# 5 Time Control

The difference between a tree search procedure and a game-playing algorithm is an important and underestimated one. The latter will include a tree search procedure, but something more is required — it must also have a method for terminating searches at some point. Basically, a tree search procedure looks upon a single move as the whole problem, whilst a game-playing algorithm looks upon the problem as that of managing a series of moves, with concomitant searches, and makes rational decisions about how to share any transferable resources (i.e. time[18]) between them. The role of time control may therefore be summarised below:

$$\boxed{\text{Game Playing} = \text{Time Control} + \text{Tree Search}}$$

While tree search algorithms have been the subject of much research, very little has been published in the artificial intelligence literature on time control. This paucity of literature demands an explanation, particularly when one considers how poorly the standard 'flat rate' approach models that of human experts. The method of controlling time which still provides the basis for many otherwise refined game-playing programs is to apportion the search time between the moves on a simple pro-rata basis, so that the same amount of search is spent on every move. This is a very crude method of resource allocation, most obviously in cases where the player has only one legal move

[18] Also *information*, but analysis of this issue is beyond the scope of this text.

at his disposal[19]. Any human gameplayer will confirm that some moves are more important than others, in the sense that a player with a fixed time limit for the whole game will naturally wish to consider for longer before making them.

It is almost tautological to state that a player spends the longest on the positions he finds the most difficult, which are the ones in which it is hardest for him to determine which move it is best for him to play. Nevertheless, this statement contains the key to the issue of time control, and provides at least a partial explanation as to why so little work has been done on formal methods of time control:- *to be able to apportion time in an efficient fashion, an 'understanding' of the consequences is essential.*

By their very nature, brute force methods — which were until very recently ubiquitous amongst the top game-playing programs — do not have any such understanding. Subsection 1.1.4 introduced the method of singularity extensions as a refinement to alpha-beta search. It is also possible to view it as a time control algorithm, since it increases the amount of time spent searching variations which involve singular moves. This hints at another important observation about game-playing algorithms:- *the distinction between a search algorithm and a time control algorithm is to some extent an artificial one.* Any human game player knows this, since no one first decides in advance of thinking about a move how much time to spend doing so. If the

[19]Of course, in practice programs have an *if* statement to detect for this, but the very existence of such a special case illustrates the ad hoc nature of this approach.

human thinking process is to be mimicked, then the time allocation process must be responsive to the findings of the search algorithm, as depicted below.

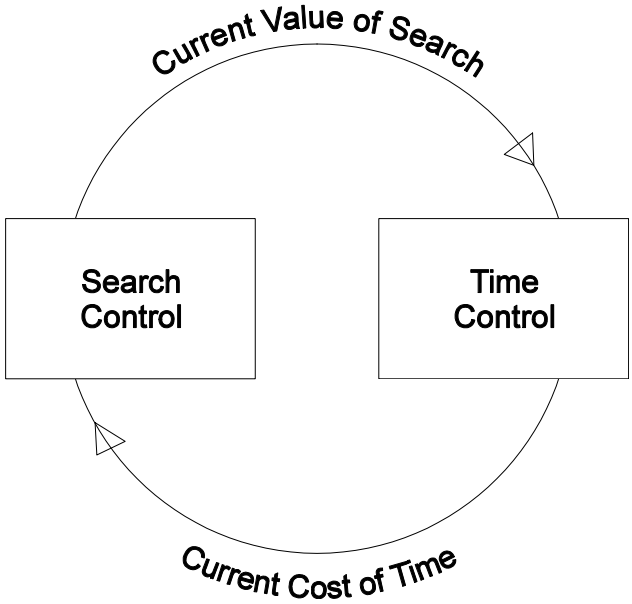


Figure 30: The Relationship between Time Control and Search Control

The reason why most examples in the literature simply have a fixed budget or some other very simple policy is that the 'brute force' nature of their alpha-beta-based search algorithms offers so little understanding of the position. It is worth formalising the justification for this time control policy; uniform time control assumes that the utility to be gained from searching is the same for every move throughout a game. The quality of this approximation depends largely upon the game.

## 5.1 Previous Methods

To pursue a 'flat rate' time allocation policy is to ignore the information about the tree that is gathered by the searching process. Information of this kind is readily available to a selective search since it is required by the selective nature of the algorithm. This suggests the general form of time control appropriate to selective search. Since the selectivity of the search algorithm depends upon some information being accrued about the tree as search progresses, search can be terminated as soon as the information built up meets some criterion. The nature of this criterion depends of course upon the time allowance, the game being played and the specifics of the selective search algorithm. Flaws in the conception of a search algorithm will lead to corresponding failures of the time control algorithm, since the time control algorithm can only access information gathered by the search algorithm.

For example, consider the paradigm behind the B* family of search algorithms described in Subsections 1.2.3–1.2.5. Originally, B* search was suggested as a method of searching adversary trees which had accurate bounds on their scores. Accordingly, no time control was required; the algorithm terminated as soon as it had found *the answer*. As part of a game-playing algorithm, problems arise with the initially appealing goal of finding the best move. In a real game, some moves are more important than others, a fact which is not recognised by B* or any of its probabilistic variants. Indeed, the notion of a unique 'best' move may be flawed. It is a frequent occurrence in the game of Go, for example, that a pair of moves have *exactly* the same

game theoretic value. The general problem with use of B* in a game-playing problem is illustrated overleaf in Figure 31, which shows four types of uncertainty about the best available move, of which the top left is the most worthy of further search. Since the B* paradigm only looks at the probability that one move dominates the others, and ignores the amount, it does not take into account the variance of the move estimates. Since it therefore fails to recognise that the left hand cases are better candidates for further search, it will not be a good basis for a time allocation policy.

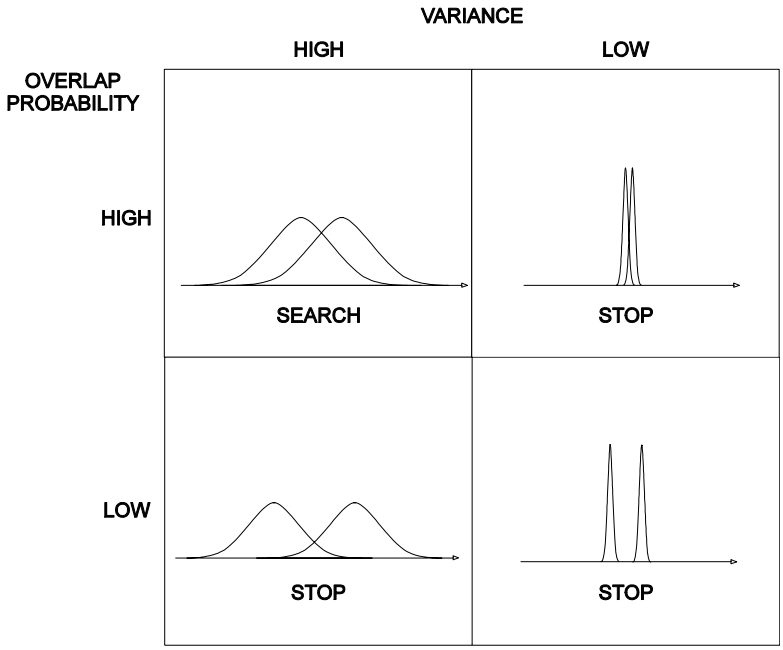


Figure 31: The Problem with the B* Paradigm

We now examine the range of approaches to time control applied to the more important selective search algorithms, in reverse order of sophistication.

### 5.1.1 Best First Minimax

In the theoretical description of the best first minimax algorithm, Korf and Chickering [47] make the bald statement that

> "While in principle we could make a move at any time, we chose to make a move when the length of the principal variation exceeds a given bound. This ensures that the chosen move has been explored to some depth."

Since this is the sum total of their remarks on time control in their substantial paper, one is forced to assume that they have paid little attention to this topic. One possible explanation for this is to be found in their conclusions, in which they state that "in practice, memory is not a problem because in a two-player game, a move must be made every few minutes". The indication here is that practical rather than theoretical considerations may have determined the nature of the time control policy; as well as being in keeping with the spirit of the best first minimax algorithm — neat and simple — the setting of such a depth threshold is expedient in that it avoids practical problems caused by the space required to store the tree in memory. Where such a compromise is being made to expediency it should be admitted, and the time control policy does not seem to have any theoretical justification.

### 5.1.2 Conspiracy Numbers

In his original paper on the subject, McAllester [50] describes the method of conspiracy numbers not as a game-playing algorithm, but as a 'procedure ... for growing min-max game trees' and so does not address time control. The results he obtained were achieved by terminating the search after a fixed number of nodes.

Schaeffer [73] is credited with adding the notion of iterative deepening to conspiracy number search. The usage of the term in this context is an analogy with its use in the context of alpha-beta search; instead of increasing the fixed depth of the alpha-beta search by one each time, the size of the conspiracies searched for is increased by one. The most promising results of his paper concern tactical Chess problems of the 'Black to play and win' type. These however are single searches and so do not require any means of time allocation. Schaeffer acknowledges that time control remains a difficult issue.

A natural, if rather straightforward, approach to time control would be to impose a ceiling on the maximum conspiracy size that is searched for. This would have the desired effect of spending less time on positions where the best move was relatively clear and more where it was unclear. This is, however, not merely game specific, but also implementation specific; the 'granularity'[20] of the evaluation function is very important in determining

[20] the precision with which positions are evaluated

the conspiracy thresholds reached by the search algorithm.

The extent of this problem is revealed by the application of Schaeffer's conspiracy numbers algorithm to a set of Chess positions. The granularity of the position evaluation function was $\frac{1}{10}$ of a pawn. For some positions, the algorithm was still hunting for conspiracies of size 2 after an hour of C.P.U. time! Naive use of the conspiracy threshold is therefore not an appropriate method of time control. A 'flat rate' time allocation policy is particularly wasteful for a selective search algorithm, and so the current lack of any feasible alternative must represent a serious drawback of conspiracy number search.

### 5.1.3 PSVB*

The original paper on the B* search algorithm, like that on conspiracy number search, did not contain any discussion of time control. For his PSVB* search, Palay [62] was therefore forced to deduce an appropriate time control mechanism from first principles. He drew his inspiration for the task from a major strength of B*, namely, the ability to terminate search once it becomes clear that one move is 'better' than the others, even if the score of that move is not certain.

The 'better than' criterion of B* is replaced in PSVB* by a rather dubious[21]

[21] Baum and Smith[6] highlight a weakness of this criterion by giving an example of distributions $A$, $B$ and $C$, such that $A$ dominates $B$,$B$ dominates $C$,and $C$ dominates $A$, all with probability $> .5$.

'dominates with probability $p$' criterion. Another failing is that Palay's method owes too much to the standard 'flat rate' approach to time control, since he assumes that each move is allocated (presumably uniformly) a 'time limit'. His basic philosophy is summarised below:

> "While there is plenty of time remaining, the search should be continued, unless it is certain that one move is better than the remaining moves. As the amount of time used by the search increases the levels of domination used should start to decrease. At first the decrease should be gradual; however, as the amount of time used by the search approaches the amount of time allocated to the search, the levels of dominance should decrease rapidly."

He also recognised that the need for further search is influenced by the relative winning chances of the two players. Again, whilst the idea is not without merit, the choice of implementation is tailored specifically for the game of Chess.

> "The choice of the actual function was somewhat arbitrary. The guiding principle in selecting this function was that as the achieved value of a move increases, the level of domination needed for termination decreases."

Palay's approach to time control was very innovative at the time at which the work was carried out, and he was one of the first authors to pay specific

attention to this topic. However, the fact that he never worked out how to play from a position in which one was almost hopelessly behind merely underlines the piecemeal nature of the approach he took to the issue of time control.

### 5.1.4 MGSS*

The time control algorithm of MGSS* is based on the following algorithm proposed in 1968 by Good [30]:

1. *If no leaves have positive* $U()$*, stop and make the move which appears best on the basis of the current search tree, otherwise proceed to step* 2.
2. *Search the leaf with the greatest* $U()$*, recalculate the* $U()$ *values as required, and go to step* 1.

The rationale for such a schema is clear, since *if the meta-calculation costs are ignored* and *given a suitable definition of* $U()$, this time control algorithm is indeed optimal. However, as we have seen, neither of these is possible, and so this point is of theoretical rather than practical interest. Indeed, the problem of determining a tractable approximation of $U()$ proved such a difficult one that this method was all but ignored by the mainstream of artificial intelligence research for around twenty years. The efficacy of such a policy hinges upon the utility function, a topic which we shall address in

Section 6.5. The MGSS* algorithm uses $U(S) = E[V(S)] - TC(S)$, where $V(S)$, the value of search, is defined in Subsection 1.2.6.

Russell and Wefald [72] are uncharacteristically vague about the choice of $TC()$ used to implement MGSS*. The simplest choice would be to use a constant function. For games in which different numbers of moves are available from different positions, a more accurate $TC()$ would — since the units of computation are the expansion of a single node — be proportional to the number of children of that node.

### 5.1.5 BP

The time control policy of the BP algorithm is analogous to that of MGSS*. Since BP expands leaves one gulp at a time rather than one leaf at a time, their expression for the net utility of further search is as follows:

$$\frac{U_{\text{gulp}}}{t_{\text{gulp}}} - TC(t, m)$$

In this expression, $t_{\text{gulp}}$ is the amount of time needed to carry out the next gulp of search, while $U_{\text{gulp}}$ is the expected increase in utility from doing so, as explained in Subsection 1.2.7. The chief advance over the time control of MGSS* is the use of a more advanced expression for the cost of time, $TC()$. This takes parameters $t$, the amount of time left, and $m$, the number of moves until the next time control. This makes the assumption that the number of moves until the next time control is known, which is not valid if there is an overall limit for the game unless the game is a fixed number of

moves. The following two expressions are suggested for $TC()$:

$$TC() = c_6 \frac{m}{t} U\left(log_B\left[\frac{t}{m}\right]\right) = \begin{cases} c_3 \frac{m/t}{\log_B (t/m)^2} & (Levy) \\ c_4 \left(\frac{m}{t}\right)^1 + c_5 & (Szabo) \end{cases}$$

The $B$ in the Levy formula is the expected branching factor of the search. The two formulae are derived from suggested expressions for the value of time by Levy and by Szabo [84]. Baum and Smith [6] explain their efforts to fit these two formulae — by judicious choice of $c$. The data from which they derive their observed utility function, $U()$, is a table of Newborn's [58] of how likely a Chess program is to change its opinion of the best move if deeper search is carried out.

## 5.2 Marginal Value of Information

The appeal to some kind of 'law of diminishing returns' on time spent searching is a very convincing one, following almost as a corollary of the exponential explosion of nodes in the game tree. The exact nature of this principle remains obstinately difficult to codify in general terms.

The 'boxes' problem of Section 3.1 is one in which the principle can be seen to yield the optimal policy in a very straightforward way. Box $i$ has reward rate $\emptyset_i = p_i/t_i$, so the probability of finding an object is $\emptyset_i t_i$, yielding an expected probability of termination proportional to $\emptyset_i$ for every time unit spent searching a box of type $i$. The optimal policy of searching boxes in

decreasing order of type $\emptyset_i$ equates to a law of diminishing returns over each single box searched.

The addition of linear precedence constraints brings a further level of complication. The $\emptyset_i$ of searches by the optimal policy are no longer monotonically decreasing, since searches are made not solely for their probability of discovering an object, but also to reveal other nodes for search. If we consider the units of the search to be maximal indivisible blocks rather than individual nodes, the law of diminishing returns still applies.

The stochastic OR-tree model of Section 3.2 is similar in principle. The chunking process, however, is affected by the random nature of the model. A law of diminishing expected returns applies, where the unit of search is a single exhaustive search. The *corrected* reward rates, $\emptyset_i^*$, take account of this.

Unfortunately, the AND-OR model that is the subject of Chapter 4 is harder to model in this fashion, due to the way the '$\cap$' branches affect the return from searching. An expression such as '$A \cap B$' presents a problem since the return from searching '$A$' depends upon the results of searching '$B$' and vice versa. As explained in Section 4.4, considering the composite term '$A \cap B$' as a single unit of search presents problems, not least because search may expand '$A$' and '$B$' into compound terms.

## 5.3 Marginal Value of Search

We now revise the two models presented in Chapter 2, so as to give the player a choice as to how much time he uses to search. Instead of having a fixed time limit, an unlimited supply of time units are available to the player, at a cost of $\lambda > 0$ each. This formulation has a natural interpretation in a practical context, and is mathematically easy since the time spent so far is a sunk cost, that is, one which does not affect future play. Accordingly, only very minor adaptations are necessary to the mechanics of the two models. The only extra notational device which we shall use in this section is the symbol $\Lambda(\tau)$, to stand for the marginal value of time, that is, the cost of time at which, if the player must purchase all the time units before the game[22], he is indifferent about purchase of a further unit of time once $\tau$ units have already been bought.

In the one-player tree search game of Section 2.2, Lemma 2.4 implies that for $\lambda \geq \kappa$, it is optimal not to buy any time, and for $\lambda = \kappa$, it is optimal to buy up to $H$ time units. For $\lambda \leq \kappa$, it may be optimal to buy more, though this depends upon the values of $Y$ observed in relation to the specific distribution of $Y$. If $\lambda = \kappa$ the only circumstances in which it is optimal to buy more search is if the last two $Y$ values observed were both equal to 0, in which case it is optimal to 'backtrack' up the searched tree, as illustrated

[22]This restriction is introduced as a slight simplification of the presentation that follows, which is nevertheless sufficient to illustrate the principles involved. Without it, $\Lambda(\tau)$ is a random variable, since it depends upon the findings of earlier search.

on page 51 in Figure 8.

One extreme occurs if $Y$ is distributed as follows:

$$Y \sim \begin{pmatrix} -\kappa & \kappa \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

In this case, only $h$ searches are ever required to find the optimal path down the game tree, so $\Lambda(\tau) = 0$ for $\tau > h$. At the other extreme, if $Y$ has support along the whole real line, $\Lambda(\tau)$ decreases in $\tau$, only reaching 0 for $\tau = 2^h - 1$, at which point the tree is completely searched.

It is understood, although not formally proved, that $\Lambda(\tau)$ is a decreasing function in $\tau$ for this game. This seems clear from the way in which search time is exchanged for information. As the time already spent increases, the most profitable search opportunities have already been exhausted, so resulting in some 'duplication of effort'. If the variance of $|Y|$ is low, $M(\tau)$ decreases quickly from an initially high value, since the searches yield a lot of information. By contrast, if the variance of $|Y|$ is high, then $M(\tau)$ decreases more slowly because of the greater chance of revising previous decisions about which move was best from a particular node.

In Section 2.3, we saw how to calculate $V_n()$, the payoff of the fuel control problem with $n$ units of fuel available. We can therefore calculate the marginal value of fuel exactly since $\Lambda_\tau() = V_{\tau+1}() - V_\tau()$. In contrast to the one-player tree search model, $\Lambda_\tau()$ need not be monotonic, a consequence of its more general structure. Consider the construction shown in Figure 32.

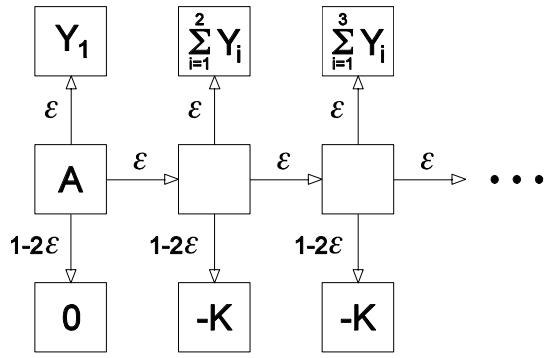


Figure 32: Construction to Show the Generality of $\Lambda_\tau(A)$

By choosing $\epsilon > 0$ to be small, and $K$ to be sufficiently large, this shows that $\Lambda_\tau(A)$ can be any non-negative sequence, $Y_1, Y_2, Y_3 \ldots Y_N$.

## 5.4 A Two-Player Search Game

We consider the following two-player zero-sum game, similar to the one defined in Section 2.2. The game tree is an infinite binary tree. Players make alternate moves. Each move is associated with score, $Y$, an independent random variable of known distribution. The player whose move it is has the option of carrying out one or more search actions before moving. Each search makes known to that player the values of the two moves possible from the node searched. The set of nodes searched must at all times form a tree. This is almost the infinite height case of the familiar search structure from Chapter 2. This model differs from the model described in Section 2.2, since the $Y$ values of sisters are independent of each other, rather than being constrained so that $Y_{i1} + Y_{i2} = 0$.

The running reward, $R_\tau$, paid from player 2 to player 1 in time unit $\tau$ is

given below:

$$R_\tau = \begin{cases} -\lambda & | \text{ Player 1 carries out a search.} \\ \lambda & | \text{ Player 2 carries out a search.} \\ y & | \text{ Player 1 makes a move with value } y. \\ -y & | \text{ Player 2 makes a move with value } y. \end{cases}$$

It is useful to discount the running rewards by $\alpha^N$, where $N$ is the number of moves made so far and $\alpha \in (0, 1]$. As in Section 2.2 we specify that $E[Y] = 0$ and $E[|Y|] = \kappa$.

We now consider how to capture the value of search already carried out. If the moves in the search information which leads down to leaf, $L$, have scores $y_1, y_2 \ldots y_{h_i}$, then the *net value* of leaf $L$ is defined as follows:

$$nv(L) = \sum_{i=1}^{h_i} (-\alpha)^{i-1} y_i$$

This is the net value to player 1 from playing the game until leaf $L_i$ is reached.

We now define the net values of internal nodes of the search information as follows:

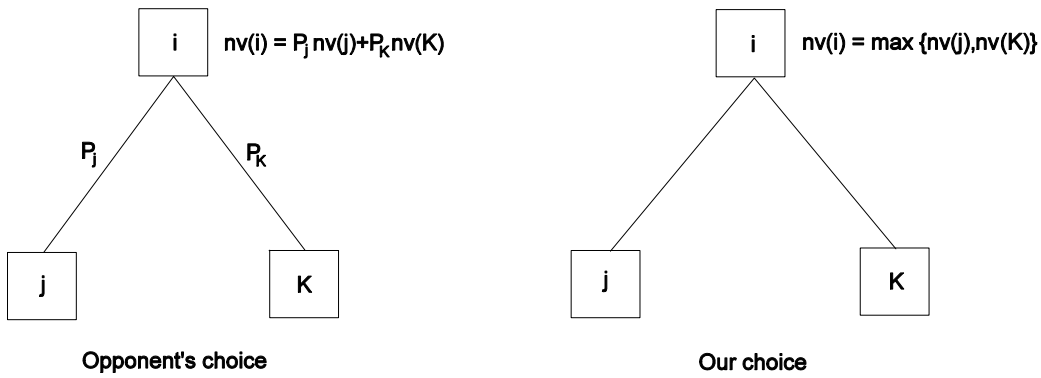


Figure 33: Calculation of Net Value of Internal Nodes

This defines $nv()$ as the superharmonic majorant corresponding to our opponent making random choices while we select the move which leads to the node with the greatest net value. Where $L$ refers to a tree, we shall abbreviate $nv(\text{root}(L))$ as $nv(L)$.

We suppose the left and right subtrees of $L$ to be $L_L$ and $L_R$, and the left and right moves from root to have scores $Y_L$ and $Y_R$, as illustrated overleaf in Figure 34.

**Theorem 5.1** *For $\lambda \geq \kappa$, the value of the game is 0, and Policy $\pi^*$, described below, is the optimal policy for both players.*

| |
|---|
| Policy $\pi^*$ moves to the daughter of root with the higher net value, and moves at random if there is no more search information. |

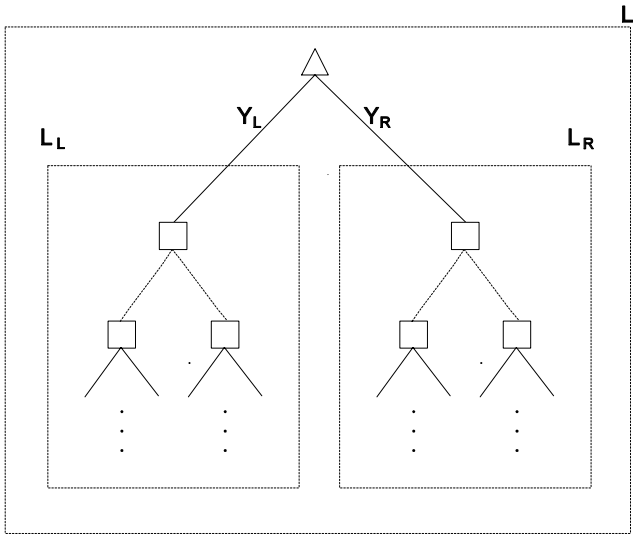


Figure 34: Information Notation

**Proof**: We denote by $V_{\text{Left }\pi}()$ the payoff from moving left and then following policy $\pi$ and $V_{\text{Right }\pi}()$ the payoff from moving right and then following policy $\pi$. We consider the optimality equation for player 1, assuming his opponent pursues a policy of moving at random:

$$V(L) = \max\left\{E[V_{\text{Left}}(L)], E[V_{\text{Right}}(L)], E\left[\max_i\{V(L+i)\}\right]\right\}$$

Substituting in policy $\pi^*$:

$$\begin{aligned} V_{\pi^*}(L) &= \max\left\{E[V_{\text{Left }\pi^*}(L)], E[V_{\text{Right }\pi^*}(L)], E\left[\max_i\{V_{\pi^*}(L+i)\}\right]\right\} \\ &= \max\left\{E[Y_L + \alpha nv(L_L)], E[Y_R + \alpha nv(L_R)], E\left[\max_i\{nv(L+i)\}\right] - \lambda\right\} \\ &= \max\left\{nv(L), E\left[\max_i\{nv(L+i)\}\right] - \lambda\right\} \end{aligned} \tag{15}$$

If there is no search information, $Y_L$ and $Y_R$ are unknown, and $L_L = L_R = \phi$. Hence:

$$nv(L_L) = nv(L_R) = E[Y_L] = E[Y_R] = 0$$

$$E[Y_L + \alpha nv(L_L)] = E[Y_R + \alpha nv(L_R)] = nv(L)$$

Equation (15) therefore follows. If there is information, it follows directly from the definition of net value.

Now, let us consider how the expansion of a leaf $i$ of $L$ affects $nv(L)$. First, we note that if the opponent is to play from $i$ then there is no expected change to $nv(L)$ caused by expanding it. If the searcher can play then expansion of $i$ increases the net value of that leaf by $\alpha^{h_i} E[\max\{Y_{i1}, Y_{i2}\}]$, where $h_i$ is the depth of leaf $i$.

From the definition of net value, the net value of a the root of a tree cannot change by more than the change to one of its leaves. Hence, for any $i$:

$$\begin{aligned} nv(L+i) - \lambda &\leq nv(L) + E[\max\{Y_{i1}, Y_{i2}\}] - \lambda \\ &\leq nv(L) \qquad\qquad \text{for } \lambda \geq \kappa \end{aligned}$$

Substitution of this in (15) shows us that $V_{\pi^*}(L) = nv(L)$ satisfies the optimality equation, so $\pi^*$ is the optimal policy.

We have shown that if player 2 pursues a policy of always playing randomly, then player 1 can do no better than pursue an identical policy. This is a Nash equilibrium, and so both players must therefore be playing optimally, since the game is zero-sum. ■

**Corollary 5.2** *For $\lambda = \kappa$, the value of the game is 0. Policy $\pi^{1*}$, described below, is optimal, as well as policy $\pi^*$.*

| |
|---|
| Policy $\pi^{1*}$ moves to the daughter of root with the highest net value, if search information is available. If no search information is available, it expands root. |

**Proof**: Theorem 5.1 establishes the optimality of policy $\pi^*$ for $\lambda = \kappa$, and so we evaluate the payoff from playing policy $\pi^{1*}$ against policy $\pi^*$. Policy $\pi^*$ has been proved optimal above. The only case in which policy $\pi^{1*}$ differs from policy $\pi^*$ is if there is no search information. In this case, observe from the definition of net value that any action is optimal:

$$\begin{aligned} nv(L + \text{Root}) - \lambda &= nv(L) + E[\max\{Y_L, Y_R\}] - \lambda = nv(L) = 0 \\ &= E[V_{\text{Left } \pi^{1*}}(L)] = E[V_{\text{Right } \pi^{1*}}(L)] \end{aligned}$$

■

## 5.5 Summary

We have extended the models of Chapter 2 to allow the player to choose how much time to spend. The modifications made were minor, but relatively powerful, since they allow several of these simple games to be grouped together as part of a common problem, with the overall reward of a linear combination of the payoffs of each individual problem.

The result proved about the two-player search game presented in Section 5.4 is perhaps a somewhat unsurprising but nevertheless significant one.

It may be the first time that an optimal policy has been proved for a two-player game with such a realistic search structure, and it demonstrates that dynamic stochastic control can be used for this purpose.

The gap between the theoretical results proved in this chapter and the practical methods of time control described in Section 5.1 is large indeed. Of the time control policies described, those of the MGSS* and BP algorithms are the most theoretically interesting. As one-step lookahead policies, they both make the tacit assumption of decreasing marginal value of search: that if the cost of the next step exceeds the expected utility of the information, the cost of the next two steps will exceed the expected utility of the information they yield. As explained, this seems a fairly reasonable approximation for most games[23].

We have seen how the issue of time control is related to search control, and that for either to be satisfactory, a theoretically sound approximation must be found to the notion of 'utility'. The considerable problems surrounding this concept will be discussed in Section 6.5.

[23]Doubtless, pathological games could be deduced to show the inefficiency of this, just as Nau [55, 54, 56, 57] deduced pathological games to do the same for the minimax backing-up principle.