# 2 Markov Chains

For a fully satisfactory model of a game being played by two agents, the statespace must be rich enough to represent not merely the state of the game[8], but also the amount of time left and a representation of the 'mind' — be it human or silicon — of each of the two agents (i.e. which portion of the game tree they have investigated, and how they have evaluated it). Human gameplayers have comparatively little difficulty taking into account their opponent's 'state of mind'. Bluffs, trick plays and psychological ploys are common currency among human game players, as is playing on the opponent's weaknesses. All these are strategems which involve not merely the state of the game itself, but also inferred knowledge about the opponent's state of mind.

When in a position which it has already established to be a game-theoretic draw, Schaeffer's draughts program, Chinook, chooses the line of play it estimates to be the most difficult for its opponent [75], so as to maximise its chance of winning through a blunder on its opponent's part — an idea he first suggested in [74]. This is a very primitive step towards the goal of properly modelling the opponent's deliberations, but is unmistakably a step in that direction.

As far as I am aware, no work has yet been published about a game-playing program, either actual or virtual, which involves modelling its oppo-

[8] typically the board position and whose move it is

nent's model[9]. Such meta-modelling would seem to lie beyond the realms of what it is currently useful to implement in a game-playing program. However, its inclusion in a program would have useful consequences, such as allowing the program to bluff and to exploit its opponent's known weaknesses. It is therefore my confident prediction that its importance will become apparent as computing power increases, and a point will be reached at which its implementation becomes desirable. This view is broadly supported by Berliner, Goetsch, Campbell and Ebeling [15] who conclude (for the game of Chess) that as computing power increases, so does the need to deploy it with greater selectivity.

Note that even a meta-model is an approximation. In fact it is impossible to derive a completely satisfactory model of the state of the game, because of the need to have a full model of the opponent's state of mind; since we must assume that the opponent is in turn is modelling our state of mind, we need to model his model of our model, leading to an infinite recursion. There is an obvious parallel here with Russell and Wefald's comments about how the need to optimally control meta-calculations incurs meta-meta-calculations, and so on. Their response to this recursive dilemma was to adopt the meta-greedy assumption, which requires the meta-meta-calculations to be of a particularly simple nature. Once this issue eventually becomes relevant to the playing strength of programs, a compromise of this kind seems likely to

[9]The closest reference I have found to such an idea is Korf's [45] study of the alpha-beta technique for players who use different evaluation functions.

be an adequate solution for many games[10].

## 2.1 Modelling Game Trees

We now consider the essential properties of a game tree. For the purposes of developing a game-independent model of the game-playing process, a flexible if simple mathematical model is to be preferred to a carefully carried out empirical study of some specific game or games. In choosing a model, a balance must be struck between simplicity on the one hand and descriptive power on the other.

As a first concession to simplicity, we assume the game tree to be, in fact a proper *tree*, as opposed to a DAG (directed acyclic graph). This is a standard assumption. Various authors have achieved minor speed-ups in their domain specific game-playing programs by the incorporation of a 'transposition table'[11], capitalising upon the fact that certain sequences of moves can be interchanged and still lead to the same position. Although no systematic study of this has yet been carried out, it seems likely that this is a second order consideration the importance of which does not justify its inclusion in the model at this stage.

The essential property of game trees, which underpins all search algo-

[10] As programs become more advanced, it will be interesting to see how the importance of such meta-modelling will vary depending upon the game involved. It seems likely to be of greatest importance for games of imperfect information, such as Bridge or Scrabble.

[11] A hash table of previously searched positions.

rithms, is that there is a correlation between evaluations of nodes at different points on the tree. Specifically, a node with a high score is likely to have, by and large, descendants which also have high scores. This is fundamental if meaningful information is to be inferred from partially explored game trees.

One simple method in which a tree with this property may be defined is by directly relating the scores of the parents and the children. Two important ways in which this can be done are deducing the parents' scores from those of their children ('upward percolation'), or generating the scores of the child nodes from the score of their parents ('downward percolation'). The former approach is, in one sense, the most natural, since it reflects how game theoretic values are actually defined. However, the consideration of future searches requires analysis of the possible children given the parent node, rather than the other way round, so this is not a convenient mathematical formulation to analyse[12]. Accordingly, most of the models we consider will assume downward percolation of node scores. We shall use a simple additive error structure: the difference between a node's score and its child's score is a random variable of known expectation.

[12]Some very rudimentary models with upward percolation of scores have been constructed, to analyse the minimax backing-up procedure. This approach was first applied by Nau [55, 54, 56, 57], and subsequently used by Pearl [63, 64] and Beal [7, 8].

## 2.2 A One-player Search Model

A one-player model allows for very substantial simplifications on the general framework set out above, since it avoids the difficult issue of modelling the opponent. The state is made up of the *game tree*, $G$, the remaining *search time*, $\tau$, and the subtree of $G$ which has been searched, termed the search *information*, $L$. Since $G \backslash L$ is unknown we exclude it from the state. The control decision to be made is a choice between moving and increasing the search information by spending one unit of search time to expand a leaf of the information.

Consider a one-player game in which on $h$ occasions, a choice must be made between two alternative moves. With the added condition that each attainable position may be reached by a unique sequence of moves, the corresponding game tree is a binary tree of height $h$. Each one of the $2^h$ possible different terminal positions has a score associated with it. The player's goal is to terminate the game at a node with as great a score as possible.

At the outset, the only score that is known to the player is the score at root, which we assume without loss of generality to be 0. Upon making a move, the player becomes aware of the score associated with the node at which they arrive. The player has a limited amount of time, $\tau$, to spend conducting an exploratory search of the game tree. The only nodes which may be searched are daughters of a node the value of which is already known. Therefore, the actions available are *making a move* and *expanding a leaf of the search information*.

The effects of making a move are shown below. Making a move rules out half of the end positions which are currently possible, while moving closer to the other half. It therefore corresponds to discarding half of the game tree, while the root of the remaining half becomes the new overall root.

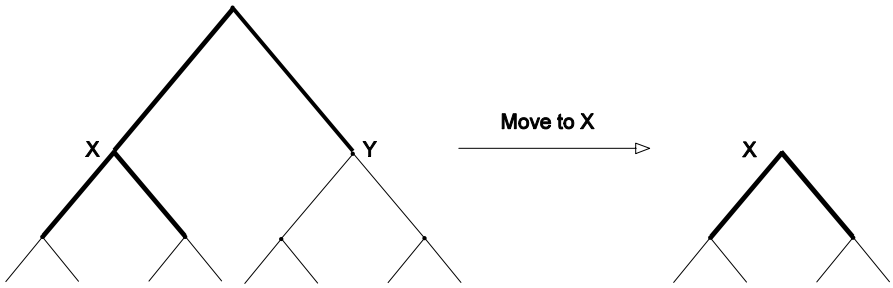


Figure 7: Making a Move

The distribution of scores among the nodes of the game tree is random, but the nature of the uncertainty is known *a priori*. There are in fact game-specific patterns of correlations between node scores at different heights – as exploited in the game of Othello by the Probcut model of Section 1.1.5. However, at this point we make the simplifying assumption that the game tree is a Markov field[13]. Specifically, the model assumes that the score associated with a parent node is always the mean of its two daughters, so one is better by a random variable, $Y$, and one is worse by the same amount. Since this is a one player game, there is an optimal policy amongst the class of non-randomised policies. In our discussion of one-player games we shall therefore

[13]That is, conditional upon knowledge of the score associated with its parent, a node's score is independent of the scores of any of its other ancestors.

only consider deterministic policies from now on, so in ensuing analyses of one-player games, any policies mentioned may be assumed non-randomised.

We now make another observation about the optimal policy. Let us define the set of policies, $S$, to be those which only ever make a move down the game tree if it is part of an uninterrupted sequence of moves to an unsearched node or to the end of the tree. The below result implies that there is an optimum policy in $S$.

**Theorem 2.1** *For every policy, $\pi \notin S$, there is a policy $\pi' \in S$ which obtains the same payoff.*

**Proof**: Consider a policy $\pi \notin S$. There must therefore be states in which it performs actions $\{m_1, \ \ldots \ m_n, \ s\}$ where $m_1, \ \ldots \ m_n$ are a sequence of move actions that do not reach an unsearched node, and $s$ is the immediately subsequent search action. Define $\pi'$ to mimic $\pi$ except when $\pi$ performs such a sequence. When this occurs, let $\pi'$ take actions $\{s, \ m_1, \ \ldots \ m_n\}$. Since the action sequence $\{m_1, \ \ldots \ m_n\}$ does not move to an unsearched node, it yields no information, so policy $\pi'$, which obtains the same payoff as $\pi$, is admissible. ■

This result is in line with intuition: since searching reveals information without restricting future choices whilst moving to an already searched node does exactly the opposite, it is to be expected that the optimal policy will search first and move later. The case of moving to an unsearched node is different, since it reveals that node's score and so makes it cheaper to search that node's descendants.

Policies in $S$ never move down the tree unless as part of a sequence of moves that terminates either at the bottom of the tree or at an unsearched node. That is, they take an action from the following set: {*Move beyond* i, *Expand* i, *Move to* e *and terminate*}, where $i$ is a leaf of the search information, and $e$ is a leaf of the search information with height 0.

The conditional independence of a node's score given the score of its parent means that the scores of the leaves of the search information comprise a sufficient statistic for inference of the unknown values. The leaves are therefore the only nodes of the search information which are relevant to future search. Hence, the tree of search information may be represented by a vector of leaves. The statespace may therefore be summarised by *the search time left*, $\tau$, and *the leaves of the search information*, $\mathbf{L}$. Suppose the game starts at the root of a binary tree which has height $h$. We shall address the problem of how best to spend a number $\tau \leq h$ units of search time.

As demonstrated above, for the purposes of deducing optimal policy the search information may be summarised by its leaves, so, if it has $N$ leaves, then it may be written $(L_1 \ \ldots \ L_N) \equiv \mathbf{L}$. A leaf $i$ of height $h_i$ and score $v_i$ shall be denoted $L_i = (h_i, v_i)$. Every nonterminal node $L_i$ has two daughters, which we denote $L_{(i,1)}$ and $L_{(i,2)}$. Since they are one level further down the tree:

$$h_{(i,1)} = h_{(i,2)} = h_i - 1$$

Each node's $Y_i$ value is an independent and identically distributed symmetric random variable. We shall use $\kappa$ to denote $E[|Y_i|]$, the expected difference

between a node's score and those of its daughters. The scores of $L_{(i,1)}$ and $L_{(i,2)}$ satisfy the following:

$$v_{(i,1)} = v_i + Y_i \qquad\qquad v_{(i,2)} = v_i - Y_i$$

Define $V_{M_i}(\mathbf{L}, \tau)$ to be the value of the game $(\mathbf{L}, \tau)$ subject to the constraint that the next action taken will be to *move* beyond leaf $L_i$, or, if $L_i$ is a terminal node, to terminate:

$$V_{M_i}(\mathbf{L}, \tau) = \begin{cases} E[V(L_{(i,1)}, \tau)] & | \quad h_i > 0 \\ v_i & | \quad h_i = 0 \end{cases}$$

Further define $V_{S_i}(\mathbf{L}, \tau)$, for $\tau > 0$, the value of the game $(\mathbf{L}, \tau)$ subject to the constraint that the next action taken will be to *search* beyond leaf $L_i$, or, if $L_i$ is a terminal node, to discard a unit of time:

$$V_{S_i}(\mathbf{L}, \tau) = \begin{cases} E[V(L_1 \ldots L_{i-1}, L_{(i,1)}, L_{(i,2)}, L_{i+1} \ldots L_N, \tau - 1)] & | \quad h_i > 0 \\ E[V(\mathbf{L}, \tau - 1)] & | \quad h_i = 0 \end{cases}$$

### 2.2.1 Optimal Policy for $\tau \leq h$

Lemmas 2.2, 2.3 and 2.4 all use the following induction hypothesis. The last of these will show that if it is true for $\tau = N$, then it is true for $\tau = N + 1$. We first note that it is true for $\tau = 0$.

$$V(\mathbf{L}, \tau) \leq \max_j \{v_j\} + \tau\kappa \tag{1}$$

**Lemma 2.2** *If inequality (1) holds for $\tau = N$, then for any $i$:*

$$V_{S_i}(\mathbf{L}, N) \leq \max_j \{v_j\} + N\kappa \tag{2}$$

**Proof**: For any $i$:

$$\begin{aligned}
V_{S_i}(\mathbf{L}, N+1) &= E\left[V(L_1 \ldots L_{i-1}, L_{(i,1)}, L_{(i,2)}, L_{i+1}, \ldots L_N, N)\right] \\
&\leq E\left[\max_{j \in \{1 \ldots i-1,(i,1),(i,2),i+1 \ldots N\}} \{v_j\}\right] + N\kappa \,\text{from (1)} \\
&\leq E\left[\max\left\{\max_{j \in \{v_{(i,1)}, v_{(i,2)}\}} \{v_j\}, \max_{j \neq i}\{v_j\}\right\}\right] + N\kappa \\
&\leq E\left[\max\left\{v_i + Y, v_i - Y, \max_{j \neq i}\{v_j\}\right\}\right] + N\kappa \\
&\leq E\left[\max\left\{v_i + |Y|, \max_{j \neq i}\{v_j\}\right\}\right] + N\kappa \\
&\leq \max_j\{v_j\} + (N+1)\kappa \qquad \text{since E[|Y|]} = \kappa
\end{aligned}$$

■

**Lemma 2.3** *If inequality (1) holds for $\tau = N$, then for any $i$:*

$$V_{M_i}(\mathbf{L}, N+1) \leq \max_j\{v_j\} + (N+1)\kappa \tag{3}$$

**Proof**: We prove this by induction on $H$, the number of search units required to completely explore the search information. The lemma holds if $H = 0$, since $V_{M_i}(\mathbf{L}, N+1) = \max_j\{v_j\}$. So, for the remaining cases where $H > 0$,

for any $i$:

$$\begin{aligned} V_{M_i}(\mathbf{L}, N+1) &= E[V(L_{(i,1)}, N+1)] \\ &= E[V((h_{(i,1)}, v_{(i,1)}), N+1)] \end{aligned}$$

We observe that the search information, $\mathbf{L}$, only contains a single leaf and so the node that decides the terminal payoff must be a direct descendant of the node $L_{(i,1)} = (h_{(i,1)}, v_{(i,1)})$. The additive structure of the tree therefore implies that $V(\mathbf{L})$ is linear in $v_{(i,1)}$. Hence:

$$\begin{aligned} V_{M_i}(\mathbf{L}, N+1) &= V((h_{(i,1)}, E[v_{(i,1)}]), N+1) \\ &= V((h_i - 1, v_i), N+1) \\ &= \max\{V_{M_1}((h_i - 1, v_i), N+1), V_{S_1}((h_i - 1, v_i), N+1)\} \\ &\leq \max\{V_{M_1}((h_i - 1, v_i), N+1), v_i + (N+1)\kappa\} \text{ from (2)} \end{aligned}$$

The result is proved with the observation that since the search information $(h_i - 1, v_i)$ has a lower $H$ value than $\mathbf{L}$, equation (3) implies by induction on $H$:

$$V_{M_i}((h_i - 1, v_i), N+1) \leq v_i + (N+1)\kappa$$

■

**Lemma 2.4**

$$\forall \tau : \quad V(\mathbf{L}, \tau) \leq \max_j\{v_j\} + \tau\kappa$$

**Proof**: Lemmas 2.2 and 2.3 show that if (1) is true for $\tau = N$, then (2) and (3) also hold, so

$$\max\left\{\max_i\{V_{M_i}(\mathbf{L}, N+1)\}, \max_i\{V_{S_i}(\mathbf{L}, N+1)\}\right\} \leq \max_j\{v_j\} + (N+1)\kappa$$

The optimality equation for $V$ is:

$$V(\mathbf{L}, N+1) = \max\left\{\max_i\{V_{M_i}(\mathbf{L}, N+1)\}, \max_i\{V_{S_i}(\mathbf{L}, N+1)\}\right\}$$

Combining these we see if that inequality (1) is true for $\tau = N$ then this implies that $V(\mathbf{L}, N+1) \leq \max_j\{v_j\} + (N+1)\kappa$, which is inequality (1) for $\tau = N+1$ so the result is proved by induction on $\tau$. ■

**Theorem 2.5** *For $\tau \leq h$, the optimal payoff is $\tau\kappa$. Policy $\pi^*$, defined below, achieves this payoff and so is therefore optimal.*

| Policy $\pi^*$ is to pick a leaf $L_i$ which satisfies $v_i = \max_{1\leq j\leq N}\{v_j\}$ and |
|---|
| if $\tau > 0$ then *Search* it, |
| if $\tau = 0$ and $h_i > 0$ then *Move* to a random daughter of it, |
| if $\tau = 0$ and $h_i = 0$ then move to it. |

**Proof**: The payoff from playing policy $\pi^*$, $V_{\pi^*}$, from a state where $\min_i\{h_i\} \geq \tau$ is given by:

$$V_{\pi^*}(\mathbf{L}, \tau) = \max_i\{v_i\} + \tau\kappa$$

The value of the game played under the optimum policy, $V$, is an upper bound on the value of the game played under any other policy, so combining this with Lemma 2.4, we see:

$$V_{\pi^*}(\mathbf{L}, \tau) \leq V(\mathbf{L}, \tau) \leq \max_i\{v_i\} + \tau\kappa = V_{\pi^*}(\mathbf{L}, \tau)$$

■

### 2.2.2 Optimal Policy for $\tau > h$

The optimal policy for $\tau > h$ depends upon the properties of the distribution $P()$. At one extreme, if $Y$ takes only the values -1 and 1, the paths of the game tree represent all possible courses of a 1-dimensional random walk of length $h$. In this case, the set of terminal node values is completely determined, and $\tau = h$ is sufficient time to discover the path to the best one, so any extra search time is superfluous.

Conversely, for any $P()$ with support along the whole real line, all $2^h$ terminal nodes must be searched before it can be definitely established which one of them has the greatest score, so search time only becomes superfluous for $\tau \geq 2^h - 1$. We conclude our study of this model with a short theoretical section in which we shall prove a restriction on the optimal policy for $\tau > h$ and outline how it may be tackled as an optimal stopping problem.

Recall our earlier definition of $S$, as the set of policies which only ever make a move down the game tree if it is part of a sequence of moves to an unsearched node or to the end of the tree. Define $S^* \in S$ to be set of such policies which never move with $\tau > 0$.

**Lemma 2.6** *There is an optimal policy in $S^*$.*

**Proof**: We show that for every policy $\pi' \in S \backslash S^*$, there is a policy $\pi^* \in S^*$ which obtains the same payoff. Once again we use induction on $H$, the number of time units required to search all of the search information. The result is trivially true for states in which H=0, so we now consider cases in

which $H > 0$.

Firstly, let $\pi^*$ mimic $\pi'$ until the first state in which $\pi'$ suggests a move action with $\tau > 0$ . Since $\pi' \in S$, such an action must be the first of a consecutive sequence which leads either to the end of the game or to an unsearched node. The first of these cases is trivial; $\pi^*$ can achieve an identical payoff by taking the same sequence of moves once it has exhausted the search time by making random searches until $\tau = 0$.

The remaining case is that in which $\tau > 0$ and $\pi'$ moves to an unsearched node, $L_{(i,1)}$, achieving the following payoff:

$$\begin{aligned} V_{M_i\pi'}(\mathbf{L}, \tau) &= E[V_{\pi'}(L_{(i,1)}, \tau)] \\ &= E[V_{\pi'}((h_{(i,1)}, v_{(i,1)}), \tau)] \\ &= V_{\pi'}((h_i - 1, E[v_{(i,1)}]), \tau) \\ &= V_{\pi'}((h_i - 1, v_i), \tau) \end{aligned} \tag{4}$$

Since $L_{(i,1)}$ has a lower H value than $L_i$, the induction hypothesis tells us that the below is true for some $\pi'^* \in S^*$:

$$V_{\pi}((h_i - 1, v_i), \tau) \leq V_{\pi'^*}((h_i - 1, v_i), \tau) \tag{5}$$

Policy $\pi^*$ omits the move action of $\pi'$, but ignores all nodes of $\mathbf{L}$ except $L_i$.

$$V_{\pi}(\mathbf{L}, \tau) = V_{\pi}((h_i, v_i), \tau) \tag{6}$$

Policy $\pi^*$ then carries out the same sequence of actions from $((h_i, v_i), \tau)$ as policy $\pi'^*$ does from $((h_i - 1, v_i), \tau)$. If concludes by making a random move

to reach the end of the game, leaving the expected payoffs of the two policies equal.

$$V_{\pi'^*}((h_i - 1, v_i), \tau) = V_{\pi^*}((h_i, v_i), \tau) \tag{7}$$

Policy $\pi^*$ carries out a sequence of searches, mimics $\pi'^* \in S^*$ and concludes with a move to an unsearched node, and is therefore also in $S^*$. From (4), (5), (6) and (7):

$$\begin{aligned} V_{M_i\pi'}(\mathbf{L}, \tau) &= V_{\pi'}((h_i - 1, v_i), \tau) \\ &\leq V_{\pi'^*}((h_i - 1, v_i), \tau) \\ &= V_{\pi^*}((h_i - 1, v_i), \tau) \\ &= V_{\pi^*}(\mathbf{L}, \tau) \end{aligned}$$

■

We now suggest the following conjecture as a step towards proof of the optimal policy for $\tau > h$.

**Conjecture 2.1** *For $\tau \geq h$, there is an optimal policy in $S$ which is certain to reach a state in which for some node $L_i$, $v_i = max_j\{v_j\}$ and $\tau = h_i$ without moving to an unsearched node.*

If this is proved true, then the following result is also proved.

**Corollary 2.7** *For $\tau \geq h$, there is an optimal policy, $\pi^* \in S^*$, that is, one which never moves to an unsearched node.*

**Proof**: From Conjecture 2.1, there is an optimal policy in $S$ which is certain to reach a state in which for some node $L_i$, $v_i = max_j\{v_j\}$ and $\tau = h_i$, with moving to an unsearched node Theorem 2.5 establishes the following upper bound on the optimal payoff, obtainable from such a point:

$$V(\mathbf{L}, \tau) \leq max_j\{v_j\} + \tau\kappa$$

This applies to states in which for all $i$, $h_i > \tau$ . This case is more restricted, but the bound can be achieved by discarding all nodes other than $L_i$ and playing optimally as described in Theorem 2.5. Since $\tau = h_i$, such a policy searches to the very end of the tree. ■

Corollary 2.7 tells us that there is an optimal policy which never moves to an unsearched node for $\tau = h + 1$. This therefore defines the shapes of the search information which it may be optimal to search; since it must include an unbroken path of search from root down to a leaf, one unit of search remains, which — assuming $P()$ is such that it is needed — will form a bifurcation at some point down this strand.

Once this extra unit of time has been spent, the problem is reduced to the '$\tau = h$' case, and optimal policy is established. The problem may therefore be treated as one of optimal stopping — the 'stopping' being interpreted as using up the extra unit of time to create a bifurcation of the search information. The practical upshot of this is that the optimal policy with $h + 1$ units of time, as shown overleaf, is to search down a single variation, always picking the better daughter, until the $Y$ value is sufficiently small. When this occurs, it is optimal to 'backtrack' up the variation one level and search the sister

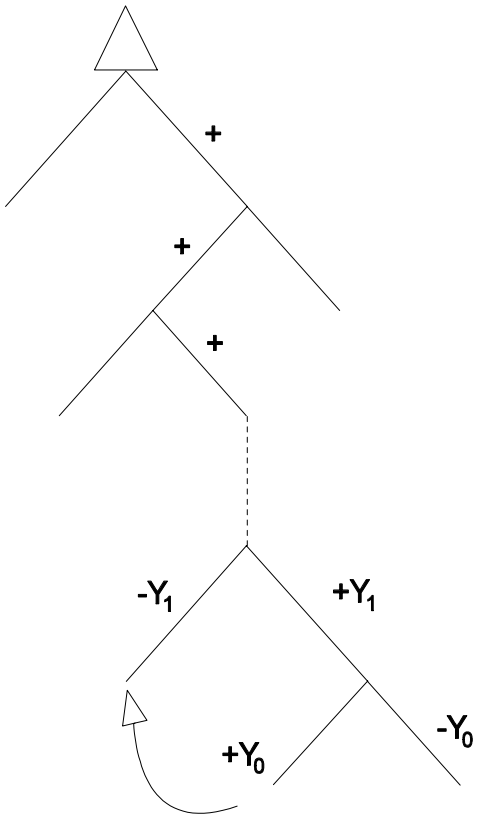


Figure 8: Optimal Policy for $\tau = h + 1$

of the last node just searched. The threshold value of $Y$ below which $Y$ is deemed to be 'sufficiently small' is a function of the last two $Y$ values found. It also decreases as does the height of the tree remaining. If the bottom of the tree is reached without the extra time unit having been used, it is optimal to use it whatever the last two values of $Y$, since there is no further use for it.

## 2.3 A Fuel Control Problem

We now consider a fuel control problem inspired by the tree search game of the previous section. As before, the game consists of moving to the bottom of a finite tree, possibly with the aid of one or more searches. Similarly, the reward from playing the game is the value of the leaf arrived at.

In contrast, however, to the original problem, we assume that the set of leaf values is known. Moreover the value of each separate leaf node, $i$, is $f(i)$, which is known. The uncertainty in this game concerns the *moves*. The player at root has perfect information of the game tree ahead of him, but cannot distinguish which of the two moves available will lead to which half of the game tree. The action 'move' causes the player to choose one random half of the game tree and move there. The 'search' option reveals this information to him. This is equated to the expenditure of fuel (or search time).

Faced with a height $h$ game tree, $h$ time units are sufficient to achieve the optimal result. The following policy is optimal:

1. Select a leaf node, $L$, which achieves the greatest reward.
2. Carry out a search to discern which move is which, and move to the half of the tree which contains node $L$. Go to 2.

The optimal policy for less than $h$ time units is more interesting. It is useful to introduce some notation at this point. We shall define two sequences of functions on the game tree, $V_n()$ and $V_n^*()$ . Each function accords values

to each node of the tree. We denote by $p_{ij}$ the probability of arriving at $j$ if a move is made at random from $i$, so $p_{i(i,1)} = p_{i(i,2)} = \frac{1}{2}$.

Define $V_0()$ as follows:

$$V_0(i) = \begin{cases} f(i) & \mid \ i \text{ is a leaf} \\ \sum_j p_{ij} V_0(j) & \mid \ \text{otherwise} \end{cases}$$

That is to say, $V_0()$ is the harmonic extension of the $f()$ values at the leaves.

Define $V_0^*()$ with reference to $V_0()$ as follows:

$$V_0^*(i) = \begin{cases} f(i) & \mid \ i \text{ is a leaf} \\ \max_{j|p_{ij}>0} \{V_0(j)\} & \mid \ \text{otherwise} \end{cases}$$

Since $V_0()$ is the harmonic extension of $f()$, $V_0^*()$ is a majorant of $V_0()$. It is equal to the payoff from a policy of expending one unit of time straight away, and then making random moves thereafter. We now define the following function $V_1()$ with respect to $V_0^*()$:

$$V_1(i) = \begin{cases} f(i) & \mid \ i \text{ is a leaf} \\ \max \left\{ V_0^*(i), \sum_j p_{ij} V_1(j) \right\} & \mid \ \text{otherwise} \end{cases}$$

Theorem 2.8 shows that this function gives the value of the game when one unit of time remains and the optimal policy is followed. An example of these two functions is given overleaf in Figure 9.

The $V_1^*()$ tree is now defined with respect to $V_1()$ in a similar fashion to the way in which $V_0^*()$ was defined in terms of $V_0()$. This process is repeated, allowing computation of $V_\tau()$ for any $\tau$.

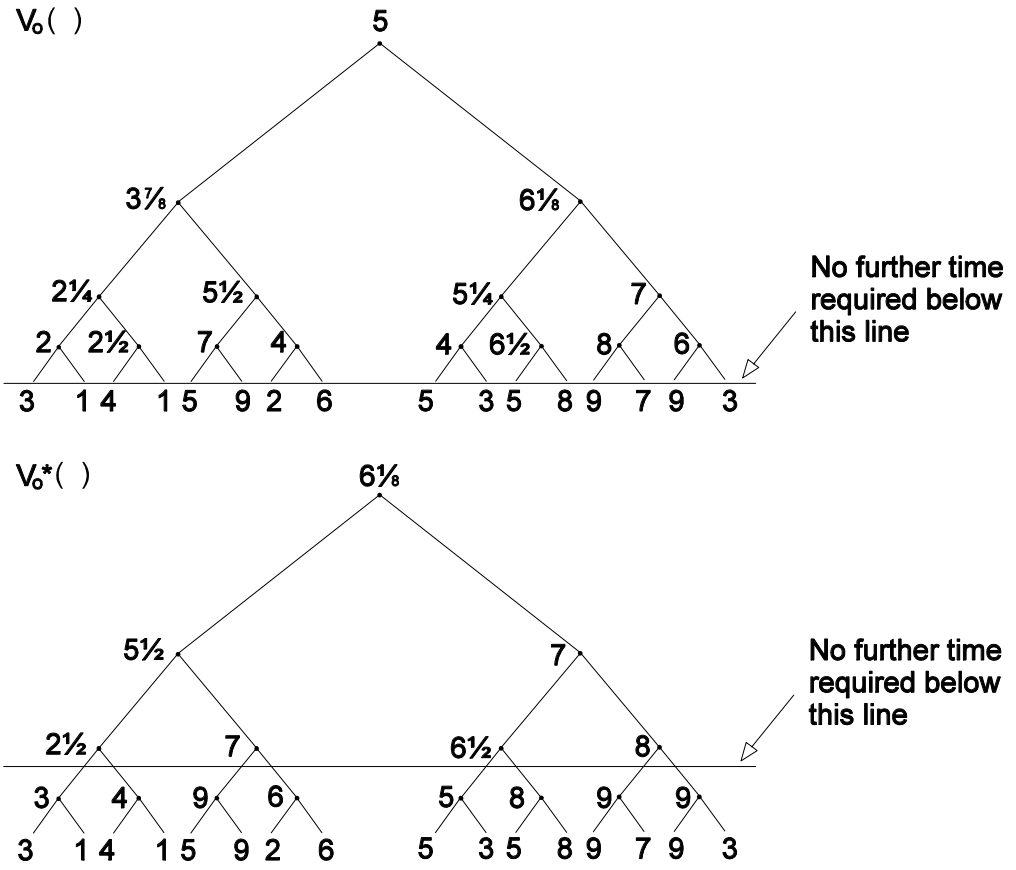


Figure 9: Example $V_0()$ and $V_0^*()$ Values

**Theorem 2.8** *The payoff from playing optimally with $\tau$ units of time is given by $V_{\tau+1}()$.*

**Proof**: If $\tau = 0$, the only available policy is to move at random, so the result is true since $V_0()$ is the harmonic extension of $f()$. We now apply induction, assuming that it is true for $\tau = n$. The optimality equation for $V_{n+1}()$ is as

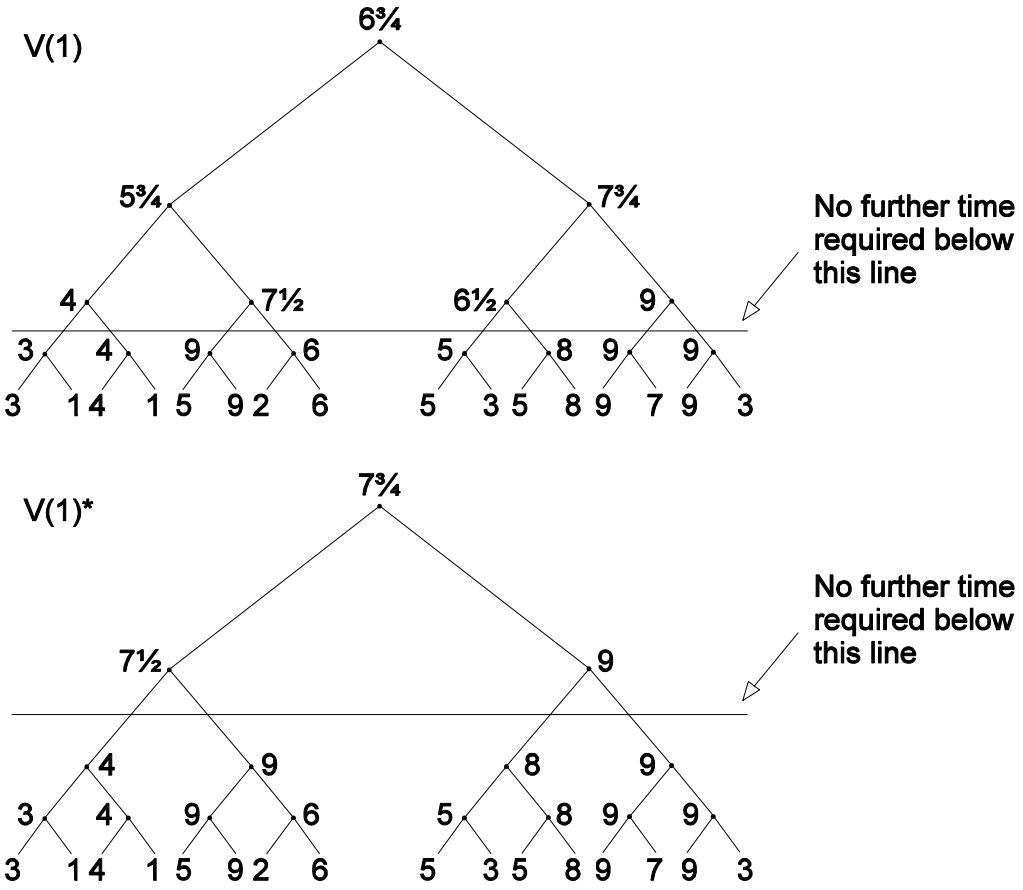


Figure 10: Example $V_1()$ and $V_1^*()$ Values

follows:

$$V_{n+1}(i) = \max\left\{V_n^*(i), \sum_j p_{ij} V_{n+1}(j)\right\}$$

This shows it is the minimal superharmonic majorant of $V_n^*()$. Hence it is the optimality equation for optimally stopping $V_n^*$. ■

**Corollary 2.9** *The function $V_\tau^*()$ gives the optimal payoff if $\tau + 1$ units of fuel are available, and one of them has to be expended immediately.*

**Proof**: If $i$ is a leaf, then only one payoff is possible, so the corollary is trivially true. Otherwise:

$$V_\tau^*(i) = \max_{j|p_{ij}>0} \{V_\tau(j)\}$$

■

For $n \geq h$, the process is trivial since $V_n() = V_{n+1}()$, reflecting the fact that no more than $h$ search units are required to ensure an optimal choice of moves. The problem is now solved, since the optimal action from a node with $\tau$ time units available for search can be deduced from consideration of the $V_\tau()$ values of that node and its children. The remaining $V_n()$ and $V_n^*()$ functions for the example are shown in Figures 10 and 11.

Having understood this simple model, we now consider several generalisations. Firstly, the game tree need not be binary, indeed it need not have a constant branching factor. Furthermore, it need not have a fixed height, since any finite irregular tree of maximum height $h$, maximum branching factor $b$ can be modelled as a regular tree of height $h$, branching factor $b$, simply by the addition of 'dummy' branches where each daughter has the same score as the parent node.

What is more, the structure of the game need not be a tree; a DAG can be treated in an identical fashion. This is perhaps most easily seen by observing that any DAG may be expanded to an equivalent out-tree, by simply making copies of the nodes which have in-degree of greater than one, as shown below. We note that in practice it is not even necessary to perform a transformation.

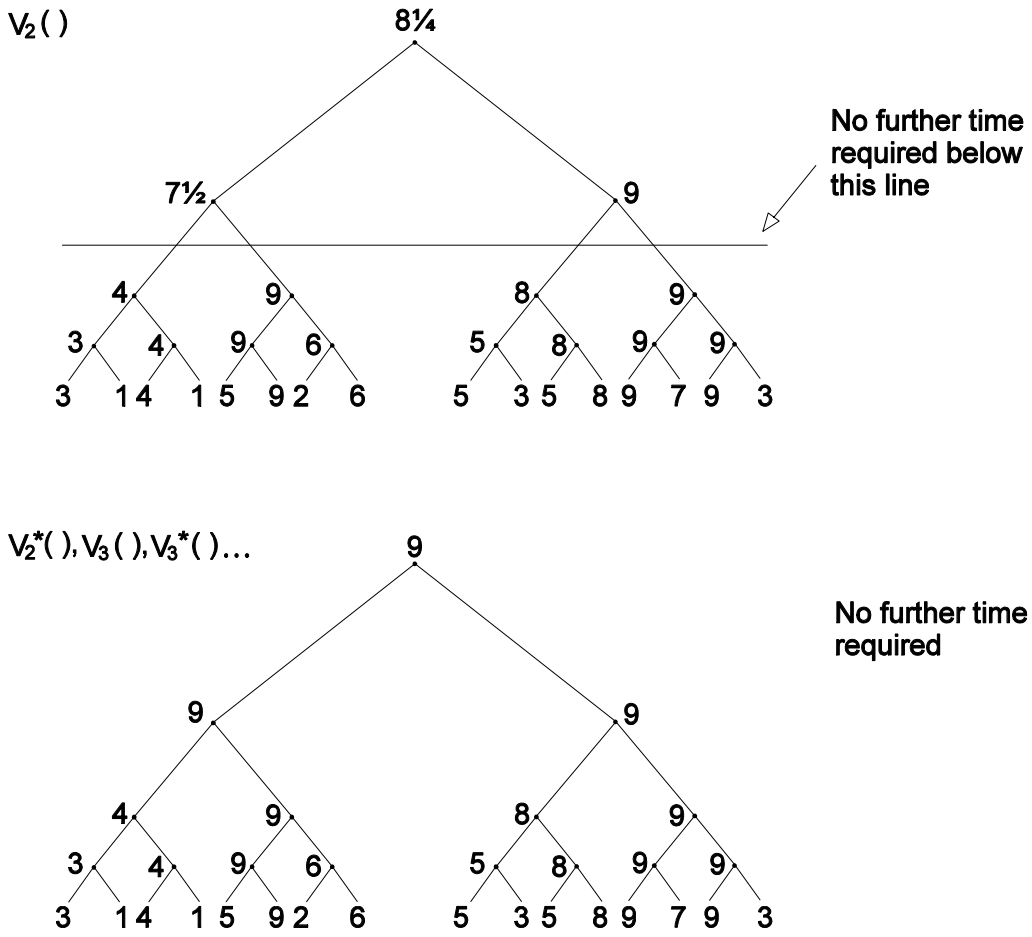


Figure 11: Example $V_2()$ and $V_2^*()$ Values

We have now seen how the method developed for the binary tree case may be applied without difficulty to any finite DAG. We now broaden the scope of the model still further by re-interpreting the nature of the uncertainty involved in 'moving' in a more general fashion. Let us replace the notion of spending a unit of time to 'discover which move is which' by that of 'expending a unit of fuel in order to control our movement'. This allows specification of a general probability distribution over a node's possible daughters. The

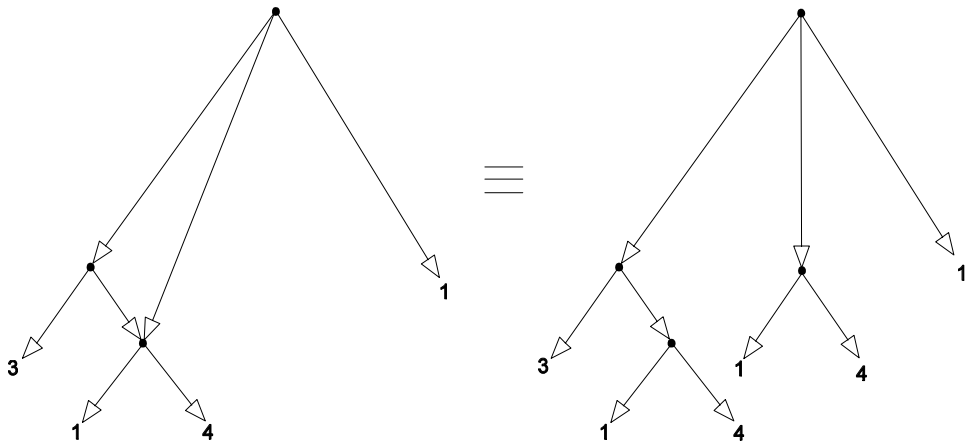


Figure 12: Expansion of a DAG to an Equivalent Out-tree

previous model, which assumed that random movement to each of a node's daughters was equally likely, is thus reduced to a special case.

We are now in a position to understand the problem description as a fuel allocation problem. The problem is defined on a DAG, with one source node, at which some 'particle' begins, and one or more sink nodes, at which it can end the game. Each sink node, $i$, is associated with a finite payoff, $f(i)$, the value of the game if the particle reaches that node. Every arc is associated with a positive weight. We naturally require that the sum of weights associated with the outarcs of a node be 1, unless the node is a sink, in which case it has no outarcs, by definition.

If no fuel is expended, the particle makes a random transition, following an outarc of the node where it currently resides with a probability equal to the associated weight. If the player chooses to expend a unit of fuel he may select any of the outarcs from the particle's current position and move the particle along it.

If enough time units are available, it is possible to control every transition made by the particle, directing it to reach a sink with the greatest reward, which is of course optimal. This will be reflected in the calculation of the $V_\tau()$ value of root.

We now consider one final generalisation — that of allowing cyclical graphs. This causes no problem for the theoretical definition of the $V()$ and $V^*()$ values. In practice, they become harder to calculate, since the straightforward 'bottom up' recursive calculation method is no longer possible if the graph contains possible cyclical particle trajectories, owing to the mutual dependence of the $V()$ values of the nodes in the cycle.

In this case we resort to application of the policy improvement algorithm. We calculate $V_0()$, the harmonic extension of $f()$ as before. To calculate $V_0^*()$, the policy improvement algorithm starts with a policy which everywhere expends a unit of fuel and then proceeds at random. It then proceeds by iteratively decreasing $E$, the set of points at which it is optimal to expend fuel. The policy of expending the fuel unit at a node in set $E$ has a payoff $V_0^E()$, defined as follows:

$$V_0^E(i) = \begin{cases} f(i) & \mid \; i \text{ is a sink node} \\ \max\limits_{j \mid p_{ij} > 0} \{V_0(j)\} & \mid \; i \in E \\ \sum\limits_j p_{ij} V_0^E(j) & \mid \; \text{otherwise} \end{cases}$$

The steps of the policy improvement algorithm are:

1. Let $E$ contain all the non-sink nodes of the graph.
2. Delete from $E$ any nodes $i$ such that:

$$\sum_j p_{ij} V_0^E(j) > \max_{j|p_{ij}>0} \{V_0(j)\}$$

   (If there are no such nodes, go to step 4.)
3. Recalculate the $V_0^E()$ values. The fact that the new $V_0^E()$ function is a strict majorant of the old one is a consequence of the definition of $V_0^E()$ and the choice of points to omit from $E$. Go to step 2.
4. Since this choice of $E$ is optimal, $V_1() \equiv V_0^E()$. Deduce the set $A_1$ of nodes with $V_1(i) = max_j\{f(j)\}$. This set contains points from which the optimal payoff is attainable without further time units.

A similar method can be used to deduce $V_{N+1}()$ from $V_N()$, for any $N$. Examination of step 2 shows that termination occurs after a number of iterations not exceeding the number of nodes. Note the criterion for removing a point from $E$:

$$\sum_j p_{ij} V_0^E(j) > \max_{j|p_{ij}>0} \{V_0(j)\}$$

Since the $V_N^E()$ values are never adjusted downwards, it is never optimal to add a node back into $E$ once it has been removed. This, together with

the criterion for reaching step 4, prove the assertion that the choice of $E$ is optimal at this point. Finally, we note that once a node is added to $A_i$ it need not be included in subsequent iterations. Iterations are therefore likely to involve less and less calculation as time proceeds, since more and more nodes have been added to $A_i$.

We conclude our discussion of this problem with a special case — grids which have a translationally invariant transition structure. Define the *interior* of a translationally independent lattice to be the set of points which are not sink nodes, and which have a transition structure identical to each of their neighbours. i.e. those more than one step away from any sink nodes, edges or other irregularities in the lattice.

**Theorem 2.10** *It is optimal never to expend the last remaining unit of fuel from a state in the interior of a translationally independent lattice.*

**Proof**:

Let node $x$ be a node on the interior of a translationally invariant lattice, with neighbours denoted $x+1, \ldots x+n$. Suppose that if no fuel is expended, cell $x+i$ is reached with probability $p_i$. Note that because $x$ is in the interior of the lattice, $(x+i)+j = (x+j)+i$, since making a move in direction $i$ followed by one direction $j$ is equivalent to making these moves in the reverse order.

$$V_{N+1}(x) = \max \left\{ \max_i \{V_N(x+i)\}, \sum_{j=1}^{n} p_j V_{N+1}(x+j) \right\}$$

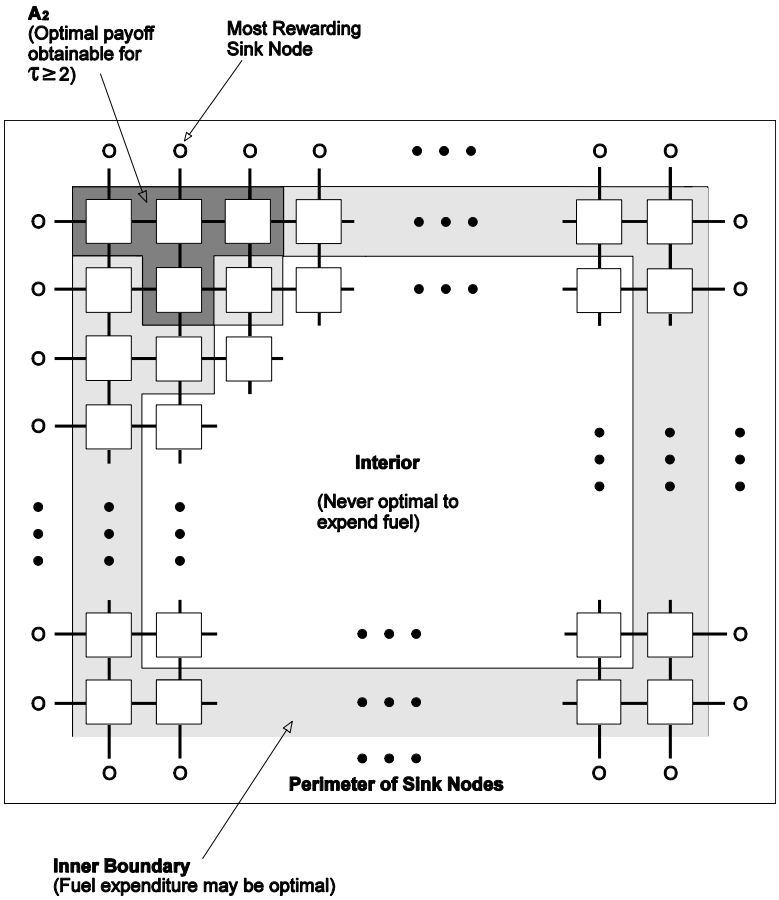


Figure 13: Calculation of $V_3()$ on a Translationally Invariant Grid

Consider the optimality equation above. We show that the second of the terms of the maximisation dominates the first.

$$\begin{aligned}
\sum_{j=1}^{n} p_j V_{N+1}(x+j) &\geq \sum_{j=1}^{n} p_j V_N((x+j)+i) && \forall i \\
&\geq \sum_{j=1}^{n} p_j V_N((x+i)+j) && \forall i \\
&\geq \max_i \left\{ \sum_{j=1}^{n} p_j V_N((x+i)+j) \right\} \\
&\geq \max_i \{V_N(x+i)\}
\end{aligned}$$

Specifically, if $N = 0$, we observe:

$$\sum_{j=1}^{n} p_j V_1(x+j) \geq \max_i \{V_0(x+i)\}$$

■

This result speeds up application of the policy improvement algorithm for cases in which some subset of the graph is a translationally invariant lattice, as is illustrated in above. It can be understood in familiar terms. Not expending fuel equates to making a random move, while expending fuel entails making a decision about how to do so. For positions in which the nearest sink or edge is more than one transition away, a random move causes a reversible change to the node's position, since if desired, a time unit may be expended to move back. Therefore, for every policy which first expends fuel and then moves randomly there is an equivalent policy which interchanges these actions. Not only does this entail no loss, it is better in general, since more information is available (the direction of one more random transition) upon which to base the decision about how to move.

## 2.4 Summary

In the general case of the game-playing problem as we have discussed it, strictly 'optimal' play is, even in theory, an unachievable concept, due to the complications in trying to anticipate the opponent's policy. We must therefore study less complicated models if a strictly optimal policy is sought. One such is the one-player game of Section 2.2. The optimal policy for the case of $\tau \leq h$ is fairly trivial, but considerable work had to be put into its proof. Similarly, the results for $\tau > h$ represent the fruits of some considerable effort, even though they stop short of actually specifying an optimal policy, and are probably of theoretical rather than practical interest.

The fuel control model of Section 2.3 was deduced as something of a 'spin-off' of investigation into the main one-player tree search model. It has a pleasing generality, in that it goes so far beyond the binary tree case of its parent model. I therefore believe that it may well turn out to be of practical use without further modification, and its serendipitous discovery is further evidence of the value of developing tree search models.