal node is at least 2, so $m \geq 2(N-d)$. Since each such arc enters a distinct internal node, different than the root of the tree, so $m < N$. It follows that $d > \frac{N}{2}$. □

Definition 12 *A **sibling-class** is a non-empty set of all leaf children of one internal node of an* `and-or` *tree.*

For example the `and-or` tree T_d shown in Figure 2.3a contains three sibling-classes: $L_1 = \{a_1, a_2\}$, $L_2 = \{b_1, b_2, b_3\}$ and $L_3 = \{c_1, c_2\}$. The number of sibling-classes of an `and-or` tree is the number of leaf-parents in the tree. 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 S is an optimal strategy for T that fulfills the conditions of Theorem 10. While evaluating T using S, we gradually reduce our `and-or` tree (namely 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 S. Assume that I still contains m_i tests from the sibling-class L_i. If $m_i < |L_i|$, then the remaining tests from L_i were already performed. Since we always query tests with higher R-ratio before sibling tests with lower R-ratio, the m_i tests still present in I must have the lowest R-ratios among all tests from L_i.

That means that for any d-tuplet $(m_1, m_2, ..., m_d)$, $0 \leq m_i \leq |L_i|$, there is only one reduced tree which 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-tuplet.

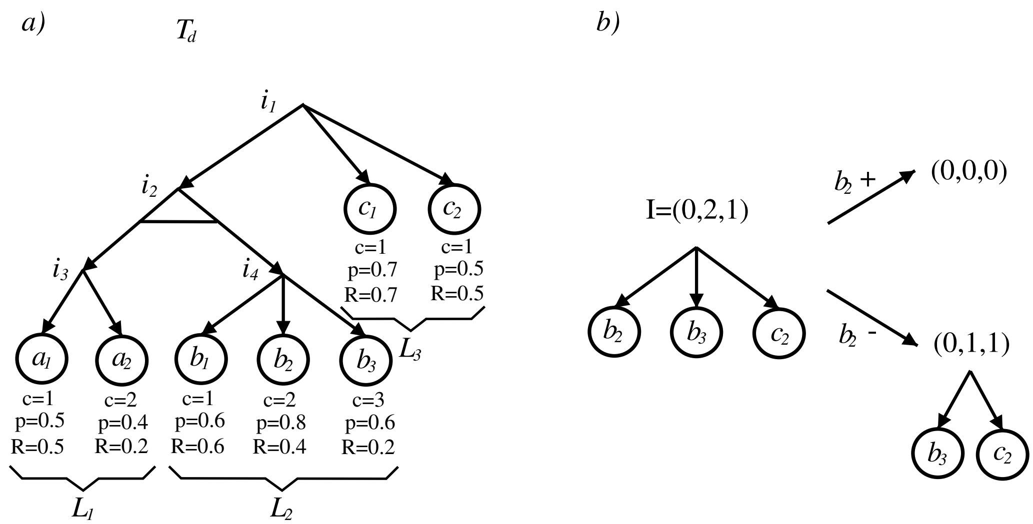

Figure 2.3: a) An `and-or` tree T_d with sibling classes L_1, L_2, L_3. For each test the cost, success probability and R-ratio values are denoted respectively by c, p and R. b) The reduced tree $I = (0, 2, 1)$ obtained from T_d and reduced trees obtained from I when b_2 succeeds and when b_2 fails.

For example consider the tree T_d shown in Figure 2.3a. T_d is represented by the 3-tuplet $(2, 3, 2)$ while the 3-tuplet $(0, 2, 1)$ corresponds to the reduced tree I shown in Figure 2.3b.

Thus 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 in $O\left(n^d\right)$ where $n = \sum_{i=1}^{d} |L_i|$ is the number of tests (leaf nodes) in T.

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 d sibling-classes.

Assume that we are given a structure of internal nodes of an `and-or` tree T such that for each internal node we know its parent node as well as its internal child nodes, and each leaf-parent is associated with its sibling-class. Assume also that we are given a reduced tree I obtained from T encoded by a d-tuplet. We shall now discuss how for some sibling class L one can calculate the d-tuplets I_L^+ and I_L^- corresponding to the reduced trees obtained from I when the test with maximum R-ratio from L in I respectively succeeds and fails.

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 I is one, then x_L is the only test in the tree and I_L^+ as well as I_L^- is the empty tree.

Otherwise we find the parent node of x_L in collapsed I. To do this we firstly 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 tests' number 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 w on the path from the root of T to v such that w has the same label (`or`/`and`) as v and w is the root of T, or w is a child node of the root of T, or the subtree rooted at the parent node of w contains at least one sibling-class with non-zero number of tests outside the subtree rooted at w.

Given the parent node w of x_L in I we can easily modify the d-tuplet I in order to obtain I_L^+ or I_L^-. If x_L resolves its parent w, the required modification is setting to zero the number of tests for each sibling-class inside the subtree rooted at w. Otherwise, the modification is done by decrementing the number of tests from the sibling-class L by one.

Notice that the operation of finding the parent w 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, thus also linear in d.

Consider as an example the reduced tree $I = (0, 2, 1)$ obtained from the tree T_d from Figure 2.3a. In I the test x_{L_2} with maximum R-ratio in L_2 is b_2. We want to find $I_{L_2}^+$ and $I_{L_2}^-$. Using the algorithm described above we first find the node v which is i_4. Then we find the parent node w which is i_1. The parent node i_1 is `or`. If x_{L_2} 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 i_1. Thus $I_{L_2}^+ = (0, 0, 0)$. If x_{L_2} fails, we just need to decrement the number of tests in L_2 by one: $I_{L_2}^- = (0, 1, 1)$. See Figure 2.3b.

We shall now describe Dynamic Programing Algorithm (DPA) for PAOTR. While performing the algorithm we enumerate all possible $(|L_1| + 1) \times (|L_2| + 1) \times \ldots \times (|L_d| + 1)$ reduced trees and identify them with d-tuplets. In other words we identify each reduced tree with one entry in a d-dimensional matrix of the 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 tree I we 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.

Once these attributes are stored for each reduced tree, an optimal strategy for the input tree T is encoded. The strategy starts with performing the test `FirstTest`$[T]$ and then depending on the value of this test either `TrueArc` $[T]$ or `FalseArc`$[T]$ is followed; the tree pointed by it is the reduced tree that needs to be evaluated at this point. The procedure is carried on until the empty tree is reached: if it is reached by a `TrueArc`, the value of the tree is `true`, otherwise its value is `false`.

The algorithm deals with reduced trees in the order of the number of tests, starting with the empty tree.

Dynamic Programing Algorithm for PAOTR is presented in Figure 2.4. The input `and-or` tree is strictly alternating and such that each internal node has out-degree at least two. The tree is encoded by the set of its internal nodes, such that for each internal node we know its parent node and all internal nodes that are its children. Moreover each leaf-parent is associated with the array of its leaf children.

```
        DPA (and-or tree T)
        Output:  optimal strategy for T

 (1)    For each sibling-class L of T
 (2)        order tests of L by ratio R
 (3)    End For
 (4)    For each tree I
 (5)        Cost[I]:=∞
 (6)        Calculate the number M of tests in I
 (7)        Add I to the list of trees with M tests
 (8)    End For
 (9)    Cost[empty tree]:=0
(10)    FirstTest[empty tree]:=NIL
(11)    For M = 1 to # of tests in T
(12)        For each tree I with M tests
(13)            For each sibling-class L of T that is not empty in I
(14)                x_L:=test from L in I with maximum R
(15)                I_L^+:=tree obtained from I if x_L succeeds
(16)                I_L^-:=tree obtained from I if x_L fails
(17)                C := c(x_L) + p(x_L) ×Cost[I_L^+] + p̄(x_L) × Cost[I_L^-]
(18)                If C <Cost[I]
(19)                    Cost[I] := C
(20)                    FirstTest[I] := x_L
(21)                    TrueArc[I] is pointed to I_L^+
(22)                    FalseArc[I] is pointed to I_L^-
(23)                End If
(24)            End For
(25)        End For
(26)    End For
```

Figure 2.4: Dynamic Programming Algorithm (DPA) for PAOTR.

Theorem 12 *DPA produces an optimal strategy for* and-or *trees. The time complexity of the algorithm is in* $O\left(d^2 n^d\right)$ *and the space complexity is in* $O\left(n^d\right)$*, where* n *is the total number of tests (leaf nodes) of a tree and* d *is the number of leaf-parents in a tree.*

Proof: The correctness of the algorithm follows from Theorem 10, as discussed above. Since as shown above the total number of reduced trees is in $O\left(n^d\right)$ and we store a constant amount of data for each reduced tree, the space complexity is in $O\left(n^d\right)$.

Lines (1)-(3) order tests inside each sibling class, so the time required to perform them is in $O(n \log n)$. Line (6) takes time $O(d)$ and is called once for each reduced

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_3	$(0,0,0)$	$(0,0,0)$
$(1,0,0)$	2	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_2	$(0,0,0)$	$(0,1,0)$
$(1,0,1)$	2	c_2	$(0,0,0)$	$(1,0,0)$
$(1,1,0)$	3.2	a_2	$(0,1,0)$	$(0,0,0)$
$(2,0,0)$	2	a_1	$(0,0,0)$	$(1,0,0)$

Table 2.1: Parameters of reduced trees obtained from the `and-or` tree T_d from Figure 2.3 with less than three tests.

tree thus time required to perform lines (4)-(8) is in $O(dn^d)$.

We will show that time required by lines (11)-(26) is in $O(d^2n^d)$ which ends the proof. Loop (13) has d iterations and is called once for each of $O\left(n^d\right)$ reduced trees. To complete the proof we need to show that time required by the operations inside this loop is in $O(d)$. 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 14). As we have shown before, we can calculate I_L^+ or I_L^- in time $O(d)$ (lines (15) and (16)). Also, since we identify trees with d-tuplets, we may find data for trees I_L^+ and I_L^- (line 17) in $O(d)$ time, as elements of a d-dimensional matrix. □

The corollary below follows immediately from the previous theorem.

Corollary 13 *Probabilistic and-or tree resolution for* `and-or` *trees with a bounded number of internal nodes is in P.*

As an example consider again the `and-or` tree T_d shown in Figure 2.3a. Assume that we already processed all reduced trees with less than three tests. The calculated parameters for each of these trees is given in Table 2.1. We now want to calculate the optimal strategy for the reduced tree $I = (0,2,1)$ with three tests; see Figure 2.3b. The sibling-class L_1 is empty in I. Now consider the sibling-class L_2. The test x_{L_2} with maximum R ratio from L_2 in I is the test b_2 and $I_{L_2}^+ = (0,0,0)$, $I_{L_2}^- = (0,1,1)$. Thus we now have

$$\begin{aligned} C &= c\left(b_2\right) + p\left(b_2\right) \cdot \textbf{Cost}\left[I_{L_2}^+\right] + \bar{p}\left(b_2\right) \cdot \ \textbf{Cost}\left[I_{L_2}^-\right] = \\ &= 2 + 0.8 \cdot 0 + 0.2 \cdot 2.5 = 2.5. \end{aligned} \tag{2.9}$$

Thus we set `Cost`$[I]$ to 2.5 and `FirstTest`$[I]$ to b_2, we point `TrueArc` $[I]$ to $(0,0,0)$ and `FalseArc`$[I]$ to $(0,1,1)$. Now we proceed to the sibling-class L_3. We have $x_{L_3} = c_2$ and $I_{L_3}^+ = (0,0,0)$, $I_{L_3}^- = (0,2,0)$. Thus

$$\begin{aligned} C &= c\left(c_2\right) + p\left(c_2\right) \cdot \textbf{Cost}\left[I_{L_3}^+\right] + \bar{p}\left(c_2\right) \cdot \ \textbf{Cost}\left[I_{L_3}^-\right] = \\ &= 1 + 0.5 \cdot 0 + 0.5 \cdot 2.6 = 2.3. \end{aligned} \tag{2.10}$$

Since this cost is lower than current $\texttt{Cost}[I]$, we set $\texttt{Cost}[I]$ to 2.3 and $\texttt{FirstTest}[I]$ to c_2, we point $\texttt{TrueArc}\ [I]$ to $(0,0,0)$ and $\texttt{FalseArc}[I]$ to $(0,2,0)$. These parameters, together with the parameters from Table 2.1, encode the optimal strategy for the reduced tree I. This strategy is presented as a binary tree in Figure 2.5.

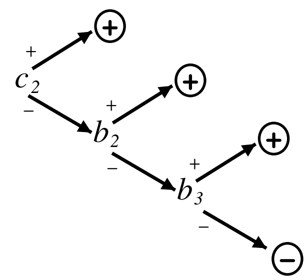

Figure 2.5: An optimal strategy for the tree I shown in Figure 2.3.

2.3 Simplifying And-Or Trees Using the Twins Lemma

The Twins Lemma 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 an effective cost and probability corresponding to performing all tests from the R-class. By Observation 9 the order of performing the tests from the R-class is arbitrary.

Simple calculations yield the parameters of such a meta-test.

Observation 14 *Let W be an R-class and let R be the R-ratio of the tests from W. In searching for an optimal strategy we can replace W by a single meta-test w with the following parameters:*

$$p(w) = \begin{cases} 1 - \prod_{x \in W} \bar{p}(x) & \textit{if tests from } W \textit{ have an } \texttt{or}\textit{-parent}, \\ \prod_{x \in W} p(x) & \textit{if tests from } W \textit{ have an } \texttt{and}\textit{-parent}, \end{cases} \tag{2.11}$$

$$c(w) = \begin{cases} \sum_{x \in W} c(x) & \textit{if } R = 0, \\ \frac{p(w)}{R} & \textit{if } R > 0 \textit{ and tests from } W \textit{ have an } \texttt{or}\textit{-parent}, \\ \frac{1-p(w)}{R} & \textit{if } R > 0 \textit{ and tests from } W \textit{ have an } \texttt{and}\textit{-parent}. \end{cases} \tag{2.12}$$

The following observation follows immediately from the previous one.

Observation 15 *The meta-test w replacing an R-class W has the same R-ratio as tests from W.*

The simplification described above allows us to prove that DFA produces an optimal strategy for depth three parameter-uniform `and-or` trees. Recall that an `and-or` tree is parameter-uniform if all tests have unit cost and the same success probability.

Observation 16 *Let T be a depth three* `and-or` *tree such that for each internal node v of depth two, all tests that are child nodes of v have the same R-ratio. Then DFA produces an optimal strategy for T.*

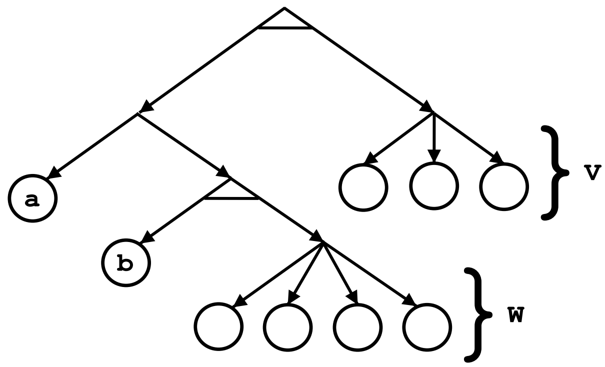

Figure 2.6: A parameter-uniform `and-or` tree T_u. Each test has cost one and probability of success 0.2. W and V denote depth one subtrees.

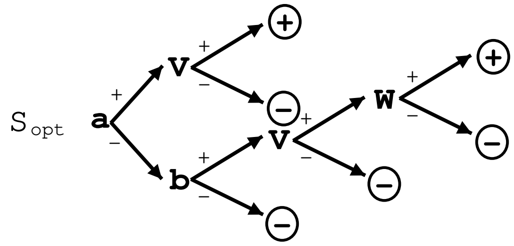

Figure 2.7: The unique optimal strategy S_{opt} for the `and-or` tree T_u where nodes labeled by W and V denote evaluation of the corresponding subtrees.

Proof: By Theorem 3 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 with depth two has only one child: a single meta-test. Thus T' collapses to depth two. By Theorem 4 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 Lemma is optimal for T. □

The following theorem follows immediately from Observation 16.

Theorem 17 *DFA produces an optimal strategy for depth three parameter-uniform* `and-or` *trees.*

However, this property does not always 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. For example the strategy S_{opt} in Figure 2.7 is the unique optimal strategy for T_u in Figure 2.6, but it is not depth-first.

2.4 Parameter-Uniform Ladders

In the previous section we have shown that PAOTR for parameter-uniform depth three `and-or` trees can be solved with DFA. By Theorem 6 we also know that an optimal strategy can be found efficiently for any parameter-uniform tree that is balanced.

In this section we present another particular subclass of parameter-uniform `and-or` trees for which the optimal strategy has a very simple description. PAOTR for this type of trees is in P.

First let us state two observations; the first of these is an unpublished result due to Greiner, Hayward, and Molloy.

Observation 18 *Let T be an* `and-or` *tree and let x be a child test of the root of T. If a strategy S after performing the first test, performs x regardless of the value of the first test, then there is a strategy starting with x whose expected cost is less than or equal to the expected cost of S.*

Proof: Let y be the first test performed by S. Assume that T is `or`-rooted; the proof for the other case is symmetric. Let S_+ be the substrategy followed when y is `true` and x is `false`, let S_- be the substrategy followed when y is `false` and x `false` (if x is `true`, the value of T is `true`). The strategy $S' = x : +(\texttt{true}); -(y : +(S_+); -(S_-))$ is nonredundant, correct for T, and

$$\begin{aligned} C(S') - C(S) &= c(x) + \bar{p}(x)c(y) + \bar{p}(x)p(y)C(S_+) + \bar{p}(x)\bar{p}(y)C(S_-) + \\ &\quad - [c(y) + c(x) + p(y)\bar{p}(x)C(S_+) + \bar{p}(y)\bar{p}(x)C(S_-)] = \\ &= -p(x)c(y) \leq 0. \end{aligned} \tag{2.13}$$

□

Observation 19 *Let T be an* `and-or` *tree whose root has a test child x. If the root of T is* `or` *(respectively* `and`*) and for any test y in T $p(y)/c(y) \leq p(x)/c(x)$ (respectively $\bar{p}(y)/c(y) \leq \bar{p}(x)/c(x)$), then there is an optimal strategy for T that starts with performing x.*

Proof: By induction on the number of the tests in T. The observation holds if T has only one test. Now assume that the observation holds for any tree which has fewer tests than T has.

Let S be an optimal strategy for T and assume that it starts with performing a test y different than x. By the Siblings Lemma and Twins Lemma we may assume

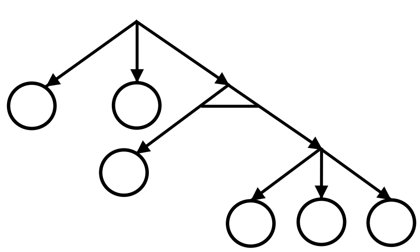

Figure 2.8: An example of an `and-or` ladder.

that S does not perform any sibling test of x before x, so y is not a child of the root of T. Since we may assume that each internal node of T has out-degree at least two, that means that the reduced trees obtained from T when y is `true` and y is `false` both contain x and have fewer tests than T. So by the inductive assumption, after testing y, S performs x, regardless of the value of y. But x is a child of the root of T, so by Observation 18 there is a strategy for T that has not higher expected cost than S and starts with performing x. □

The above observation generalizes an unpublished result of Omid Madani.

An *and-or ladder* is an `and-or` tree such that each internal node is a parent of at most one internal node. Figure 2.8 shows an example of an `and-or` ladder. From the Observation 19 follows immediately that if all tests of an `and-or` ladder are identical, there is a very simple way of finding an optimal strategy which we now formalize.

Observation 20 *For any parameter-uniform* `and-or` *ladder T there is an optimal linear strategy S that performs tests "from the top to the bottom", that is that performs first, in an arbitrary order, tests of depth one, and as long as the value of T is not determined, after performing in an arbitrary order tests of depth k, performs in an arbitrary order tests of depth $k+1$.*

2.5 Reduction to Unit-Cost PAOTR

In Section 2.3 we explained how an `and-or` tree can be simplified by replacing an R-class by a single meta-test. On the other hand, for any test with cost greater than one, we may consider replacing the test with a depth one subtree having identical (same probability) unit-cost tests. Notice that if we could find for each test from the original tree an appropriate replacement collection of identical unit cost tests such that, for each original test, the replacement subtree probability and evaluation cost would equal the original leaf probability and cost, then by the Twins Lemma an optimal strategy for the new unit-cost tree would have the same expected cost as an optimal strategy for the original tree. Such a replacement subtree is shown schematically on Figure 2.9.

As we shall show, while it is not always possible to find a unit-cost tree that yields such an exact correspondence, by keeping sufficient precision it is possible to obtain by such replacements a unit-cost tree whose optimal resolution cost is arbitrarily close to the optimal resolution cost of the original tree. We now describe such a reduction in detail.

We start by considering depth one subtrees with identical leaf nodes in a unit-cost tree. Let u be the chosen unit of cost, that is the cost of each test in the tree. Let A be a depth one subtree with k identical leaf nodes and let p, $\bar{p}$ be the respective success, failure probability of each test from A. We will denote the optimal resolution cost of A by $C(A)$, the probability that A evaluates to `true` by $p(A)$, and the probability that A evaluates to `false` by $\bar{p}(A)$.

If A is `and`-rooted then

$$p(A) = p^k, \tag{2.14}$$

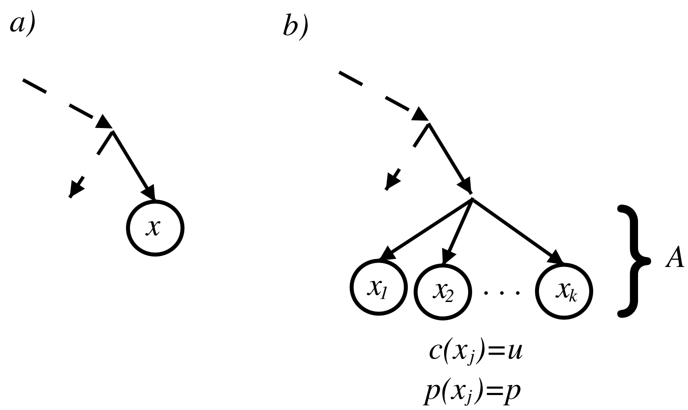

Figure 2.9: a) A test x in an `and-or` tree. b) A subtree A replacing the test x. There is k tests in A, each with the same cost u and the same success probability p.

$$C(A) = \begin{cases} u\frac{1-p^k}{1-p} & \text{if } p \neq 1, \\ uk & \text{if } p = 1. \end{cases} \tag{2.15}$$

If A is `or`-rooted then

$$\bar{p}(A) = \bar{p}^k, \tag{2.16}$$

$$C(A) = \begin{cases} u\frac{1-\bar{p}^k}{1-\bar{p}} & \text{if } p \neq 0, \\ uk & \text{if } p = 0. \end{cases} \tag{2.17}$$

Now assume that we want to use the subtree A to replace a test x with the cost c_x and the success probability p_x. The cost of x expressed in units u is c_x/u. We require that $c_x/u > 1$. which means that x is not just a unit-cost test (in which case there is no need for replacing it).

We would like to obtain the exact correspondence between the test x and the subtree A, so we require that $p(A) = p_x$ and $C(A) = c_x$. Observe, that it is not possible if $p_x = 0$ and A is `and`-rooted because we would need $p(A) = 0$, which requires $p = 0$, but then we have $C(A) = u$ (meaning that it is always enough to perform just one test to evaluate A to `false`). Similarly, it is impossible to replace x by an `or`-rooted subtree if $p_x = 1$.

Simple calculations lead to the following expressions for k and p, yielding $p(A) = p_x$ and $C(A) = c_x$.

For an `and`-rooted subtree A

$$k = \begin{cases} \frac{\ln(p_x)}{\ln\left(1-\frac{1-p_x}{c_x/u}\right)} & \text{if } 0 < p_x < 1, \\ c_x/u & \text{if } p_x = 1, \end{cases} \tag{2.18}$$

$$p = {p_x}^{1/k}, \tag{2.19}$$

whereas for an `or`-rooted subtree A

$$k = \begin{cases} \frac{\ln(\bar{p}_x)}{\ln\left(1-\frac{1-\bar{p}_x}{c_x/u}\right)} & \text{if } 0 < p_x < 1, \\ c_x/u & \text{if } p_x = 0, \end{cases} \tag{2.20}$$

$$\bar{p} = {\bar{p}_x}^{1/k}. \tag{2.21}$$

Before we address the difficulties related to the above conditions, let us investigate the dependency of the number k on p_x and c_x/u.

As we shall show at the end of this section in Lemma 23, for an `and`-rooted subtree k monotonically decreases from ∞ to c_x/u as p_x increases from 0 to 1. By symmetry, as p_x increases from 0 to 1, k for an `or`-rooted subtree increases from c_x/u to ∞, and the value of k for $p_x = 1/2$ is the same for both kinds of subtrees. Thus we can minimize k by replacing x by an `and`-rooted subtree if $p_x \geq 1/2$ and by an `or`-rooted subtree if $p_x < 1/2$. Then for a given c_x we have the maximum k when $p_x = 1/2$.

We shall show (Lemma 24) that if $p_x = 1/2$ then $k \leq 2\ln 2\frac{c_x}{u} < 1.4\frac{c_x}{u}$. Therefore, given our way of selecting the root of the subtree, for an arbitrary probability p_x it holds that

$$k < 1.4\frac{c_x}{u}. \tag{2.22}$$

If the conditions (2.18) and (2.19) for an `and`-rooted subtree or (2.20) and (2.21) for an `or`-rooted subtree are satisfied, then the subtree has exactly the same cost and probability of being `true` as the original test x. There are however two obvious obstacles. Firstly, since k is the number of leaf nodes of A, k must be an integer. Thus we have to round the value of k given by (2.18) or (2.20) to some integer k'. Moreover, since the values of tests' probabilities (${p_x}^{1/k'}$ or $1 - {\bar{p}_x}^{1/k'}$) are not always rational then assuming that we want to store and use these values as input to some algorithms, we need to round them using finite number of digits.

Therefore the exact correspondence between a test x and a replacement subtree A is not always possible to achieve. But by choosing the cost unit u small enough and by keeping enough precision in probabilities of new tests, we can reach an arbitrary small error in the optimal resolution cost for the new tree, as described in the following theorem.

Theorem 21 *Given an* `and-or` *tree T^* with the optimal resolution cost C^* and a real number r, $0 < r < 1$, there is a unit-cost* `and-or` *tree T_u with optimal resolution cost C_u such that $C^*(1-r) \leq C_u \leq C^*(1+r)$.*

For n being the number of tests in T, c_{max} the maximum cost over all tests in T and B the maximum number of digits used to represent a probability over all tests in T, the construction of T_u runs in time polynomial in n, $1/r$, c_{max} and B.

Moreover, for each leaf-parent of T_u all its test children are identical. If instead of constructing all leaf nodes of T_u we rather keep for each leaf-parent of T_u the number of its test children and the (uniform) success probability of each test, then such a construction of T_u runs in time polynomial in n, $\log(1/r)$, $\log(c_{max})$ and B.

The theorem was stated by Omid Madani, with whom we collaborated to prove it.

We now describe the construction of the unit-cost tree T_u.

Let r be as stated in the theorem. For the original `and-or` tree T^* let n be the number of tests in T^*, and for each test x of T^* let $p_{x0} = min\,\{p_x, \bar{p}_x\}$.

In the unit-cost `and-or` tree T_u each test has cost $u \leq \frac{r}{3}c_{min}$, where c_{min} is the smallest cost of a test in T^*. We construct T_u by replacing each test x of T^* by a depth one subtree A with k' identical tests, such that

i) if $p_x \geq 1/2$, A is `and`-rooted, otherwise A is `or`-rooted,

ii) $k' = \lfloor k \rfloor$, where k is given by (2.18) if A is `and`-rooted, and by (2.20) if A is `or`-rooted,

iii) given $p' = {p_x}^{1/k'}$ for an `and`-rooted A and $\bar{p}' = {\bar{p}_x}^{1/k'}$ for an `or`-rooted A, the success probability $\tilde{p}$ of tests from an `and`-rooted A (respectively the failure probability $\tilde{\bar{p}}$ of tests from an `or`-rooted A) is obtained by rounding p' up to $\tilde{p}$ so that $p' \leq \tilde{p} \leq p'(1+\varepsilon)$ (respectively rounding $\bar{p}'$ up to $\tilde{\bar{p}}$ so that $\bar{p}' \leq \tilde{\bar{p}} \leq \bar{p}'(1+\varepsilon)$), for any ε satisfying $\varepsilon \leq \frac{p_{x0}r}{4nk'}$.

Notice that T_u is not necessarily strictly alternating since we may replace a test child of an `or`-node (respectively `and`-node) by a `or`-rooted subtree (respectively `and`-rooted subtree).

Since $k' \geq k - 1 \geq \frac{c_x}{u} - 1 \geq \frac{c_x 3}{c_{min} r} - 1 \geq 2$, each test of T^* is replaced by a depth one subtree with at least two tests. Thus for each leaf-parent of T_u, all its tests children have the same probability. Now Theorem 21 follows immediately from the theorem below.

Theorem 22 *Let T^* be an arbitrary* `and-or` *tree, C^* the optimal resolution cost of T^* and $0 < r < 1$. If T_u is constructed as described above, then for the optimal resolution cost C_u of T_u we have $C^*(1-r) \leq C_u \leq C^*(1+r)$, and for each depth one subtree replacing a test x, the number of leaf nodes k' is $O\left(\frac{c_x}{r}\right)$, and the number of significant digits required to obtain the probability precision defined in (iii) is $O\left(\log\left(\frac{nc_x}{p_{x0}r}\right)\right)$.*

Proof: We will begin by proving the order of k' and of the number of required significant digits of probabilities.

Since $k' = \lfloor k \rfloor$, then from (2.22) and the fact that $1/u = O(1/r)$ it follows that $k' = O\left(\frac{c_x}{r}\right)$.

For an `and`-rooted subtree, we round up the probability $p' = {p_x}^{1/k'}$ to $\tilde{p}$ and require that $\tilde{p} \leq p'(1+\varepsilon) = p' + p'\varepsilon$. So we need to keep a number of significant digits in order of $\log\left(\frac{1}{p'\varepsilon}\right)$, or

$$O\left(\log\left(\frac{1}{p'}\right) + \log\left(\frac{1}{\varepsilon}\right)\right) = O\left(\frac{1}{k'}\log\left(\frac{1}{p_x}\right) + \log\left(\frac{4nk'}{p_{x0}r}\right)\right) =$$
$$= O\left(\log\left(\frac{nc_x}{p_{x0}r^2}\right)\right) = O\left(\log\left(\frac{nc_x}{p_{x0}r}\right)\right), \tag{2.23}$$

where we used the fact that $O\left(\log\left(1/p_x\right)\right) = 1$, since for an `and`-rooted subtree $p_x \geq 1/2$. The argument for an `or`-rooted subtree is symmetric.

We will now prove that the optimal resolution cost of T_u is approximately the same as of T^*.

Consider a strategy S^* for T^* and the corresponding strategy S_u for T_u, that is the strategy that in place of each test performed by S^* evaluates the corresponding depth 1 subtree of T_u. We will show that for the expected costs of S^* and S_u the following condition is satisfied:

$$C\left(S^*\right)(1-r) \leq C\left(S_u\right) \leq C\left(S^*\right)(1+r), \tag{2.24}$$

from which, by the Twins Lemma, the desired relation between the optimal resolution costs follows. For the expected cost of S^* we have:

$$C\left(S^*\right) = \sum_{x \in Tests(T^*)} P_p(x) c_x, \tag{2.25}$$

where the sum is taken over all tests of the tree T^* and $P_p(x)$ is the sum of probabilities of all paths from the root of S^* to nodes labeled by the test x.

If $P_p(A)$ is the sum of probabilities of all paths from the root of S_u to nodes labeled by the first test of a replacement subtree A, then

$$C\left(S_u\right) = \sum_{A \in RepSubtrees(T_u)} P_p(A) C(A), \tag{2.26}$$

where the sum is taken over all replacement subtrees in T_u.

Now assume that after the reduction, for any subtree A of T_u replacing a test x, the cost of evaluating A is perturbed at most by δ_c in comparison with c_x and the values of the probability that A is `true` and `false` are perturbed at most by δ_p in comparison with p_x and $\bar{p}_x$, respectively, meaning:

$$c_x \left(1 - \delta_c\right) \leq \; C(A) \; \leq c_x \left(1 + \delta_c\right), \tag{2.27}$$

$$p_x \left(1 - \delta_p\right) \leq \; p(A) \; \leq p_x \left(1 + \delta_p\right), \tag{2.28}$$

$$\bar{p}_x \left(1 - \delta_p\right) \leq \; \bar{p}(A) \; \leq \bar{p}_x \left(1 + \delta_p\right). \tag{2.29}$$

Since each path from the root of strategy S^* includes at most n nodes labeled by tests of T^*, we have for any subtree A replacing a test x:

$$P_p(x) \left(1 - \delta_p\right)^n \leq P_p(A) \leq P_p(x) \left(1 + \delta_p\right)^n. \tag{2.30}$$

This gives us the following relation between the expected costs of S^* and S_u:

$$C\left(S^*\right) \left(1 - \delta_p\right)^n \left(1 - \delta_c\right) \leq C\left(S_u\right) \leq C\left(S^*\right) \left(1 + \delta_p\right)^n \left(1 + \delta_c\right). \tag{2.31}$$

If $\delta_p \leq r/4n$, then $(1 - \delta_p)^n \geq 1 - n\delta_p \geq 1 - r/3$ by Observation 27i, and $(1 + \delta_p)^n \leq 1 + r/3$ by Observation 27ii. Thus if we have

$$\delta_p \leq r/4n, \tag{2.32}$$

$$\delta_c \leq r/3, \tag{2.33}$$

then

$$\left(1 - \delta_p\right)^n \left(1 - \delta_c\right) \geq \left(1 - r/3\right)^2 > 1 - r, \tag{2.34}$$

$$\left(1 + \delta_p\right)^n \left(1 + \delta_c\right) \leq \left(1 + r/3\right)^2 < 1 + r. \tag{2.35}$$

Therefore as long as the conditions (2.27), (2.28), (2.29) hold, with δ_p and δ_c satisfying (2.32) and (2.33), the desired relation (2.24) is fulfilled. We will now prove these bounds for an `and`-rooted subtree. The case of an `or`-rooted subtree is symmetric.

First let $C'(A)$ be the cost of the subtree A if the conditions (i) and (ii) of the reduction are satisfied, but the success probability of each test is equal to $p' = {p_x}^{1/k'}$, that is it is not rounded to $\tilde{p}$. The subtree A has k' tests, $k' = \lfloor k \rfloor$, where k is given by (2.18). Thus $k \geq k' \geq k - 1$. If $p_x = 1$ then $p' = 1$, $C'(A) = uk'$ and $k = c_x/u$

so $c_x \geq C'(A) \geq u\,(k-1) = c_x - u$. Lemma 25 implies that also for $p_x \neq 1$ we have $c_x - u \leq C'(A) \leq c_x$. Since $u \leq \frac{r}{3} c_{min}$, it holds that

$$c_x(1 - r/3) \leq C'(A) \leq c_x. \tag{2.36}$$

We now need to incorporate the effect of rounding the probability p' to $\tilde{p}$, according to the condition (iii) of the reduction. We have $p_x \geq 1/2$ (A is `and`-rooted), and $p_{x0} = \bar{p}_x \leq 1/2$.

For $p_x = 1$ we have $\tilde{p} = p' = 1$, so $C(A) = C'(A)$. Now assume $\frac{1}{2} \leq p_x < 1$. Observe that $\varepsilon \leq \frac{p_{x0} r}{4nk'} = \frac{3p_{x0}}{4nk'}\frac{r}{3} \leq \frac{3}{8nk'}\frac{r}{3} \leq \frac{1}{2(k'-1)}\frac{r}{3}$. Therefore we can use Lemma 26i and conclude that

$$C'(A) \leq C(A) \leq C'(A)(1 + r/3). \tag{2.37}$$

From (2.37) and (2.36) it follows that

$$c_x\,(1 - r/3) \leq C(A) \leq c_x\,(1 + r/3)\,, \tag{2.38}$$

which means that (2.27) and (2.33) hold.

Now let us consider the probability $p(A)$ that A evaluates to `true`, $p(A) = \tilde{p}^{k'}$. For $p_x = 1$ we have $\tilde{p} = 1$ thus $p(A) = 1 = p_x$. If $\frac{1}{2} \leq p_x < 1$, it follows from Lemma 26ii and the bound $\varepsilon \leq \frac{p_{x0} r}{4nk'} = \frac{p_{x0}}{k'}\frac{r}{4n}$ that

$$p_x \leq p(A) \leq p_x\left(1 + \frac{r}{4n}\right), \tag{2.39}$$

$$\bar{p}_x\left(1 - \frac{r}{4n}\right) \leq \bar{p}(A) \leq \bar{p}_x, \tag{2.40}$$

so (2.28), (2.29) and (2.32) are satisfied. □

In the remainder of this section we present the proofs of the lemmas used above.

Lemma 23 *If $k(p,c) = \frac{\ln(p)}{\ln\left(1-\frac{1-p}{c}\right)}$, where $0 < p < 1$ and $c > 1$, then $k(p,c)$ is a monotone function of p, decreasing from ∞ to c as p increases from 0 to 1.*

Proof:

$$\lim_{p\to 0} \frac{\ln(p)}{\ln\left(1 - \frac{1-p}{c}\right)} = \infty. \tag{2.41}$$

$$\lim_{p\to 1} \frac{\ln(p)}{\ln\left(1 - \frac{1-p_x}{c}\right)} = \lim_{p\to 1} \frac{(p)^{-1}}{\left(1 - \frac{1-p}{c}\right)^{-1}\frac{1}{c}} = c. \tag{2.42}$$

$$\frac{\partial k(p,c)}{\partial p} = \frac{c\left(1 - \frac{1-p}{c}\right)\ln\left(1 - \frac{1-p}{c}\right) - p\ln(p)}{p\left(1 - \frac{1-p}{c}\right) c \ln^2\left(1 - \frac{1-p}{c}\right)}. \tag{2.43}$$

The above derivative exists for $0 < p < 1$ and $c > 1$.
We will show that $c\left(1 - \frac{1-p}{c}\right)\ln\left(1 - \frac{1-p}{c}\right) - p\ln(p) < 0$.
Let $g(p,c) = c\left(1 - \frac{1-p}{c}\right)\ln\left(1 - \frac{1-p}{c}\right)$. We will firstly show that $g(p,c)$ is a monotone decreasing function of c.

$$\frac{\partial g(p,c)}{\partial c} = \ln\left(1 - \frac{1-p}{c}\right) + \frac{1-p}{c} \tag{2.44}$$

and the above derivative exists for $0 < p < 1$ and $c > 1$.

For $-1 < x < 1$, $\ln(1-x) = -(x + \frac{x^2}{2} + \frac{x^3}{3} + \ldots)$, so for $0 < x < 1$ $\ln(1-x) < -x$.

Since $0 < \frac{1-p}{c} < \frac{1}{c} < 1$, so $\ln\left(1 - \frac{1-p}{c}\right) < -\frac{1-p}{c}$. Thus:

$$\frac{\partial g(p,c)}{\partial c} < 0. \tag{2.45}$$

Moreover

$$\lim_{c \to 1} g(p,c) = p \ln p. \tag{2.46}$$

Thus $g(p,c) < p \ln p$ and $\frac{\partial k(p,c)}{\partial p} < 0$. □

Lemma 24 *For $c \geq 1$, $\frac{\ln(0.5)}{\ln\left(1-\frac{0.5}{c}\right)} \leq 2\ln 2\, c$.*

Proof: Let $k(c) = \frac{\ln(0.5)}{\ln\left(1-\frac{0.5}{c}\right)}$ and $g(c) = k(c)/c$. Then

$$g(c) = \frac{\ln 2}{\ln\left(\frac{c}{c-0.5}\right) c}, \tag{2.47}$$

and

$$\lim_{c \to \infty} g(c) = \lim_{c \to \infty} \frac{\ln 2\left(-c^{-2}\right)}{\frac{-0.5}{c(c-0.5)}} = 2 \ln 2. \tag{2.48}$$

We will show that $g(c)$ is a monotone, increasing function of c, from which it follows that $k(c)/c \leq 2\ln 2$. It is enough to show that $h(c) = \ln\left(\frac{c}{c-0.5}\right) c$ is monotonically decreasing while c is increasing.

$$\frac{dh(c)}{dc} = \ln\left(\frac{c}{c-0.5}\right) - \frac{0.5}{c-0.5}. \tag{2.49}$$

The above derivative exists for $c \geq 1$. Let $b(c) = \ln\left(\frac{c}{c-0.5}\right) - \frac{0.5}{c-0.5}$. We will show that $b(c) \leq 0$. Since

$$b(1) = \ln 2 - 1 < -0.3, \tag{2.50}$$

$$\lim_{c \to \infty} b(c) = 0, \tag{2.51}$$

so it is enough to show that $b(c)$ monotonically increases with c:

$$\frac{db(c)}{dc} = \frac{0.5}{(c-0.5)^2}\left(1 - \frac{c-0.5}{c}\right) > 0 \tag{2.52}$$

and the above derivative exists for $c \geq 1$. □

Lemma 25 *Let $0 < p < 1$, $c > u$ for some positive u, $k = \frac{\ln(p)}{\ln\left(1-\frac{1-p}{c/u}\right)}$, $k-1 \leq k' \leq k$, $p' = p^{1/k'}$ and $c' = u\frac{1-p'^{k'}}{1-p'}$. Then $c - u \leq c' \leq c$.*

Proof: First notice that from Lemma 23 it follows that $k > 1$. Thus $k' > 0$ and p' exists.

$$c' = u\frac{1 - p'^{k'}}{1 - p'} = u\frac{1 - p}{1 - p^{1/k'}}. \tag{2.53}$$

From (2.18), (2.19) follows that

$$c = u\frac{1 - (p^{1/k})^k}{1 - p^{1/k}} = u\frac{1 - p}{1 - p^{1/k}}. \tag{2.54}$$

The inequality $c' \leq c$ follows now from $k' \leq k$.

$$c - c' = u(1 - p)\left(\frac{1}{1 - p^{1/k}} - \frac{1}{1 - p^{1/k'}}\right) \leq u(1 - p)\left(\frac{1}{1 - p^{1/k}} - \frac{1}{1 - p^{1/(k-1)}}\right). \tag{2.55}$$

Let $B(p, k) = (1 - p)\left(\frac{1}{1-p^{1/k}} - \frac{1}{1-p^{1/(k-1)}}\right)$. We will show that $B(p, k) < 1$, which ends the proof.

$$\lim_{k\to\infty} B(p, k) = \frac{-1 + p}{\ln(p)} = \frac{-(1 - p)}{\ln(1 - (1 - p))} < 1, \tag{2.56}$$

where we used the fact that $\ln(1 - x) < -x$ for $0 < x < 1$ and calculated the first equality using asymptotic expansion into series for k approaching ∞ from the "Maple" packet. Now it is enough to show that $B(p, k)$ increases when k increases.

$$\frac{\partial B(p,k)}{\partial k} = (1 - p)\ln(p)\left[\frac{p^{\frac{1}{k-1}}}{\left(1 - p^{\frac{1}{k-1}}\right)^2 (k - 1)^2} - \frac{p^{\frac{1}{k}}}{\left(1 - p^{\frac{1}{k}}\right)^2 k^2}\right] =$$

$$= -(1 - p)\ln(p)\frac{p^{\frac{1}{k-1}}}{\left(1 - p^{\frac{1}{k-1}}\right)^2 k^2}\left[\left(\frac{1 - p^{\frac{1}{k-1}}}{1 - p^{\frac{1}{k}}}\right)^2 \frac{1}{p^{\frac{1}{k(k-1)}}} - \frac{k^2}{(k - 1)^2}\right]. \tag{2.57}$$

The above derivative exists for $0 < p < 1$, $k > 1$. Let $Z(p, k) = \left(\frac{1-p^{\frac{1}{k-1}}}{1-p^{\frac{1}{k}}}\right)^2 \frac{1}{p^{\frac{1}{k(k-1)}}}$. We will show that $Z(p, k) - \frac{k^2}{(k-1)^2} > 0$, which ends the proof.

$$\lim_{p\to 0} Z(p, k) = \infty. \tag{2.58}$$

$$\lim_{p\to 1} Z(p, k) = \left(\lim_{p\to 1}\frac{1 - p^{\frac{1}{k-1}}}{1 - p^{\frac{1}{k}}}\right)^2 = \left(\lim_{p\to 1}\frac{k}{k - 1}p^{\frac{1}{k-1} - \frac{1}{k}}\right)^2 = \frac{k^2}{(k - 1)^2}. \tag{2.59}$$

So now it is enough to show that $Z(p, k)$ is a monotone (decreasing) function of p. Let us change the variables. Define $y(k, p) = p^{\frac{1}{k(k-1)}}$. Now $Z(y(k, p), k) = \left(\frac{1-y^k}{1-y^{k-1}}\right)^2 \frac{1}{y}$. $y(k, p)$ increases when p increases and $0 < y(k, p) < 1$. It is enough to show that $Z(y(k, p), k)$ decreases when y increases.

$$\frac{\partial Z(y(k,p),k)}{\partial y} = \frac{1 - y^k}{(1 - y^{k-1})^3 y^2}\left[y^{2k-1} - (2k - 1)y^k + (2k - 1)y^{k-1} - 1\right]. \tag{2.60}$$

The above derivative exists for $0 < p < 1$, $k > 1$.

Let $W(y,k) = y^{2k-1} - (2k-1)y^k + (2k-1)y^{k-1} - 1$. We will show that $W(y,k) < 0$.

$$\lim_{y \to 0} W(y,k) = -1. \tag{2.61}$$

$$\lim_{y \to 1} W(y,k) = 0. \tag{2.62}$$

Now we need to show that $W(y,k)$ is a monotone (increasing) function of y.

$$\begin{aligned} \frac{\partial W(y,k)}{\partial y} &= (2k-1)y^{2k-2} - (2k-1)ky^{k-1} + (2k-1)(k-1)y^{k-2} = \\ &= (2k-1)y^{k-2}\left[y^k - ky + k - 1\right]. \end{aligned} \tag{2.63}$$

The above derivative exists for $0 < p < 1$, $k > 1$. Let $J(y,k) = y^k - ky + k - 1$. We will show that $J(y,k) > 0$.

$$\lim_{y \to 0} J(y,k) = k - 1. \tag{2.64}$$

$$\lim_{y \to 1} J(y,k) = 0. \tag{2.65}$$

Now it is enough to show that $J(y,k)$ is a monotone (decreasing) function of y.

$$\frac{\partial J(y,k)}{\partial y} = ky^{k-1} - k = k\left(y^{k-1} - 1\right) < 0. \tag{2.66}$$

The above derivative exists for $0 < p < 1$, $k > 1$. □

Lemma 26 *Let $\frac{1}{2} \le p < 1$, $p' = p^{1/k'}$, $p' \le \tilde{p} \le p'(1+\varepsilon)$, for integer $k' \ge 2$. Let $c' = u\left(1 + p' + p'^2 + \ldots + p'^{k'-1}\right)$ and $\tilde{c} = u\left(1 + \tilde{p} + \tilde{p}^2 + \ldots + \tilde{p}^{k'-1}\right)$, for $u > 0$. For any $0 < \delta < 1$:*

i) if $\varepsilon \le \frac{1}{2(k'-1)}\delta$ then $c' \le \tilde{c} \le c'(1+\delta)$,

ii) if $\varepsilon \le \frac{1-p}{k'}\delta$ then $p \le \tilde{p}^{k'} \le p(1+\delta)$ and $(1-p)(1-\delta) \le 1 - \tilde{p}^{k'} \le 1 - p$.

Proof:

i)

Since $\tilde{p} \ge p'$ so $\tilde{c} \ge c'$.

$$\begin{aligned} \tilde{c} &= u\left(1 + \tilde{p} + \tilde{p}^2 + \ldots + \tilde{p}^{k'-1}\right) \le \\ &\le u\left(1 + p'(1+\varepsilon) + p'^2(1+\varepsilon)^2 + \ldots + p'^{k'-1}(1+\varepsilon)^{k'-1}\right) \le \\ &\le (1+\varepsilon)^{k'-1} u\left(1 + p' + p'^2 + \ldots + p'^{k'-1}\right) = \\ &= (1+\varepsilon)^{k'-1} c' \le c'\left(1 + \frac{\delta}{2(k'-1)}\right)^{k'-1} \le c'(1+\delta), \end{aligned} \tag{2.67}$$

where the last inequality follows from Observation 27ii.

ii)

Since $\tilde{p} \ge p'$ so $\tilde{p}^{k'} \ge p'^{k'} = p$ and $1 - \tilde{p}^{k'} \le 1 - p$.

$$\tilde{p}^{k'} \le \left[p'(1+\varepsilon)\right]^{k'} = p(1+\varepsilon)^{k'} \le p\left(1 + \frac{1-p}{k'}\delta\right)^{k'} = p\left(1 + \frac{\delta}{\frac{1}{1-p}k'}\right)^{k'}. \tag{2.68}$$

Since $\frac{1}{1-p} \geq 2$ so by Observation 27ii we have

$$p\left(1+\frac{\delta}{\frac{1}{1-p}k'}\right)^{k'} \leq p\left(1+\frac{\delta}{\frac{1}{1-p}-1}\right) = p\left(1+\delta\frac{1-p}{p}\right). \tag{2.69}$$

Now since $\frac{1-p}{p} \leq 1$ we obtain eventually $\tilde{p}^{k'} \leq p\,(1+\delta)$.
For $1-\tilde{p}^{k'}$ it holds that

$$1-\tilde{p}^{k'} \geq 1-p\left(1+\delta\frac{1-p}{p}\right) = (1-p)(1-\delta). \tag{2.70}$$

□

Observation 27 *For integer $k \geq 1$ and real x, $0 \leq x \leq 1$, the following inequalities hold:*

$$i)\quad (1-x)^k \geq 1-kx, \tag{2.71}$$

$$ii)\quad \left(1+\frac{x}{ak}\right)^k \leq 1+\frac{x}{a-1} \text{ for } a \geq 2. \tag{2.72}$$

Proof:
i)
By induction on k. Assume it holds for k.

$$(1-x)^{k+1}-[1-(k+1)x] \geq (1-x)(1-kx)-[1-(k+1)x] = kx^2 \geq 0. \tag{2.73}$$

ii)
Trivial for $x=0$. Now assume $0 < x \leq 1$.

$$\left(1+\frac{x}{ak}\right)^k = \sum_{i=0}^{k}\binom{k}{i}\left(\frac{x}{ak}\right)^i \leq \sum_{i=0}^{k}\left(\frac{x}{a}\right)^i = \frac{1-\left(\frac{x}{a}\right)^{k+1}}{1-\frac{x}{a}}, \tag{2.74}$$

and

$$\frac{1-\left(\frac{x}{a}\right)^{k+1}}{1-\frac{x}{a}}-\left(1+\frac{x}{a-1}\right) = \left(1-\frac{x}{a}\right)^{-1}\left[-\left(\frac{x}{a}\right)-\frac{x(1-x)}{a(a-1)}\right] \leq 0. \tag{2.75}$$

□

Chapter 3

Conjectures and Counterexamples

3.1 Best Test of a Subtree

The natural approach to solving PAOTR is to try to exploit the local structures in the input trees. We expected that the Siblings Lemma can be generalized so that for each immediate subtree of a given tree we could find, independently on other subtrees, the "best" test, in the sense that there is an optimal strategy such that the first performed test is the best test from its subtree. However it turns out not the case.

Consider as an example the `and-or` tree T_c shown in Figure 3.1a. Tests a_1, a_2, b_1 and b_2 are grandchildren of the same `and`-node, but the relative order in which these tests are queried by an optimal strategy, varies with the probability of success of test c; depending on $p(c)$, an optimal strategy starts with a_1 or with b_1.

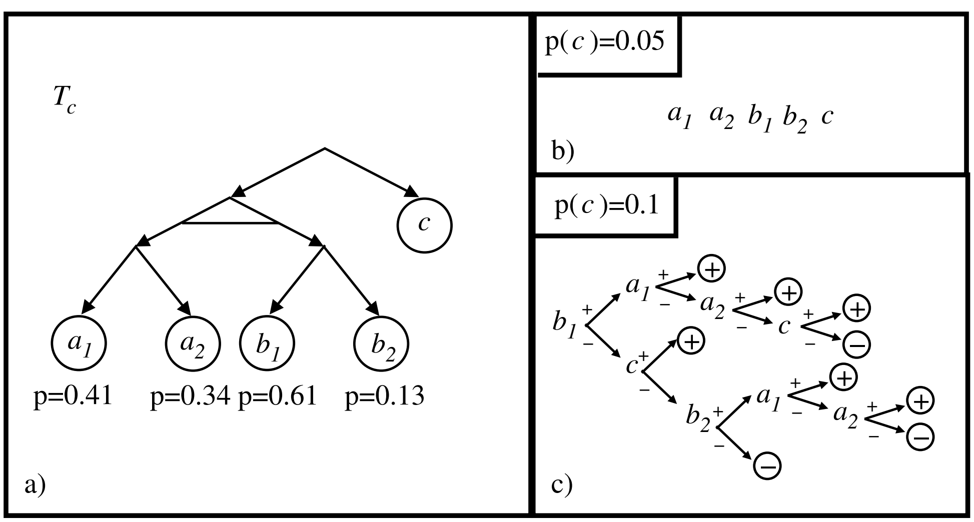

Figure 3.1: (a) An `and-or` tree T_c with all costs unit. (b) The unique optimal strategy for T_c if $p(c) = 0.05$, encoded by the fixed order of tests, starting with a_1. (c) The unique optimal strategy for T_c if $p(c) = 0.1$, starting with b_1.

For example, if $p(c) = 0.05$, the unique optimal strategy starts with testing a_1 and then follows the linear strategy shown in Figure 3.1b. But if $p(c) = 0.1$, the unique optimal strategy is the strategy shown in Figure 3.1c, which starts with testing b_1.

3.2 Prime Implicants and Implicates

A minimal set with some property P is a set with the property P which does not include any set with the property P as its proper subset. A *prime implicant* of an `and-or` tree is a minimal set of tests with the property that if all tests from the set are `true` then the entire tree evaluates to `true`. A *prime implicate* in 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` (respectively `false`) if and only if there is at least one prime implicant (respectively prime implicate) whose tests are all `true` (respectively `false`). To see that, assume that the value of T is `true`, but each prime implicant of T contains at least one `false` test. But then the set of all `true` tests includes a prime implicant for T, contradiction. Whenever, while performing a correct strategy, we conclude that an `and-or` tree evaluates to `true` (respectively to `false`), it is only after all tests of some prime implicant (respectively prime implicate) have been performed and succeeded (respectively failed).

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

One might conjecture that for an `or`-rooted `and-or` tree, all tests performed on the true path of an optimal strategy are from exactly one prime implicant. In other words, we expected that in `or`-rooted tree, if the first test performed by an optimal strategy succeeds, the strategy will perform tests from this prime implicant, as long as they are `true`. Notice that it would mean that the optimal strategy does not leave one child subtree of the `or`-root as long as the performed tests succeed. The conjecture is equivalent to the one that all tests performed on the `false` path of an optimal strategy for an `and`-rooted tree are from exactly one prime implicate.

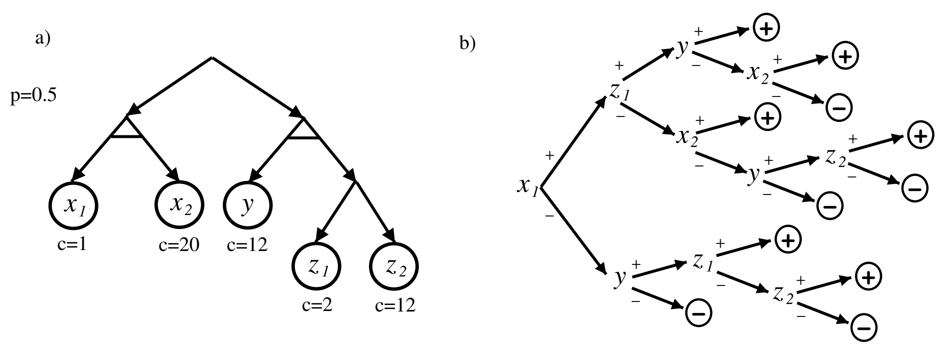

Figure 3.2: a) An `and-or` tree T_i. All tests have probability of success 0.5. b) The unique optimal strategy S_i for T_i. The tests performed on the true path of S_i are from two prime implicants of T_i.

This conjecture turned out to be false. Consider the `or`-rooted tree T_i in Figure 3.2a. The prime implicants are the sets $\{x_1, x_2\}$, $\{y, z_1\}$ and $\{y, z_2\}$. The unique optimal strategy S_i for T_i, shown in Figure 3.2b, starts with performing x_1, but then, if x_1 is `true`, it leaves the subtree. The true path of S_i, whose first node is labeled by x_1, contains then the nodes labeled by z_1 and y: tests from another prime implicant.

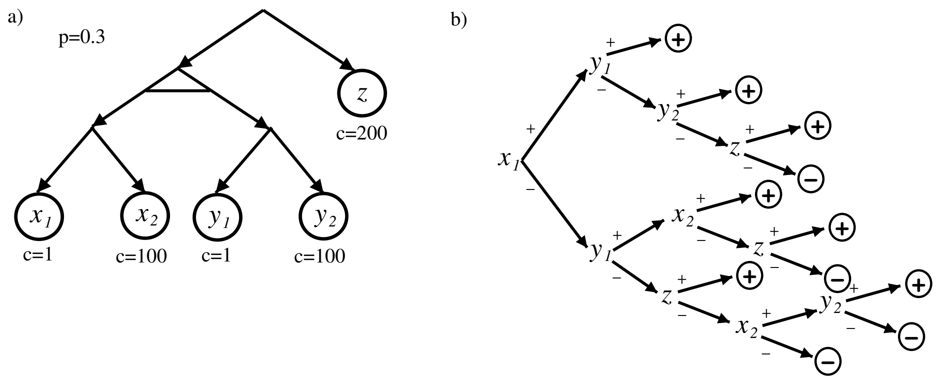

Figure 3.3: a) An `and-or` tree T_e. All tests have probability of success 0.3. b) The unique optimal strategy S_e for T_e. The tests performed on the false path of S_e are from two prime implicates of T_e.

Notice that the tests performed on the false path of S_i are from only one prime implicate, $\{x_1, y\}$. But it also is not always the case. On Figure 3.3a we depict an `or`-rooted tree T_e, with prime implicates $\{x_1, x_2, z\}$ and $\{y_1, y_2, z\}$. The false path of the unique optimal strategy S_e for T_e shown in Figure 3.3b contains nodes labeled by tests x_1, z, x_2, interlaid by a node labeled by a test y_1 from another prime implicate.

We still expect though that an optimal strategy will either complete one prime implicant or one prime implicate.

Conjecture 1 *For any* `and-or` *tree there is an optimal strategy S such that either all tests performed on the true path of S are from exactly one prime implicant, or all tests performed on the false path of S are from exactly one prime implicate.*

3.3 Cograph Representation

There is a representation of the `and-or` trees that may be helpful in studying prime implicants and implicate. We now describe this representation. We start with basic definitions related to graphs. We follow the definitions from [5].

In undirected graphs, we consider unordered pairs of nodes (edges) as opposed to the ordered pairs (arcs) in directed graphs. An *undirected graph* G is an ordered pair (V, E), where V is a finite set (whose elements are called *nodes* of G) and E is the set of unordered pairs of nodes of G (whose elements are called *edges* of G). We denote an edge by (v, w), understanding that (v, w) and (w, v) is the same edge.

Let $G = (V, E)$. If $(u, v) \in E$, we say that u and v are *adjacent*. The notion of a path, subgraph and the relation of being reachable for nodes is defined as for a directed graph (if v is reachable from w then w is reachable from v). G is *connected* if every vertex is reachable from every other vertex, and *disconnected* otherwise. The *connected components* of G are the equivalence classes of nodes for the relation of being reachable from. The graph $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{(v, w) : v, w \in V, v \neq w, (v, w) \notin E\}$ is called the *complement* of G.

A set of nodes $V' \subseteq V$ is called a *clique* if the nodes in V' are pairwise adjacent, and is called an *independent set* if no two nodes from V' are adjacent. A clique (respectively independent set) V' is *maximal* if there is no clique (respectively independent set) V'' such that $V' \subset V''$.

Let $G_1 = (V_1, E_1), G_2 = (V_2, E_2), \ldots, G_k = (V_k, E_k)$, $k \geq 2$ be graphs with disjoint sets of nodes. G is the *union* of $G_1, G_2, \ldots, G_k$, if $V = V_1 \cup V_2 \cup \ldots \cup V_k$ and $E = E_1 \cup E_2 \cup \ldots \cup E_k$. G is the *join* of $G_1, G_2, \ldots, G_k$, if $V = V_1 \cup V_2 \cup \ldots \cup V_k$ and $E = E_1 \cup E_2 \cup \ldots \cup E_k \cup \{(v, w) : v \in V_i, w \in V_j, i \neq j, i, j \leq k\}$. By the *operation of taking complement*, the *operation of union*, the *operation of join* we mean constructing the graph that is the complement of a graph, union of graphs, join of graphs, respectively.

A *cotree* of a graph G is a directed rooted tree T whose leaf nodes are nodes of G, internal nodes are labeled 0 or 1 and two nodes v and w of G are adjacent if and only if the least common predecessor (that is the common predecessor with maximum depth) of v and w in T is labeled 1.

The class of *cographs* (*complement reducible graphs*) is the class of graphs that can be constructed from single nodes using the operations of union and taking complement.

There is a number of important characterization of cographs.

Theorem 28 *[15, 3] The following statements are equivalent:*

- *G is a cograph,*
- *there exists a cotree of G,*
- *G does not contain a path on four nodes as an induced subgraph,*
- *the complement of each connected induced subgraph of G with more than one node is disconnected,*
- *in every induced subgraph H of G, the intersection of any maximal clique of H and any maximal independent set of H contains precisely one node.*

The representation of a cograph by its cotree enables a linear-time recognition of cographs [4], and allows to use cographs to recognize whether a Boolean function is a read-once function [9].

We say that a graph G *represents* an `and-or` tree T if each node of G is associated with a leaf of T and two nodes v and w of G are adjacent if and only if the least common predecessor of v and w in T is labeled `and`. From the definition of a cotree and Theorem 28 it follows that a graph G whose nodes are associated with distinct independent tests represents an `and-or` tree if and only if G is a cograph. Figure 3.4 shows a cograph representing an `and-or` tree.

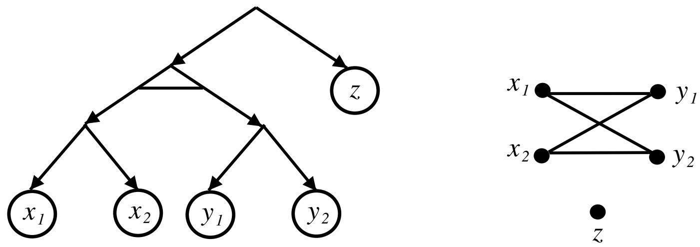

Figure 3.4: An `and-or` tree and the cograph representing the tree.

Observe that a join of graphs G_1 and G_2 is the complement of the union of complements of G_1 and G_2. It is easy to see how to build a unique cograph representing given `and-or` tree, using the operations of union and join. If T is a single leaf, then the graph with a single node represents T. Let $\Psi = \{T_1, T_2, \ldots, T_k\}$, $k \geq 2$, be a set of `and-or` trees and let $G_1, G_2, \ldots, G_k$ be the graphs representing $T_1, T_2, \ldots, T_k$, respectively. Now consider the `and-or` tree $T = \langle r, \Psi \rangle$ (that is rooted at r and whose set of immediate subtrees is Ψ) and let G be a graph that represents T. For any two tests from the same subtree T_i their least common predecessor in T belong to T_i, whereas for any two tests from different subtrees T_i, T_j, their last common predecessor in T is the root r. Thus if r is labeled `and`, G is the join of $G_1, G_2, \ldots, G_k$, whereas if r is labeled `or`, G is the union of $G_1, G_2, \ldots, G_k$.

On the other hand, for any cograph G whose nodes are associated with distinct tests one can construct an `and-or` tree T that is represented by G. If G is a single node, then T is a single test. Otherwise, if G is disconnected, then the root of T is `or`, and each of the immediate subtrees of T is represented by one connected component of G, whereas when G is connected, then the root of T is `and` and each of its immediate subtrees is represented by one connected component of the complement of G.

There is a correspondence between prime implicants of `and-or` trees and maximal cliques of cographs as well as between prime implicates and maximal independent sets.

Observation 29 *Let T be an* `and-or` *tree and let G be the cograph representing T. A set W of tests of T is a prime implicant (respectively prime implicate) if and only if the set of nodes of G associated with the tests from W is a maximal clique (respectively a maximal independent set).*

Proof: We will prove the correspondence between prime implicants and maximal cliques. The proof of the other correspondence is analogous. The proof is by induction on the depth of T. The observation trivially hold for a depth zero tree, namely for a single test. Now assume that T is of depth at least one and that the observation is true for any tree shallower than T.

Let $T_1, T_2, \ldots, T_k$, $k \geq 2$ be the immediate subtrees of T and $G_1, G_2, \ldots, G_k$ the cographs representing $T_1, T_2, \ldots, T_k$, respectively.

Assume that T is `and`-rooted. W is a prime implicant of T if and only if $W = W_1 \cup W_2 \cup \ldots \cup W_k$, where for $i \leq k$ W_i is a prime implicant of T_i. By the inductive assumption W_i is a prime implicant of T_i if and only if the set of nodes of G_i associated with tests from W_i is a maximal clique. But since all nodes from different subgraphs G_i, G_j are adjacent in G, so the set of nodes V' of G is a maximal clique if and only if $V' = V'_1 \cup V'_2 \cup \ldots \cup V'_k$, where for $i \leq k$ V'_i is a maximal clique of G_i.

Now assume that T is `or`-rooted. W is a prime implicant of T if and only if W is a prime implicant of one of $T_1, T_2, \ldots, T_k$. So the observation for this case follows from the inductive assumption and the fact that since G is the union of $G_1, G_2, \ldots, G_k$, a set of nodes of G is a maximal clique if and only if it is a maximal clique in one of $G_1, G_2, \ldots, G_k$. □

We can define a problem related to cographs that is equivalent to PAOTR.

We are given a graph G such that each node of G is assigned one of two colours: it can be either black or white, independently on other nodes. We call any given assignment of colours to nodes a *colouring* of nodes. For each node we know a non-negative cost of checking the colour of the node and the probability that the node is black. The *strategy for the graph G* is an algorithm that for any colouring of nodes determines whether there is a maximal clique in G whose nodes are all black, via sequential checking colours of nodes. For any given colouring, the cost of a strategy on this colouring is the total cost of performed colour checking. The expected cost of a strategy is the average cost of the strategy, over all colourings of the nodes of G. An *optimal strategy* for G is a strategy with the smallest expected cost.

From the discussion above it follows that the problem of finding an optimal strategy for a cograph is equivalent to PAOTR.

Our Conjecture 1 can be now rephrased as follows: For any cograph G there is an optimal strategy S such that S either does not leave one maximal clique as long as the checked nodes are black, or does not leave one maximal independent set, as long as the checked nodes are white.

Generalizing, we could ask for an optimal strategy for an arbitrary graph (which is not related to the `and-or` tree problem). We will show that this problem is NP-hard.

Observation 30 *Finding an optimal strategy for an arbitrary graph is NP-hard.*

Proof: Let G be an arbitrary graph on n nodes. For each node, let the cost of checking the colour be 1 and the probability of being black be $q = \left(1 - \frac{1}{2n}\right)^{\frac{1}{n+1}}$. Let S be a strategy for G, represented by a binary tree. The root-to-leaf path of the strategy that is followed when all checked nodes are black is called the *black path.* Let k be the number of internal nodes of the black path of S. Observe that each other root-to-leaf path of S contains at least 1 and at most n internal nodes. Analogously as in the proof of Theorem 1, we obtain following bounds on the expected cost of S:

$$C(S) \leq q^k k + (1 - q^k)n = n - (n-k)\left(1 - \frac{1}{2n}\right)^{\frac{k}{n+1}} <$$
$$< n - (n-k)\left(1 - \frac{1}{2n}\right) = k + \frac{n-k}{2n} \leq k + \frac{1}{2} \qquad (3.1)$$

and

$$C(S) \geq q^k k + (1-q^k)1 = 1 + (k-1)\left(1 - \frac{1}{2n}\right)^{\frac{k}{n+1}} >$$
$$> 1 + (k-1)\left(1 - \frac{1}{2n}\right) = k - \frac{k-1}{2n} \geq k - \frac{1}{2}. \tag{3.2}$$

Therefore there is a strategy for G with expected cost not greater than $k + \frac{1}{2}$ if and only if there is a strategy for G that checks at most k nodes along its "black path" if and only if there is a maximal clique in G with at most k nodes.

But the problem of determining whether a graph G has a maximal clique with at most k nodes (called the Minimum Maximal Clique Problem) is NP-complete by reduction from the Minimum Maximal Independent Set Problem, that is the problem of determining whether a graph G has a maximal independent set with at most k nodes [7].

Therefore, by reduction from the Minimum Maximal Clique Problem finding an optimal strategy for an arbitrary graph is NP-hard. □

3.4 Resolving Subtrees

A depth-first strategy, which is optimal for depth two `and-or` trees, does not leave a given subtree until its value is determined. Such approach is not necessary optimal for deeper trees, but we conjectured a weaker property of an optimal strategy.

After a test from an `and-or` tree is performed, let the *highest resolved node* in the tree be the root of the maximal subtree whose value has been determined. For example, assume that a test x is `false`. If the parent of x is labeled `or` and x has a sibling, then the highest resolved node is x itself. But if the parent of x is `and` and is a child of an `or`-node, then the highest resolved node is the parent of x.

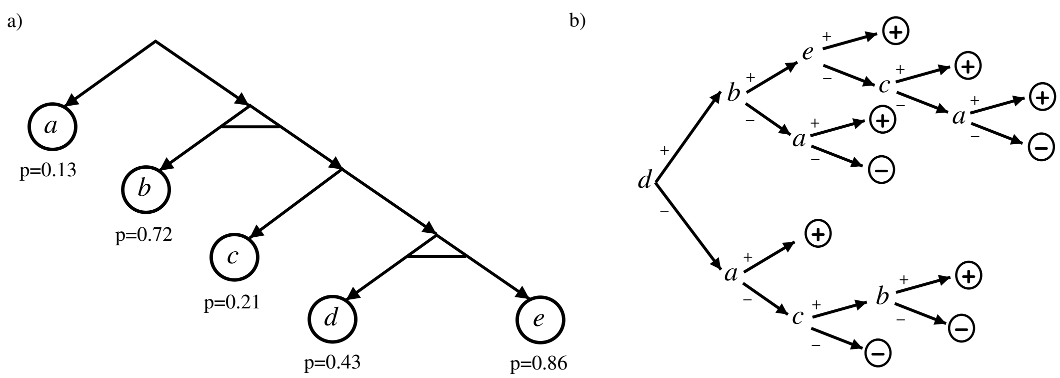

Figure 3.5: a) An `and-or` tree T_l. All tests have unit cost. b) The unique optimal strategy for T_e. After the first performed test is `true` as well as after the first test is `false`, the strategy leaves the subtree rooted at the parent of the highest resolved node.

We expected that after a test x is performed, an optimal strategy would at least in one case (when x is `true`, or if x is `false`) performs after x a test from the subtree rooted at the parent of the highest resolved node. However it turns out that it is not always so. Consider the tree T_l in Figure 3.5a. The unique optimal strategy for T_l is presented in Figure 3.5b. After performing the first test d, when d is `true`, as well as when d is `false`, the strategy performs the next test from outside the subtree rooted at the parent of the highest resolved node. We thank Leah Hackman and Martha Lednicky, WISEST 2003 participants, whose experimentation with instances of `and-or` trees led to the discovery of this example.

3.5 Tests Ordering for Ladders

As defined in Section 2.4, in an `and-or` ladder each internal node has at most one internal child node. In an `and-or` ladder a test y is called *better* than a test x if y is a child of a predecessor v of x and either v is labeled `or` and $p(y)/c(y) > p(x)/c(x)$, or v is labeled `and` and $\bar{p}(y)/c(y) > \bar{p}(x)/c(x)$. See Figure 3.6 for an example. For

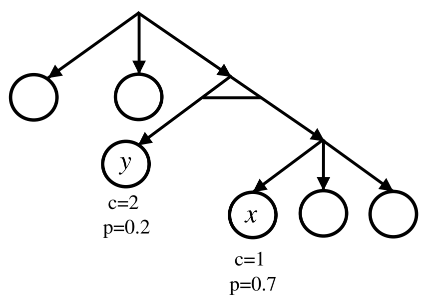

Figure 3.6: An `and-or` ladder. Test y is better than test x.

a large number of ladders, there is a pattern in the order of tests in an optimal strategy, observed by Leah Hackman and Martha Lednicky, which can be described by the following conjecture.

Conjecture 2 *Let T be an* `and-or` *ladder. There is an optimal strategy S for T such that for any tests x and y such that y is better than x, x is not performed before y on any root-to-leaf path of S.*

For `and-or` ladders the conjecture generalizes the Siblings Lemma and the Observation 19. For depth one ladders, it is equivalent to the Siblings Lemma. For depth two ladders, the correctness of the conjecture follows by the Observation 32 from the fact that DFA produces an optimal strategy for depth two `and-or` trees (Theorem 4). Though a large number of numerically checked examples of ladder trees justify the conjecture, we were able to prove it only for a special case of a depth three `and-or` ladder:

Observation 31 *If an* `and-or` *ladder T has depth one or two, or T has depth three and has only two tests with depth three, then Conjecture 2 holds for T.*

Proof: We prove the observation by induction on the number of the tests of T. The observation trivially holds for a ladder with only one test. Now assume that the observation hold for any ladder fulfilling the conditions of the observation, that has less tests than T. Since, as discussed above, the observation hold for any depth 1 or depth 2 ladder, assume that T is depth 3 and that the last internal node of T has exactly 2 tests. Assume that T is `or`-rooted; the proof for the other case is symmetric.

Let a, b and c be the tests with the highest R-ratio among the tests with depth 1, depth 2 and depth 3, respectively. By the Sibling and Twins Lemma there is an optimal strategy that starts with a or with b or with c. If there is a test that is better than c, then it has either depth 1 or 2. If a test with depth 2 is better than c, then b is better than c. Then by Lemma 35 there is an optimal strategy for T that starts either with a or b. If a test with depth 1 is better than c, then a is better than c and by Lemma 34 there is an optimal strategy that starts with a. Now if there is a test better than b, then it has depth 1, and thus a is better than b. In this case, by Lemma 34, there is an optimal strategy for T that starts with a. No test in T can be better than a. It follows that there is an optimal strategy for T whose root is labeled by a test for which there is no better test in T.

Now let T' be the reduced tree obtained from T when the first test performed by S is, say, `true`. Assume that a test y is better than x in T. If y and x are still present in the tree T', then y is better than x in T'. Thus by the inductive assumption there is an optimal strategy for T' that never performs y before x. The same holds for the reduced tree obtained when the first test preformed by S is `false`. Therefore there is an optimal strategy for T that fulfills the condition of Conjecture 2. □

In the remainder of the section we present the proofs of the observations used above.

Observation 32 *Let A be a depth one* `and-or` *tree with at least two leaf nodes and let A' be the tree obtained from A by removing one leaf. Assume that the parent nodes of A and A' have different labels than the root of A. Let $C(A)$ ($C(A')$) be the expected cost of the optimal strategy to evaluate A (respectively A') and let $p^r(A)$ ($p^r(A')$) be the probability that A (respectively A') resolves its parent node. Then $\frac{p^r(A')}{C(A')} \geq \frac{p^r(A)}{C(A)}$.*

Proof: Assume that the root of A is `and`. The proof for the other case is symmetric. Now $p^r(A)$ ($p^r(A')$) is the probability that A (respectively A') evaluates to `true`.

Let $x_1, x_2, \ldots, x_k$, $k \geq 2$, be the tests of A and assume that $R(x_1) \geq R(x_2) \geq \ldots \geq R(x_k)$. Then by Theorem 2 the above order of tests is the order of performing them by the optimal strategy for A. Let x_m, $1 \leq m \leq k$, be the test that is removed from A to create A'.

Thus we have $p^r(A) = \prod_{i \leq k} p(x_i)$, $p^r(A') = \prod_{i \leq k, i \neq m} p(x_i)$, and $C(A) = c(x_1) + \sum_{2 \leq i \leq k} c(x_i) \prod_{1 \leq j < i} p(x_j)$.

We introduce the following notation:

$$C_1 = \begin{cases} 0 & \text{if } m = 1, \\ c(x_1) + \sum_{2 \leq i \leq m-1} c(x_i) \prod_{1 \leq j < i} p(x_j) & \text{if } m \geq 2, \end{cases} \tag{3.3}$$

$$p_1 = \begin{cases} 1 & \text{if } m = 1, \\ \prod_{1 \leq i \leq m-1} p(x) & \text{if } m \geq 2, \end{cases} \tag{3.4}$$

$$C_2 = \begin{cases} 0 & \text{if } m = k, \\ c(x_{m+1}) + \sum_{m+2 \le i \le k} c(x_i) \prod_{m+1 \le j < i} p(x_j) & \text{if } m < k. \end{cases} \tag{3.5}$$

Then we can use the following expressions:

$$C(A) = C_1 + p_1 c(x_m) + p_1 p(x_m) C2, \tag{3.6}$$
$$C(A') = C_1 + p_1 C_2. \tag{3.7}$$

Now we obtain

$$\frac{p^r(A')}{c(A')} - \frac{p^r(A)}{c(A)} = \frac{p^r(A')}{c(A')c(A)} \left[(1 - p(x_m)) C_1 + p_1 c(x_m) \right] \ge 0. \tag{3.8}$$

□

Lemma 33 *Let T be an* `and-or` *tree and let x and y be different tests from T. Assume that y is a child of a predecessor of x and that there is an optimal strategy S_x for T that starts with performing the test x. If*

i) y is the first test performed by S_x after x is `false` *and $R(y) \ge \frac{p(x)}{c(x)}$, or*

ii) y is the first test performed by S_x after x is `true` *and $R(y) \ge \frac{\bar{p}(x)}{c(x)}$,*

then there is an optimal strategy S_y for T that starts with performing y.

Proof: We will prove the theorem for the case when the condition (i) is fulfilled. The proof for the condition (ii) is symmetric. Assume that y is the first test performed by S_x after x is `false` and $R(y) \ge \frac{p(x)}{c(x)}$.

If x and y are child nodes of the same `or`-node then we may assume that the substrategies followed when x is `true`, and when x is `false`, y is `true`, are the same (because if they are not, we may replace them by such strategies). Then, by Observation 8, the strategy S_y obtained by switching labels x and y is optimal.

Now assume that x and y are not child nodes of the same `or`-node. Since y is a child node of a predecessor of x, so after x is `true`, y is still present in the reduced tree. Let S_{+x}, S_{-x} be the substrategies of S_x followed when x is `true`, `false`, respectively. The root of S_{-x} is labeled by y. Let S^{-}_{+y}, S^{-}_{-y} be the substrategies of S_{-x} followed when y is `true`, `false`, respectively. Let $M \ge 1$ be the number of nodes of S_{+x} labeled by test y, let $S_{y1}, S_{y2}, \ldots, S_{yM}$ be the subtrees of S_{+x} rooted at nodes labeled by y, and for $k = 1, 2, \ldots, M$, let S_{+y_k}, S_{-y_k} be the substrategies of S_{y_k} followed in the case when y is `true`, y is `false`, respectively. Also let S^r denote the (possibly empty) part of S_{+x} that contains all nodes outside $S_{y_1}, S_{y_2}, \ldots, S_{y_M}$.

The following strategy S'_{+x} is nonredundant and correct for the tree obtained from T when x is `true`, and may replace the substrategy S_{+x}:
$S'_{+x} = y : + \left(S'^{+}_{+y}\right); - \left(S'^{+}_{-y}\right)$, where $S'^{+}_{+y} = S_{+x}\left(S_{y_1} \triangleleft S_{+y_1}, \ldots, S_{y_M} \triangleleft S_{+y_M}\right)$,
$S'^{+}_{-y} = S_{+x}\left(S_{y_1} \triangleleft S_{-y_1}, \ldots, S_{y_M} \triangleleft S_{-y_M}\right)$.

Assume that y is a child of an `or`-node. The proof for the other case is symmetric. Thus

$$R(y) = \frac{p(y)}{c(y)} \ge \frac{p(x)}{c(x)}. \tag{3.9}$$

Let $C(S_r)$ denote the expected cost of performing S_r, that is the sum of costs of tests labeling nodes of S_r, factored by the probabilities of paths from the root of S_{+x} to a given node (if S_r is empty, then $C(S_r) = 0$). For any k, let p_{y_k} be the probability of the path from the root of S_{+x} to the labeled by y root node of S_{y_k}.

Then we have

$$C\left(S'^{+}_{+y}\right) = C(S_r) + \sum_{k=1}^{M} p_{y_k} C(S_{+y_k}). \tag{3.10}$$

If y is `true`, its parent node is resolved. Since y is a child of a predecessor of x, so the reduced trees evaluated by S'^{+}_{+y} and S^{-}_{+y} are identical. Since S^{-}_{+y} is a substrategy of the optimal strategy S_x, so $C\left(S^{-}_{+y}\right) \leq C\left(S'^{+}_{+y}\right)$, that is

$$C\left(S^{-}_{+y}\right) \leq C(S_r) + \sum_{k=1}^{M} p_{y_k} C(S_{+y_k}). \tag{3.11}$$

Now it is obvious that the following strategy S_y is nonredundant and correct for T: $S_y = y : +\left(S^{-}_{+y}\right); -\left(x : +\left(S'^{+}_{-y}\right); -\left(S^{-}_{-y}\right)\right)$. For the expected costs of S_x and S_y we have

$$\begin{aligned} C(S_x) &= c(x) + p(x)\left[C(S_r) + \sum_{k=1}^{M} p_{y_k}\Big(c(y) + p(y)C(S_{+y_k}) + \bar{p}(y)C(S_{-y_k})\Big)\right] + \\ &\quad + \bar{p}(x)\left[c(y) + p(y)C\left(S^{-}_{+y}\right) + \bar{p}(y)C\left(S^{-}_{-y}\right)\right], \qquad (3.12)\\ C(S_y) &= c(y) + p(y)C\left(S^{-}_{+y}\right) + \\ &\quad + \bar{p}(y)\left[c(x) + p(x)\left(C(S_r) + \sum_{k=1}^{M} p_{y_k} C(S_{-y_k})\right) + \bar{p}(x)C\left(S^{-}_{-y}\right)\right], \qquad (3.13) \end{aligned}$$

thus

$$\begin{aligned} C(S_y) - C(S_x) &= -\left[p(y)c(x) - p(x)c(y)\right] + \\ &\quad -p(x)p(y)\left[C(S_r) + \sum_{k=1}^{M} p_{y_k} C(S_{+y_k}) - C\left(S^{-}_{+y}\right)\right] + \\ &\quad -p(x)c(y)\sum_{k=1}^{M} p_{y_k} \leq \\ &\leq 0, \qquad (3.14) \end{aligned}$$

where the inequality holds by (3.9), (3.11). Therefore S_y is optimal for T. □

Lemma 34 *Let T be an `or`-rooted depth three ladder. Let a, b and c be each a test with the highest R-ratio among all tests with depth one, depth two and depth three respectively. If $\frac{p(a)}{c(a)} \geq \min\left\{\frac{p(c)}{c(c)}, \frac{p(b)}{c(b)}\right\}$ then there is an optimal strategy for T that starts with a.*

Proof: First assume that $\frac{p(a)}{c(a)} \geq \frac{p(c)}{c(c)}$. The proof is by induction on the number of tests with depth 3. If there is only one such test, then the tree collapses to depth 2 and by Observation 32 from the fact that $\frac{p(a)}{c(a)} \geq \frac{p(c)}{c(c)}$ it follows that DFA prefers a

to the and-rooted subtree of the root. Now assume that the lemma hold for a depth 3 ladder that has fewer tests with depth 3 than T has.

Let S be an optimal strategy for T. Assume that S starts with c. If c is `false`, the reduced tree is a ladder fulfilling the condition of the lemma (because each sibling of c has lower ratio of success probability to the cost, than c). So by the inductive assumption S performs a after c is `false`, thus by Lemma 33 there is an optimal strategy for T that starts with a.

Assume that S starts with b. Let $b_1 = b$ and let $b_2, b_3, \ldots, b_k$, $k \geq 1$, be siblings of b_1 such that S performs b_{i+1} after b_i is `true`, and S does not perform a sibling of b_1 after b_k. When any b_i is `false`, the optimal strategy performs a (because the only tests left in the reduced tree are child tests of the root). After b_k is `true` the value of the tree is not resolved yet; let T' be the reduced tree obtained at this point of the strategy and let S' be the optimal strategy for T'. If S' starts with a, then by Observation 18 we can "move" a to the root of the strategy S, that is there is an optimal strategy for T that starts with a. If b_k was the last test with depth 2 then T' collapses to depth 1. Then, since $\frac{p(a)}{c(a)} \geq \frac{p(c)}{c(c)}$, the S' starts with a. So assume the there is still at least one test sibling of b and S' starts with c. But then the reduced tree obtained from T' when c is `false` fulfills the conditions of the lemma, so by the inductive assumption S' performs a after c is `false`. Thus by Lemma 33 there is an optimal strategy for T' that starts with a.

Now assume that $\frac{p(a)}{c(a)} \geq \frac{p(b)}{c(b)}$. The proof is by induction on the number of the depth 3 tests. If there is only one such test, then the tree collapses to depth 2 and by Observation 32 from the fact that $\frac{p(a)}{c(a)} \geq \frac{p(b)}{c(b)}$ follows that DFA prefers a than the and-rooted child subtree of the root. Now assume that the lemma hold for a depth 3 ladder that has less tests of depth 3 than T has.

Let S be an optimal strategy for T. Assume that S starts with b. If b is `false`, S performs a (because the only tests left in the reduced tree are child tests of the root). So by Lemma 33 there is an optimal strategy for T that starts with a. Assume that S starts with c. By the inductive assumption, S performs a after c is `false`. If c is `true`, the reduced tree collapse to depth 2 or depth 1, if b does not have any test sibling. From the fact that $\frac{p(a)}{c(a)} \geq \frac{p(b)}{c(b)}$ follows that in both cases a is the first test to perform by an optimal substrategy (for depth 2 tree, it follows by Observation 32). Thus S performs a after c regardless of the value of c, so by Observation 18 there is an optimal strategy for T that starts with a. $\square$

Lemma 35 *Let T be an* or*-rooted depth three ladder such that there are only two tests with depth three. Let a, b and c be the tests with the highest R-ratio among the tests with depth one, depth two and depth three, respectively. If $\frac{\bar{p}(b)}{c(b)} \geq \frac{\bar{p}(c)}{c(c)}$ then there is an optimal strategy for T that starts either with a or with b.*

Proof: Let S be an optimal strategy for T. Assume that S starts with performing c. Let BC be the subtree rooted at the and-child of the root and let B be the depth 1 subtree obtained from BC when c is `true` and B' be the depth 1 subtree obtained from BC if c is `false` (after collapsing the or-node); B' has all child tests of B and additionally the test that was a sibling of c in BC.

If the first test performed by S after c is `true` is a, it means that DFA prefers a than B. Thus by Observation 32 DFA also prefers a than B', and a is also the first test performed when c is false. But a is a child of the root of T, so there is an optimal strategy for T that starts with a (see Observation 18).

So assume that the first test performed after c is `true` is b. But then by Lemma 33 there is an optimal strategy for T that starts with b. $\square$

Chapter 4

Preconditioned And-Or Trees

4.1 Smith's Algorithm

In this chapter we deal with a generalization of `and-or` trees, namely with preconditioned `and-or` trees. As explained in Section 1.3.6, in a preconditioned `and-or` tree both leaf nodes and internal `or`-nodes and `and`-nodes are probabilistic tests, with given success probabilities and performance costs. The value of a leaf node is the output of the associated test. Each internal node is additionally associated with a required value, `true` or `false`. The tests that are child nodes of a given `and`-node (respectively `or`-node) v may be queried only if the test associated with v was performed and returned its required value: in such a case v evaluates to the logic `and` (respectively `or`) of the child nodes' values. If the output of the test associated with v is not the required value of v, the node v evaluates to the output of this test. The value of a tree is the value of its root node.

To understand better the notion of the required value of an internal node, consider the following example. A company uses three tests x, y and z to evaluate candidates for a certain position. There are precedence constraints: the test x has to be performed before y and z.

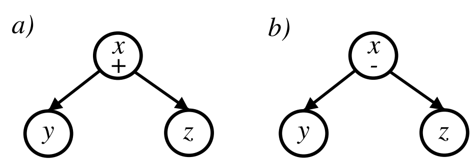

Figure 4.1: Preconditioned `or`-trees. a) The test x has the required value `true`. The tests y and z can be performed only after x succeeds. b) The test x has the required value `false`. The tests y and z can be performed only after x fails.

First consider the case when a candidate is rejected if he or she fails test x (that is passing this test is a necessary condition for accepting the candidate). Additionally the successful candidate has to pass either y or z. To describe this situation, we use the preconditioned `or`-tree presented in Figure 4.1a, where the `or`-node x has the required value `true`. If the output of x is `false` then the tree evaluates to `false`. If the output of x is `true` then the tree evaluates to `or` of values of y and z. Thus the value of the tree is the value of the expression $e_1 = x$ `and` $(y$ `or` $z)$.

But what if a candidate to be accepted needs just to pass one of the tests x, y or z? In such a case we use the preconditioned `or`-tree from Figure 4.1b, where the `or`-node x has the required value `false`. If x has the value `true` then the tree evaluates to `true`, otherwise it evaluates to `or` of the values of y and z. Thus the tree evaluates to the value of the expression $e_2 = x$ `or` y `or` z.

Consider the negation of a preconditioned `and-or` tree. For the tree in Figure 4.1a we have $\neg e_1 = (\neg x)$ `or` $[(\neg y)$ `and` $(\neg z)]$, which is equivalent to the preconditioned `and`-tree with tests $\neg x$, $\neg y$ and $\neg z$, in which the required value of the `and`-node $\neg x$ is `false`. For the tree in Figure 4.1b $\neg e_2 = (\neg x)$ `and` $(\neg y)$ `and` $(\neg z)$. This expression describes the preconditioned `and`-tree in which the required value of the `and`-node $\neg x$ is `true`. In general, we obtain the negation of a preconditioned `and-or` tree by negating the output of any test, changing each `or`-node to an `and`-node and vice versa, and negating the required value of each internal node.

Smith [26] presented an efficient algorithm to find an optimal strategy for preconditioned `or`-trees (that is preconditioned `and-or` trees without `and`-nodes) if the required value of each `or`-node is `true`. We will call this algorithm SA (Smith's Algorithm).

We will first describe the idea of SA, then explain the natural generalization of the algorithm that deals with both `true` and `false` required values of internal nodes in a preconditioned `or`-tree, and present the pseudo-code of the generalized algorithm.

SA operates on blocks, that is sequences of tests. Each block has to obey the precedence constraints in a given tree. For a block a we can calculate the expected cost $C(a)$ of performing the tests from a and the resolving probability $P^r(a)$, that is the probability that performing the tests from a will cause the entire tree to evaluate to `true`. The R-ratio $R(a)$ for a block a is defined as $R(a) = \frac{P^r(a)}{C(a)}$. Notice that for a block that contains a single leaf test it is equivalent to our previous definition of R-ratio in `and-or` trees. A block is *rooted* at a test x if it starts with x and contains only tests from the subtree rooted at x.

The *best block for a node*, called "best strategy" in [26], is the block that maximizes the R-ratio, over all blocks rooted at this node. Consider any two tests x and y such that y is inside the subtree rooted at x. It turns out that if the best block for x contains y, then it also contains the entire best block for y, not interlaid by other tests. A *maximal best block* in a tree is the best block for some node that is not included in the best block for any other node. The optimal strategy for the entire `or`-tree performs one maximal best block after another, ordered by nonincreasing blocks' R-ratios. An equivalent description of an optimal strategy may be used. Instead of constructing and storing the maximal best blocks, we can rather store with each test the R-ratio of the best block for this test, called the *worth* of the test. Then the best-first strategy, that is the strategy that always performs the test with the highest worth over all available tests, is optimal for the tree.

The best block for a node is created by starting with the block that contains only the single test associated with the node, and then building it up as long as we can improve its R-ratio. Nodes are processed from bottom up, that is any test x is processed only after the best block for each node inside the subtree rooted at x has been already found. As mentioned before, whenever a test is added to a block, at the same time the entire best block for this test needs to be added to it.

Since any two tests are never considered separately after they have been included in one block, a block is treated as a single meta-node. Two blocks are combined (one sequence added at the end of another) by merging two meta-nodes into a single one. At the beginning each original node is a single block. Notice that at this stage if there is a directed path from a block a to a block b, then b has to be performed after a, but if there is no directed path between a and b then there is no restriction on the order of performing a and b. This property is maintained because a parent block a can be only merged with its child block b, by replacing a with ab, with the set of child nodes being the union of child nodes of a and b.

For a leaf node, the smallest, single-test block is the best block. Now consider an internal test x and the block a rooted at x that is initialized with x. We first recursively find the maximal best blocks for each child subtree of x. We then select the child block b of a with the highest R-ratio. If the R-ratio of b is not less than the R-ratio of a, then we combine a and b together, as described above: the new block will have higher R-ratio than a previously had. We repeat this process until no child block of a has higher R-ratio than a, at which point a is the best block for x and we are left with the maximal best blocks for the subtree rooted at x.

Smith proved the correctness of the algorithm under the assumption that the required value of each `or`-node is `true`, that is that performing an internal test can never resolve the entire tree. But the algorithm builds a best block by combining nodes together and then treats it as a single meta-node. In doing so, it creates nodes that are internal (that is have children) but that can resolve the entire tree (because they result from combining internal and leaf nodes). This is the intuitive argument why in fact Smith's Algorithm can deal with the presence of `or`-nodes with the required value `false` (that is internal tests whose success resolves the entire tree). We now explain formally this generalization to arbitrary required values of `or`-nodes.

SA and the proof of its correctness [26] deal with blocks and are based on the following expressions for the expected cost $C(a)$ and the resolving probability $P^r(a)$ of a block a:

For any test x we have

$$C(x) = c(x), \tag{4.1}$$

if a test x is a leaf node

$$P^r(x) = p(x), \tag{4.2}$$

if a test x is an internal node with the required value `true`

$$P^r(x) = 0. \tag{4.3}$$

Moreover if x is an internal node with the required value `true`, a parameter $L(x)$ is defined as follows:

$$L(x) = p(x). \tag{4.4}$$

Notice that the value of $L(x)$ is the value of the probability that child tests of x became available to perform after querying x.

For a sequence a of internal tests, $L(a)$ is defined as

$$L(a) = \prod_{x \in a} L(x), \tag{4.5}$$

where the product is taken over all tests from a.

Given blocks a and b, where b is rooted at its first test, the following expression calculates the expected cost and the resolving probability of the bigger block ab regardless of the required values of nodes:

$$C(ab) = C(a) + P_0(a,b)C(b), \tag{4.6}$$

$$P^r(ab) = P^r(a) + P_0(a,b)P^r(b), \tag{4.7}$$

where $P_0(a,b)$ is the probability that one starts performing the block b after performing a, that is that the block a fails to resolve the tree, but in such a way that performing b is still possible.

In the case when all internal nodes in an `or`-tree have the required value `true`, no internal node can resolve the tree, and performing b is still possible only if all internal nodes of a that are predecessors of the first test (thus of all tests) of b succeeded. Thus the expression for $P_0(a,b)$ used in [26] is

$$P_0(a,b) = L(a_b)(1 - P^r(a_{\bar{b}})), \tag{4.8}$$

where a_b is the subsequence of internal tests of a that are predecessors of the first test of b, $a_{\bar{b}}$ is the remaining subsequence of a.

Now assume that in a tree internal nodes can have both required values. For a block containing a single internal test x with the required value `false`, the resolving probability is $p(x)$. Moreover, in a block ab (with b rooted at its first test) we start performing the block b if and only if all internal tests from a that are predecessors of the first test of b, returned their required values (notice that at the same time it means that none of these tests resolved the tree) and the remaining subsequence of a did not resolve the tree. Observe that we may use exactly the same formulae (4.5) and (4.8) if we use the following expressions for internal `or`-nodes with required value `false`:

if x is an internal node with the required value `false`:

$$P^r(x) = p(x), \tag{4.9}$$

$$L(x) = 1 - p(x). \tag{4.10}$$

Thus if in addition to formulae (4.1) and (4.2) we use (4.3) and (4.4) for the `or`-nodes with the required value `true` and (4.9) and (4.10) for the `or`-nodes with the required value `false`, we obtain correct values of the expected cost and the resolving probability for single-test blocks and the same expressions to calculate these parameters for bigger blocks as the ones on which SA and the proof of its correctness rely, namely (4.5), (4.8), (4.6) and (4.7).

Smith's Algorithm for preconditioned `or`-trees is presented in Figure 4.2. The procedure Create_Blocks(x) builds maximal best blocks for a tree rooted at test x; the procedure Combine(x,y) combines blocks x and y into one block xy. Instead of using (4.8) directly to calculate $P_0(a,b)$, we rather keep for each block b the value $P_0(b) = P_0(a,b)$, where a is the node (block) that is the current parent of b. Notice that this is all we need since we add a block b at the end of a only in the case when a is the parent of b. If the parent a is a single test x then $P_0(x,b) = L(x)$. If a is being combined with other block c, we update $P_0(b)$ using the following expressions:

SA(preconditioned or tree T)

(1) Create_Blocks(root test of T)

(2) Order maximal best blocks from T by nonincreasing $R()$

Create_Blocks(test x)

(1) $C(x) := c(x)$

(2) If x is a leaf

(3) $\quad P^r(x) := p(x)$

(4) $\quad R(x) := \frac{P^r(x)}{C(x)}$

(5) Else

(6) $\quad P^r(x) := \begin{cases} 0 & \texttt{if required value of } x \texttt{ is true} \\ p(x) & \texttt{if required value of } x \texttt{ is false} \end{cases}$

(7) $\quad R(x) := \frac{P^r(x)}{C(x)}$

(8) $\quad$ For each child y of x

(9) $\quad\quad P_0(y) := \begin{cases} p(x) & \texttt{if required value of } x \texttt{ is true} \\ \bar{p}(x) & \texttt{if required value of } x \texttt{ is false} \end{cases}$

(10) $\quad\quad$ Create_Blocks(y)

(11) $\quad$ End For

(12) $\quad$ While x has child blocks

(13) $\quad\quad$ Find child y_{best} of x with maximum $R()$

(14) $\quad\quad$ If $R(y_{best}) < R(x)$ Then Go To (17)

(15) $\quad\quad$ Combine (x,y_{best})

(16) $\quad$ End While

(17) End Else

Combine (block x,block y)

(1) $P^r(x) := P^r(x) + P_0(y)P^r(y)$

(2) $C(x) := C(x) + P_0(y)C(y)$

(3) $R(x) := \frac{P^r(x)}{C(x)}$

(4) For each child y' of x other than y

(5) $\quad P_0(y') := P_0(y')(1 - P^r(y))$

(6) End For

(7) For each child z of y

(8) $\quad P_0(z) := P_0(y)P_0(z)$

(9) End For

(10) $x := xy$

(11) Add all children of y to the set of children of x

(12) Discard y

Figure 4.2: Smith's Algorithm (SA).

if block a is being added at the end of c, which means that in the current tree a is a child of c, we have

$$\begin{aligned} P_0(ca, b) &= L((ca)_b)\,[1 - P^r((ca)_{\bar{b}})] = L(c_a)L(a_b)\,[1 - P^r(c_{\bar{a}})]\,[1 - P^r(a_{\bar{b}})] = \\ &= P_0(c, a)P_0(a, b), \end{aligned} \tag{4.11}$$

whereas if block c is being added at the end of a, it means that in the current tree block c is a child of a, thus a sibling of b, therefore

$$\begin{aligned} P_0(ac, b) &= L((ac)_b)\,[1 - P^r((ac)_{\bar{b}})] = L(a_b)\,[1 - P^r(a_{\bar{b}})]\,[1 - P^r(c)] = \\ &= P_0(a, b)\,[1 - P^r(c)]\,. \end{aligned} \tag{4.12}$$

Observe that if each `or`-node from a tree has either the required value `false`, or the probability of success 1, then for any block b rooted at its first test and the parent block a of b we have $P_0(a, b) = 1 - P^r(a)$. Therefore in such a case we do not need to store $P_0(b)$.

Let n be the number of all tests (nodes) in the tree. Notice that except for the initialization of blocks which is performed n times, SA combines blocks at most n times. Each combining two blocks requires time linear in the number of child nodes of the combined blocks, so the worst case complexity of SA is $O(n^2)$.

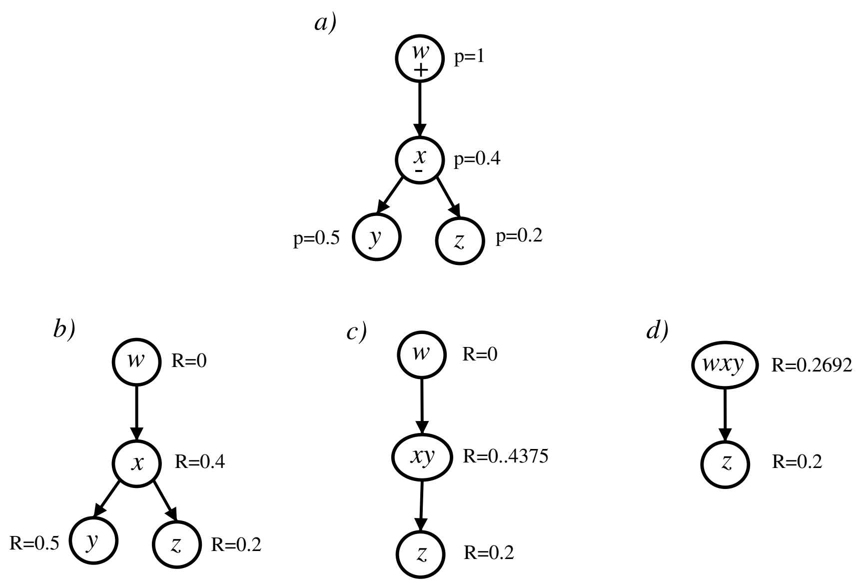

Figure 4.3: An example of using Smith's Algorithm. a) An algorithm's input: preconditioned `or`-tree T_p. All tests have unit cost. b) The initial blocks built by SA. c) The blocks after combining blocks x and y together. c) The blocks after combining blocks w and xy together. These blocks are maximal best blocks for T.

We will now discuss a simple example of using SA. The input preconditioned `or`-tree T_p is given in Figure 4.3a. Firstly after the initialization of blocks the structure of blocks is given in Figure 4.3b. Since all tests have cost one, so all blocks have cost one as well. The other parameters of the blocks have the values shown in Table 4.1.

$P^r(w) = 0$	$R(w) = 0$	
$P^r(x) = p(x) = 0.4$	$R(x) = 0.4$	$P_0(x) = p(w) = 1$
$P^r(y) = p(y) = 0.5$	$R(y) = 0.5$	$P_0(y) = \bar{p}(x) = 0.6$
$P^r(z) = p(z) = 0.2$	$R(z) = 0.2$	$P_0(z) = \bar{p}(x) = 0.6$

Table 4.1: Parameters of initial blocks shown in Figure 4.3.

Blocks z and y do not have any children. Now we process the block x. Among its child blocks, y has the highest R-ratio and $R(y) > R(x)$. So we combine blocks x and y together. The resulting blocks are shown in Figure 4.3c. The following parameters change:

$$\begin{aligned} P^r(xy) &= P^r(x) + P_0(y)P^r(y) = 0.4 + 0.6 \cdot 0.5 = 0.7, \\ C(xy) &= C(x) + P_0(y) \cdot C(y) = 1 + 0.6 \cdot 1 = 1.6, \\ R(xy) &= \frac{P^r(xy)}{C(xy)} = 0.4375, \\ P_0(z) &= P_0(z)\,(1 - P^r(y)) = 0.6 \cdot 0.5 = 0.3. \end{aligned} \tag{4.13}$$

The only child of the block xy is the block z and it has less R-ratio than xy has. Thus now we process the block w. This block has only one child xy whose R-ratio is higher than $R(w)$. Thus we combine w and xy and in this way obtain the blocks shown in Figure 4.3d. Notice that, according to the algorithm, $P_0(xy) = P_0(x)$. The following expressions describe parameters update:

$$\begin{aligned} P^r(wxy) &= P^r(w) + P_0(xy)P^r(xy) = 0 + 1 \cdot 0.7 = 0.7, \\ C(wxy) &= C(w) + P_0(xy) \cdot C(xy) = 1 + 1 \cdot 1.6 = 2.6, \\ R(wxy) &= \frac{P^r(wxy)}{C(wxy)} = \frac{7}{26} \approx 0.2692, \\ P_0(z) &= P_0(xy)P_0(z) = 1 \cdot 0.3 = 0.3. \end{aligned} \tag{4.14}$$

Since $R(z) < R(wxy)$, z and wxy are the maximal best blocks of T. By ordering them according to nonincreasing R-ratios we obtain the optimal strategy for T: $wxyz$.

As discussed at the beginning of this section, a preconditioned `and`-tree is equivalent, up to the negation of its value, to some preconditioned `or`-tree, namely the tree obtained by changing every `and`-node into `or`-node, negating output of every test and negating the required value for each internal node. In a preconditioned `and-or` tree the resolving probability is the probability that the entire tree evaluates to `false`. From this it follows that the obvious modification in lines (3) and (6) of the Create_Blocks procedure allow us to use the algorithm for preconditioned `and`-trees.

The *alternation number of a path* in an `and-or` tree is the number of arcs of the path whose ends are internal nodes with different (`or`/`and`) labels. The *alternation* *number of a tree* is the maximum alternation number of a path over all root-to-leaf paths in the tree. For a strictly alternating `and-or` tree the alternation number is equal to the tree's depth minus one. We call a preconditioned `and-or` tree a *k-alternation* tree if its alternation number is k.

Using the above definition, we may summarize the main result of the section that SA finds an optimal strategy for 0-alternation preconditioned `and-or` trees.

To end this section let us discuss a feature of the optimal strategy produced by SA that we will use in the next section. Consider an internal test x and let W be the set of all maximal best blocks from all child subtrees of x. Because any maximal best block has higher R-ratio than any of its child maximal best blocks (otherwise it would be combined with the best child block), so the best block for x is grown by combining it with maximal best blocks from W in the order of their nonincreasing R-ratio until all remaining blocks from W have lower R-ratio than the current block for x. Now assume that x is the root test of the entire tree. Observe that in this case it does not really matter whether we create the best block for x or not, because after building the best block for x, all remaining blocks from W are added at its end in the order of their nonincreasing R-ratios, to create the entire strategy. Therefore we can use the following description of the optimal strategy calculated by SA: given a set W of all maximal best blocks for all child subtrees of the root test x of a tree, the optimal strategy first perform the root test x and then, if the tree is not resolved yet, performs the blocks from W, ordered by their nonincreasing R-ratios.

4.2 1-Alternation And-Or Trees

We will present an extension of SA that finds an optimal strategy for some 1-alternation preconditioned `and-or` trees.

Let T be a 1-alternation preconditioned `and-or` tree and let A be a subtree of T. A is a *maximal pure included* subtree if A is 0-alternation subtree, is not a leaf node, and the parent node of the root of A has different label (`or`/`and`) than the internal nodes of A. If a maximal pure included subtree is an `or`-subtree (respectively `and`-subtree), we call the subtree a maximal `or`-subtree (respectively `and`-subtree).

We call an internal node *degenerate* if it is associated with a test with cost 0, success probability 1 and required value `true`. Observe that `and-or` trees are preconditioned `and-or` trees whose internal nodes are all degenerate. Also, once a test associated with an internal node has returned its required value, the node becomes degenerate.

The following algorithm, proposed in [12] is called DFA*: in a 1-alternation preconditioned `and-or` tree T, first run SA on each maximal pure included subtree of T. Then replace each maximal pure included subtree A by a leaf meta-test, whose cost is equal to the expected cost of the calculated strategy for A and whose success probability is equal to the probability that A evaluates to `true`. Given maximal best blocks for A found by SA, the expected cost and the resolution probability of the entire strategy can be easily calculated in the same way as SA calculates them for combined blocks; the resolution probability of the entire strategy is equal to the probability that the tree evaluates to `true`, if it is an `or`-tree, and to the probability that the tree evaluates to `false`, if it is an `and`-tree. After such a replacement, the tree is 0-alternation: use SA again to find an optimal strategy for it. In the resulting strategy we understand that any meta-test denotes the sequence of tests for the corresponding original subtree.

We call a strategy S *contiguous on a subtree* T' if on any root-to-leaf path of S, whenever a test from T' is performed, no test from outside T' is performed until the value of T' is determined. Since SA calculates an optimal strategy for any 0-alternation tree, so DFA* produces an optimal strategy for a 1-alternation tree T if and only if there exists an optimal strategy for T that is contiguous on any maximal pure included subtree of T. Unfortunately this is not true for all 1-alternation trees, as we shall discuss at the end of this section, but there are trees for which this condition hold. The following theorem specifies such trees.

Theorem 36 *DFA* produces an optimal strategy for a 1-alternation preconditioned* `and-or` *tree T if for each maximal pure included subtree A of T one of the following conditions is fulfilled:*

i) A is depth one and rooted at a degenerate internal node, or

ii) the required value of each internal node of A is `true` *if T is* `or`*-rooted, or* `false` *if T is* `and`*-rooted.*

Proof: Assume that T is an `or`-rooted preconditioned `and-or` tree fulfilling the conditions of the theorem. The proof for the other case is symmetric.

If a test from a maximal `and` subtree in T fails, the entire subtree evaluates to `false`. A strategy S is not contiguous on some maximal `and`-subtrees if and only if there is at least one node v of S such that v is labeled by a test from a maximal `and`-subtree A, the substrategy followed when this test is `true` starts with performing a test not in A, but contains at least one node labeled by a test from A. We will call such a node v a *violating node* of a subtree.

Let S be an optimal strategy for T. Let k be the number of violating nodes of S. We will show that there is an optimal strategy for T that is contiguous on any maximal `and` subtree (from which the theorem follows). The proof is by induction on k. The base case when $k = 0$ is trivial. Now assume that the statement holds if $k < K$, for some $K \geq 1$, and let S contain K violating nodes.

Let v be a violating node of S such that both child substrategies of v do not contain any violating node (there is at least one such node). Let S' be the strategy rooted at v and let T' be the corresponding reduced tree, evaluated by S'. Let x be the test that labels v and let A be the maximal `and`-subtree of T' that contains test x. Let T'_+, T'_- be the reduced trees obtained from T' when x is `true`, `false` respectively and let S'_+, S'_- be the substrategies of S' followed when x is `true`, `false` respectively.

If T' contains any `and`-subtree other than A, then it is evaluated as one meta-test by both S'_+ and S'_-. Thus we may replace any such subtree by a single meta-test, in other words we may assume that A is the only `and`-subtree of T'.

A test from a preconditioned `and-or` tree is available to perform if it is a root test, or if all internal nodes on the path from the root of the tree to the test are degenerate. We can collapse any degenerate node that is a child of a node with the

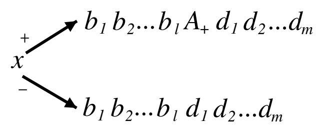

Figure 4.4: An illustration for the proof of Theorem 36; the substrategy S'.

same label (and/or), thus a test from a tree can be perform only if it is the root test or there is a path of degenerate internal nodes from the root of the tree to the parent node of the test strictly alternating between and-nodes and or-nodes.

Therefore in T' test x is either a root of A or a child of the root of A, and the root of A is a child of the root of T'. If x is false, the entire subtree A, and only it, disappears. Let A_+ be the reduced subtree obtained from A_+ when x is true. Notice that S'_+ evaluates A_+ as one meta-test. Thus we can treat A_+ as a one node, observe that it is a leaf child of the root of T'_+.

We may assume that S'_+ and S'_- are obtained by running SA on T'_+ and T'_- respectively (because if they are not, they may be replaced by such optimal strategies). Let W_+ and W_- be the sets of all maximal best blocks for all child subtrees of the root of T'_+ and T'_- respectively. S'_+ (respectively S'_-) first performs the degenerate root test and then preforms blocks from W_+ (respectively W_-) in order of their R-ratio. Since A_+ is a single leaf child of the root, so it is a separate maximal best block in W_+. All other subtrees of the root are the same for both T'_+ and T'_-, so $W_+ = W_- \cup \{A_+\}$. Let $b_1, b_2, \ldots, b_l$, $l \geq 1$ be all maximal best blocks from W_+ performed (in this order) by S'_+ before A_+, and $d_1, d_2, \ldots, d_m$, $m \geq 0$ be all (if any) maximal best blocks performed (in this order) by S'_+ after A_+. This means that S'_- performs maximal blocks $b_1, b_2, \ldots, b_l, d_1, d_2, \ldots, d_m$, in this order. S' is shown on Figure 4.4.

Let b be the block consisting of $b_1, b_2, \ldots, b_l$. Since A_+ is a child of the root test, so it does not depend on any internal test from the block b; A_+ is performed if b does not resolve the tree. For any $1 \leq i \leq m$ let $P_0(d_i)$ be the probability that $b, d_1, d_2, \ldots, d_{i-1}$ fail to resolve the tree but in such a way that performing d_i is still possible. Thus we have the following expression for the expected cost of the strategy S':

$$
\begin{aligned}
C(S') &= c(x) + \\
&\quad +p(x)\left[C(b) + (1 - P^r(b))C(A_+) + (1 - P^r(A_+))\sum_{i=1}^{m} P_0(d_i)C(d_i)\right] + \\
&\quad +\bar{p}(x)\left[C(b) + \sum_{i=1}^{m} P_0(d_i)C(d_i)\right] = \\
&= c(x) + C(b) + p(x)(1 - P^r(b))C(A_+) + \\
&\quad +\Big[p(x)(1 - P^r(A_+)) + \bar{p}(x)\Big]\sum_{i=1}^{m} P_0(d_i)C(d_i). \qquad (4.15)
\end{aligned}
$$

Now let S'^* be the linear strategy consisting of blocks $b, x, A_+, d_1, d_2, \ldots, d_m$ in this order. Test x also does not depend on any internal node from b. Thus the expected cost of S'^* is given by

$$
\begin{aligned}
C(S'^*) &= C(b) + (1 - P^r(b))c(x) + (1 - P^r(b))p(x)C(A_+) + \\
&\quad + \Big[\bar{p}(x) + p(x)(1 - P^r(A_+))\Big] \sum_{i=1}^{m} P_0(d_i)C(d_i) = \\
&= C(S') - P^r(b)c(x) \leq \\
&\leq C(S').
\end{aligned} \tag{4.16}
$$

Since S' is optimal for T', thus so is S'^*. We can replace the substrategy S' in the optimal strategy S by S'^*. But there is no violating node in S'^*, so after this replacement S has $K-1$ violating nodes, so by the inductive assumption there is an optimal strategy for T that is contiguous on any maximal `and`-subtree. □

If a 1-alternation tree is `and`-rooted and some `or`-nodes have the required value `true` or, equivalently, it is `or`-rooted and some `and`-nodes have the required value `false`, the strategy produced by DFA* is not necessary optimal. Figure 4.5 presents an `and`-rooted tree, for which the unique optimal strategy is not contiguous on the maximal `or`-subtree.

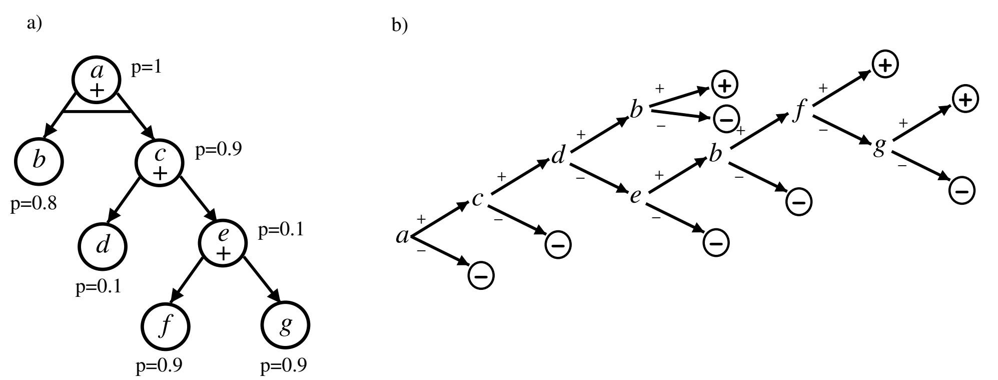

Figure 4.5: a) A 1-alternation `and`-rooted preconditioned `and-or` tree T_p. The required value of each internal node is `true`. All tests have unit costs. b) The unique optimal strategy for the tree T_p. This strategy is not contiguous on the maximal `or`-subtree.

While DFA* may also produce an suboptimal strategy for a tree that contains a depth one maximal pure included subtree rooted at a non-degenerate node, we can use a simple procedure to transform any depth one maximal pure included subtree into a depth one maximal pure included subtree rooted at a degenerate node. Let A be a depth one maximal pure included subtree of a 1-alternation tree T and let x be the test associated with the root of A.

Let L_1 be the label (`and`/`or`) of the root of T and let L_2 be the (different) label of the root of A. Now consider the following subtree A': the root v of A' is associated with test x but has label L_1. The root v has only one child node w which is a degenerate node with label L_2. The leaf nodes of A are the child nodes of w. Figure 4.6 illustrates the subtrees A and A'. Subtrees A and A' are equivalent. If we replace A by A' in the tree T, then instead of a depth one maximal pure

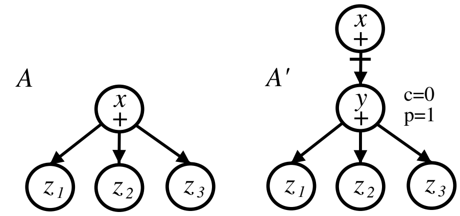

Figure 4.6: Two equivalent preconditioned `and-or` trees A and A'. The root of A is an `or`-node associated with a test x. The root of A' is an `and`-node associated with the test x, A' contains a depth one `or`-subtree rooted at a degenerate node.

included subtree rooted at a non-degenerate node, we have a depth one maximal pure included subtree with a degenerate root node.

Therefore, if all maximal pure included subtrees with depth greater than one fulfill the condition (ii) of Theorem 36, then after processing all depth one maximal pure included subtrees in a tree as described above we obtain a tree for which DFA* computes an optimal strategy. The processing takes constant time for every such subtree.

Chapter 5

Conclusion

The problem of finding an optimal strategy for an arbitrary `and-or` tree remains open. In this thesis we showed that for tests that are child nodes of the same internal node, there exists a relative order of performing the tests by an optimal strategy that does not depend on the other parts of the tree. Moreover, some of such sibling tests are always performed together. These findings led to the design of the Dynamic Programming Algorithm (DPA) to find an optimal strategy for `and-or` trees which runs in time $O(d^2n^d)$, where n is the number of tests in the tree and d is the number of internal nodes that are leaf-parents. For `and-or` trees with a bounded number of internal nodes this time is clearly polynomial in the trees' size. We also showed that the known efficient algorithm DFA produces an optimal strategy for depth three `and-or` trees whose all tests are identical (have the same cost and probability of success). For other type of trees with identical tests: parameter-uniform ladders, an optimal strategy also can be found in a simple, efficient way. On the other hand, we showed that the probabilistic and-or tree resolution for trees whose all tests have the same cost, but may have different success probability, can be used as an approximation of the problem with arbitrary costs.

We also studied a subclass of probabilistic Boolean expressions with precedence constraints imposed on the set of tests, called preconditioned `and-or` trees. We showed that an extension of Smith's Algorithm produces an optimal strategy for some type of such expressions.

We hope that the optimal order of performing sibling tests we described may be helpful in designing a polynomial-time algorithm to solve the problem, either for general `and-or` trees or at least for further subclasses (for example for depth three `and-or` trees). Such an algorithm does not necessary have to construct the entire strategy at once (as DPA or DFA does); it would be sufficient to show how to find in polynomial time the first test to be performed, as we can simply reduce the original tree, given the value of the first test and recurse.

If it turns out that PAOTR is NP-hard, then it would be of interest to find an approximation algorithm. The present known algorithms cannot be used in this way: DPA does not run in polynomial time for general `and-or` trees, whereas the strategy produced by DFA may be arbitrarily worse than the optimal one for some trees.

In the thesis we showed how to find an optimal strategy for two types of parameter-uniform trees. It is interesting whether PAOTR for parameter-uniform trees is simpler than the general problem; whether the algorithm to find an optimal strategy for `and-or` trees with identical tests can be designed.

Preconditioned `and-or` trees generalize `and-or` trees. We showed how to find an optimal strategy for a subset of 1-alternation preconditioned `and-or` trees; the important first step on the way to solving the general problem would be to discover an algorithm for an arbitrary 1-alternation tree.

Bibliography

[1] Jeffrey A. Barnett. How much is control knowledge worth? A primitive example. *Artificial Intelligence*, 22:77-89, 1984.

[2] 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.

[3] D. G. Corneil, H. Lerchs, L. Stewart Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3:163-174, 1981.

[4] D. G. Corneil, Y. Pearl, L. K. Stewart. A linear recognition algorithm for cographs. *SIAM Journal of Computing*, 14:926-934, 1985.

[5] Thomas H. Cormen. Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms* The MIT Press, Cambridge, Massachusetts, 2001.

[6] Michael R. Garey. Optimal task sequencing with precedence constraints. *Discrete Mathematics*, 4:37-56, 1973

[7] 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.

[8] Dan Geiger and Jeffrey A. Barnett. Optimal satisficing tree searches. *Proceedings of AAAI-91*, 441-445, 1991.

[9] Martin C. Golumbic, Aviad Mintz, Udi Rotics. Factoring and Recognition of Read-Once Functions using Cographs and Normality. *Proceedings of the 38th Design Automation Conference*, 109-114, 2001.

[10] Russell Greiner. Finding optimal derivation strategies in redundant knowledge bases. *Artificial Intelligence*, 50:95-115, 1991.

[11] Russell Greiner, Pekka Orponen. Probably approximately optimal satisficing strategies. *Artificial Intelligence*, 82:21-44, 1996.

[12] Russell Greiner, Ryan Hayward and Michael Molloy. Optimal depth-first strategies for and-or trees. *Proceedings of AAAI-02*, 725-730, 2002.

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

[14] Richard M. Karp and Yanjun Zhang. Bounded branching process and AND/OR tree evaluation. *Random Structures and Algorithms*, 7(2):97-116, 1995.

[15] H. Lerchs. On cliques and kernels. Technical Report, Dept. of Computer Science, University of Toronto, 1971.

[16] David Francis Manlove. Minimaximal and maximinimal optimisation problems: a partial order-based approach. PhD Thesis, University of Glasgow, 1998.

[17] K. S. Natarajan. Optimizing depth-first search of AND-OR trees. Report RC-11842, IBM Watson Research Center, 1986.

[18] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reasing, MA, 1995.

[19] Judea Pearl. Asymptotic properties of minimax trees and game-searching procedures. *Artificial Intelligence*, 14(2):113-138, 1980.

[20] Judea Pearl. *Heuristic*. Addison-Wesley, Reading, MA, 1984.

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

[22] Sartaj Sahni. Computationally related problems. *SIAM Journal on Computing*, 3(4):262-279, 1974.

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

[24] Miklos Santha. On the Monte Carlo boolean decision tree complexity of read-once formulae. *Random Structures and Algorithms* 6(1):75-88, 1995.

[25] Herbert A. Simon and Joseph B. Kadane. Optimal problem-solving search: all-or-none solutions. *Artificial Intelligence*, 6:235-247, 1975.

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

[27] Michael Tarsi. Optimal search on some game trees. *Journal of ACM*, 30:389-396, 1983.

[28] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134-1142, 1984.