3 OR-Tree Search

This chapter addresses a problem which we shall refer to as 'OR-tree' search[14]. The model that we address can be viewed as an investigation into the truth of a logical expression, L, which is a finite conjunction of logical primitives, l_i. L is a logical expression iff one of the following is true:

1. L is a logical primitive.
2. $L \equiv (X \cup Y)$, where X and Y are both logical expressions.

The symbol '$\cup$' represents the customary binary Boolean 'OR' operator, which is associative, and so the above definition implies the following:

$$L \equiv l_1 \cup l_2 \cup \dots l_N$$

For proofs, different types of logical primitive will be differentiated by numerical subscripts. However, when giving examples, we shall refer to logical primitives by capital letters, A, B, $C \dots$ so that a numerical subscript can be added to distinguish between logical primitives of the same type.

Sometimes it will be useful to have a label for logical expressions other than primitives. e. g. $Y \equiv (A \cup B)$. These we shall refer to by using uppercase letters, starting from X. Again, independent copies will be distinguished by use of numerical subscripts.

[14]The reader interested in rule-based methods of theorem proving should note that this chapter — indeed this thesis — does not contain any material of *direct* relevance.

3.1 Deterministic Case

We begin our consideration of this problem by restricting our attention to the case in which every logical primitive is a *box*. By *box* we mean a logical primitive which, when searched, will be shown for certain to be either true or false. The time taken to search a box may be a random variable, but we require that the expected time required to search it, t, is known. The probability, p, that it contains an object is also known.

We take the term 'box' from a classic and well-discussed case — see Dean [22], Joyce [35], Mitten [52] and Sweat [83] amongst others — which involves the search of a set of N boxes. In terms of the original model, p_i is the probability that box i contains an object, which is found if search is carried out on that box. We can view L, therefore, as the statement "one or more of the boxes $l_1 \ldots l_N$ contains an object". We seek to deduce a search policy which can determine, for sure, the truth of a general logical expression, L, in the smallest possible expected time.

This search is termed *satisficing*, since it aims to find a solution which is *good enough* for some specified purpose. It is to be contrasted with an optimising search which aims to find the *best* solution available. In contrast to an optimisation, satisficing search does not necessarily have a solution – in this case search proceeds until all the possible solutions have been shown not to meet the constraints which define what is 'good enough', whereupon the search terminates unsuccessfully. If, however, a solution is found, search may be terminated immediately.

We shall assume for simplicity that a search of a box which contains an object is certain to be successful, that is, an object will be found, although this restriction will be relaxed in Section 3.7.

Search continues until an object is detected, or until all the boxes have been unsuccessfully searched. A policy is termed *optimal* if it searches boxes in such a way that it minimises the payoff, V, equal to the expected time until termination.

In the model described above, each policy can be identified with what Kadane and Simon [37, 77] term a *strategy*, that is, a permutation of the integers from 1 to N. A strategy, A, represents the policy which searches the boxes in the order of the integers given by A, ceasing as soon as an object is detected or if all the boxes have been searched.

Consider an arbitrary strategy, $A = \{a_1, a_2 \dots\ a_n\}$. Using $\mathbf{x}$ to represent the state, we denote by $V_A(\mathbf{x})$ the expected termination time of applying strategy A to search the boxes. We shall use q_i to denote the probability that an object is not found when box i is searched, and $p_i = 1 - q_i$.

Hence:

$$V_A(\mathbf{x}) = \sum_{i=1}^{N} t_{a_i} \prod_{j=1}^{i-1} q_{a_j}$$

We now modify A by interchanging the k^{th} and $k+1^{th}$ elements to obtain $A' = \{a_1, a_2 \dots a_{k-1}, a_{k+1}, a_k, a_{k+2}, \dots\ a_n\}$, and consider the payoff from applying this modified strategy:

$$V_{A'}(\mathbf{x}) = \sum_{i=1}^{k-1} t_{a_i} \prod_{j=1}^{i-1} q_{a_j} + t_{a_{k+1}} \prod_{j=1}^{k-1} q_{a_j} + q_{a_{k+1}} t_{a_k} \prod_{j=1}^{k-1} q_{a_j} + \sum_{i=k+2}^{N} t_{a_i} \prod_{j=1}^{i-1} q_{a_j}$$

It is instructive to consider the expected difference in the time taken by policies A and A':

$$\begin{aligned} V_A(\mathbf{x}) - V_{A'}(\mathbf{x}) &= \sum_{i=k}^{k+1} t_{a_i} \prod_{j=1}^{i-1} q_{a_j} - t_{a_{k+1}} \prod_{j=1}^{k-1} q_{a_j} - q_{a_{k+1}} t_{a_k} \prod_{j=1}^{k-1} q_{a_j} \\ &= \prod_{j=1}^{k-1} q_{a_j} (t_{a_k} + (1 - p_{a_k}) t_{a_{k+1}} - t_{a_{k+1}} - t_{a_k}(1 - p_{a_{k+1}})) \\ &= \prod_{j=1}^{k-1} q_{a_j} (p_{a_{k+1}} t_{a_k} - p_{a_k} t_{a_{k+1}}) \\ &> 0 \qquad \Leftrightarrow \frac{p_{a_{k+1}}}{t_{a_{k+1}}} > \frac{p_{a_k}}{t_{a_k}} \end{aligned}$$

This simple interchange argument shows that strategy A' is an improvement upon strategy A if the exchanged boxes, a_k and a_{k+1}, were not in decreasing order of $\frac{p}{t}$. If we define the *reward rate*, $\emptyset_i = \frac{p_i}{t_i}$, of a box i, we see that the optimal policy must therefore be to search the boxes in decreasing order of $\emptyset$.

3.1.1 Linear Precedence Constraints

We now suppose that some restrictions exist as to the order in which the boxes can be searched. Specifically, we consider the case of linear precedence constraints, as illustrated below.

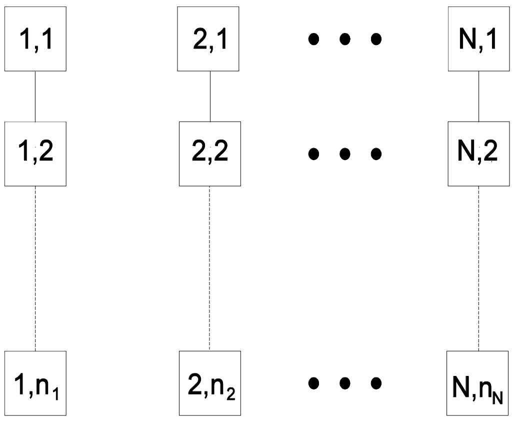

Figure 14: Linear Precedence Constraints

Such a constraint structure arises if we assume that the boxes are arranged in N stacks, so that box (i, j) is the j^{th} box in stack i, and the searcher is limited to searching only the uppermost unsearched box of each stack. The goal of search, to discover whether there is an object, remains unchanged.

There are now two factors that influence the choice of box; not only the immediate possibility of finding an object, but also the future benefit from access to boxes below must be considered. This problem has a straightforward solution, first shown by Mitten [52].

Let us begin by considering briefly the case in which the boxes are arranged so that the $\emptyset_i$ decreases from the top of each stack to the bottom.

In this case, it is possible to select the boxes in decreasing order of $\emptyset_i$, and so the same payoff is possible. This is, therefore, optimal by the trivial observation that restricting the possible actions cannot possibly improve the payoff. We now show that it is possible for any boxes problem with linear precedence constraints to find an equivalent one with this structure.

Suppose $\emptyset_{(i,j)} < \emptyset_{(i,j+1)}$. We show that this implies that boxes (i,j) and $(i,j+1)$ belong to the same *indivisible block*. By this we mean that it cannot be optimal to immediately follow search of (i,j) by search of any box other than $(i,j+1)$. To see this, we compare the payoffs of the following three policies:

π_1: Search some other box(es), k, then box (i,j), then box $(i,j+1)$.

π_2: Search box (i,j), then some other box(es), k, then box $(i,j+1)$.

π_3: Search box (i,j), then box $(i,j+1)$, then some other box(es), k.

(It is understood that the policies only search if necessary — i.e. that they terminate if an object is found). Let us denote by p_k the overall probability that the other box(es) contain an object and by t_k the expected time to search them.

$$\begin{aligned}
V_{\pi_1} &= t_k + q_k t_{(i,j)} + q_k q_{(i,j)} t_{(i,j+1)} \\
V_{\pi_2} &= t_{(i,j)} + q_{(i,j)} t_k + q_k q_{(i,j)} t_{(i,j+1)} \\
V_{\pi_3} &= t_{(i,j)} + q_{(i,j)} t_{(i,j+1)} + q_{(i,j)} q_{(i,j+1)} t_k
\end{aligned}$$

Hence:

$$\begin{aligned}
V_{\pi_2} - V_{\pi_1} &= t_{(i,j)} + q_{(i,j)}t_k - t_k - q_k t_{(i,j)} \\
&= p_k t_{(i,j)} - p_{(i,j)} t_k \\
&= t_{(i,j)} t_k (\emptyset_k - \emptyset_{(i,j)}) \qquad (8) \\
V_{\pi_2} - V_{\pi_3} &= q_{(i,j)} t_k + q_k q_{(i,j)} t_{(i,j+1)} - q_{(i,j)} t_{(i,j+1)} - q_{(i,j)} q_{(i,j+1)} t_k \\
&= q_{(i,j)} (p_{(i,j+1)} t_k - p_k t_{(i,j+1)}) \\
&= q_{(i,j)} t_k t_{(i,j+1)} (\emptyset_{(i,j+1)} - \emptyset_k) \qquad (9)
\end{aligned}$$

Equation (8) implies that if $\emptyset_k > \emptyset_{(i,j)}$ then policy π_1 achieves a lower payoff than policy π_2, while equation (9) implies that if $\emptyset_k < \emptyset_{(i,j+1)}$ then policy π_3 achieves a lower payoff than policy π_2. At least one of these conditions applies, since $\emptyset_{(i,j+1)} > \emptyset_{(i,j)}$, and so policy π_2 is therefore not optimal.

The solution then proceeds by processing the boxes, starting from the bottom of each stack, grouping them into indivisible blocks wherever possible. This process of grouping together separate searches into a single unit we shall refer to as *chunking*. Once no more chunking can be carried out, the indivisible blocks are said to be *maximal*. In this case, it is a consequence of the criterion for chunking that the maximal indivisible blocks within a stack must now be in order of decreasing $\emptyset$, and so the optimum policy is just to search them in decreasing order of $\emptyset$.

3.2 Stochastic Case

We now increase the scope of the model to cater for a more general class of logical primitives, to allow dynamic revelation of nodes (previously referred to as boxes) in the course of search. We assume that all nodes belong to one of n different types, the details of which are known.

Satisficing search on an out-tree differs from more familiar tree search models, such as shortest path search, in that the structure of nodes already searched is of no importance in guiding further search. Apart from the set of nodes now available for search, the only relevant result of previous search is whether or not an object has been found. The state of an ongoing search problem may therefore be represented by $\mathbf{x}$, a vector of length n that contains the number of nodes of each type that are currently visible.

Subsection 3.3.2 compares the values of different problems and so there the state of a search will be represented $(\mathbf{x}, \mathbf{d})$ where $\mathbf{d}$ is a vector of length n which summarises the details of each type of node.

$$\mathbf{d} = (d_1, d_2 \ \dots \ d_n) \text{ where } d_i = (p_i, t_i, f_i)$$

The definitions of p_i and t_i are unchanged, whilst we use $f_i(s)$ to denote the *offspring distribution* of node i. This is defined as the n-dimensional probability distribution of extra nodes revealed when a type i node is searched.

We shall use familiar notation, $V_\pi(\mathbf{x}, \mathbf{d})$, to refer to the expected time taken to terminate starting from state $\mathbf{x}$ with nodes of type $\mathbf{d}$, when policy π is applied. In cases where $\mathbf{d}$ is fixed, we shall abbreviate this as $V_\pi(\mathbf{x})$. The optimal value, $V(\mathbf{x})$ is given by:

$$V(\mathbf{x}) \;=\; min_{\pi}\{V_{\pi}(\mathbf{x})\}$$

A (non-randomised) policy π is a (deterministic) function of a state $\mathbf{x}_\mathbf{i}^\pi$, with past history, $\mathbf{H}_\mathbf{i} = (\mathbf{x_0}^\pi, a_1, \mathbf{x_1}^\pi, a_2 \;\ldots \mathbf{x_i}^\pi)$, where $a_i \in \{1 \;\ldots\; n\}$. Denote the $\tau + 1^{\text{th}}$ action taken by π, as $\pi_\tau(\mathbf{H}_\tau)$. A Markov policy is a policy which does not take the past history of states or actions into account, so can be expressed $\pi(\mathbf{x}_\mathbf{i}^\pi) \;=\; a_i$. Since this is a one player game, there is an optimal policy amongst the class of non-randomised Markov policies. We shall therefore restrict our attention for the rest of this paper to these policies, so any policy mentioned may be assumed to be both non-randomised and Markov.

We shall expand the state space by adding the special state, $\langle \mathrm{T} \rangle$, which corresponds to having found an object. This must be a trapping state. The expected time to search for an object from this state is always 0. From any other state, action a may only be taken if there is a node of that type available for search. In such a case, the expected time required to take action a is a constant, t_a, so the cost, $c(\mathbf{x}, a)$, of taking action a from state $\mathbf{x}$ satisfies:

$$E[c(\mathbf{x}, a)] = t_a I_{\mathbf{x} \neq \langle \mathrm{T} \rangle}$$

A node type is termed *most rewarding* if it achieves the maximum reward rate. A node is termed *most rewarding* if it is of a most rewarding type. In the exposition that follows, we use I^* to denote the set of most rewarding node types.

We define an *exhaustive search* of type i nodes as a sequence of searches of type i nodes, which terminates either upon finding an object or when there are no more nodes of type i available for search, whichever happens first.

Now define a *k-exhaustive search* of nodes of type i as a sequence of searches of type i nodes which terminates either as soon as it finds an object or when there are no more nodes of type i are available, or when it has carried out k searches, whichever happens first. The above defined exhaustive search is therefore equivalent to an ∞-exhaustive search by this definition.

A *simple-minded* policy is defined as a policy which carries out an exhaustive search of the most rewarding node type(s) whenever possible.

We now explain the equivalence with tree search. Each logical primitive corresponds to a node in the search tree. If search reveals the node to contain an object, this is interpreted as discovering that the corresponding logical primitive is true. We shall disregard degenerate search 'opportunities' — nodes which have no probability of containing an object — and so the only way to conclude for certain that L is false is to show that each of the L_i is false. This corresponds to the game ending when all the nodes are exhausted.

3.3 Nature of the Optimal Policy

Our analysis proceeds in three stages. In Subsection 3.3.1, we prove that the optimal policy must be simple-minded. This establishes the optimal action from states in which there are any most rewarding nodes available for search. We then show how it is possible to treat an exhaustive search in a similar fashion to the search of a single node. The next section then uses this to prove Theorem 3.2, which then establishes a connection between a problem with n node types and a problem with $n-1$ transformed node types.

Finally, Subsection 3.3.3 establishes that dynamic programming may be used to solve the problem by recursive application of Theorem 3.2, proving the optimal policy to be a simple priority order rule. The structure of this proof is identical to that of the proof by Tsitsiklis[85] for semi-Markov bandits.

3.3.1 A Restriction on the Optimal Policy

To simplify the following theorem, we slightly modify our conception of the game. Suppose that, rather than stopping once an object has been found, all policies keep searching as long as any boxes are available for search. The equivalence between the games is maintained by assuming that searches carried out once an object has been discovered are made at no cost. This construction is useful since it allows conditioning upon the states encountered to be independent of whether or not an object has been found.

The state we denote as $(W, \mathbf{X})$, where X has its previous meaning, and W keeps track of whether an object has been found, assuming value 0 if an object has been found, and value 1 if not. Hence:

$$E[c(W, \mathbf{X}, a)] = Wt_a$$

Theorem 3.1 *A policy which is not simple-minded cannot be optimal.*

Proof: Consider an arbitrary policy, π, which is not simple-minded. It is shown that there exists a policy, π', which achieves a strictly better payoff. Define the stopping time, $N < \infty$, as the time at which policy π first breaks the simple-minded criterion. (That is $\pi_N(\mathbf{x}_N) \notin I^*$, although for some $i^* \in I^*$, $\mathbf{x}^{\pi}_{N_{i^*}} > 0$ and so i^* would have been a legal action in state $\mathbf{x}^{\pi}_N$). The payoff achieved by a policy π, $V_\pi[\mathbf{x}]$, we break into three parts. The expected cost of actions taken from states $\mathbf{X}_0$ up to $\mathbf{X}_{N-1}$ is written A. The expected cost of actions taken from the state in which policy π first misses a chance of searching a node of type i^*, $\mathbf{X}_N$, until the state in which it next takes the chance, $\mathbf{X}_M$, is written B. The expected cost of actions after this point is written C. Note that the time at which policy π first takes action i^* again, $M \leq \infty$, is a stopping time. (If policy π never does, $M = \infty$ and $C = 0$).

$$
\begin{aligned}
V_\pi[\mathbf{x}] &= E\left[\sum_{i=0}^{\infty} c(W_i^\pi, \mathbf{X}_i^\pi, \pi_i(\mathbf{X}_i^\pi))\right] = A + B + C \\
\text{where :} & \\
A &= E\left[\sum_{i=0}^{N-1} c(W_i^\pi, \mathbf{X}_i^\pi, \pi_i(\mathbf{X}_i^\pi))\right] \\
B &= E\left[\sum_{i=N}^{M} c(W_i^\pi, \mathbf{X}_i^\pi, \pi_i(\mathbf{X}_i^\pi))\right] \\
&= E\left[\sum_{i=N}^{M-1} c(W_i^\pi, \mathbf{X}_i^\pi, \pi_i(\mathbf{X}_i^\pi)) + c(W_M^\pi, \mathbf{X}_M^\pi, i^*)\right] \\
C &= E\left[\sum_{i=M+1}^{\infty} c(W_i^\pi, \mathbf{X}_i^\pi, \pi_i(\mathbf{X}_i^\pi))\right]
\end{aligned}
$$

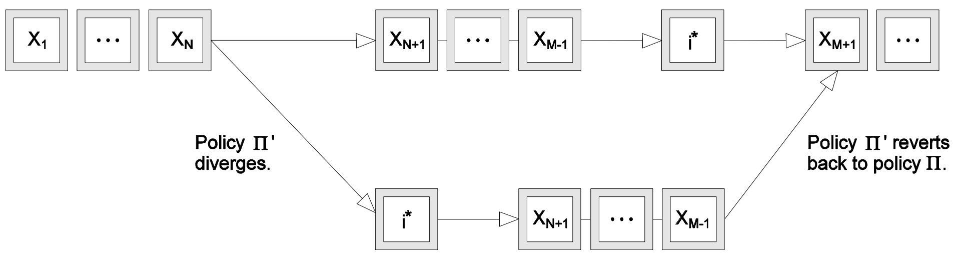

Figure 15: Alternative Policy π'

Consider a policy π' which mimics policy π until time N, when π first misses an opportunity to search a node of some type $i^* \in I^*$. Policy π' now deviates by searching a node of type i^*. This is admissible, since N is a stopping time. The search of a node of type i^* may result in extra nodes of other types being added to the state, but it cannot result in such nodes being removed. It is therefore an admissible policy to mimic the actions taken by policy π once more. Policy π' does this until time M. If $M < \infty$, then at this time policy π searches the node of type i^* which π' searched earlier, and the two policies reconverge and play identically once more.

Denoting by $\mathbf{x} + y$ the state arrived at from $\mathbf{x}$ when a node of type i^* is expanded, policy π' leads to the following series of states:

$$\{\mathbf{X}_0^\pi, \mathbf{X}_1^\pi, \ \ldots \ \mathbf{X}_N^\pi, \mathbf{X}_N^\pi + y, \mathbf{X}_{N+1}^\pi + y, \ldots \ \mathbf{X}_{M-1}^\pi + y, \mathbf{X}_{M+1}^\pi, \mathbf{X}_{M+2}^\pi, \ \ldots \}$$

The payoff achieved by policy π' we express, as before, as the sum of three parts: A' is the expected cost of the first N actions, B' of the next $M - N$, and C' of the remainder:

$$V_{\pi'}[\mathbf{x}] = E\left[\sum_{i=0}^{\infty} c(\mathbf{X}_i^{\pi'}, \pi_i'(\mathbf{X}_i^{\pi'}))\right] = A' + B' + C'$$

Where:

$$\begin{aligned}
A' &= E\left[\sum_{i=0}^{N-1} c(W_i^{\pi'}, \mathbf{X}_i^{\pi'}, \pi_i'(\mathbf{X}_i^{\pi'}))\right] = A \\
B' &= E\left[\sum_{i=N}^{M} c(W_i^{\pi'}, \mathbf{X}_i^{\pi'}, \pi_i'(\mathbf{X}_i^{\pi'}))\right] \\
&= E\left[c(W_i^{\pi'}, \mathbf{X}_N^{\pi'}, i^*) + \sum_{i=N+1}^{M} c(W_i^{\pi'}, \mathbf{X}_i^{\pi'}, \pi_i'(\mathbf{X}_i^{\pi'}))\right] \\
C' &= E\left[\sum_{i=M+1}^{\infty} c(W_i^{\pi'}, \mathbf{X}_i^{\pi'}, \pi_i'(\mathbf{X}_i^{\pi'}))\right] = C
\end{aligned}$$

$$\begin{aligned}
V_\pi[\mathbf{x}] - V_{\pi'}[\mathbf{x}] &= (A + B + C) - (A' + B' + C') = B - B' \\
&= E\left[[E\ D|\, \pi_N(\mathbf{X}_N^\pi) = a_N, \ldots \pi_M(\mathbf{X}_M^\pi) = a_M]\right]
\end{aligned}$$

We consider D, the difference in expected payoff of policies π and π', conditional upon the sequence of actions taken.

$$
\begin{aligned}
D &= \sum_{i=N}^{M-1} c(W_i^{\pi}, \mathbf{X}_i^{\pi}, \pi(\mathbf{X}_i^{\pi})) + c(W_M^{\pi}, \mathbf{X}_M^{\pi}, i^*) \\
&\quad -c(W_N^{\pi'}, \mathbf{X}_N^{\pi'}, i^*) - \sum_{i=N+1}^{M} c(W_i^{\pi'}, \mathbf{X}_i^{\pi'}, \pi_i(\mathbf{X}_i^{\pi'})) \\
&= \sum_{i=N}^{M-1} W_i^{\pi} t_i + W_M^{\pi} t_{i^*} - W_N^{\pi'} t_{i^*} - \sum_{i=N+1}^{M} W_{i-1}^{\pi'} q_{i^*} t_{a_{i-1}} \\
&= p_{i^*} \sum_{i=N}^{M-1} t_{a_i} \prod_{j=N}^{i-1} q_{a_j} + t_{i^*}(\prod_{j=N}^{M} q_j - 1) \\
&= t_{i^*} \left(\sum_{i=N}^{M-1} \frac{p_{i^*}}{t_{i^*}} t_{a_i} \prod_{j=N}^{i-1} q_{a_j} + \prod_{j=N}^{M} q_j - 1 \right)
\end{aligned}
$$

Since i^* is a most rewarding node, $\frac{p_{i^*}}{t_{i^*}} > \frac{p_i}{t_i}\ \forall i \notin I^*$.

$$
\begin{aligned}
&= t_{i^*} \left(\sum_{i=N}^{M-1} \frac{p_{a_i}}{t_{a_i}} t_{a_i} \prod_{j=N}^{i-1} q_{a_j} + \prod_{j=N}^{M} q_j - 1 \right) \\
&= t_{i^*} \left(\sum_{i=N}^{M-1} p_{a_i} \prod_{j=N}^{i-1} q_{a_j} + \prod_{j=N}^{M} q_j - 1 \right) \\
&> 0
\end{aligned}
$$

■

We have now established that the optimal policy is simple-minded. This is equivalent to stating that it carries out an exhaustive search for most rewarding nodes at every opportunity. Rather than considering a single search we now consider search of a type i node followed by an exhaustive search for nodes of type i^*. We denote as $\hat{p}_i(i^*)$ the probability of finding an object with such a chuck of search. The expected time taken to search such a chunk we denote $\hat{t}_i(i^*)$, and let us use $\hat{f}_i(i^*)$ to represent the distribution of nodes revealed. We can now use these values to enable the mathematical treatment of this chunk of search as if it were the expansion of a single node.

3.3.2 An Equivalence Between Search Problems

This section leads to a theorem which establishes an identity between $V(\mathbf{x}, \mathbf{d})$ and the value of a modified problem $V(\mathbf{x}', \mathbf{d}')$ which has one less node type.

Theorem 3.2 *If i^* is a most rewarding node type then*

$$\begin{gathered} V((x_1 \dots x_{i^*-1}, 0, x_{i^*+1} \dots x_N), (d_1 \dots d_N)) \\ = V((x_1 \dots x_{i^*-1}, x_{i^*+1} \dots x_N), (d'_1 \dots d'_{i^*-1}, d'_{i^*+1} \dots d'_N)) \end{gathered}$$

where the transformed node types satisfy

$$d'_i = (\hat{p}_i(i^*), \hat{t}_i(i^*), \hat{f}_i(i^*))$$

Proof: Denote by U' the space of strategies applicable to the game $(\mathbf{x}', \mathbf{d}')$, and by U the space of simple-minded strategies applicable to the game $(\mathbf{x}, \mathbf{d})$. There is a bijection between the two, since any $u' \in U'$ may be expressed as a sequence of substrategies, each of which corresponds to searching a single node, i.e. $u' = (u'_1, u'_2 ...)$, whilst any $u \in U$ may be represented as a conjunction of sub-strategies e_k and u_k, where the $\{e_.\}$ are substrategies that carry out a (possibly zero length) exhaustive search of nodes of type i^*, and the $\{u_.\}$ are searches of a single node of any type other

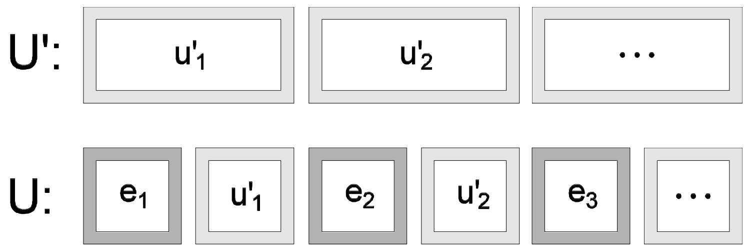

Figure 16: Policy Spaces U and U'

than k. The strategy u may be represented as $u = (e_1, u_1, e_2, u_2 \ldots)$ because it is simple-minded.

The value obtained by applying strategy $u = (e_1, u_1, e_2, u_2 \ldots)$ to game $(\mathbf{x}, \mathbf{d})$ is exactly that obtained by applying strategy $s(u) = (u'_1, u'_2 ...)$ to game $(\mathbf{x}', \mathbf{d}')$, since the effects of carrying out an exhaustive search of nodes of type i^* are accounted for by the transformations $d'_i = (\hat{p}_i(i^*), \hat{t}_i(i^*), \hat{f}_i(i^*))$.

The above bijection argument is easily extensible to policies; if policy π applies strategy u to game $(\mathbf{x}, \mathbf{d})$, define policy $s(\pi)$ to be the policy which applies strategy $s(u)$ to game $(\mathbf{x}', \mathbf{d}')$. This includes the optimum policy, π^*, of game $(\mathbf{x}', \mathbf{d}')$ which Theorem 3.1 tells us is simple-minded:

$$V(\mathbf{x}', \mathbf{d}') = V_{s(\pi^*)}(\mathbf{x}', \mathbf{d}') = V_{\pi^*}(\mathbf{x}, \mathbf{d}) = V(\mathbf{x}, \mathbf{d})$$

$\blacksquare$

3.3.3 Proof of Optimal Policy

Theorem 3.3 *For a game with n types of node there exists a permutation, $(y_1, \ldots y_n)$ of the integers 1 to n, termed the* optimal ordering*, such that policy π is optimal if it searches a node of type y_k iff $k = min\{i : y_i > 0\}$.*

Proof: The theorem is trivially true for n=1. For $n > 1$, an induction argument applies. Theorem 3.1 proves that any optimal policy, π, for the game $(\mathbf{x}, \mathbf{d})$ is simple-minded. Since π searches nodes of type i^* iff $x_{i^*} > 0$, let $y_1 = i^*$. The optimal policy has only to be determined when $x_{i^*} = 0$. In this case, Theorem 3.2 proves that for

$$\begin{aligned}\mathbf{x}' &= (x_i \ldots\ x_{i^*-1}, x_{i^*+1}, \ldots\ x_n) \\ \mathbf{d}' &= (d'_i \ldots\ d'_{i^*-1}, d'_{i^*+1}, \ldots\ d'_n) \\ d'_i &= (\hat{p}_i(i^*), \hat{t}_i(i^*), \hat{f}_i(i^*))\end{aligned}$$

$$V(\mathbf{x}, \mathbf{d}) = V(\mathbf{x}', \mathbf{d}')$$

Now let $(y_2,\ \ldots\ y_n)$ be the optimal ordering of the game $(\mathbf{x}', \mathbf{d}')$, which involves only $n - 1$ different types of node, so optimal play is a consequence of the induction hypothesis. ■

The proof considers n different games, since the equivalence proved by Theorem 3.2 is applied $n - 1$ times. Each node type is optimal to search in at least one of these games, so consider a node type in a game for which it is optimal. The transformed node details of such a node type account for the chunking of any more rewarding offspring. We therefore define p^*_i, the *corrected* probability that a node of type i contains an object, as the p_i value of this game. We define t_i^*, the *corrected* expected time taken to search a node of type i, in a similar way. The ratio of these two we term the *corrected reward rate*, denoted $\varnothing_i^*$. This is a true reflection of the box type's reward rate in the sense that $1/\varnothing_i^*$ is the expected time to find an object when the only available boxes are an unlimited supply of boxes of type i.

3.4 The OR-Tree Model in Practice

The model may be used for situations with a problem which requires individual searches to be carried out in an effort to find a 'solution' (an object). The domain of applicability is narrowed by the satisficing nature of the model. This imposes the following requirements:

1. A problem may have one, none or many solutions.
2. All the solutions are of equal value.

The model also requires that the performance indicator is the *expected use* of some resource (typically time or money) while the problem is being investigated. For the model to be of use, the search must also be of a kind which can be broken down into independent units, of which there are finitely many categories, each of which has its own expected resource requirement, its own known chance of yielding a solution and of creating other search units. The original model also allows search to terminate only when a solution is found, or when no further units remain to be searched, although we shall relax this requirement in Section 3.6.

Note that ‘finding a solution’ does not necessarily imply a favourable outcome. In the drug-testing example that follows, if a solution is found then the drug being tested has failed the required tests. Another such example is the task of scheduling a difficult multi-stage manufacturing process. Each stage of the manufacturing process is a node type. Its resource cost is straightforward, while the chance of its finding a solution corresponds to the chance it goes wrong and permanently destroys the product.

A useful consequence of the generality of the model is that it can be applied without modification to cases in which some of the tree is known in advance. This may be achieved by the addition of extra node types. In the general case there is a one-to-one correspondence between the extra node types and the nodes in the out-tree which are known. The search time, t_i and probability of containing an object, p_i, of a node type are set so as to match these values of the node in the known out-tree. The internal nodes in the known tree are represented by node types which have offspring distributions which reflect the fact that it is known for certain exactly which nodes will be made available. In the case where the previously known out-tree contains subtrees which are identical, the number of node types added to the model may be reduced by assigning the matching nodes in the tree to the same node type. The problem of searching a set of boxes or a tree which is wholly pre-determined [23, 52] is thus a special case of this model.

3.4.1 Complexity and Ill-conditioning

Selection of the most rewarding node requires $O(n)$ computations and must be done n times, so this part of the algorithm has complexity $O(n^2)$. The time required to transform the node type details by the equations of Theorem 3.2 depends upon the properties of the descendant distributions, f_i. Repeated slight inaccuracies during calculation of the values of $\hat{p}()$, $\hat{t}()$ and $\hat{f}()$ may be compounded and hence lead to greater inaccuracies later on in calculations, because of the recursive nature of the algorithm. Complexity problems are therefore likely to arise in determining the optimal policy in cases with large numbers of node types and complicated descendent distributions. Such ill-conditioning arises particularly when two or more node types have very similar reward rates, and so the reward rates of the nodes concerned must be calculated to a great accuracy to determine which is optimal. However the practical consequence of such ill-conditioning is that even a sub-optimal policy can be expected to achieve a payoff which is very close to optimal.

If the model is applied to problems which are to be studied repeatedly in real time then the fact that it is computationally expensive to *deduce* the optimal policy may be of little relevance. The model has the great advantage that the optimal policy, once deduced, is very easy to store and apply; it need only be deduced once, and can then be applied swiftly in all states that arise. This feature makes the model suitable for applications such as computer game playing in which speed of exercising sequences of optimal control decisions is at a premium.

In practical applications with a large number of node types which exhibit ill-conditioning, some degree of approximation may well be required. The tension between the wish to deduce a strictly optimal policy and the need to limit the complexity of the calculations required is a matter to be determined by the relative costs involved.

3.4.2 Optimal Search for Conspiracies of Size 1

Although conspiracy numbers may be applied to the usual multi-valued trees, we now consider their application to trees in which all the nodes are scored with one of two values. With a certain loss of information, any evaluation of any game tree may be treated in this way.

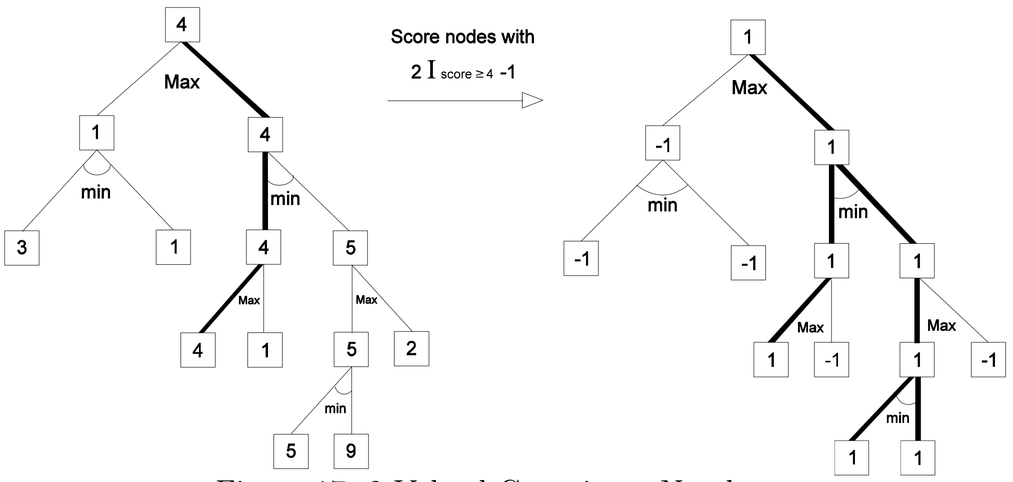

Figure 17: 2-Valued Conspiracy Numbers

Two-valued evaluation functions simplify the calculations about and expositions of conspiracy-based techniques, since the only nodes which can conspire are nodes on the principal variation (marked in bold in Figure 17). Consider an evaluation function which assigns nodes one of two values, $H(x) \mapsto \{-1, 1\}$. These may be thought of as 'probable win' and 'probable loss' nodes. We follow Knuth and Moore's negamax notation [44]. Hence, assume that there are two sorts of node, -1 and 1. For the sake of simplicity of exposition, let us assume that the game tree and evaluation function are such that:

$$\begin{aligned} d \text{ is a daughter of } p \quad \Rightarrow \quad P[H(d) = -H(p)] &= 1 - \delta \\ P[H(d) = H(p)] &= \delta \end{aligned}$$

We now show how the tree search model we have developed can be used to analyse the process of searching for conspiracies of size one, and to deduce the optimal search order. The first step of the original conspiracy number search, upon being given a game tree, is to search it for conspiracies of size one. That is, to search it either until expansion of a single leaf has the effect of changing the minimax backed up score at the root, or until we can conclude that there are no more *single* leaf nodes which can do this.

This problem can be seen to fit into the satisficing out-tree search model framework without further adjustment. The state of the investigation may be represented by L, a list of the critical leaves. The search terminates if investigation of a critical leaf finds a conspiracy, or if the list of critical leaves becomes empty. The leaf investigated is removed from L, and any descendants which are critical are placed on L. We assume, for simplicity, that both nodes take one time unit to expand. Optimal policy is to search the nodes in the order $\{1, -1\}$, as deduced from solving the problem involving the following two node types:

$$\begin{aligned} d_1 &= (1-(1-\delta)^2, 1, {s_{-1}}^2) \\ d_{-1} &= (\delta^2, 1, \tfrac{2\delta(1-\delta)}{1-\delta^2} s_1 + \tfrac{(1-\delta)^2}{1-\delta^2}) \end{aligned}$$

We have seen in Section 1.2 that several authors have derived algorithms which evaluate game positions not only with a scalar score but also with a measure of uncertainty, and that these can yield significant performance gains. In this, simplified example, we suppose, that as well as assigning them a score $\in \{-1, 1\}$ the evaluation function separates its estimates into two classes, types a and b, with different amounts of reliability. There are therefore four types of nodes: $\{+1(a), +1(b), -1(a), -1(b)\}$. A realistic estimate of δ is required for each node type, and could be deduced empirically. If we assume that a node's daughters have a probability, $\dot{p}$, of having the same type as their parent, then as shown overleaf in Figure 18 this model requires the following 4 node types:

$$\begin{aligned} d_{1a} &= (1-(1-a)^2, 1, (\dot{p}s_{-1a} + \dot{q}s_{-1b})^2) \\ d_{1b} &= (1-(1-b)^2, 1, (\dot{p}s_{-1b} + \dot{q}s_{-1a})^2) \\ d_{-1a} &= (a^2, 1, \tfrac{2a(1-a)}{1-a^2}(\dot{p}s_{1a} + \dot{q}s_{1b}) + \tfrac{(1-a)^2}{1-a^2}) \\ d_{-1b} &= (b^2, 1, \tfrac{2b(1-b)}{1-b^2}(\dot{p}s_{1b} + \dot{q}s_{1a}) + \tfrac{(1-b)^2}{1-b^2}) \end{aligned}$$

This problem is in fact the one which inspired the investigation of the tree search model which forms the main subject of Chapter 3. The tree search model, as we have seen, is capable of solving models of considerably

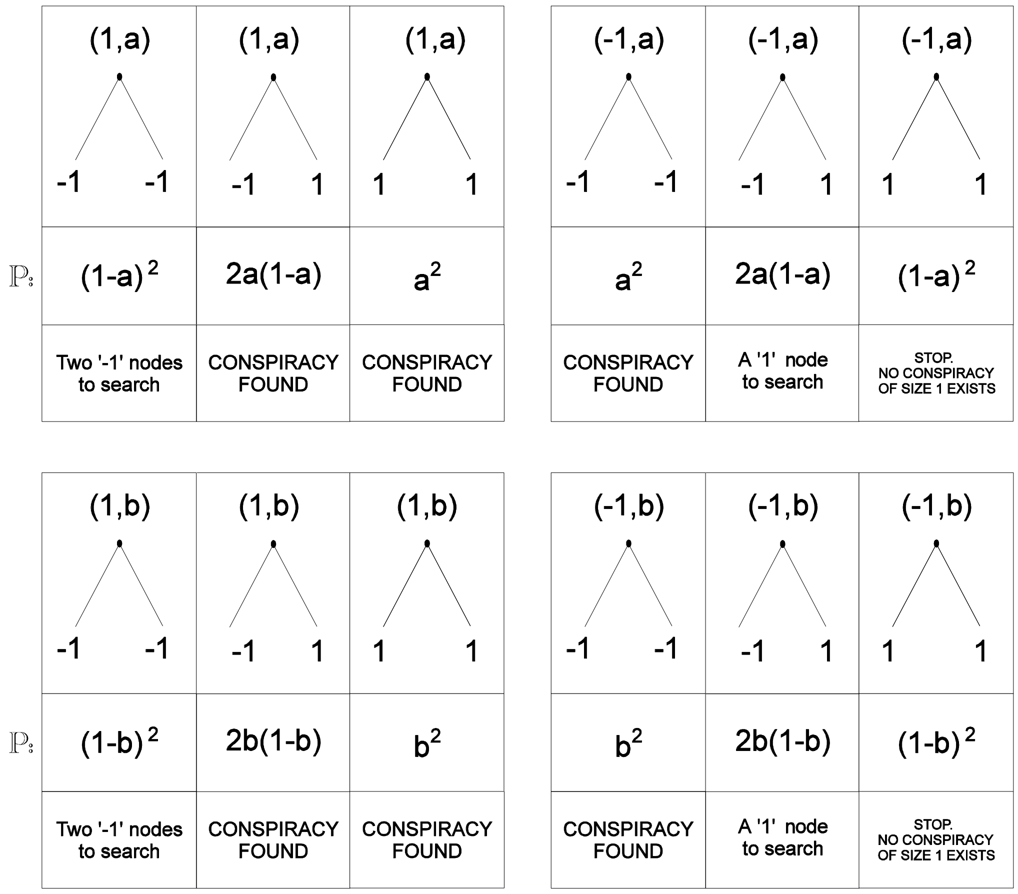

Figure 18: Possible Node Expansion Results

greater generality; the branching factor may be varied as required, as may the number of classes of uncertainty, or the rules about how nodes of one type give rise to nodes of another. Any of these things may be correlated, so as to reflect observations made about a specific game. It is also possible to model the partial expansion of nodes – by which we refer to the creation of less than a full set of node's daughters.

3.4.3 Mathematical Example

For the sake of convenience in the following two examples we shall replace the offspring distribution, f_i, by a multivariate generating function, $G_i(s)$. Consider the search problem involving the three node types below:

$$\begin{aligned} d_1 &= \left(\frac{1}{4}, 6, \frac{1}{3} + \frac{1}{3}s_1s_2 + \frac{1}{3}s_3\right) \\ d_2 &= \left(\frac{1}{2}, 9, \frac{1}{4} + \frac{1}{4}s_1 + \frac{1}{2}s_2\right) \\ d_3 &= \left(0, 1, \frac{2}{3}s_1s_2 + \frac{1}{3}s_1s_2^2s_3\right) \end{aligned}$$

The first step is calculate the reward rates:

$$\begin{aligned} \emptyset_1 &= \tfrac{1}{4}/6 &= \tfrac{1}{24} \\ \emptyset_2 &= \tfrac{1}{2}/9 &= \tfrac{1}{18} \\ \emptyset_3 &= 0/1 &= 0 \end{aligned}$$

Hence, node type 2 is the most rewarding one. In order to deduce the second element of the optimal ordering we require $(\hat{p}_2(1), \hat{t}_2(1), \hat{G}_2(s)(1))$ and $(\hat{p}_2(3), \hat{t}_2(3), \hat{G}_2(s)(3))$, the modified details of the remaining node types. To do this we first chunk together a single search of the most rewarding node type with an exhaustive search of its descendants, to deduce $\hat{p}_2(2)$, $\hat{t}_2(2)$ and $\hat{G}_2(s)(2)$, the characteristics that define an exhaustive search of type 2 nodes carried out upon a single type 2 node.

$$\hat{p}_2(2) = \frac{1}{2} + \frac{1}{4}\hat{p}_2(2)$$

$$\begin{aligned}
&= \frac{2}{3} \\
\hat{t}_2(2) &= 9 + \frac{1}{4}\hat{t}_2(2) \\
&= 12 \\
\hat{G}_2(s)(2) &= \frac{1}{4} + \frac{1}{4}s_1 + \frac{1}{2}\hat{G}_2(s)(2) \\
&= \frac{1}{2} + \frac{1}{2}s_1
\end{aligned}$$

Solution of the above equation for $\hat{G}_2(s)$ is a straightforward matter in this case, since a search of the most rewarding node never reveals more than one more node of this same type. This is the case for several classes of problem, including the multi-armed bandit model described in Section 3.5. Numerical methods are required in other cases.

$$d_2 = \left(\frac{2}{3}, 12, \frac{1}{2} + \frac{1}{2}s_1\right)$$

The next step is to amend d_1 and d_3 to reflect what we already know about the optimal ordering, i.e. that if a node of type 2 is revealed then it will be optimal to search it.

$$\begin{aligned}
\hat{p}_2(1) &= \tfrac{1}{4} + \tfrac{3}{4}\tfrac{1}{3}\hat{p}_2(2) &&= \tfrac{5}{12} \\
\hat{t}_2(1) &= 6 + \tfrac{3}{4}\tfrac{1}{3}\hat{t}_2(2) &&= 9 \\
\hat{G}_2(s)(1) &= \tfrac{1}{3} + \tfrac{1}{3}s_1\hat{G}_2(s) + \tfrac{1}{3}s_3 &&= \tfrac{1}{3} + \tfrac{1}{6}s_1 + \tfrac{1}{6}s_1^2 + \tfrac{1}{3}s_3
\end{aligned}$$

$$\begin{aligned}
\hat{p}_2(3) &= 0 + 1\tfrac{2}{3}\hat{p}_2(2) + 1\tfrac{1}{3}(1-(1-\hat{p}_2)^2)) &&= \tfrac{20}{27} \\
\hat{t}_2(3) &= 1 + 1\tfrac{1}{3}\hat{t}_2(2) + 1\tfrac{2}{3}(\hat{t}_2(2) + (1-\hat{p}_2(2))\hat{t}_2(2)) &&= \tfrac{43}{3} \\
\hat{G}_2(s)(3) &= \tfrac{2}{3}s_1\hat{G}_2(s)(2) + \tfrac{1}{3}s_1(\hat{G}_2(s)(2))^2 s_3 \\
&= \tfrac{1}{3}s_1 + \tfrac{1}{3}s_1^2 + \tfrac{1}{12}s_1s_3 + \tfrac{1}{6}s_1^2s_3 + \tfrac{1}{12}s_1^3s_3
\end{aligned}$$

The modified node details are therefore

$$\begin{aligned}
d_1' &= \left(\frac{5}{12}, 9, \frac{1}{3} + \frac{1}{6}s_1 + \frac{1}{6}s_1^2 + \frac{1}{3}s_3\right) \\
d_3' &= \left(\frac{20}{27}, \frac{43}{3}, \frac{1}{3}s_1 + \frac{1}{3}s_1^2 + \frac{1}{12}s_1s_3 + \frac{1}{6}s_1^2s_3 + \frac{1}{12}s_1^3s_3\right)
\end{aligned}$$

We now compute the reward rates as shown below and conclude that type 3 is the new most rewarding node type since $\emptyset_3 > \emptyset_1$, so the optimal ordering is $\{2,\ 3,\ 1\}$.

$$\begin{aligned}
\emptyset_1 &= \tfrac{5}{12}/9 &&= \tfrac{5}{108} \\
\emptyset_3 &= \tfrac{20}{27}/\tfrac{43}{3} &&= \tfrac{20}{387}
\end{aligned}$$

3.4.4 Drug Testing Example

As a simplified example of a realistic application, suppose permission is being sought to market a newly developed drug. Before a license can be given there are various statutory tests which must be carried out. Assume that legislation requires that a drug be tested for its allergic potential, a, interaction with other drugs, i, and for effectiveness, e.

The anti-allergenic trial, A, requires that the drug not trigger an allergic reaction in any of four test subjects particularly susceptible to allergic reactions. The drug interaction trial, I, consists of a test upon three test subjects, and the drug is deemed to pass if at most one patient shows evidence of a negative interaction. In order to be marketed the drug also must show evidence of a sufficiently high rate of effectiveness. The drug is deemed to pass the effectiveness trial, E, if it has a therapeutic effect on at least two out of four sufferers.

Suppose that the relative costs of testing a single subject in trials A, I and E are 1:5:4, while previous work developing the drug is such that the prior belief about the parameters is as follows:

$$a_p \sim Beta(1,22) \qquad e_p \sim Beta(44,6) \qquad i_p \sim Beta(5,25)$$

The problem now is to determine *the order in which the trials should be conducted so as to minimise the expected cost until completion.* Define the following nodes. For the anti-allergenic trial:

$$\begin{aligned} d_{a_0} &= (\tfrac{1}{23}, 1, s_{a_1}) \qquad & d_{a_1} &= (\tfrac{1}{24}, 1, s_{a_2}) \\ d_{a_2} &= (\tfrac{1}{25}, 1, s_{a_3}) \qquad & d_{a_3} &= (\tfrac{1}{26}, 1, 1) \end{aligned}$$

For the effectiveness trial:

$$\begin{aligned}
d_{e00} &= (0, 4, \tfrac{6}{50}s_{e10} + \tfrac{44}{50}s_{e11}) & d_{e11} &= (0, 4, \tfrac{6}{51}s_{e21} + \tfrac{45}{51}) \\
d_{e10} &= (0, 4, \tfrac{7}{51}s_{e20} + \tfrac{44}{51}s_{e21}) & d_{e21} &= (0, 4, \tfrac{7}{52}s_{e31} + \tfrac{45}{52}) \\
d_{e20} &= (\tfrac{8}{52}, 4, s_{e31}) & d_{e31} &= (\tfrac{8}{53}, 4, 1)
\end{aligned}$$

For the interaction trial:

$$\begin{aligned}
d_{i00} &= (0, 5, \tfrac{5}{30}s_{i10} + \tfrac{25}{30}s_{i11}) & d_{i10} &= (\tfrac{6}{31}, 5, s_{i21}) \\
d_{i11} &= (0, 5, \tfrac{5}{31}s_{i21} + \tfrac{26}{31}) & d_{i21} &= (\tfrac{6}{32}, 5, 1)
\end{aligned}$$

The process of deducing the most rewarding node type and updating the details of other nodes types accordingly is repeated just as before, to deduce the optimal ordering: $\{A_0$, A_1, A_2, I_{10}, A_3, E_{20}, E_{31}, I_{21}, I_{00}, E_{10}, I_{11}, E_{00}, $E_{11}\}$. The optimal ordering is sufficient to calculate optimal policy. The problem starts with three nodes available for search, one each of types A_0, E_{00} and I_{00}, of which A_0 has the highest priority, so should be searched first. Then, assuming no allergic reaction is observed, there will be a node of type A_1, and so that will be have the highest priority of the available nodes, and so on. The optimal policy is therefore to carry out the anti-allergenic trial first. Similarly, since all of the $I_{..}$ node types have a higher priority than the node type E_{00}, it will optimal next to carry out the interaction trial to its conclusion, and then, if necessary, the effectiveness trial.

Further detail can be added as required. If for example, trial I has a set up cost of α which must be paid before any experimentation can take place it suffices to add one more node type:

$$d_i = (0, \alpha, s_{i00})$$

Now suppose, in addition, that the number of separate drug interaction trials required is itself variable and can only be determined by a preliminary investigation at a cost of β. The number of interaction trials required may be modelled by a general discrete distribution, but for the purposes of illustration, let us suppose that it was a *geometric*(k) distribution; the extra node type added would be:

$$d_{pi} = \left(0, \beta, (1-k)\sum_{n=0}^{\infty}(ks_{i00})^n\right)$$

3.4.5 Computer Software Example

We now present an example of an application that motivates the model extension described in Section 3.6. Suppose that a software developer contracts another firm to check the reliability of a piece of a new product before it is released. For this process, the software involved is broken down into a set of individual modules of code, each of which is supplied with its own specification that describes the intended functionality.

The consultancy firm is paid a certain amount for each module of code investigated. Modules in which a deviation from the accompanying specification is detected are returned to the original company together with details of the bug. Modules in which no fault is detected are guaranteed 'OK' by the consultancy. The nature of the agreement is such that the consultancy is obliged to pay a certain levy for every module which is guaranteed 'OK' but later discovered to contain a bug.

Modern methods of software design mean that each module of code may be well modelled as a separate problem. The 'object' that is searched for is a fault in the code. The node types are submodules of code. Upon investigation, each submodule of code may be shown to be functioning incorrectly (object detected), to be functioning correctly (no object detected, no offspring generated), or to have a functionality that depends upon the conjunction of the functionality of one or more other submodules (no object detected, offspring generated).

Early retirement is an essential feature for application of the model to this problem, since in the marketplace it is simply too time consuming to test a program to the point where one can have 100% confidence that no bugs exist. With the extra assumption that once the software is released onto the marketplace, any module containing a bug will eventually be detected as such, this means that the expected cost of ceasing investigations into a particular module and declaring it 'OK' is proportional to the probability that it contains a bug, which is the form of the early retirement function specified in Subsection 3.6.1.

3.5 Bandits

We now review a class of standard models referred to as *bandit problems.* A one armed-bandit is in one of a finite number of states. When activated it returns a random payoff the expectation of which is dependent upon that state. Each activation also entails a random transition to another state, according to known probability distribution. A multi-armed bandit is a problem composed of a set of one-armed bandits, in which the operator of the bandits can choose which project to activate, and is aiming to do so in a manner that maximises his expected reward.

If there is no discounting, the multi-armed bandit problem is a discrete Markov problem. With discounting, the length of time taken for each activation becomes important. If this is allowed to vary randomly, the problem becomes semi-Markov – as well as the states of the projects, the time at which the last transition occurred is also relevant. In this model future rewards are continuously discounted by α.

3.5.1 Gittins Indices

In 1979, Gittins[25] proved that it is optimal to allocate each 'arm' of the bandit a separate index [15] depending only upon the state of that project, and then the projects with the greatest available indices. Gittins' proof proceeds by comparing projects with a 'standard project' which yields a constant stream of rewards.

Whittle[89] produced a more natural proof of the optimality of Gittins indices, and provided an interpretation of them by introducing the idea of a 'retire' option. *Retirement* from a bandit process results in receipt of a single amount referred to as a *terminal reward.*

[15] Gittins termed them *dynamic allocation indices*, but at the suggestion of Whittle[89], *Gittins index* has become standard

3.5.2 Branching

Whittle[89] developed his proof of the optimality of Gittins indices to address a problem considered by Nash[53], in which the number of projects is not constant. He assumed, for convenience, that projects fall into one of finitely many classes, each of which has finitely many states. He proved the optimality of the Gittins index policy for the case in which new projects arrive at a known random rate. He referred to this model as the 'arm-acquiring bandit'.

The 'branching bandits' model of Weiss[88] is a very powerful generalisation of the semi-Markov multi-armed bandit model. It allows the number of new projects that arrive, termed 'descendants' by Weiss, to depend in a general fashion upon the project activated, and so the arm-acquiring model, in which it is not, becomes a simple special case. Weiss proves the optimality of a Gittins index policy.

A welcome development in the theory of semi-Markov multi-armed bandits was a very simple proof of the optimality of index policies by Tsitsiklis[85]. This owes quite a lot to Weiss' paper and is generalisable to the branching bandit case. Its key feature, however, an induction on the cardinality of the bandit's statespace, makes it considerably simpler. Its structure is identical to the independently discovered proof of the optimal policy for the OR-tree model given in Subsection 3.3.3.

The important paper of Bertsimas and Niño-Mora[17] establishes a framework with which to analyse a wide range of stochastic and dynamic scheduling problems in a radically different fashion. Their approach is not based around dynamic programming, but they characterise the optimal policy by using a linear program. This approach yields closed formulae for the maximum reward of a multi-armed bandit. Glazebrook and Garbe[28] use this to develop simpler dynamic programming proofs of the optimality of Gittins index policies for finite state branching bandits, as well as suboptimality bounds.

3.5.3 Search Problem Applications

In his comment on Gittins[25], Kelly[42] points out how multi-armed bandits can be used to solve the 'boxes' search problem, as described in Section 3.1, with overlook probabilities. He does this by considering a family of alternative bandit processes, with no transition costs, that give a reward of α^t it the object is found at time t. The issue of stopping is conveniently dealt with by assuming that the searcher does not know whether the object has been found. Kelly also explains how another classic problem, the 'gold-mining', or 'bombing', problem first formulated by Bellman[11] can be solved by applying the multi-armed bandit framework.

As the solution to both of these problems was already known, he remarks, the real advantage of formulating them as bandit problems is that this illustrates how slightly more general problems may also be fitted into the framework. As an example, he supposes the problem in which the probabilities of some of the boxes were not known exactly; the learning that results from unsuccessful search can easily be fitted into the bandit framework.

The paper by Kadane and Simon[37] of two years earlier established the optimal policy for the boxes and slices case, and contains a flawed proof of the case in which search of boxes is constrained by a general partial ordering. This case, both with and without discounting, was proved independently by Gittins and Glazebrook[29].

Glazebrook[27] had earlier proved the case in which the precedence constraints formed an out-tree. As pointed out by Gittins[26], it is a simple matter to solve this problem by constructing a branching bandit process.

3.5.4 Link with OR-Tree Model

The 'boxes' search model of Section 3.1 specifies parameters p_i as the probability that a box contains an object, and t_i as the time taken to search it. We now show how it can be understood by using the framework for semi-Markov bandits. As suggested by Kelly in his comment on Gittins[25], each box has an equivalent bandit.

The cost structure, however, is different. The bandits have two states - 'searched' and 'unsearched'. The retirement penalty and costs of searching an already searched bandit should be sufficiently large that it is optimal to search all the unsearched bandits, in some order, and then retire at once.

The expected cost of searching a box, t_i, is minus the expected running reward, R_i, of the activating the corresponding bandit in the 'unsearched' state, suitably adjusted to account for its being paid at time T_i, assuming continuous discounting at rate $\alpha \in (0,1]$:

$$R_i = -e^{\alpha T_i} t_i$$

The discounting of the bandits is associated with the possibility of finding the object in the boxes search. To this end, the bandits have the following activation times:

$$T_i = -\frac{ln(1-p_i)}{\alpha}$$

The boxes problem minimises $\sum t_i$, whilst the associated semi-Markov bandit problem maximises $\sum R_i$.

Construction of an equivalent semi-Markov bandit problem for the linear precedence constraints model of Section 3.1 is straightforward, once the reward rate for semi-Markov bandits has been calculated. From the above formulae:

$$\emptyset_i = \frac{p_i}{t_i} = \frac{1-e^{-\alpha T_i}}{-R_i e^{-\alpha T_i}} = \frac{1-e^{\alpha T_i}}{R_i}$$

We observe that this is the reciprocal of the conventional form of the Gittins index.

The stochastic search case is equivalent to the semi-Markov branching bandit model, as described by Weiss[88]. The distribution of descendants, $g_i(s, z_1 \ldots z_N)$ is the the offspring distribution f_i described in Section 3.2.

Tsitsiklis[85] writes in his proof of the Gittins index theorem for semi-Markov bandits:

> The proof given here is very simple and it is quite surprising that it was not known earlier. Perhaps a reason is that for the proof

to go through, we have to consider semi-Markov bandits rather than the usual discrete-time Markov bandits.

It is correct that the method of proof does not work on the standard discrete-time Markov bandit model. However, as made clear by the above there is an equivalence between the semi-Markov bandit model and the Markov tree search model. This clarifies the position of the independently discovered proof given in Subsection 3.3.3. The tree search model has the 'probability of discovering an object' which is equivalent to a nodetype-specific discount factor. This enables the crucial induction step of the proof by providing a means to carry out the chunking of node types.

3.6 Retiring Early

Retirement was introduced to the theory of multi-armed bandits by Whittle[89]. This is the option of permanently rejecting all the bandits, and earning instead a single payoff, termed a *retirement reward*, $M \in \mathbf{R}$. To see how this can be conveniently brought within the existing multi-armed bandit framework, consider a bandit with a single state which yields reward $M/(1-\alpha)$ when activated. Activiating this bandit does not change the state. Therefore once it becomes optimal to do this it will remain so, resulting in a stream of payoffs $M/(1-\alpha)$, $\alpha M/(1-\alpha)\ldots$, equivalent to a one-off payment of M.

Seen in this way, the addition of a retirement option looks more like an interesting feature of the multi-armed bandit framework than a real extension to it. Indeed, the fact that it was introduced by Whittle principally to facilitate a shorter proof of Gittins Index Theorem would certainly support this view, and may explain why 'retirement' has not been developed much further. The inadequacy of this model of retirement is exemplified by the software development scenario presented in Subsection 3.4.5 above, in which the expected payoff from retiring should be allowed to depend upon the states of the projects. We now address this problem.

Let us add an action, *retirement*, which may be taken at any stage, with the effect of immediately terminating the search, incurring a penalty cost $M(P)$, a function of the probability that there is at least one object somewhere. Whittle's retirement option is equivalent to using a retirement function $M(P) = MI_{P\in(0,1)}$.

The previous model allowed termination only upon discovery of an object or when all search opportunities had been exhausted, and so suggests use of the following penalty function:

$$M(P) = \begin{cases} \infty & : \quad P \in (0,1) \\ 0 & : \quad P \in \{0,1\} \end{cases}$$

This new model is identical to the original one, because of the regularity conditions imposed concerning the probability, p_i'', that there is an object amongst the offspring of a type i node. The two models are not equivalent if there is an i for which $p_i'' = 1$, since an occurrence of such a node would allow an early retirement in the second model which would be prohibited in the original model (because discovery of such a node shows that an object exists somewhere without actually locating it). Similarly, if there is a node type with $q_i q_i'' = 1$, then the two models again diverge.

We now consider some important retirement functions, in rough order of tractability. Only for the first of these is the optimal policy proved in the general case.

3.6.1 Retire and Say "No"

Theorem 3.4 *The optimal policy for penalty functions of the form $M(P) = I_{P<1}(a + bP)$, with $a \in [0, \infty]$, $b \in (-a, \infty)$ is to search nodes in decreasing order of Ø^*, until either an object has been found or all the remaining node types have $\text{Ø}^* < (a + b)^{-1}$, at which point it is optimal to retire.*

Proof: We first show that it cannot be optimal to retire if there is a node type i available which has $\text{Ø}_i^* > (a + b)^{-1}$. Then we show by an interchange argument that, in this situation, it is optimal to search these nodes in order of Ø^*. Finally, we use a one step lookahead argument to show that if there is no node type i available with $\text{Ø}_i^* > (a + b)^{-1}$ then it is optimal to retire. The case of $a = \infty$ has already been proved, so we assume $a < \infty$.

$M(P)$ is bounded below by 0. Since this bound is achieved for $P = 1$, it must always be optimal to retire if an object has been found.

When a node of type i is expanded, suppose that with probability q_i no object is found, and that the probability that no object exists amongst nodes revealed by the search is q_i''. We shall let q_i' be the probability that there is no object amongst the other nodes or their descendants. Hence:

$$P = 1 - q_i q_i' E[q_i'']$$

Denote by V_{π_0} the expected value of immediate retirement, and by $V_{\pi_{i0}}$ the expected value of searching a node of type i and then retiring. Let us now consider the relative merits of these two courses of action.

$$\begin{aligned}
V_{\pi_0} &= a + bP \\
&= a + b(1 - q_i q_i' E[q_i'']) \\
V_{\pi_{i0}} &= t_i + q_i E[M(1 - q_i' q_i'')] \\
&= t_i + q_i M(E[1 - q_i' q_i'']) \quad \text{as } \mathrm{q_i'q_i''} > 0 \text{ and M() is linear in } [0,1) \\
&= t_i + q_i(a + b(1 - q_i' E[q_i''])) \\
V_{\pi_{i0}} - V_{\pi_0} &= t_i + q_i(a + b(1 - q_i' E[q_i''])) - (a + b(1 - q_i q_i' E[q_i''])) \\
&= t_i + (a + b)(q_i - 1) \\
&= t_i - (a + b)p_i \qquad (10)
\end{aligned}$$

Retirement is therefore not optimal if there are any nodes of type i available, where $(a + b)p_i > t_i$, that is, if $\emptyset_i > (a + b)^{-1}$.

We now extend the scope of the above line of reasoning to deal not only with boxes, but also with chunks of search. A consequence of their definition is that the reward rate, $\emptyset$, of a partially searched chunk is strictly less than that of the remaining unsearched part. This implies that, as we argued in Subsection 3.1.1 that it is not optimal to intercalate any other search in the middle of searching a chunk. It is not optimal to retire in the midst of a chunk, since at least one of the following is always true:

1. The searched portion had $\emptyset > (a+b)^{-1}$.

2. The unsearched portion has $\emptyset > (a+b)^{-1}$.

In the former case, equation (10) implies that it would have been better to retire before starting search of the chunk, whilst in the latter, it implies that it would be better to complete search of the chunk before retiring.

Up to now, we have seen that search should proceed in chunks, and should not stop as long as there are no nodes available with $\emptyset^* \geq (a+b)^{-1}$. Equation (10) implies that, once this point has been reached, immediate retirement is a better policy than carrying out one (or by induction, many) further searches and then retiring. This establishes the set of nodes which it is worth searching before retiring as those with $\emptyset^* > (a+b)^{-1}$. The optimal policy must search those in the order which minimises the expected time taken to discover an object. This is exactly the problem of Section 3.3, which was shown in Theorem 3.3 to be solved by searching the chunks in decreasing order of $\emptyset^*$. ■

3.6.2 Retire and Guess

Now suppose that upon retirement a guess is taken as to whether or not an object exists, and a constant cost of K is incurred for an incorrect guess.

This corresponds to a retirement function of $M(P) = K(\frac{1}{2} - |P - \frac{1}{2}|)$, where P is the overall probability that an object exists.

Lemma 3.5 *If $P \geq \frac{1}{2}$, it is not optimal to search a box or sequence of boxes and then retire unless at this point $P < \frac{1}{2}$, or an object has been found.*

Proof: We show that the lemma holds for a single box, from which the result for a sequence follows immediately by induction. Let policy π_i be the policy of searching box i and then retiring.

$$\begin{aligned}
V_{\pi_{i0}} &= t_i + q_i M(1 - q_i') \\
V_{\pi_0} &= M(1 - q_i q_i') = K q_i q_i' \\
V_{\pi_{i0}} - V_{\pi_0} &= t_i + q_i M(1 - q_i') - K q_i q_i'
\end{aligned}$$

If the policy retires such that $1 - q_i' \geq \frac{1}{2}$:

$$V_{\pi_{i0}} - V_{\pi_0} = t_i + q_i K q_i' - q_i K q_i' = t_i > 0$$

■

Theorem 3.6 *If all the nodes available for search are boxes, the optimal policy with retirement penalty function $M(P) = K(\frac{1}{2} - |P - \frac{1}{2}|)$ is either to retire immediately or to search the boxes in decreasing order of $\emptyset$ until an object is found or until no boxes remain with $\emptyset > K^{-1}$.*

Proof:

$P \in [0, \frac{1}{2}] \cup \{1\}$:

We observe that no sequence of searches can cause P to assume a value in the interval $(\frac{1}{2}, 1)$, successful search fixes P as 1 and unsuccessful search decreases it. Over this domain $M(P) = K(\frac{1}{2} - |P - \frac{1}{2}|)$ assumes identical values to the function $M(P) = I_{P<1}KP$, so Theorem 3.4 proves the result.

$P \in (\frac{1}{2}, 1)$:

Lemma 3.5 establishes that an optimal policy which searches at all must continue to do so until $P \in [0, \frac{1}{2}] \cup \{1\}$. Such an optimal policy, therefore, cannot leave unsearched any boxes with $\emptyset > K^{-1}$, from application of Theorem 3.4, always assuming no object is found. This establishes that an optimal policy must be prepared to search all the boxes with $\emptyset > K^{-1}$. The standard interchange argument proves that unless it does so in decreasing order of $\emptyset$ it can be improved upon by a policy which plays a suitably permutated strategy.

To see that it is optimal not to include in the search any boxes with $\emptyset \leq K^{-1}$, we consider the payoff of such a policy, π_A.

$$\begin{aligned} V_{\pi_A} &= \sum_{i=1}^{m} t_i \prod_{j=1}^{i-1} q_j + \prod_{i=1}^{m} q_i M \left(1 - \prod_{j=m+1}^{n} q_j\right) \\ &= \sum_{i=1}^{m} t_i \prod_{j=1}^{i-1} q_j + \prod_{i=1}^{m} q_i K \left(1 - \prod_{j=m+1}^{n} q_j\right) \\ &= \sum_{i=1}^{m} t_i \prod_{j=1}^{i-1} q_j + K \prod_{i=1}^{m} q_i - K \prod_{i=1}^{n} q_i \end{aligned}$$

Now consider the payoff of a policy, $\pi_{A'}$, which omits search of box m:

$$\begin{aligned}
V_{\pi_{A'}} &= \sum_{i=1}^{m-1} t_i \prod_{j=1}^{i-1} q_j + \prod_{i=1}^{m-1} q_i M \left(1 - \prod_{j=m}^{n} q_j\right) \\
&\leq \sum_{i=1}^{m-1} t_i \prod_{j=1}^{i-1} q_j + \prod_{i=1}^{m-1} q_i K \left(1 - \prod_{j=m}^{n} q_j\right) \\
&\leq \sum_{i=1}^{m} t_i \prod_{j=1}^{i-1} q_j + K \prod_{i=1}^{m-1} q_i - K \prod_{i=1}^{n} q_i
\end{aligned}$$

Hence:

$$\begin{aligned}
V_{\pi_A} - V_{\pi_{A'}} &= t_m \prod_{j=1}^{m-1} q_j - p_m K \prod_{i=1}^{m-1} q_i \\
&= \prod_{j=1}^{m-1} q_j (t_m - p_m K) \\
&\geq 0 \qquad \text{if } \emptyset_m \leq K^{-1}
\end{aligned}$$

Since policy π_A considers m boxes in decreasing order of $\emptyset_i$, this establishes that it is optimal not to consider any which have $\emptyset_i \leq K^{-1}$. ■

We assume for convenience that the boxes are indexed in decreasing order of $\emptyset$, so $i < j$ implies that $\emptyset_i \geq \emptyset_j$. Theorem 3.6 implies that the following policy is optimal either for $j = 0$ or for the largest j such that $\emptyset_j \geq K^{-1}$.

Policy π_j searches boxes $1 \ldots j \leq n$ until it finds an object. If no object is found, it then retires.

Corollary 3.7 *If $P > \frac{1}{2}$, policy π_j may be optimal iff $\prod_{i=1}^{n} q_i > 1 - \prod_{i=j+1}^{n} q_i$. In this case, the critical value of K, for which immediate retirement gives the*

same payoff as a policy π_j above, is given by the below:

$$K^{-1} = \frac{\prod_{i=j+1}^{n} q_i + \prod_{i=1}^{n} q_i - 1}{\sum_{i=1}^{j} t_i \prod_{k=1}^{i-1} q_k}$$

Proof:

$$\begin{aligned}
V_{\pi_j} - V_{\pi_0} &= \sum_{i=1}^{j} t_i \prod_{k=1}^{i-1} q_k + M\left(1 - \prod_{i=j+1}^{n} q_i\right) - M\left(1 - \prod_{i=1}^{n} q_i\right) \\
&= \sum_{i=1}^{j} t_i \prod_{k=1}^{i-1} q_k + K\left(1 - \prod_{i=j+1}^{n} q_i\right) - K\left(\prod_{i=1}^{n} q_i\right) \\
&= \sum_{i=1}^{j} t_i \prod_{k=1}^{i-1} q_k + K\left(1 - \prod_{i=j+1}^{n} q_i - \prod_{i=1}^{n} q_i\right) \\
&> 0 \qquad\qquad \text{if } \left(1 - \prod_{i=1}^{n} q_i \geq \prod_{i=j+1}^{n} q_i\right)
\end{aligned}$$

We have shown that if $\left(1 - \prod_{i=1}^{n} q_i \geq \prod_{i=j+1}^{n} q_i\right)$ then it is always optimal to retire. In the remaining cases, the critical value of K is calculated as follows:

$$\begin{aligned}
\sum_{i=1}^{j} t_i \prod_{k=1}^{i-1} q_k + K\left(1 - \prod_{i=j+1}^{n} q_i - \prod_{i=1}^{n} q_i\right) &= 0 \\
\sum_{i=1}^{j} t_i \prod_{k=1}^{i-1} q_k &= K\left(\prod_{i=j+1}^{n} q_i + \prod_{i=1}^{n} q_i - 1\right) \\
K^{-1} &= \frac{\prod_{i=j+1}^{n} q_i + \prod_{i=1}^{n} q_i - 1}{\sum_{i=1}^{j} t_i \prod_{k=1}^{i-1} q_k}
\end{aligned}$$

■

Linear Precedence Constraints. We now add to the model constraints about the order in which the boxes may be searched, as illustrated overleaf. We suppose the boxes are indexed so that for $j > 0$, box $(i, j+1)$ cannot be searched until box (i, j) has been searched.

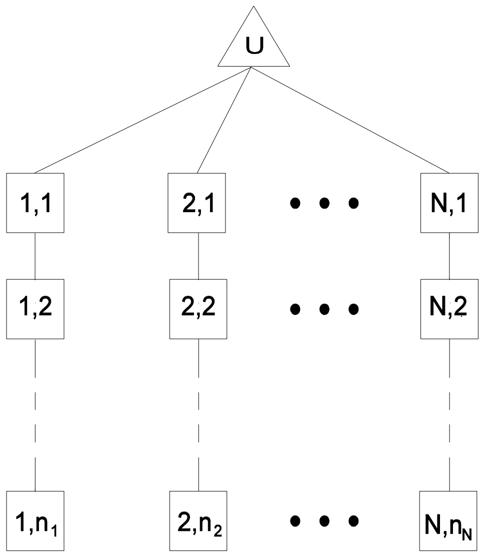

Figure 19: Linear Precedence Constraints

This structure is that of the model of Subsection 3.1.1, and the problem can be analysed in the same fashion. That is, we work backwards from the end of each stack, parsing the tree into sections that must be searched together.

Consider the last two boxes of the ith stack. If $\O_{(i,n_i-1)} < \O_{(i,n_i)}$ then these two boxes are part of a single chunk of search. The chunking procedure used is identical to that in Subsection 3.1.1. The validity of the chunking process is established by the result below, which will be applied recursively to each stack:

Theorem 3.8 *If* $\emptyset_{(i,n_i-1)} < \emptyset_{(i,n_i)}$ *then any optimal policy which searches box* $(i, n_i - 1)$ *must search box* (i, n_i) *next, unless an object is found.*

Proof:

If $\emptyset_{(i,n_i)} < K^{-1}$:

Theorem 3.6 implies that no optimal policy would search either box (i, n_i) or box $(i, n_i - 1)$ even if there were no constraints. Adding constraints can never increase the payoff from searches, and so the result is proved because no optimal policies search $(i, n_i - 1)$.

If $\emptyset_{(i,n_i)} \geq K^{-1}$:

Let $\pi_{(i,n_i-1)}$ be a policy which searches box $(i, n_i - 1)$, then some (possibly empty) set of boxes, j, then retires, leaving box (i, n_i) unsearched. Denoting by q'_{ij} the probability of there being no object amongst the boxes other than those in stack i or set j, the payoff of policy $\pi_{(i,n_i-1)}$ can be expressed as follows:

$$V_{\pi_{(i,n_i-1)}} = t_{(i,n_i-1)} + q_{(i,n_i-1)}t_j + q_{(i,n_i-1)}q_j M(1 - q_{(i,n_i)}q'_{ij})$$

We shall compare the payoff of this this with that of two others, policy π_1, which differs from policy $\pi_{(i,n_i-1)}$ in that it searches box (i, n_i) before retiring and with policy π_0, that of immediate retirement. These policies have the following payoffs:

$$\begin{aligned} V_{\pi_0} &= M(1 - q_{(i,n_i-1)}q_{(i,n_i)}q_j q'_{ij}) \\ V_{\pi_1} &= t_{(i,n_i-1)} + q_{(i,n_i-1)}t_j + q_{(i,n_i-1)}q_j t_{(i,n_i)} + q_{(i,n_i-1)}q_j q_{(i,n_i)} M(1 - q'_{ij}) \end{aligned}$$

If $q_{(i,n_i)}q'_{ij} \geq \frac{1}{2}$, policy π_1 is an improvement:

$$\begin{aligned}
V_{\pi_{(i,n_i-1)}} - V_{\pi_1} &= q_{(i,n_i-1)}q_j M(1 - q_{(i,n_i)}q'_{ij}) \\
&\quad -q_{(i,n_i-1)}q_j t_{(i,n_i)} - q_{(i,n_i-1)}q_j q_{(i,n_i)} M(1 - q'_{ij}) \\
&= q_{(i,n_i-1)}q_j (K(1 - q_{(i,n_i)}q'_{ij}) - t_{(i,n_i)} - q_{(i,n_i)}K(1 - q'_{ij})) \\
&= q_{(i,n_i-1)}q_j (Kp_{(i,n_i)} - t_{(i,n_i)}) \\
&= q_{(i,n_i-1)}q_j K t_{(i,n_i)} (\text{Ø}_{(i,n_i)} - K^{-1}) \ > \ 0
\end{aligned}$$

If $q_{(i,n_i)}q'_{ij} < \frac{1}{2}$, policy π_0 is an improvement:

$$\begin{aligned}
V_{\pi_{(i,n_i-1)}} - V_{\pi_0} &= t_{(i,n_i-1)} + q_{(i,n_i-1)}t_j + q_{(i,n_i-1)}q_j M(1 - q_{(i,n_i)}q'_{ij}) \\
&\quad -M(1 - q_{(i,n_i-1)}q_{(i,n_i)}q_j q'_{ij}) \\
&= t_{(i,n_i-1)} + q_{(i,n_i-1)}t_j + q_{(i,n_i-1)}q_j K(q_{(i,n_i)}q'_{ij}) - K(q_{(i,n_i-1)}q_{(i,n_i)}q_j q'_{ij}) \\
&= t_{(i,n_i-1)} + q_{(i,n_i-1)}t_j \ > \ 0
\end{aligned}$$

We now consider the remaining case, that in which a policy π searches box (i, n_{i-1}) and then intercalates search of a non-empty set of boxes, j, before searching (i, n_i). Since either $\text{Ø}_{(i,n_i)} > \text{Ø}_j$ or $\text{Ø}_j > \text{Ø}_{(i,n_{i-1})}$, policy π can be improved by a simple interchange argument, as the constraint structure does not debar swapping j with either (i, n_{i-1}) or, once (i, n_{i-1}) has been searched, with box (i, n_i). ■

The case of linear precedence constraints can therefore be solved by applying the same chunking process that was used on the model without retirement. This reduces it, as before, to the unconstrained boxes case treated above.

We now consider what can be deduced about the general OR-tree model with early retirement function $M(P) = K(\frac{1}{2} - |P - \frac{1}{2}|)$. From the form of this function, we see that as $K \to 0$, the cost of early retirement becomes vanishingly small, and so, for small enough K, the optimal policy is to retire immediately. As $K \to \infty$, early retirement becomes increasingly expensive, so the optimal policy tends to that of the model without the retirement option. Note, however, that the models are only asymptotically equivalent, since for any fixed K, $M(P) \to 0$ as extra nodes are added, and so retirement is still optimal from states with small enough P.

By comparing V_{π_0} with $V_{\pi_{i0}}$ we can prove the following necessary (though not sufficient) condition for states in which it is optimal to retire. We denote by q'_i the probability that no object exists outside box i or its offspring.

Theorem 3.9 *For it to be optimal to retire there must not be a node of type i available with* $q'_i \geq \frac{1}{2}, \O_i > K^{-1} + I_{q'_i q_i \leq \frac{1}{2}} \frac{1-2q'_i q_i}{t_i}$.

Proof: If node i is a box:

$$\begin{aligned}
V_{\pi_0} &= M(1 - q'_i q_i) \\
V_{\pi_{i0}} &= t_i + q_i M(1 - q'_i) \\
V_{\pi_{i0}} - V_{\pi 0} &= t_i + q_i M(1 - q'_i) - M(1 - q'_i q_i) \\
&= t_i + q_i K(\tfrac{1}{2} - |\tfrac{1}{2} - q'_i|) - K(\tfrac{1}{2} - |\tfrac{1}{2} - q'_i q_i|)
\end{aligned}$$

$$\begin{aligned}
\text{For } q'_i q_i \geq \tfrac{1}{2}: \quad &= t_i + q_i K(1 - q'_i) - K(1 - q'_i q_i) \\
&= t_i + K(q_i - q'_i q_i - 1 + q'_i q_i) \\
&= t_i + K(q_i - 1) \\
&= t_i - K p_i \\
&< 0 \text{ iff } \emptyset_i > K^{-1}
\end{aligned}$$

$$\begin{aligned}
\text{For } q'_i \geq \tfrac{1}{2} \geq q'_i q_i: \quad &= t_i + q_i K(1 - q'_i) - K(q'_i q_i) \\
&= t_i + K(q_i - q'_i q_i - q'_i q_i) \\
&= t_i + q_i K(1 - 2q'_i) \\
&< 0 \text{ iff } q_i K(2q'_i - 1) > t_i \\
&\quad \Leftrightarrow \tfrac{2q'_i q_i - q_i}{t_i} > K^{-1} \\
&\quad \Leftrightarrow \tfrac{2q'_i q_i t_i - 1}{t_i} + \tfrac{p_i}{t_i} > K^{-1} \\
&\quad \Leftrightarrow \emptyset_i > K^{-1} + \tfrac{1 - 2q'_i q_i}{t_i}
\end{aligned}$$

$$\begin{aligned}
\text{For } \tfrac{1}{2} > q'_i: \quad &= t_i + q_i K(q'_i) - K(q'_i q_i) = t_i \\
&> 0
\end{aligned}$$

Thus:

$$V_{\pi_{i0}} - V_{\pi_0} \quad \begin{cases} < 0 & | \quad q_i' q_i \geq \frac{1}{2}, \qquad \emptyset_i > K^{-1} \\ < 0 & | \quad q_i' \geq \frac{1}{2} \geq q_i' q_i, \quad \emptyset_i > K^{-1} + \frac{1 - 2q_i' q_i}{t_i} \\ \geq 0 & | \quad \text{otherwise} \end{cases} \tag{11}$$

Adding more descendants to a node can only ever increase the desirability of searching it, and so, since we assumed the node type i had no descendants, the result is also valid for any nodes of type i with probability p_i of containing an object. ■

Useless node types. We define a node type as being *useless*, if there is an optimal policy which will never search it, no matter which other boxes are available for search.

Corollary 3.10 *A box of type i with $\emptyset_i \leq K^{-1}$ is* useless.

Proof: Let π_{iA} be an arbitrary policy which starts by searching a box of a type i, with $\emptyset_i \leq K^{-1}$. By A we denote the sequence of searches it carries out before retiring if the initial search is unsuccessful. Inequality (11) of Theorem 3.9 above implies that if A is empty, then policy π_{iA} is no better than a policy of immediate retirement, which we shall denote π_0.

We now address the case of non-empty A. Suppose the expected time to carry out search A is t_A, and that an object is revealed with expected probability p_A. We denote by p_A'' the expected probability that an object exists amongst the nodes revealed by searching A, and by p'_{Ai} the probability that an object exists amongst the nodes unsearched by policy π_{Ai}.

For $\emptyset_A \geq K^{-1}$:

$$\begin{aligned}
V_{\pi_{iA}} &= t_i + q_i(t_A + q_A M(1 - q'_{Ai}q''_A)) \\
V_{\pi_{Ai}} &= t_A + q_A(t_i + q_i M(1 - q'_{Ai}q''_A)) \\
\text{So} : V_{\pi_{iA}} - V_{\pi_{Ai}} &= t_i + q_i(t_A + q_A M(1 - q'_{Ai}q''_A)) \\
&\quad -t_A - q_A(t_i + q_i M(1 - q'_{Ai}q''_A)) \\
&= p_A t_i - p_i t_A \\
&= t_A t_i(\emptyset_A - \emptyset_i) \geq 0
\end{aligned}$$

In this case therefore, policy π_{Ai} is at least as good as policy π_{iA}.

For $\emptyset_A < K^{-1}$:

$$\begin{aligned}
V_{\pi_{iA}} &= t_i + q_i(t_A + q_A M(1 - q'_{Ai}q''_A)) \\
V_{\pi_0} &= M(1 - q_i q_A q'_{Ai}q''_A) \\
\text{So} : V_{\pi_{iA}} - V_{\pi_0} &= t_i + q_i(t_A + q_A M(1 - q'_{Ai}q''_A)) - M(1 - q_i q_A q'_{Ai}q''_A)
\end{aligned}$$

For $q_i q_A q'_{Ai} q''_A \geq \frac{1}{2}$:

$$\begin{aligned}
&= t_i + q_i(t_A + q_A K(1 - q'_{Ai}q''_A)) - K(1 - q_i q_A q'_{Ai}q''_A) \quad (12) \\
&= t_i + q_i t_A + K(q_i q_A - q_i q_A q'_{Ai}q''_A - 1 + q_i q_A q'_{Ai}q''_A) \\
&= t_i + q_i t_A + K(q_i q_A - 1)
\end{aligned}$$

Since $\emptyset_A < K^{-1}$ and $\emptyset_i \leq K^{-1}$, $Kp_A < t_A$ and $Kp_i \leq t_i$. Thus:

$$\begin{aligned}
&> Kp_i + q_i K p_A + K(q_i q_A - 1) \\
&> K(p_i + q_i p_A + q_i q_A - 1)
\end{aligned}$$

$$> \quad 0$$

For $q''_A q'_{Ai} \geq \frac{1}{2} \geq q_i q_A q''_A q'_{Ai}$:

$$\begin{aligned} &= \quad t_i + q_i(t_A + q_A K(1 - q'_{Ai} q''_A)) - K(q_i q_A q'_{Ai} q''_A) \\ &\geq \quad t_i + q_i(t_A + q_A K(1 - q'_{Ai} q''_A)) - K(1 - q_i q_A q'_{Ai} q''_A) \\ &> \quad 0 \qquad \text{from (12)} \end{aligned}$$

For $\frac{1}{2} \geq q''_A q'_{Ai}$:

$$\begin{aligned} &= \quad t_i + q_i(t_A + q_A K(q'_{Ai} q''_A)) - K(q_i q_A q'_{Ai} q''_A) \\ &= \quad t_i + q_i t_A \\ &> \quad 0 \end{aligned} \tag{13}$$

In this case therefore, π_0 is better than π_{iA}.

We have now proved that for any policy π_{iA}, which starts by searching a box of type i, with $\emptyset_i \leq K^{-1}$, there exists an alternative policy which does not, and which achieves a payoff at least as good. The theorem therefore follows by recursive application of this result. ■

Identification of useless node types allows a compression of the description of the state, since the number of such nodes available is relevant only in so far as it influences P, the probability that an object exists somewhere. Thus, if node types $X_{j+1} \dots X_n$ are useless, they can be 'hidden' in our description of the state, and the vector $(X_1, \dots X_n)$ replaced by the vector $(X_1, \dots X_j, h)$ where h represents the 'hidden probability', that is, the probability that an object exists somewhere amongst the nodes of type $j+1 \dots n$ or their descendants. The h value has the effect of changing the effective cost of retirement:

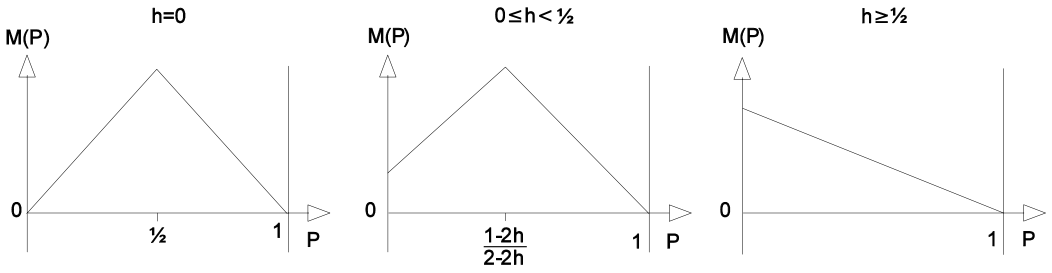

Figure 20: Effective M() for Different Hidden Probabilities

Since the effective $M(P)$ is linear for $h \geq \frac{1}{2}$, the optimal policy from such a state is to retire immediately.

One-step Lookahead Policies. We now consider the class of one-step lookahead policies for this retirement function. A one-step lookahead policy is optimal for the boxes case without retirement, as shown at the start of this chapter. This may be understood to be a consequence of the simplicity of the motives for searching a box. In the case of linear precedence constraints, this simplicity is upset. However, we have seen that if the nodes are grouped into maximal indivisible blocks, and search of a maximal indivisible block is taken to be a single step then the optimal policy is a one-step lookahead policy. The OR-tree search problem may be treated in a similar fashion, with stochastic chunks of search taking the place of the deterministic maximal indivisible blocks. One is therefore prompted to wonder whether the OR-tree search problem with retirement function $M(P) - K(\frac{1}{2} - |\frac{1}{2} - P|)$ can also be solved by a similar process of chunking together a node with some subset of its descendants.

Suppose that there are three node types, as defined below, with ϵ small but positive.

$$\begin{aligned} A &= (0, \epsilon, \tfrac{10}{11}s_B + \tfrac{1}{11}s_C) \\ B &= (\tfrac{1}{100}, K, 1) \\ C &= (\tfrac{1}{10}, K, 1) \end{aligned}$$

Consider a state in which there are seven nodes of type A available. Observe that $p''_A = \frac{54}{55}$, so the overall probability that an object exists is $P(\mathbf{x}) = 1 - (\frac{54}{55})^7 < \frac{1}{2}$. We note that if each of the type A nodes is searched and gives rise to a type C node, then the probability that an object exists will rise to $1-(\frac{9}{10})^7 > \frac{1}{2}$. Hence, $M()$ is concave but not linear over the range of possible values taken by P if all the type A nodes are searched. Thus, the expected payoff from retirement decreases if these searches are carried out, and it will be optimal to do some searching if ϵ is small enough.

We observe that expansion of a single type A node can only cause P to adopt a value of either $1 - \frac{9}{10}(\frac{54}{55})^6$ or $1 - \frac{99}{100}(\frac{54}{55})^6$. Since $M(P)$ is linear over this range, a one-step lookahead policy would retire rather than carry out any search, and so fail to play optimally.

Hence, *if the optimal policy is to be a one-step lookahead policy, the step size must be such as to consider expansion of all the type A nodes in one go.* This would require a 'horizontal' equivalent of the chunking concept, that is, one which treats sets of siblings as a single chunk[16]. The interaction between these, new, chunks, and the 'vertical' chunking process which defines maximal indivisible blocks does not seem to be tractable. A fuller understanding of the problems with one-step lookahead policies may lead to the development of a more theoretical approach to the choice of step size than that presented in Section 6.3.

3.6.3 Retire and Say "Yes"

Now suppose that upon retirement, a cost of K is incurred if no object exists. This corresponds to a retirement function of $M(P) = I_{P>0}K(1-P)$.

Theorem 3.11 *In the boxes case, if* $K \leq \sum_{i=1}^{n} \prod_{j=1}^{i-1} q_j t_i \,/ (1 - \prod_{i=1}^{n} q_i)$ *then it is optimal to retire immediately. If* $K \geq \sum_{i=1}^{n} \prod_{j=1}^{i-1} q_j t_i \,/ (1 - \prod_{i=1}^{n} q_i)$ *then it is optimal to search all the boxes for an object in decreasing order of* $\emptyset_i$.

Proof: Suppose that a non-empty sequence of searches, A, is carried out, which has probability p_A of revealing an object, and takes expected time t_A. Denoting by q'_A the probability that an object exists in the remaining boxes, we compare the payoff of a policy, π_0, of immediate retirement, with policy

[16]There is an analogy here with the 'conspiracy' concept mentioned in Subsection 1.2.2.

π_{A0}, which carries out search of A, and then retires:

$$\begin{aligned} V_{\pi_{A0}} &= t_A + q_A M(q'_A) \\ &= \begin{cases} t_A + q_A K q'_A & | \quad q'_A < 1 \\ t_A & | \quad q'_A = 1 \end{cases} \\ &> K q_A q'_A = V_{\pi_0} \qquad | \; q'_A < 1 \end{aligned}$$

If $q'_A < 1$ it is therefore better to retire immediately than to carry out such a sequence of searches, A, and then retire. Since we exclude boxes with $p_i = 0$, there is only sequence of searches with $q'_A = 1$. This is made up of all the boxes. The two possible optimal policies are therefore searching all available boxes or retiring immediately. The interchange argument can be used as usual to show that the optimal order in which to carry out search is in order of decreasing Ø. The payoff from searching all the boxes in this order is $\sum_{i=1}^{n} \prod_{j=0}^{i-1} q_j t_i$, while the cost of immediate retirement is $K\left(1 - \prod_{i-1}^{n} q_i\right)$. ■

The following conjecture, if proved, would give some insight onto the shape of the stopping region for the boxes case.

Conjecture 3.1 *If it is optimal in the boxes case to retire from state* $\mathbf{x}^A$ *and from state* $\mathbf{x}^B$*, then will also be optimal to retire from the state below:*
$(\left\lceil \frac{x_1{}^A + x_1{}^B}{2} \right\rceil, (\left\lceil \frac{x_2{}^A + x_2{}^B}{2} \right\rceil \ldots \left\lceil \frac{x_n{}^A + x_n{}^B}{2} \right\rceil)$.

We present a simple example to show that Conjecture 3.1 is not true in the general case. Consider the following four node types:-

$$\begin{aligned} A &= \left(0, 10, \tfrac{1}{2}s_C + \tfrac{1}{2}s_D\right) & B &= (\epsilon, 46, 0) \\ C &= (\epsilon, 16, 0) & D &= \left(\tfrac{1}{2}, K', 0\right) \end{aligned}$$

Let ϵ be positive but negligibly small. We now calculate the optimal course of action from states $(A_1 \cup A_2)$, $(B_1 \cup B_2)$ and $(A \cup B)$, assuming that $K' > K$, so that, if a node of type D is found, the optimal action will be to retire. For some value of K, we will be indifferent between immediate retirement from $(A_1 \cup A_2)$ and searching in an effort to show that no object exists.

$$V(A_1 \cup A_2) = 10 + \frac{1}{2}K + \frac{1}{2}\left(10 + \frac{1}{2}K + \frac{1}{2}(16 + 16)\right) = 23 + \frac{3}{4}K$$

So, the indifference value of K from state $(A_1 \cup A_2)$ is 92 since this is the solution to $23 + \frac{3}{4}K = K$. From $(B_1 \cup B_2)$, the cost to search and show that there is no object is 92, so this is also the indifference value of K from that state.

$$V(A \cup B) = 10 + \frac{1}{2}K + \frac{1}{2}(46 + 16) = 41 + \frac{1}{2}K$$

Solving this equation to get the indifference value of K for $(A \cup B)$, we get 82, a *lower* value than that for either $(A_1 \cup A_2)$ or $(B_1 \cup B_2)$, and so conclude that the retirement region in this case is not convex. The lower indifference value can be understood to stem from an interaction of the A and B nodes; for $K < 92$, the expression $(A_1 \cup A_2)$ is too unlikely to be worth searching, while the expression $(B_1 \cup B_2)$ cannot be searched in a way that yields any information.

3.6.4 Other Retirement Functions

We have considered above three of the most natural choices for the retirement function. The 'Retire and Guess' function of Subsection 3.6.2 could be modified without major difficulty to allow for different penalties of type I and type II errors. For many practical applications, $M()$ might not have one of the forms described above. If the node types did not have any simplifying properties, this would probably require some approximation or solution via dynamic programming since complete mathematical treatment of more complicated retirement functions seems likely to be a difficult exercise.

To underline the complexity of the optimal policy for other retirement functions, we present an example with two node types:

$$d_A = \left(\frac{1}{10}, 1, 1\right) \qquad\qquad d_B = \left(\frac{1}{10}, 1, \frac{1}{9} + \frac{8}{9}s_A\right)$$

Table 2 overleaf shows the optimal policy from various states for the retirement function $M(P) = 90I_{P>0.06}$.

```
               1111111111222222222233333333334444444
     0123456789012345678901234567890123456789012345
  0: ...........................aaaaaaaaaaaaaaaaaaa
  1: .........................=b==================
  2: .......................=bbb=================
  3: ......................aaaaaaaaaaaaaaaaaaaaa
  4: ....................=b====================
  5: ..................=bbb===================
  6: ................=bbbbb==================
  7: ..............=bbbbbbb=================
  8: ............=bbbbbbbbb================
  9: ..........=bbbbbbbbbbb===============
 10: ........=bbbbbbbbbbbbb==============
 11: ......=bbbbbbbbbbbbbbb=============
 12: .....aaaaaaaaabbbbbbbb============
 13: ...=b========bbbbbbbbb===========
 14: .=bbb=======bbbbbbbbbb==========
 15: bbbbb======bbbbbbbbbb==========
 16: bbbbb=====bbbbbbbbbb==========
 17: bbbbb====bbbbbbbbbbb=========
 18: bbbbb===bbbbbbbbbbb=========
 19: bbbbb==bbbbbbbbbbbb========
 20: bbbbb=bbbbbbbbbbbb========
 21: bbbbbbbbbbbbbbbbb========
 22: bbbbbbbbbbbbbbbbb=======
```

`.` : Optimal to retire.	`a` : Optimal to search a type A node.
`b` : Optimal to search a type B node.	`=` : Optimal to search either node.

Table 2: Optimal Policy for an Alternative Retirement Function

The number of A nodes is along the x axis, the number of B nodes is along the y axis. The optimal policy was calculated by dynamic programming. The program is included in Appendix C.

3.7 Overlook Probabilities

We now modify the model to allow the inclusion of overlook probabilities, as the original satisficing search model in the boxes case of Section 3.1 has been modified by Hall [31], Stone [82] and Wegener [87]. We now assume that when a node of type i contains an object and is searched there is a chance that the object will not be detected. This causes a subtle yet important change to the problem; if a node type with a non-zero overlook probability is available, then however many searches are carried out, it is impossible to conclude for certain that no object exists, because of the possibility of repeatedly having overlooked an object.

Let us revise our initial optimality criterion, 'expected time to termination'. We first observe that if no object exists, then it does not matter in which order the boxes are searched. Hence:

$$\begin{aligned} V_\pi &= E[V_\pi I_{\text{Object exists}} + V_\pi I_{\text{No object exists}}] \\ &= E[V_\pi I_{\text{Object exists}}] + v_{\text{All nodes}} \\ &= E[V_\pi | \text{Object exists}] P(\text{Object exists}) + v_{\text{All nodes}} \qquad (14) \end{aligned}$$

Since $v_{\text{All nodes}}$ is a constant, equation (14) shows that a policy which is optimal in the original sense of minimising 'expected time to termination' also minimises the 'expected time to termination given that an object exists'.

There is a positive probability that no object exists, which, with overlook probabilities, causes search to continue indefinitely. We are therefore required to modify the initial definition of optimality if we wish to use it to discriminate between policies, and so we restrict our attention to cases in which the choice of policy makes a difference – i.e. those cases in which an object exists. Instead of minimising expected time until termination we shall be minimising expected termination time given that an object exists. This modified definition of optimality does not conflict in any way with the one used up to this point, and indeed supersedes the previous definition which was introduced on grounds of simplicity.

Extend node type $d_i = (p_i, t_i)$ by adding $o_i \in [0, 1)$ as an extra parameter, which we shall refer to as the overlook probability. Since there is now no limit to the number of times it may be worthwhile to search a node of type i, we add another subscript to signify the number of times each node has been searched. Define node type (i, j) to be a node of type i which has been searched j times without success. Thus, to add an overlook probability of o_i to node type i, consider each type i node to be of type $(i, 0)$, and replace the single node type i by a family of node types i, j defined as follows:

$$t_{i,j} = t_i \qquad\qquad p_{i,j} = \frac{o_i^j (1 - o_i) p_i}{q_i + p_i o_i^j}$$

We modify the offspring distribution to ensure that the first time a node of type i is searched, a node of type $(i, 1)$ is generated in addition to any offspring revealed, whilst unsuccessful search of an (i, j) node reveals exactly one node, of type $(i, j + 1)$. The only theoretical complications of extending the model in this way arise from the step of the proof in which i^* is set equal to a node type which maximises $\emptyset_i$. Since there are now an infinite number of node types, we must show that there is a node type which achieves this maximum. We shall do this by showing that $\emptyset_{i,j} \to 0$ as $j \to \infty$. This proves that the maximum is achieved, since it implies that for any $\epsilon > 0$, there are only finitely many $\emptyset_{i,j} \geq \epsilon$.

Before proving this, we shall increase the generality slightly by allowing for non-constant overlook probabilities. Let $o_{i,j}$ be the overlook probability for the j^{th} time a node of type i is searched.

Lemma 3.12 *For any node type i, $\emptyset_{i,j} \to 0$ as $j \to \infty$.*

Proof: The probability that an object is discovered on the j^{th} search of a node of type i is $p_i(1 - o_{i,j}) \prod_{k=0}^{j-1} o_{i,k}$. Hence:

$$\sum_{j=1}^{\infty} p_i(1 - o_{i,j}) \prod_{k=0}^{j-1} o_{i,k} \leq 1 \quad \Rightarrow \quad p_i(1 - o_{i,j}) \prod_{k=0}^{j-1} o_{i,k} \to 0 \ \text{ as j} \to \infty.$$

$$\emptyset_{i,j} \propto p_{i,j} = p_i(1 - o_{i,j}) \prod_{k=0}^{j-1} o_{i,k} \to 0 \ \text{ as j} \to \infty.$$

■

3.8 Continuous Extension of the OR-Tree Model

We now consider an extension to the boxes case of the OR-tree search model of Section 3.1, based upon the understanding of Ø as a reward *rate*. Graphing time spent searching on the X-axis, and the probability that an object is found on the Y-axis, the original model is represented below. Also shown is a continuous representation. In this model, a box with constant reward rate Ø may be searched for time v to reveal an object with probability Øv.

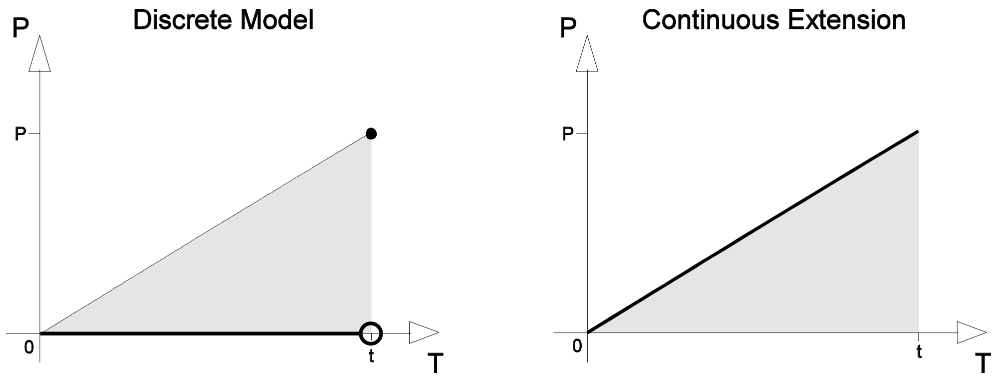

Figure 21: Continuous Extension of the Discrete Model

The boxes model of Section 3.1 has much in common with the continuous model in which box type i has a constant reward rate, Ø$_i$, and may be searched for a maximum time t_i. The optimum policies for the two models are identical, although the payoff is less for the continuous extension, because of the possibility of finding an object before a whole box is searched.

3.8.1 Linear Precedence Constraints

The case of linear precedence constraints dealt with in Subsection 3.1.1 has an immediate graphical interpretation. Consider the two boxes shown below. The gradients of the lines de and ef are $\emptyset_{(i,1)}$ and $\emptyset_{(i,2)}$ respectively, so iff $\emptyset_{(i,2)} > \emptyset_{(i,1)}$, then point e lies strictly inside the convex hull of d, e and f, in which case nodes $(i,1)$ and $(i,2)$ form a single indivisible block.

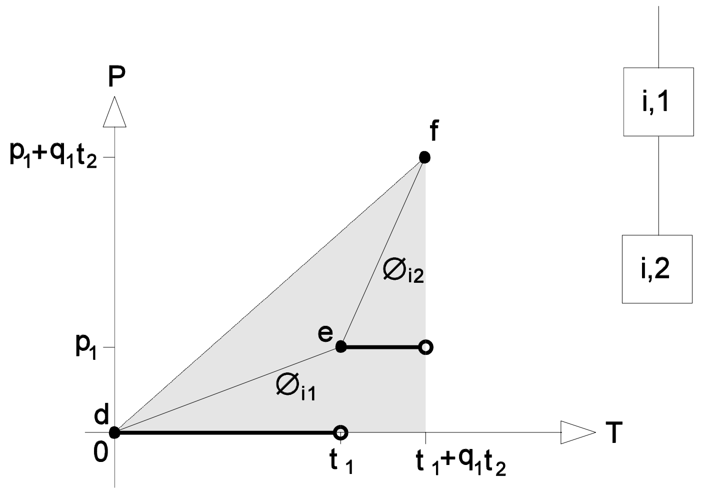

Figure 22: A Maximal Indivisible Block in the Continuous Case

By the same token, suppose further that stack i is charted in a similar fashion. If the points d and f lie on the convex hull of all the points in the stack, and this has different left- and right-gradients at these points, then the indivisible block def is maximal.

3.8.2 Concurrent Searching

We now consider how the continuous case might usefully be widened to deal appropriately with non-constant reward functions. Firstly, we note that if a box has a reward rate which is some non-decreasing function, $\Psi(t)$, of the time spent searching that box, we can consider instead $\Phi(t)$, the convex hull of the function $\Psi(t)$, for the same reason that boxes may be chunked into maximal indivisible blocks in the discrete case with linear precedence constraints. If the new rate of a node is a strictly decreasing function of t, we may wish to search that box for a vanishingly small period of time before changing to another box, which is a theoretical annoyance. Accordingly, we allow concurrent searching of boxes. Any number of boxes may be searched, with varying intensities, $i_1, \ldots i_n$ subject to the restriction that $\sum_j i_j = 1$.

As an example of how this simplifies the description of the optimal policy, suppose that two boxes are available for search, with decreasing reward rate functions $\Phi(t)$ and $\Phi(Kt)$ respectively. The optimal policy in this case is to search the boxes with respective intensities $i_1 = K/(K+1)$ and $i_2 = 1/(K+1)$, since this ensures that the reward rates of the two node types remain equal as search progresses.

3.9 Shape of $V()$

We now consider how $V(\mathbf{x})$ varies in x_i, by examining $\Delta V(\mathbf{x})_i$, defined as the increase in the payoff from adding a node of type i:

$$\Delta V(\mathbf{x})_i = V(x_1, \dots x_{i-1}, x_i + 1, x_{i+1} \dots x_n) - V(\mathbf{x})$$

The addition of an extra node of type i has two counteracting influences on $V()$:

1. *Decreasing* V: The extra node or its descendants may contain an object which can be relatively quickly found.
2. *Increasing* V: The extra node and its descendants must be searched before it is possible to terminate and conclude no object is present.

We now look at the net effect of these two influences. Observe that as $x_i \to \infty$, the probability that the existing supply of type i nodes would be exhausted tends to zero, and so both effects tend to zero. Hence $\Delta V(\mathbf{x})_i \to 0$ as $x_i \to \infty$.

Let us assume for notational convenience that the node types are indexed in order of decreasing $\O^*$, so node type n minimises $\O^*$. It is therefore possible to deduce the following value for $\Delta V(\mathbf{x})_n$ since we know that it is optimal to search the extra node last of all.

$$\begin{aligned} V(x_1, \dots x_{n-1}, x_n + 1) &= V(\mathbf{x}) + q(\mathbf{x}) V(0, 0 \dots 0, 1) \\ &= V(\mathbf{x}) + q(\mathbf{x}) t_n^* \end{aligned}$$

$$\text{Hence}: \ \Delta V(\mathbf{x})_i = q(\mathbf{x})t_n^* > 0$$

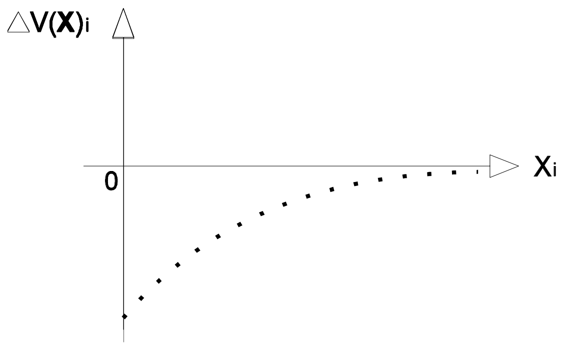

Figure 23: Shape of $\Delta V(\mathbf{x})_i$ for a Least Rewarding Node Type

As shown above, a least rewarding node type is *always* a liability. This is not true for any node types with an adjusted reward rate strictly greater than that of $\emptyset_n^*$. To see this, consider $\mathbf{x} = (0, 0 \ldots 0, N)$, for large N. As $N \to \infty$, the probability of ever terminating without finding an object becomes vanishingly small, and so does the second effect of adding a node of type i. The first effect, however, does not, and is non-zero since we required type i to have a reward rate strictly greater than $\emptyset_n$.

To see that $\Delta V(\mathbf{x})_i$ may have the shape as shown in Figure 24 overleaf, consider adding boxes of type 1 to a state $\mathbf{x}$, with $V(\mathbf{x}) > (\emptyset_\mathbf{1}^*)^{-\mathbf{1}}$. The optimal policy is to search the newly revealed boxes first, since they are of type 1. Thus:

$$V(x_1 + 1, x_2 \ldots x_n) = t_1^* + q_1^* V(\mathbf{x})$$

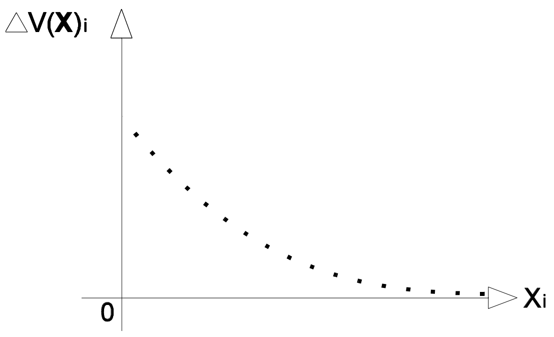

Figure 24: One Possible Shape of $\Delta V(\mathbf{x})_i$

$$\Delta V(\mathbf{x})_1 = V_1^* - p_1^* V(\mathbf{x}) = t_1^*(1 - \text{Ø}_1^* V(\mathbf{x}))$$

The shape of $\Delta V(\mathbf{x})_i$ may also have a turning point, as shown in Figure 25 below. Consider for example $\mathbf{x} = \mathbf{0}$. For any node type i, $\Delta V(\mathbf{0})_i < 0$, since $V()$ is minimised at $\mathbf{0}$.

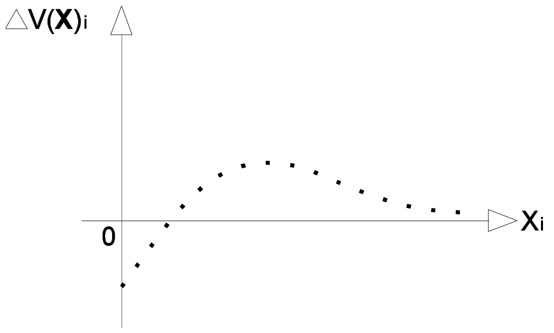

Figure 25: Second Possible Shape of $\Delta V(\mathbf{x})_i$

3.10 Summary

We have extended the branching bandits model of Weiss[88] in a number of ways. The equivalence between the semi-Markov continuous discounting case and the discrete case with transition dependent discounting highlighted in Subsection 3.5.4 could be used to derive a simple proof of the Gittins index theorem for discrete-time multi-armed bandits. The resulting proof would have much in common with Tsitsiklis' [85] proof for semi-Markov bandits.

The formulation of Ø as a reward rate and the discussion of overlook probabilities makes clear that in certain circumstances, the case of a (countably) infinite number of node types is tractable.

The non-constant 'retirement function' introduced in Section 3.6 is a powerful innovation, since it allows for its application to practical situations in which it is desirable to terminate search and make a decision based on its findings before an exact result is known. Specification of a retirement function allows for more powerful control of the search. As an illustration of its potential in the context of computer search, we note that it allows for a single search to be efficiently conducted in parallel, by allowing dynamic allocation and re-allocation of subsearches in accordance with findings.

Heuristic evaluations of nodes are commonly used only to determine which is the best of a given set of positions. This frequent under-utilisation of the information calculated means that the adoption of such an apparently simple evaluation function as that presented in Subsection 3.4.2 need not be as restrictive as at first appears. A standard evaluation function $H : x \mapsto R$

might, for the purposes of search control be replaced by $I_{H(x)>H(root)}$. This would separate out the positions into two classes, those which were an improvement on the current position and those which were the same or worse, which would suffice for some purposes.

An ability to make rational decisions about whether to terminate the search early seems likely to broaden the applicability of the model very considerably, since there are many situations where the goal of search is not to find an object but merely to discern as quickly as possible whether an object exists. Suppose, for example that a batch of goods has been manufactured. It is clearly of great value to have a procedure to automatically calculate whether the probability of eventual success is sufficient to warrant continuation of a series of quality control tests.

As currently described, the model is limited to graphs with an out-tree (or out-forest) structure. An obvious question is whether search of more general DAG's can be modelled in a similar fashion. The main problem with such an extension seems to be that the model is based upon a state which is a vector of scalars of fixed length. A more general DAG structure causes difficulties with this representation since it becomes necessary to keep track of previously searched branches.