IA4-5. Elemente de teoria jocurilor
Suport de curs

Selecţie materiale: prof. dr. ing. Florin Leon

Game Playing (Adversarial Search)

In which we examine the problems that arise when we try to plan ahead in a world where other agents are planning against us.

6.1 GAMES

Chapter 2 introduced **multiagent environments**, in which any given agent will need to consider the actions of other agents and how they affect its own welfare. The unpredictability of these other agents can introduce many possible **contingencies** into the agent's problem-solving process, as discussed in Chapter 3. The distinction between **cooperative** and **competitive** multiagent environments was also introduced in Chapter 2. Competitive environments, in which the agents' goals are in conflict, give rise to **adversarial search** problems—often known as **games**.

GAMES

Mathematical **game theory**, a branch of economics, views any multiagent environment as a game provided that the impact of each agent on the others is "significant," regardless of whether the agents are cooperative or competitive.[1] In AI, "games" are usually of a rather specialized kind—what game theorists call deterministic, turn-taking, two-player, **zero-sum games** of **perfect information**. In our terminology, this means deterministic, fully observable environments in which there are two agents whose actions must alternate and in which the utility values at the end of the game are always equal and opposite. For example, if one player wins a game of chess (+1), the other player necessarily loses (−1). It is this opposition between the agents' utility functions that makes the situation adversarial. We will consider multiplayer games, non-zero-sum games, and stochastic games briefly in this chapter, but will delay discussion of game theory proper until Chapter 17.

ZERO-SUM GAMES

PERFECT INFORMATION

Games have engaged the intellectual faculties of humans—sometimes to an alarming degree—for as long as civilization has existed. For AI researchers, the abstract nature of games makes them an appealing subject for study. The state of a game is easy to represent, and agents are usually restricted to a small number of actions whose outcomes are defined by

[1] Environments with very many agents are best viewed as **economies** rather than games.

precise rules. Physical games, such as croquet and ice hockey, have much more complicated descriptions, a much larger range of possible actions, and rather imprecise rules defining the legality of actions. With the exception of robot soccer, these physical games have not attracted much interest in the AI community.

Game playing was one of the first tasks undertaken in AI. By 1950, almost as soon as computers became programmable, chess had been tackled by Konrad Zuse (the inventor of the first programmable computer and the first programming language), by Claude Shannon (the inventor of information theory), by Norbert Wiener (the creator of modern control theory), and by Alan Turing. Since then, there has been steady progress in the standard of play, to the point that machines have surpassed humans in checkers and Othello, have defeated human champions (although not every time) in chess and backgammon, and are competitive in many other games. The main exception is Go, in which computers perform at the amateur level.

Games, unlike most of the toy problems studied in Chapter 3, are interesting *because* they are too hard to solve. For example, chess has an average branching factor of about 35, and games often go to 50 moves by each player, so the search tree has about 35^{100} or 10^{154} nodes (although the search graph has "only" about 10^{40} distinct nodes). Games, like the real world, therefore require the ability to make *some* decision even when calculating the *optimal* decision is infeasible. Games also penalize inefficiency severely. Whereas an implementation of A* search that is half as efficient will simply cost twice as much to run to completion, a chess program that is half as efficient in using its available time probably will be beaten into the ground, other things being equal. Game-playing research has therefore spawned a number of interesting ideas on how to make the best possible use of time.

We begin with a definition of the optimal move and an algorithm for finding it. We then look at techniques for choosing a good move when time is limited. **Pruning** allows us to ignore portions of the search tree that make no difference to the final choice, and heuristic **evaluation functions** allow us to approximate the true utility of a state without doing a complete search. Section 6.5 discusses games such as backgammon that include an element of chance; we also discuss bridge, which includes elements of **imperfect information** because not all cards are visible to each player. Finally, we look at how state-of-the-art game-playing programs fare against human opposition and at directions for future developments.

IMPERFECT INFORMATION

6.2 Optimal Decisions in Games

We will consider games with two players, whom we will call MAX and MIN for reasons that will soon become obvious. MAX moves first, and then they take turns moving until the game is over. At the end of the game, points are awarded to the winning player and penalties are given to the loser. A game can be formally defined as a kind of search problem with the following components:

- The **initial state**, which includes the board position and identifies the player to move.
- A **successor function**, which returns a list of (*move, state*) pairs, each indicating a legal move and the resulting state.

TERMINAL TEST

- A **terminal test**, which determines when the game is over. States where the game has ended are called **terminal states**.
- A **utility function** (also called an objective function or payoff function), which gives a numeric value for the terminal states. In chess, the outcome is a win, loss, or draw, with values $+1$, -1, or 0. Some games have a wider variety of possible outcomes; the payoffs in backgammon range from $+192$ to -192. This chapter deals mainly with zero-sum games, although we will briefly mention non-zero-sum games.

GAME TREE

The initial state and the legal moves for each side define the **game tree** for the game. Figure 6.1 shows part of the game tree for tic-tac-toe (noughts and crosses). From the initial state, MAX has nine possible moves. Play alternates between MAX's placing an X and MIN's placing an O until we reach leaf nodes corresponding to terminal states such that one player has three in a row or all the squares are filled. The number on each leaf node indicates the utility value of the terminal state from the point of view of MAX; high values are assumed to be good for MAX and bad for MIN (which is how the players get their names). It is MAX's job to use the search tree (particularly the utility of terminal states) to determine the best move.

Optimal strategies

In a normal search problem, the optimal solution would be a sequence of moves leading to a goal state—a terminal state that is a win. In a game, on the other hand, MIN has something

STRATEGY

to say about it. MAX therefore must find a contingent **strategy**, which specifies MAX's move in the initial state, then MAX's moves in the states resulting from every possible response by MIN, then MAX's moves in the states resulting from every possible response by MIN to *those* moves, and so on. Roughly speaking, an optimal strategy leads to outcomes at least as good as any other strategy when one is playing an infallible opponent. We will begin by showing how to find this optimal strategy, even though it should be infeasible for MAX to compute it for games more complex than tic-tac-toe.

Even a simple game like tic-tac-toe is too complex for us to draw the entire game tree, so we will switch to the trivial game in Figure 6.2. The possible moves for MAX at the root node are labeled a_1, a_2, and a_3. The possible replies to a_1 for MIN are b_1, b_2, b_3, and so on. This particular game ends after one move each by MAX and MIN. (In game parlance, we say

PLY

that this tree is one move deep, consisting of two half-moves, each of which is called a **ply**.) The utilities of the terminal states in this game range from 2 to 14.

Given a game tree, the optimal strategy can be determined by examining the **minimax**

MINIMAX VALUE

value of each node, which we write as MINIMAX-VALUE(n). The minimax value of a node is the utility (for MAX) of being in the corresponding state, *assuming that both players play optimally* from there to the end of the game. Obviously, the minimax value of a terminal state is just its utility. Furthermore, given a choice, MAX will prefer to move to a state of maximum value, whereas MIN prefers a state of minimum value. So we have the following:

$$\text{MINIMAX-VALUE}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MIN node.} \end{cases}$$

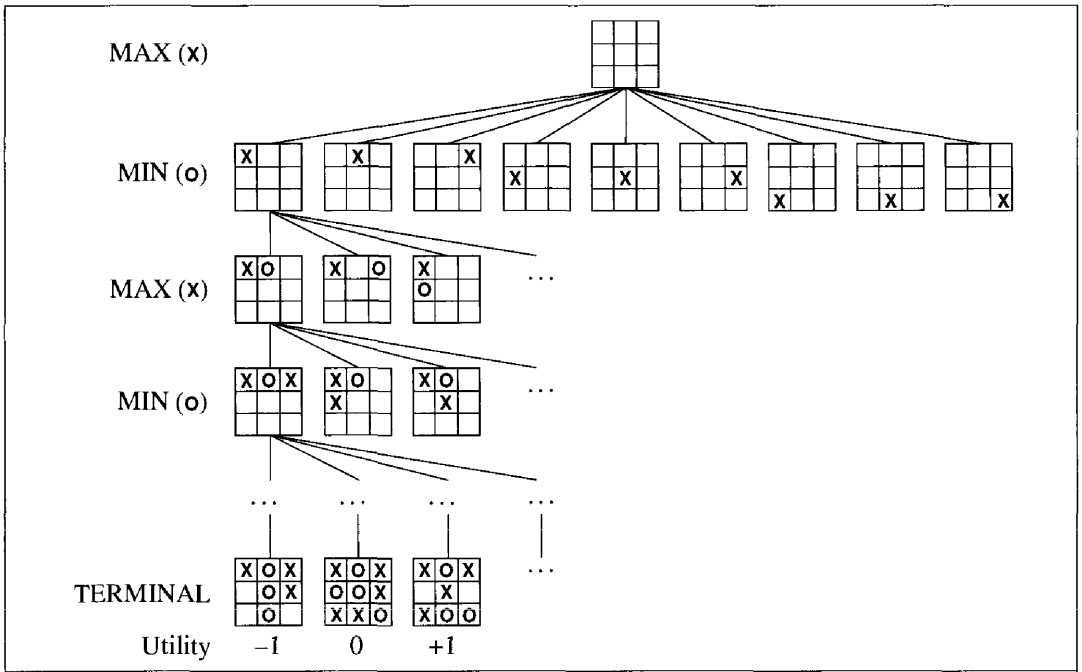


Figure 6.1 A (partial) search tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square. We show part of the search tree, giving alternating moves by MIN (O) and MAX, until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.

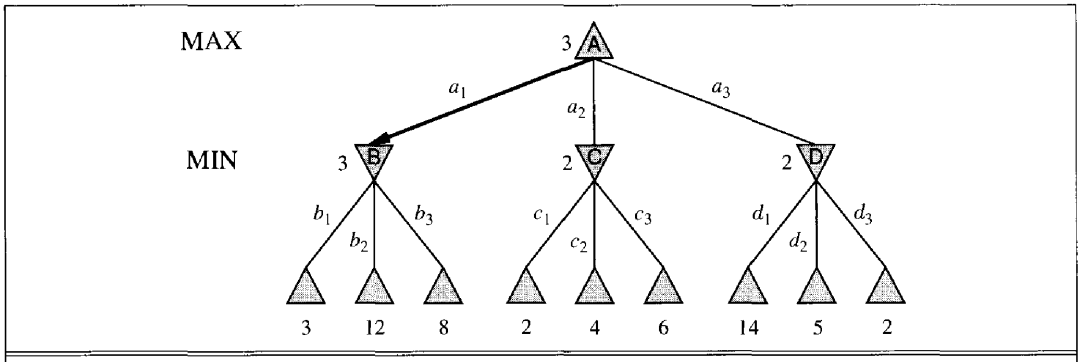


Figure 6.2 A two-ply game tree. The $\triangle$ nodes are "MAX nodes," in which it is MAX's turn to move, and the $\triangledown$ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is a_1, because it leads to the successor with the highest minimax value, and MIN's best reply is b_1, because it leads to the successor with the lowest minimax value.

Let us apply these definitions to the game tree in Figure 6.2. The terminal nodes on the bottom level are already labeled with their utility values. The first MIN node, labeled B, has three successors with values 3, 12, and 8, so its minimax value is 3. Similarly, the other two MIN nodes have minimax value 2. The root node is a MAX node; its successors have minimax

MINIMAX DECISION

values 3, 2, and 2; so it has a minimax value of 3. We can also identify the **minimax decision** at the root: action a_1 is the optimal choice for MAX because it leads to the successor with the highest minimax value.

This definition of optimal play for MAX assumes that MIN also plays optimally—it maximizes the *worst-case* outcome for MAX. What if MIN does not play optimally? Then it is easy to show (Exercise 6.2) that MAX will do even better. There may be other strategies against suboptimal opponents that do better than the minimax strategy; but these strategies necessarily do worse against optimal opponents.

The minimax algorithm

MINIMAX ALGORITHM

The **minimax algorithm** (Figure 6.3) computes the minimax decision from the current state. It uses a simple recursive computation of the minimax values of each successor state, directly implementing the defining equations. The recursion proceeds all the way down to the leaves

BACKED UP

of the tree, and then the minimax values are **backed up** through the tree as the recursion unwinds. For example, in Figure 6.2, the algorithm first recurses down to the three bottom-left nodes, and uses the UTILITY function on them to discover that their values are 3, 12, and 8 respectively. Then it takes the minimum of these values, 3, and returns it as the backed-up value of node B. A similar process gives the backed up values of 2 for C and 2 for D. Finally, we take the maximum of 3, 2, and 2 to get the backed-up value of 3 for the root node.

The minimax algorithm performs a complete depth-first exploration of the game tree. If the maximum depth of the tree is m, and there are b legal moves at each point, then the time complexity of the minimax algorithm is $O(b^m)$. The space complexity is $O(bm)$ for an algorithm that generates all successors at once, or $O(m)$ for an algorithm that generates successors one at a time (see page 76). For real games, of course, the time cost is totally impractical, but this algorithm serves as the basis for the mathematical analysis of games and for more practical algorithms.

Optimal decisions in multiplayer games

Many popular games allow more than two players. Let us examine how to extend the minimax idea to multiplayer games. This is straightforward from the technical viewpoint, but raises some interesting new conceptual issues.

First, we need to replace the single value for each node with a *vector* of values. For example, in a three-player game with players A, B, and C, a vector $\langle v_A, v_B, v_C \rangle$ is associated with each node. For terminal states, this vector gives the utility of the state from each player's viewpoint. (In two-player, zero-sum games, the two-element vector can be reduced to a single value because the values are always opposite.) The simplest way to implement this is to have the UTILITY function return a vector of utilities.

Now we have to consider nonterminal states. Consider the node marked X in the game tree shown in Figure 6.4. In that state, player C chooses what to do. The two choices lead to terminal states with utility vectors $\langle v_A = 1, v_B = 2, v_C = 6 \rangle$ and $\langle v_A = 4, v_B = 2, v_C = 3 \rangle$. Since 6 is bigger than 3, C should choose the first move. This means that if state X is reached, subsequent play will lead to a terminal state with utilities $\langle v_A = 1, v_B = 2, v_C = 6 \rangle$. Hence,

```
function MINIMAX-DECISION(state) returns an action
  inputs: state, current state in game

  v ← MAX-VALUE(state)
  return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s))
  return v
```

Figure 6.3 An algorithm for calculating minimax decisions. It returns the action corresponding to the best possible move, that is, the move that leads to the outcome with the best utility, under the assumption that the opponent plays to minimize utility. The functions MAX-VALUE and MIN-VALUE go through the whole game tree, all the way to the leaves, to determine the backed-up value of a state.

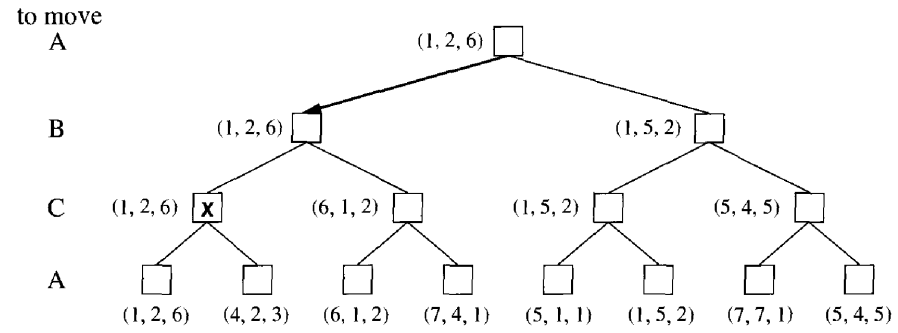


Figure 6.4 The first three ply of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

the backed-up value of X is this vector. In general, the backed-up value of a node n is the utility vector of whichever successor has the highest value for the player choosing at n.

Anyone who plays multiplayer games, such as Diplomacy^{TM}, quickly becomes aware that there is a lot more going on than in two-player games. Multiplayer games usually involve

ALLIANCES

alliances, whether formal or informal, among the players. Alliances are made and broken

as the game proceeds. How are we to understand such behavior? Are alliances a natural consequence of optimal strategies for each player in a multiplayer game? It turns out that they can be. For example suppose A and B are in weak positions and C is in a stronger position. Then it is often optimal for both A and B to attack C rather than each other, lest C destroy each of them individually. In this way, collaboration emerges from purely selfish behavior. Of course, as soon as C weakens under the joint onslaught, the alliance loses its value, and either A or B could violate the agreement. In some cases, explicit alliances merely make concrete what would have happened anyway. In other cases there is a social stigma to breaking an alliance, so players must balance the immediate advantage of breaking an alliance against the long-term disadvantage of being perceived as untrustworthy. See Section 17.6 for more on these complications.

If the game is not zero-sum, then collaboration can also occur with just two players. Suppose, for example, that there is a terminal state with utilities $\langle v_A = 1000, v_B = 1000 \rangle$, and that 1000 is the highest possible utility for each player. Then the optimal strategy is for both players to do everything possible to reach this state—that is, the players will automatically cooperate to achieve a mutually desirable goal.

6.3 ALPHA–BETA PRUNING

The problem with minimax search is that the number of game states it has to examine is exponential in the number of moves. Unfortunately we can't eliminate the exponent, but we can effectively cut it in half. The trick is that it is possible to compute the correct minimax decision without looking at every node in the game tree. That is, we can borrow the idea of **pruning** from Chapter 4 in order to eliminate large parts of the tree from consideration.

ALPHA–BETA PRUNING

The particular technique we will examine is called **alpha–beta pruning**. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

Consider again the two-ply game tree from Figure 6.2. Let's go through the calculation of the optimal decision once more, this time paying careful attention to what we know at each point in the process. The steps are explained in Figure 6.5. The outcome is that we can identify the minimax decision without ever evaluating two of the leaf nodes.

Another way to look at this is as a simplification of the formula for MINIMAX-VALUE. Let the two unevaluated successors of node C in Figure 6.5 have values x and y and let z be the minimum of x and y. The value of the root node is given by

$$\begin{aligned}
\text{MINIMAX-VALUE}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3,z,2) \qquad \text{where } z \leq 2 \\
&= 3.
\end{aligned}$$

In other words, the value of the root and hence the minimax decision are *independent* of the values of the pruned leaves x and y.

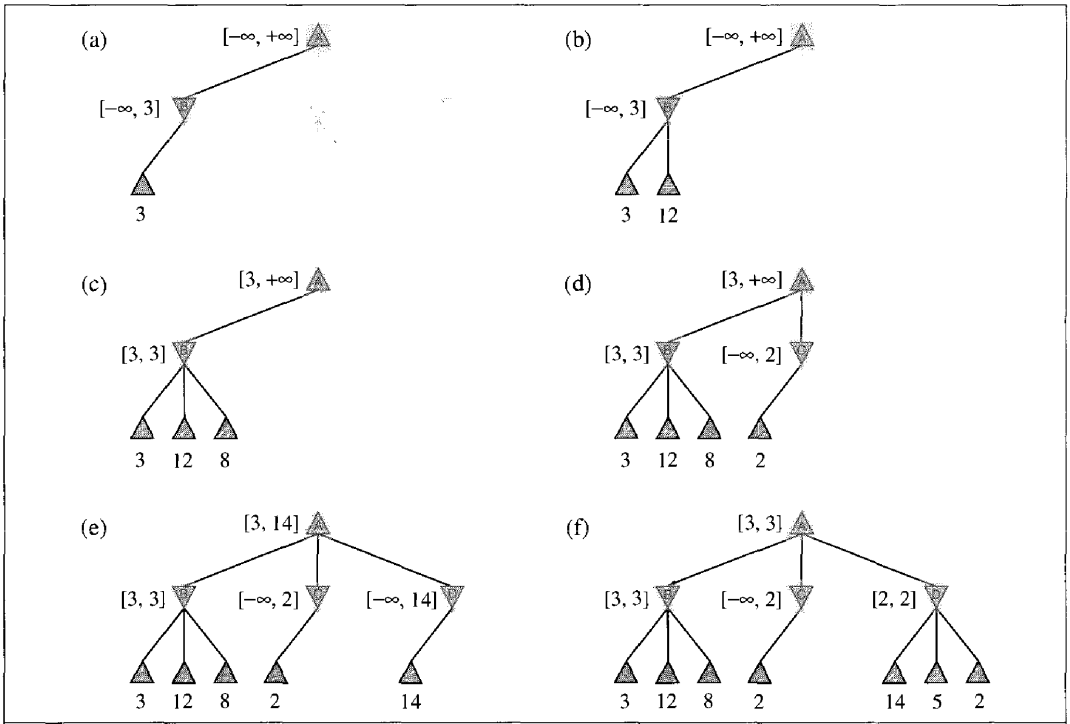


Figure 6.5 Stages in the calculation of the optimal decision for the game tree in Figure 6.2. At each point, we show the range of possible values for each node. (a) The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of *at most* 3. (b) The second leaf below B has a value of 12; MIN would avoid this move, so the value of B is still at most 3. (c) The third leaf below B has a value of 8; we have seen all B's successors, so the value of B is exactly 3. Now, we can infer that the value of the root is *at least* 3, because MAX has a choice worth 3 at the root. (d) The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of *at most* 2. But we know that B is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successors of C. This is an example of alpha–beta pruning. (e) The first leaf below D has the value 14, so D is worth *at most* 14. This is still higher than MAX's best alternative (i.e., 3), so we need to keep exploring D's successors. Notice also that we now have bounds on all of the successors of the root, so the root's value is also at most 14. (f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX's decision at the root is to move to B, giving a value of 3.

Alpha–beta pruning can be applied to trees of any depth, and it is often possible to prune entire subtrees rather than just leaves. The general principle is this: consider a node n somewhere in the tree (see Figure 6.6), such that Player has a choice of moving to that node. If Player has a better choice m either at the parent node of n or at any choice point further up, then n *will never be reached in actual play.* So once we have found out enough about n (by examining some of its descendants) to reach this conclusion, we can prune it.

Remember that minimax search is depth-first, so at any one time we just have to consider the nodes along a single path in the tree. Alpha–beta pruning gets its name from the

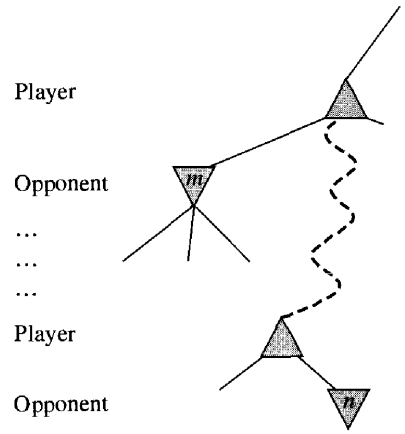


Figure 6.6 Alpha–beta pruning: the general case. If m is better than n for Player, we will never get to n in play.

following two parameters that describe bounds on the backed-up values that appear anywhere along the path:

$\alpha =$ the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.

$\beta =$ the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

Alpha–beta search updates the values of α and β as it goes along and prunes the remaining branches at a node (i.e., terminates the recursive call) as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN, respectively. The complete algorithm is given in Figure 6.7. We encourage the reader to trace its behavior when applied to the tree in Figure 6.5.

The effectiveness of alpha–beta pruning is highly dependent on the order in which the successors are examined. For example, in Figure 6.5(e) and (f), we could not prune any successors of D at all because the worst successors (from the point of view of MIN) were generated first. If the third successor had been generated first, we would have been able to prune the other two. This suggests that it might be worthwhile to try to examine first the successors that are likely to be best.

If we assume that this can be done,[2] then it turns out that alpha–beta needs to examine only $O(b^{m/2})$ nodes to pick the best move, instead of $O(b^m)$ for minimax. This means that the effective branching factor becomes $\sqrt{b}$ instead of b—for chess, 6 instead of 35. Put another way, alpha–beta can look ahead roughly twice as far as minimax in the same amount of time. If successors are examined in random order rather than best-first, the total number of nodes examined will be roughly $O(b^{3m/4})$ for moderate b. For chess, a fairly simple ordering function (such as trying captures first, then threats, then forward moves, and then backward moves) gets you to within about a factor of 2 of the best-case $O(b^{m/2})$ result. Adding dynamic

[2] Obviously, it cannot be done perfectly; otherwise the ordering function could be used to play a perfect game!

```
function ALPHA-BETA-SEARCH(state) returns an action
  inputs: state, current state in game

  v ← MAX-VALUE(state, −∞, +∞)
  return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figure 6.7 The alpha–beta search algorithm. Notice that these routines are the same as the MINIMAX routines in Figure 6.3, except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β (and the bookkeeping to pass these parameters along).

move-ordering schemes, such as trying first the moves that were found to be best last time, brings us quite close to the theoretical limit.

In Chapter 3, we noted that repeated states in the search tree can cause an exponential increase in search cost. In games, repeated states occur frequently because of **transpositions**—different permutations of the move sequence that end up in the same position. For example, if White has one move a_1 that can be answered by Black with b_1 and an unrelated move a_2 on the other side of the board that can be answered by b_2, then the sequences $[a_1, b_1, a_2, b_2]$ and $[a_1, b_2, a_2, b_1]$ both end up in the same position (as do the permutations beginning with a_2). It is worthwhile to store the evaluation of this position in a hash table the first time it is encountered, so that we don't have to recompute it on subsequent occurrences.

TRANSPOSITIONS

TRANSPOSITION TABLE

The hash table of previously seen positions is traditionally called a **transposition table**; it is essentially identical to the *closed* list in GRAPH-SEARCH (page 83). Using a transposition table can have a dramatic effect, sometimes as much as doubling the reachable search depth in chess. On the other hand, if we are evaluating a million nodes per second, it is not practical to keep *all* of them in the transposition table. Various strategies have been used to choose the most valuable ones.

6.4 IMPERFECT, REAL-TIME DECISIONS

The minimax algorithm generates the entire game search space, whereas the alpha–beta algorithm allows us to prune large parts of it. However, alpha–beta still has to search all the way to terminal states for at least a portion of the search space. This depth is usually not practical, because moves must be made in a reasonable amount of time—typically a few minutes at most. Shannon's 1950 paper, *Programming a computer for playing chess*, proposed instead that programs should cut off the search earlier and apply a heuristic **evaluation function** to states in the search, effectively turning nonterminal nodes into terminal leaves. In other words, the suggestion is to alter minimax or alpha–beta in two ways: the utility function is replaced by a heuristic evaluation function EVAL, which gives an estimate of the position's utility, and the terminal test is replaced by a **cutoff test** that decides when to apply EVAL.

CUTOFF TEST

Evaluation functions

An evaluation function returns an *estimate* of the expected utility of the game from a given position, just as the heuristic functions of Chapter 4 return an estimate of the distance to the goal. The idea of an estimator was not new when Shannon proposed it. For centuries, chess players (and aficionados of other games) have developed ways of judging the value of a position, because humans are even more limited in the amount of search they can do than are computer programs. It should be clear that the performance of a game-playing program is dependent on the quality of its evaluation function. An inaccurate evaluation function will guide an agent toward positions that turn out to be lost. How exactly do we design good evaluation functions?

First, the evaluation function should order the *terminal* states in the same way as the true utility function; otherwise, an agent using it might select suboptimal moves even if it can see ahead all the way to the end of the game. Second, the computation must not take too long! (The evaluation function could call MINIMAX-DECISION as a subroutine and calculate the exact value of the position, but that would defeat the whole purpose: to save time.) Third, for nonterminal states, the evaluation function should be strongly correlated with the actual chances of winning.

One might well wonder about the phrase "chances of winning." After all, chess is not a game of chance: we know the current state with certainty, and there are no dice involved. But if the search must be cut off at nonterminal states, then the algorithm will necessarily be *uncertain* about the final outcomes of those states. This type of uncertainty is induced by

computational, rather than informational, limitations. Given the limited amount of computation that the evaluation function is allowed to do for a given state, the best it can do is make a guess about the final outcome.

FEATURES

Let us make this idea more concrete. Most evaluation functions work by calculating various **features** of the state—for example, the number of pawns possessed by each side in a game of chess. The features, taken together, define various *categories* or *equivalence classes* of states: the states in each category have the same values for all the features. Any given category, generally speaking, will contain some states that lead to wins, some that lead to draws, and some that lead to losses. The evaluation function cannot know which states are which, but it can return a single value that reflects the *proportion* of states with each outcome. For example, suppose our experience suggests that 72% of the states encountered in the category lead to a win (utility +1); 20% to a loss (−1), and 8% to a draw (0). Then a

EXPECTED VALUE

reasonable evaluation for states in the category is the weighted average or **expected value**: $(0.72 \times +1) + (0.20 \times -1) + (0.08 \times 0) = 0.52$. In principle, the expected value can be determined for each category, resulting in an evaluation function that works for any state. As with terminal states, the evaluation function need not return actual expected values, as long as the *ordering* of the states is the same.

In practice, this kind of analysis requires too many categories and hence too much experience to estimate all the probabilities of winning. Instead, most evaluation functions compute separate numerical contributions from each feature and then *combine* them to find

MATERIAL VALUE

the total value. For example, introductory chess books give an approximate **material value** for each piece: each pawn is worth 1, a knight or bishop is worth 3, a rook 5, and the queen 9. Other features such as "good pawn structure" and "king safety" might be worth half a pawn, say. These feature values are then simply added up to obtain the evaluation of the position. A secure advantage equivalent to a pawn gives a substantial likelihood of winning, and a secure advantage equivalent to three pawns should give almost certain victory, as illustrated in Figure 6.8(a). Mathematically, this kind of evaluation function is called a **weighted linear**

WEIGHTED LINEAR FUNCTION

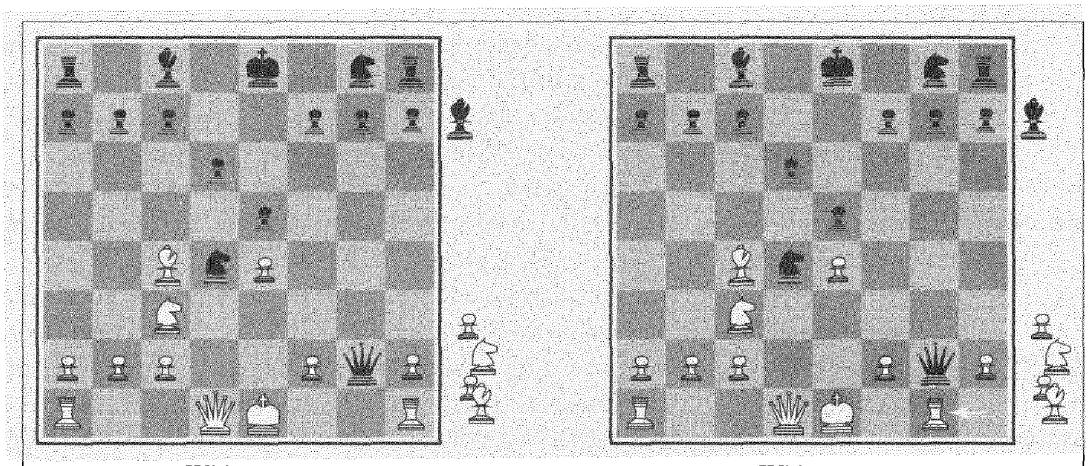
function, because it can be expressed as

$$\textsc{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s) \,,$$

where each w_i is a weight and each f_i is a feature of the position. For chess, the f_i could be the numbers of each kind of piece on the board, and the w_i could be the values of the pieces (1 for pawn, 3 for bishop, etc.).

Adding up the values of features seems like a reasonable thing to do, but in fact it involves a very strong assumption: that the contribution of each feature is *independent* of the values of the other features. For example, assigning the value 3 to a bishop ignores the fact that bishops are more powerful in the endgame, when they have a lot of space to maneuver. For this reason, current programs for chess and other games also use *nonlinear* combinations of features. For example, a pair of bishops might be worth slightly more than twice the value of a single bishop, and a bishop is worth more in the endgame than at the beginning.

The astute reader will have noticed that the features and weights are *not* part of the rules of chess! They come from centuries of human chess-playing experience. Given the



(a) White to move (b) White to move

Figure 6.8 Two slightly different chess positions. In (a), black has an advantage of a knight and two pawns and will win the game. In (b), black will lose after white captures the queen.

linear form of the evaluation, the features and weights result in the best approximation to the true ordering of states by value. In particular, experience suggests that a secure material advantage of more than one point will probably win the game, all other things being equal; a three-point advantage is sufficient for near-certain victory. In games where this kind of experience is not available, the weights of the evaluation function can be estimated by the machine learning techniques of Chapter 18. Reassuringly, applying these techniques to chess has confirmed that a bishop is indeed worth about three pawns.

Cutting off search

The next step is to modify ALPHA-BETA-SEARCH so that it will call the heuristic EVAL function when it is appropriate to cut off the search. In terms of implementation, we replace the two lines in Figure 6.7 that mention TERMINAL-TEST with the following line:

if CUTOFF-TEST(*state*, *depth*) **then return** EVAL(*state*)

We also must arrange for some bookkeeping so that the current *depth* is incremented on each recursive call. The most straightforward approach to controlling the amount of search is to set a fixed depth limit, so that CUTOFF-TEST(*state*, *depth*) returns *true* for all *depth* greater than some fixed depth d. (It must also return *true* for all terminal states, just as TERMINAL-TEST did.) The depth d is chosen so that the amount of time used will not exceed what the rules of the game allow.

A more robust approach is to apply iterative deepening, as defined in Chapter 3. When time runs out, the program returns the move selected by the deepest completed search. However, these approaches can lead to errors due to the approximate nature of the evaluation function. Consider again the simple evaluation function for chess based on material advantage. Suppose the program searches to the depth limit, reaching the position in Figure 6.8(b),

where Black is ahead by a knight and two pawns. It would report this as the heuristic value of the state, thereby declaring that the state will likely lead to a win by Black. But White's next move captures Black's queen with no compensation. Hence, the position is really won for White, but this can be seen only by looking ahead one more ply.

QUIESCENCE

Obviously, a more sophisticated cutoff test is needed. The evaluation function should be applied only to positions that are **quiescent**—that is, unlikely to exhibit wild swings in value in the near future. In chess, for example, positions in which favorable captures can be made are not quiescent for an evaluation function that just counts material. Nonquiescent positions can be expanded further until quiescent positions are reached. This extra search is called a **quiescence search**; sometimes it is restricted to consider only certain types of moves, such as capture moves, that will quickly resolve the uncertainties in the position.

QUIESCENCE SEARCH

HORIZON EFFECT

The **horizon effect** is more difficult to eliminate. It arises when the program is facing a move by the opponent that causes serious damage and is ultimately unavoidable. Consider the chess game in Figure 6.9. Black is ahead in material, but if White can advance its pawn from the seventh row to the eighth, the pawn will become a queen and create an easy win for White. Black can forestall this outcome for 14 ply by checking White with the rook, but inevitably the pawn will become a queen. The problem with fixed-depth search is that it believes that these stalling moves have avoided the queening move—we say that the stalling moves push the inevitable queening move "over the search horizon" to a place where it cannot be detected.

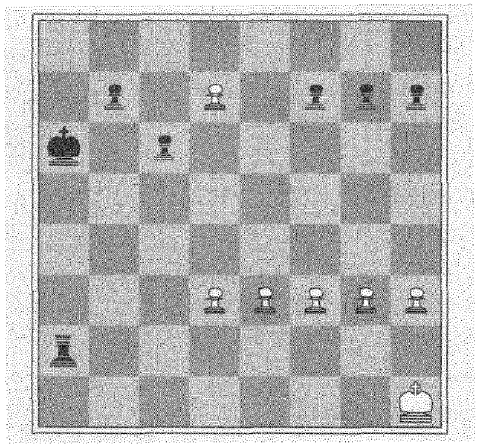
As hardware improvements lead to deeper searches, one expects that the horizon effect will occur less frequently—very long delaying sequences are quite rare. The use of **singular extensions** has also been quite effective in avoiding the horizon effect without adding too much search cost. A singular extension is a move that is "clearly better" than all other moves in a given position. A singular-extension search can go beyond the normal depth limit without incurring much cost because its branching factor is 1. (Quiescence search can be thought of as a variant of singular extensions.) In Figure 6.9, a singular extension search will find the eventual queening move, provided that black's checking moves and white's king moves can be identified as "clearly better" than the alternatives.

SINGULAR EXTENSIONS

So far we have talked about cutting off search at a certain level and about doing alpha–beta pruning that provably has no effect on the result. It is also possible to do **forward pruning**, meaning that some moves at a given node are pruned immediately without further consideration. Clearly, most humans playing chess only consider a few moves from each position (at least consciously). Unfortunately, the approach is rather dangerous because there is no guarantee that the best move will not be pruned away. This can be disastrous if applied near the root, because every so often the program will miss some "obvious" moves. Forward pruning can be used safely in special situations—for example, when two moves are symmetric or otherwise equivalent, only one of them need be considered—or for nodes that are deep in the search tree.

FORWARD PRUNING

Combining all the techniques described here results in a program that can play creditable chess (or other games). Let us assume we have implemented an evaluation function for chess, a reasonable cutoff test with a quiescence search, and a large transposition table. Let us also assume that, after months of tedious bit-bashing, we can generate and evaluate



Black to move

Figure 6.9 The horizon effect. A series of checks by the black rook forces the inevitable queening move by white "over the horizon" and makes this position look like a win for black, when it is really a win for white.

around a million nodes per second on the latest PC, allowing us to search roughly 200 million nodes per move under standard time controls (three minutes per move). The branching factor for chess is about 35, on average, and 35^5 is about 50 million, so if we used minimax search we could look ahead only about five plies. Though not incompetent, such a program can be fooled easily by an average human chess player, who can occasionally plan six or eight plies ahead. With alpha–beta search we get to about 10 ply, which results in an expert level of play. Section 6.7 describes additional pruning techniques that can extend the effective search depth to roughly 14 plies. To reach grandmaster status we would need an extensively tuned evaluation function and a large database of optimal opening and endgame moves. It wouldn't hurt to have a supercomputer to run the program on.

6.5 Games That Include an Element of Chance

In real life, there are many unpredictable external events that put us into unforeseen situations. Many games mirror this unpredictability by including a random element, such as the throwing of dice. In this way, they take us a step nearer reality, and it is worthwhile to see how this affects the decision-making process.

Backgammon is a typical game that combines luck and skill. Dice are rolled at the beginning of a player's turn to determine the legal moves. In the backgammon position of Figure 6.10, for example, white has rolled a 6–5, and has four possible moves.

Although White knows what his or her own legal moves are, White does not know what Black is going to roll and thus does not know what Black's legal moves will be. That means White cannot construct a standard game tree of the sort we saw in chess and tic-tac-toe. A

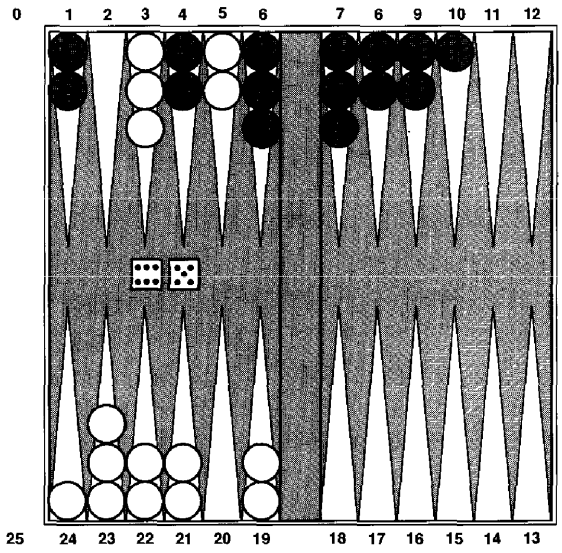


Figure 6.10 A typical backgammon position. The goal of the game is to move all one's pieces off the board. White moves clockwise toward 25, and black moves counterclockwise toward 0. A piece can move to any position unless there are multiple opponent pieces there; if there is one opponent, it is captured and must start over. In the position shown, White has rolled 6–5 and must choose among four legal moves: (5–10,5–11), (5–11,19–24), (5–10,10–16), and (5–11,11–16).

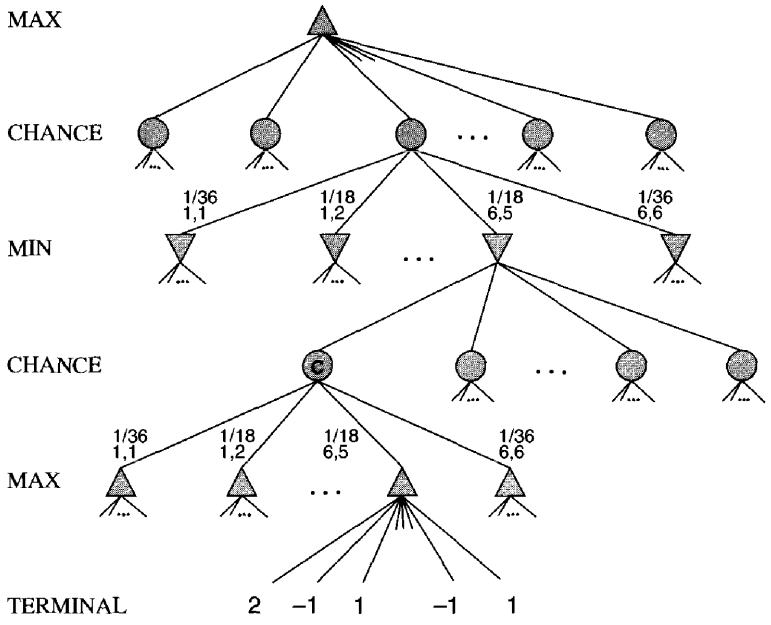


Figure 6.11 Schematic game tree for a backgammon position.

CHANCE NODES

game tree in backgammon must include **chance nodes** in addition to MAX and MIN nodes. Chance nodes are shown as circles in Figure 6.11. The branches leading from each chance node denote the possible dice rolls, and each is labeled with the roll and the chance that it will occur. There are 36 ways to roll two dice, each equally likely; but because a 6–5 is the same as a 5–6, there are only 21 distinct rolls. The six doubles (1–1 through 6–6) have a 1/36 chance of coming up, the other 15 distinct rolls a 1/18 chance each.

The next step is to understand how to make correct decisions. Obviously, we still want to pick the move that leads to the best position. However, the resulting positions do not have definite minimax values. Instead, we can only calculate the **expected value**, where the expectation is taken over all the possible dice rolls that could occur. This leads us to generalize

EXPECTIMINIMAX VALUE

the **minimax value** for deterministic games to an **expectiminimax value** for games with chance nodes. Terminal nodes and MAX and MIN nodes (for which the dice roll is known) work exactly the same way as before; chance nodes are evaluated by taking the weighted average of the values resulting from all possible dice rolls, that is,

$$\text{EXPECTIMINIMAX}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in Successors(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MIN node} \\ \sum_{s \in Successors(n)} P(s) \cdot \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a chance node} \end{cases}$$

where the successor function for a chance node n simply augments the state of n with each possible dice roll to produce each successor s and $P(s)$ is the probability that that dice roll occurs. These equations can be backed up recursively all the way to the root of the tree, just as in minimax. We leave the details of the algorithm as an exercise.

Position evaluation in games with chance nodes

As with minimax, the obvious approximation to make with expectiminimax is to cut the search off at some point and apply an evaluation function to each leaf. One might think that evaluation functions for games such as backgammon should be just like evaluation functions for chess—they just need to give higher scores to better positions. But in fact, the presence of chance nodes means that one has to be more careful about what the evaluation values mean. Figure 6.12 shows what happens: with an evaluation function that assigns values [1, 2, 3, 4] to the leaves, move A_1 is best; with values [1, 20, 30, 400], move A_2 is best. Hence, the program behaves totally differently if we make a change in the scale of some evaluation values! It turns out that, to avoid this sensitivity, the evaluation function must be a *positive linear* transformation of the probability of winning from a position (or, more generally, of the expected utility of the position). This is an important and general property of situations in which uncertainty is involved, and we discuss it further in Chapter 16.

Complexity of expectiminimax

If the program knew in advance all the dice rolls that would occur for the rest of the game, solving a game with dice would be just like solving a game without dice, which minimax

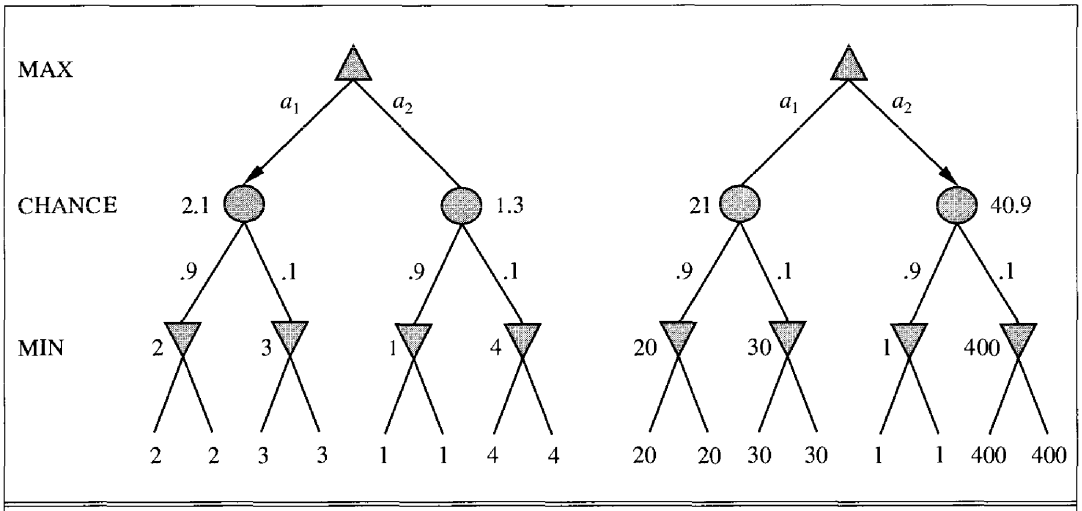


Figure 6.12 An order-preserving transformation on leaf values changes the best move.

does in $O(b^m)$ time. Because expectiminimax is also considering all the possible dice-roll sequences, it will take $O(b^m n^m)$, where n is the number of distinct rolls.

Even if the search depth is limited to some small depth d, the extra cost compared with that of minimax makes it unrealistic to consider looking ahead very far in most games of chance. In backgammon n is 21 and b is usually around 20, but in some situations can be as high as 4000 for dice rolls that are doubles. Three plies is probably all we could manage.

Another way to think about the problem is this: the advantage of alpha–beta is that it ignores future developments that just are not going to happen, given best play. Thus, it concentrates on likely occurrences. In games with dice, there are *no* likely sequences of moves, because for those moves to take place, the dice would first have to come out the right way to make them legal. This is a general problem whenever uncertainty enters the picture: the possibilities are multiplied enormously, and forming detailed plans of action becomes pointless, because the world probably will not play along.

No doubt it will have occurred to the reader that perhaps something like alpha–beta pruning could be applied to game trees with chance nodes. It turns out that it can. The analysis for MIN and MAX nodes is unchanged, but we can also prune chance nodes, using a bit of ingenuity. Consider the chance node C in Figure 6.11 and what happens to its value as we examine and evaluate its children. Is it possible to find an upper bound on the value of C before we have looked at all its children? (Recall that this is what alpha–beta needs to prune a node and its subtree.) At first sight, it might seem impossible, because the value of C is the *average* of its children's values. Until we have looked at all the dice rolls, this average could be anything, because the unexamined children might have any value at all. But if we put bounds on the possible values of the utility function, then we can arrive at bounds for the average. For example, if we say that all utility values are between $+3$ and -3, then the value of leaf nodes is bounded, and in turn we *can* place an upper bound on the value of a chance node without looking at all its children.

Card games

Card games are interesting for many reasons besides their connection with gambling. Among the huge variety of games, we will focus on those in which cards are dealt randomly at the beginning of the game, with each player receiving a hand of cards that is not visible to the other players. Such games include bridge, whist, hearts, and some forms of poker.

At first sight, it might seem that card games are just like dice games: the cards are dealt randomly and determine the moves available to each player, but all the dice are rolled at the beginning! We will pursue this observation further. It will turn out to be quite useful in practice. It is also quite wrong, for interesting reasons.

Imagine two players, MAX and MIN, playing some practice hands of four-card two handed bridge with all the cards showing. The hands are as follows, with MAX to play first:

MAX : ♡6 ♢6 ♣9 8 MIN : ♡4 ♠2 ♣10 5 .

Suppose that MAX leads the ♣9. MIN must now follow suit, playing either the ♣10 or the ♣5. MIN plays the ♣10 and wins the trick. MIN goes next and leads the ♠2. MAX has no spades (and so cannot win the trick) and therefore must throw away some card. The obvious choice is the ♢6 because the other two remaining cards are winners. Now, whichever card MIN leads for the next trick, MAX will win both remaining tricks and the game will be tied at two tricks each. It is easy to show, using a suitable variant of minimax (Exercise 6.12), that MAX's lead of the ♣9 is in fact an optimal choice.

Now let's modify MIN's hand, replacing the ♡4 with the ♢4:

MAX : ♡6 ♢6 ♣9 8 MIN : ♢4 ♠2 ♣10 5 .

The two cases are entirely symmetric: play will be identical, except that on the second trick MAX will throw away the ♡6. Again, the game will be tied at two tricks each and the lead of the ♣9 is an optimal choice.

So far, so good. Now let's hide one of MIN's cards: MAX knows that MIN has either the first hand (with the ♡4) or the second hand (with the ♢4), but has no idea which. MAX reasons as follows:

> The ♣9 is an optimal choice against MIN's first hand and against MIN's second hand, so it must be optimal now because I know that MIN has one of the two hands.

More generally, MAX is using what we might call "averaging over clairvoyancy." The idea is to evaluate a given course of action when there are unseen cards by first computing the minimax value of that action for each possible deal of the cards, and then computing the expected value over all deals using the probability of each deal.

If you think this is reasonable (or if you have no idea because you don't understand bridge), consider the following story:

> Day 1: Road *A* leads to a heap of gold pieces; Road *B* leads to a fork. Take the left fork and you'll find a mound of jewels, but take the right fork and you'll be run over by a bus.
> Day 2: Road *A* leads to a heap of gold pieces; Road *B* leads to a fork. Take the right fork and you'll find a mound of jewels, but take the left fork and you'll be run over by a bus.
> Day 3: Road *A* leads to a heap of gold pieces; Road *B* leads to a fork. Guess correctly and you'll find a mound of jewels, but guess incorrectly and you'll be run over by a bus.

Obviously, it's not unreasonable to take Road *B* on the first two days. No sane person, though, would take Road *B* on Day 3. Yet this is exactly what averaging over clairvoyancy suggests: Road *B* is optimal in the situations of Day 1 and Day 2; therefore it is optimal on Day 3, because one of the two previous situations must hold. Let us return to the card game: after MAX leads the ♣9, MIN wins with the ♣10. As before, MIN leads the ♠2, and now MAX is at the fork in the road without any instructions. If MAX throws away the ♡6 and MIN still has the ♡4, the ♡4 becomes a winner and MAX loses the game. Similarly, If MAX throws away the ♢6 and MIN still has the ♢4, MAX also loses. Therefore, playing the ♣9 first leads to a situation where MAX has a 50% chance of losing. (It would be much better to play the ♡6 and the ♢6 first, guaranteeing a tied game.)

The lesson to be drawn from all this is that when information is missing, one must consider *what information one will have* at each point in the game. The problem with MAX's algorithm is that it assumes that in each possible deal, play will proceed *as if all the cards are visible*. As our example shows, this leads MAX to act as if all *future* uncertainty will be resolved when the time comes. MAX's algorithm will also never decide to *gather* information (or *provide* information to a partner), because within each deal there's no need to do so; yet in games such as bridge, it is often a good idea to play a card that will help one discover things about one's opponent's cards or that will tell one's partner about one's own cards. These kinds of behaviors are generated automatically by an optimal algorithm for games of imperfect information. Such an algorithm searches not in the space of world states (hands of cards), but in the space of **belief states** (beliefs about who has which cards, with what probabilities). We will be able to explain the algorithm properly in Chapter 17, once we have developed the necessary probabilistic machinery. In that chapter, we will also expand on one final and very important point: in games of imperfect information, it's best to give away as little information to the opponent as possible, and often the best way to do this is to act *unpredictably*. This is why restaurant hygiene inspectors do random inspection visits.

6.6 STATE-OF-THE-ART GAME PROGRAMS

One might say that game playing is to AI as Grand Prix motor racing is to the car industry: state-of-the-art game programs are blindingly fast, incredibly well-tuned machines that incorporate very advanced engineering techniques, but they aren't much use for doing the shopping. Although some researchers believe that game playing is somewhat irrelevant to mainstream AI, it continues to generate both excitement and a steady stream of innovations that have been adopted by the wider community.

CHESS

Chess: In 1957, Herbert Simon predicted that within 10 years computers would beat the human world champion. Forty years later, the Deep Blue program defeated Garry Kasparov in a six-game exhibition match. Simon was wrong, but only by a factor of 4. Kasparov wrote:

> The decisive game of the match was Game 2, which left a scar in my memory ... we saw something that went well beyond our wildest expectations of how well a computer would be able to foresee the long-term positional consequences of its decisions. The machine

> refused to move to a position that had a decisive short-term advantage—showing a very human sense of danger. (Kasparov, 1997)

Deep Blue was developed by Murray Campbell, Feng-Hsiung Hsu, and Joseph Hoane at IBM (see Campbell *et al.*, 2002), building on the Deep Thought design developed earlier by Campbell and Hsu at Carnegie Mellon. The winning machine was a parallel computer with 30 IBM RS/6000 processors running the "software search" and 480 custom VLSI chess processors that performed move generation (including move ordering), the "hardware search" for the last few levels of the tree, and the evaluation of leaf nodes. Deep Blue searched 126 million nodes per second on average, with a peak speed of 330 million nodes per second. It generated up to 30 billion positions per move, reaching depth 14 routinely. The heart of the machine is a standard iterative-deepening alpha–beta search with a transposition table, but the key to its success seems to have been its ability to generate extensions beyond the depth limit for sufficiently interesting lines of forcing/forced moves. In some cases the search reached a depth of 40 plies. The evaluation function had over 8000 features, many of them describing highly specific patterns of pieces. An "opening book" of about 4000 positions was used, as well as a database of 700,000 grandmaster games from which consensus recommendations could be extracted. The system also used a large endgame database of solved positions, containing all positions with five pieces and many with six pieces. This database has the effect of substantially extending the effective search depth, allowing Deep Blue to play perfectly in some cases even when it is many moves away from checkmate.

The success of Deep Blue reinforced the widely held belief that progress in computer game-playing has come primarily from ever-more-powerful hardware—a view encouraged by IBM. Deep Blue's creators, on the other hand, state that the search extensions and evaluation function were also critical (Campbell *et al.*, 2002). Moreover, we know that several recent algorithmic improvements have allowed programs running on standard PCs to win every World Computer-Chess Championship since 1992, often defeating massively parallel opponents that could search 1000 times more nodes. A variety of pruning heuristics are used to reduce the effective branching factor to less than 3 (compared with the actual branching factor of about 35). The most important of these is the **null move** heuristic, which generates a good lower bound on the value of a position, using a shallow search in which the opponent gets to move twice at the beginning. This lower bound often allows alpha–beta pruning without the expense of a full-depth search. Also important is **futility pruning**, which helps decide in advance which moves will cause a beta cutoff in the successor nodes.

NULL MOVE

FUTILITY PRUNING

The Deep Blue team declined a chance for a rematch with Kasparov. Instead, the most recent major competition in 2002 featured the program FRITZ against world champion Vladimir Kramnik. The eight game match ended in a draw. The conditions of the match were much more favorable to the human, and the hardware was an ordinary PC, not a supercomputer. Still, Kramnik commented that "It is now clear that the top program and the world champion are approximately equal."

CHECKERS

Checkers: Beginning in 1952, Arthur Samuel of IBM, working in his spare time, developed a checkers program that learned its own evaluation function by playing itself thousands of times. We describe this idea in more detail in Chapter 21. Samuel's program began as a

novice, but after only a few days' self-play had improved itself beyond Samuel's own level (although he was not a strong player). In 1962 it defeated Robert Nealy, a champion at "blind checkers," through an error on his part. Many people felt that this meant computers were superior to people at checkers, but this was not the case. Still, when one considers that Samuel's computing equipment (an IBM 704) had 10,000 words of main memory, magnetic tape for long-term storage, and a .000001–GHz processor, the win remains a great accomplishment.

Few other people attempted to do better until Jonathan Schaeffer and colleagues developed Chinook, which runs on regular PCs and uses alpha–beta search. Chinook uses a precomputed database of all 444 billion positions with eight or fewer pieces on the board to make its endgame play flawless. Chinook came in second in the 1990 U.S. Open and earned the right to challenge for the world championship. It then ran up against a problem, in the form of Marion Tinsley. Dr. Tinsley had been world champion for over 40 years, losing only three games in all that time. In the first match against Chinook, Tinsley suffered his fourth and fifth losses, but won the match 20.5–18.5. The world championship match in August 1994 between Tinsley and Chinook ended prematurely when Tinsley had to withdraw for health reasons. Chinook became the official world champion.

Schaeffer believes that, with enough computing power, the database of endgames could be enlarged to the point where a forward search from the initial position would always reach solved positions, i.e., checkers would be completely solved. (Chinook has announced a win as early as move 5.) This kind of exhaustive analysis can be done by hand for 3×3 tic-tac-toe and has been done by computer for Qubic ($4 \times 4 \times 4$ tic-tac-toe), Go-Moku (five in a row), and Nine-Men's Morris (Gasser, 1998). Remarkable work by Ken Thompson and Lewis Stiller (1992) solved all five-piece and some six-piece chess endgames, making them available on the Internet. Stiller discovered one case where a forced mate existed but required 262 moves; this caused some consternation because the rules of chess require some "progress" to occur within 50 moves.

OTHELLO

Othello, also called Reversi, is probably more popular as a computer game than as a board game. It has a smaller search space than chess, usually 5 to 15 legal moves, but evaluation expertise had to be developed from scratch. In 1997, the Logistello program (Buro, 2002) defeated the human world champion, Takeshi Murakami , by six games to none. It is generally acknowledged that humans are no match for computers at Othello.

BACKGAMMON

Backgammon: Section 6.5 explained why the inclusion of uncertainty from dice rolls makes deep search an expensive luxury. Most work on backgammon has gone into improving the evaluation function. Gerry Tesauro (1992) combined Samuel's reinforcement learning method with neural network techniques (Chapter 20) to develop a remarkably accurate evaluator that is used with a search to depth 2 or 3. After playing more than a million training games against itself, Tesauro's program, TD-GAMMON, is reliably ranked among the top three players in the world. The program's opinions on the opening moves of the game have in some cases radically altered the received wisdom.

GO

Go is the most popular board game in Asia, requiring at least as much discipline from its professionals as chess. Because the board is 19×19, the branching factor starts at 361, which is too daunting for regular search methods. Up to 1997 there were no competent

programs at all, but now programs often play respectable moves. Most of the best programs combine pattern recognition techniques (when the following pattern of pieces appears, this move should be considered) with limited search (decide whether these pieces can be captured, staying within the local area). The strongest programs at the time of writing are probably Chen Zhixing's Goemate and Michael Reiss' Go4++, each rated somewhere around 10 kyu (weak amateur). Go is an area that is likely to benefit from intensive investigation using more sophisticated reasoning methods. Success may come from finding ways to integrate several lines of local reasoning about each of the many, loosely connected "subgames" into which Go can be decomposed. Such techniques would be of enormous value for intelligent systems in general.

BRIDGE

Bridge is a game of imperfect information: a player's cards are hidden from the other players. Bridge is also a *multiplayer* game with four players instead of two, although the players are paired into two teams. As we saw in Section 6.5, optimal play in bridge can include elements of information-gathering, communication, bluffing, and careful weighing of probabilities. Many of these techniques are used in the Bridge Baron^{TM} program (Smith *et al.*, 1998), which won the 1997 computer bridge championship. While it does not play optimally, Bridge Baron is one of the few successful game-playing systems to use complex, hierarchical plans (see Chapter 12) involving high-level ideas such as **finessing** and **squeezing** that are familiar to bridge players.

The GIB program (Ginsberg, 1999) won the 2000 championship quite decisively. GIB uses the "averaging over clairvoyancy" method, with two crucial modifications. First, rather than examining how well each choice works for every possible arrangement of the hidden cards—of which there can be up to 10 million—it examines a random sample of 100 arrangements. Second, GIB uses **explanation-based generalization** to compute and cache general rules for optimal play in various standard classes of situations. This enables it to solve each deal *exactly*. GIB's tactical accuracy makes up for its inability to reason about information. It finished 12th in a field of 35 in the par contest (involving just play of the hand) at the 1998 human world championship, far exceeding the expectations of many human experts.

Two-Person Non-Zero-Sum Games

In Chapter 4, we gave a complete treatment of 2-person zero-sum games. These are the games in which the two players' interests are directly opposed. As mentioned in §4.4, zero-sum games are usually games of human inventions. Most games in realistic situations are not zero-sum: the two players can both gain (or both lose) depending on their course of actions. Unfortunately, there is no such satisfactory theory for these non-zero-sum games as there is for the strictly competitive (i.e. zero-sum) games.

In this chapter, we will discuss some of the main issues and methods for non-zero-sum games. We first see when a non-zero-sum game can be transformed into a zero-sum game. Then we talk about two types of games: cooperative games, in which such things as preplay communications, binding contracts, and side payments are allowed, and non-cooperative games, in which these are forbidden. We will discuss the Nash equilibrium of non-zero-sum non-cooperative games. We will also see the problems with using Nash equilibrium as solution to these games. Then we discuss a simple solution for cooperative non-zero-sum game that involves threats and side payment. We finally give an application of these methods to evolutionary biological system.

6.1 Zero or Not Zero?

For non-zero-sum games, we can also represent them by matrices but each entry being a pair of numbers (x, y), where x represents the payoff to Rose and y the payoff to Colin.

Example 1. Consider the game:

$$\begin{bmatrix} (27,-5) & (17,0) \\ (19,-1) & (23,-3) \end{bmatrix}.$$

The $(1,2)$-th entry $(17,0)$ represents that if Rose plays her first strategy and Colin his second, then Rose will get HK$17 from the House (whatever this means), and Colin will get nothing. We note that we can also write this matrix into two matrices:

$$R = \begin{bmatrix} 27 & 17 \\ 19 & 23 \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} -5 & 0 \\ -1 & -3 \end{bmatrix}$$

where the matrix R is the payoff matrix to Rose, and C is Colin's payoff matrix. Thus sometimes, non-zero-sum games are also called *bi-matrix games*. For zero-sum games, we have $R = -C$.

Since there is a complete theory for zero-sum games, it is natural to ask if a given non-zero-sum game can be transformed into a zero-sum game first, before we try to solve it as a non-zero-sum game. We have already seen an example of this: Example 14 in Chapter 5. We saw that a constant-sum game can be transformed into a zero-sum game if we subtract the constant sum c from one player's payoffs. But there are non-constant games that can be transformed into zero-sum games too.

Example 2. Consider the game in Example 1. It's not constant-sum. However, let us multiply each Rose's payoff by $\frac{1}{2}$ and then subtract 17 from it, i.e.

$$\frac{1}{2}\begin{bmatrix} 27 & 17 \\ 19 & 23 \end{bmatrix} - \frac{17}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 1 & 3 \end{bmatrix} = W. \tag{6.1}$$

We see that Rose's new payoff matrix W is now exactly opposite to Colin's payoff matrix C. The game becomes zero-sum.

Notice that Rose and Colin strategies for playing W should be the same as those for playing R and C. It's because W amounts to halving the bet of R and giving the House HK\$17/2 each time Rose plays the game. Think of playing "paper-stone-scissors" with your friend for HK\$1 per game. The optimal strategy is $(1/3, 1/3, 1/3)$. Now assume that the bet is reduced to HK\$0.5 and that every time you play the game, you have to give HK\$8.5 to Hong Kong Jockey Club. If you are forced to play this game, then your optimal strategy will still be $(1/3, 1/3, 1/3)$. It doesn't change with the new bet or the deduction from the Jockey Club.

For the zero-sum game W, the optimal strategies for Rose and Colin are $(2/7, 5/7)$ and $(3/7, 4/7)$ respectively, and the value of the game is $15/7$ to Rose, i.e. $-15/7$ to Colin. Thus the optimal strategies for the game in Example 1 are: $(2/7, 5/7)$ for Rose and $(3/7, 4/7)$ for Colin. The expected payoff for Colin is still $-15/7$. For Rose, the expected payoff for the game W is $15/7$, hence the expected payoff v for the original game R will be given by $\frac{15}{7} = \frac{1}{2}v - \frac{17}{2}$, i.e. $v = 149/7$.

Obviously, the main question in your mind now is how do we get the number $\frac{1}{2}$ and $\frac{17}{2}$ in (6.1). Can it always be done? The answer is no. For if yes, that would mean all bi-matrix games can be reduced to matrix games, and the world will be a better place for all of us — except for Nash, because there will be no Nash equilibria but only von Neumann minimax equilibria. But when can a bi-matrix game be transformed into a matrix game? From (6.1), we see that

$$\frac{1}{2}R - \frac{17}{2}E = W = -C,$$

where E is the matrix of all ones. Thus given an arbitrary bi-matrix games with game matrices R and C, it can be transformed into a zero-sum game if there exists two numbers $\alpha > 0$ and β, such that

$$\alpha R + \beta E = -C. \tag{6.2}$$

We need $\alpha > 0$ because αR would then be like increasing the bet α time each time we play Rose's game. So α has to be positive. But how do we verify (6.2) and get our α and β? The answer is easy.

Example 3. Consider the game:

$$\begin{bmatrix} (2,-5) & (3,-7) \\ (1,-1) & (6,4) \end{bmatrix}.$$

We have:

$$R = \begin{bmatrix} 2 & 3 \\ 1 & 6 \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} -5 & -7 \\ -1 & 4 \end{bmatrix}.$$

If

$$\alpha \begin{bmatrix} 2 & 3 \\ 1 & 6 \end{bmatrix} + \beta \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = - \begin{bmatrix} -5 & -7 \\ -1 & 4 \end{bmatrix}, \tag{6.3}$$

then from the $(1,1)$-th and the $(1,2)$-th entries, we would have

$$\begin{cases} \alpha \cdot 2 + \beta \cdot 1 = 5 \\ \alpha \cdot 3 + \beta \cdot 1 = 7 \end{cases}$$

Thus $\alpha = 2$ and $\beta = 1$. But if we put this into (6.3) and check the remaining $(2,1)$-th and $(2,2)$-th entries, we see that

$$\begin{aligned} 2 \cdot 1 + 1 \cdot 1 &\neq 1, \\ 2 \cdot 6 + 1 \cdot 1 &\neq -4. \end{aligned}$$

Thus (6.3) does not hold and the bi-matrix game cannot be reduced to a matrix game.

Example 4. Consider the game:

$$R = \begin{bmatrix} 0 & 35 & 15 & 10 & -10 \\ 25 & -5 & 5 & 25 & 50 \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} 2 & -5 & -1 & 0 & 4 \\ -3 & 3 & 1 & -3 & -8 \end{bmatrix}.$$

If $\alpha R + \beta E = -C$, then from the $(1,1)$-th and the $(1,2)$-th entries, we would have

$$\begin{cases} \alpha \cdot 0 + \beta \cdot 1 = -2 \\ \alpha \cdot 35 + \beta \cdot 1 = 5 \end{cases}$$

Thus $\alpha = \frac{1}{5}$ and $\beta = -2$. It is easy to verify indeed that

$$\begin{aligned} &\frac{1}{5} \begin{bmatrix} 0 & 35 & 15 & 10 & -10 \\ 25 & -5 & 5 & 25 & 50 \end{bmatrix} - 2 \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ = &\begin{bmatrix} -2 & 5 & 1 & 0 & -4 \\ 3 & -3 & -1 & 3 & 8 \end{bmatrix} = -C. \end{aligned}$$

Thus the bi-matrix game is reduced to a zero-sum game with game matrix:

$$\begin{bmatrix} -2 & 5 & 1 & 0 & -4 \\ 3 & -3 & -1 & 3 & 8 \end{bmatrix}$$

which is exactly the game matrix we have considered in Example 15 of Chapter 4. There we found that the optimal mixed minimax strategies for Rose and Colin are $(4/7, 3/7)$ and $(2/7, 0, 5/7, 0, 0)$ respectively. The value of the game is $1/7$. Thus the expected payoff for Colin is $-1/7$. The expected payoff v for Rose will be $\frac{1}{7} = \frac{1}{5}v - 2$. That is $v = 75/7$.

6.2 Pure Nash Equilibrium

In this section, we consider 2-person, non-zero-sum, non-cooperative games. We will introduce the Nash equilibrium and discuss some of its peculiar features. As in zero-sum games, we start with pure equilibrium. Recall that for pure minimax equilibria (i.e. saddle points), we can determine them by using either the movement diagrams or the maximin method. The same is true here. We begin with the movement diagrams.

Example 5. Consider two games:

$$\begin{bmatrix} (4,1) & (2,3) \\ (5,9) & (1,10) \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} (1,5) & (8,7) & (2,9) & (9,4) \\ (7,1) & (9,6) & (10,4) & (1,2) \\ (1,9) & (2,5) & (5,8) & (8,6) \\ (8,4) & (4,9) & (6,8) & (2,8) \end{bmatrix}. \tag{6.4}$$

For Rose, she wants to maximize her payoffs, so she would prefer the highest payoff at each column. Therefore at each payoff, we draw an arrow from it to the highest payoff at the same column. Similarly for Colin, he would want to maximize his payoffs, so he would prefer the highest payoff at each row. Therefore at each payoff, we draw an arrow from it to the highest payoff at the same row. If there exists one (or more) payoff(s) where the arrows are pointing in from every directions, then that (or those) payoff(s) will be the *pure Nash equilibrium*, and the corresponding strategies are called the *pure Nash equilibrium* strategies. For our matrices above, we have:

$$\begin{bmatrix} (4,1) & \rightarrow & \boxed{(2,3)} \\ \downarrow & & \uparrow \\ (5,9) & \rightarrow & (1,10) \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} (1,5) & \rightarrow & (8,7) & \rightarrow & (2,9) & \leftarrow & (9,4) \\ | & & \downarrow & & \downarrow & & \uparrow \\ (7,1) & \rightarrow & \boxed{(9,6)} & \leftarrow & (10,4) & - & (1,2) \\ | & & \uparrow & & \uparrow & & | \\ (1,9) & \leftarrow & (2,5) & - & (5,8) & - & (8,6) \\ \downarrow & & | & & \uparrow & & | \\ (8,4) & \rightarrow & (4,9) & \leftarrow & (6,8) & - & (2,8) \end{bmatrix}.$$

Thus for the first game, $(1,0)$ and $(0,1)$ are the Nash equilibrium strategies for Rose and Colin respectively. The payoffs to Rose and Colin are 2 and 3 respectively. For the second game, Rose's and Colin's Nash equilibrium strategies are both equal to $(0,1,0,0)$.

Next we discuss the maximin method. In zero-sum game, Colin's payoffs are negation of Rose's payoffs. For non-zero-sum games, since Colin's payoffs are given explicitly, instead of a maximin method, we expect a *maximax method.*

Example 6. Consider the two games matrices in (6.4) again. Since Rose would like to maximize her payoff at each column, so at each column, we use a left-brace "{" to mark the entry that has the highest Rose's payoff in that column. Similarly, Colin would like to maximize his payoff at each row, so at each row, we use a right-brace "}" to mark the entry that has the highest Colin's payoff in that row. Then we get:

$$\begin{bmatrix} (4,1) & \{(2,3)\} \\ \{(5,9) & (1,10)\} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} (1,5) & (8,7) & (2,9)\} & \{(9,4) \\ (7,1) & \{(9,6)\} & \{(10,4) & (1,2) \\ (1,9)\} & (2,5) & (5,8) & (8,6) \\ \{(8,4) & (4,9)\} & (6,8) & (2,8) \end{bmatrix}.$$

We see clearly that the entry that is embraced from both sides is the pure Nash equilibrium.

Why do we call these entries "equilibrium"? Consider the 2-by-2 game above. It is clear that if Colin knows that Rose will play $(1,0)$, then it is to Colin's best interest to play $(0,1)$ and get the profits of $(2,3)$ for both. Conversely, if Rose knows that Colin is going to play $(0,1)$, then it is to Rose's best interest to play $(1,0)$ and get the same profits of $(2,3)$ for both. Thus the strategies $(1,0)$ and $(0,1)$ for Rose and Colin are in a state of equilibrium. Once the players get there, they will not like to change their strategies. Similarly for the 4-by-4 games, once Colin knows that Rose plays $(0,1,0,0)$, he has no choice but to play $(0,1,0,0)$ himself, and vice versa for Rose.

Why do we call them "Nash" equilibrium? It's because it is Nash who first showed that every n-person (in particular, 2-person) non-cooperative games will have at least one such equilibrium (be it in the form of pure strategies or mixed strategies). Thus we honor his contribution and call these equilibrium the *Nash equilibrium*.

Like in the case of zero-sum games, there will be non-zero-sum games with no pure Nash equilibrium and we will have to seek the mixed Nash equilibrium. The following game is an example:

$$\begin{bmatrix} (2,4)\} & \{(1,0) \\ \{(3,1) & (0,4)\} \end{bmatrix}.$$

But before we embark on the journey for finding the mixed Nash equilibrium, let us discuss some unpleasant properties of Nash equilibrium. The first one concerns multiple Nash equilibria.

For zero-sum games, we have proved in Theorem 4.5 that if there exist more than one saddle points (i.e. pure minimax equilibria), then these saddle points are equivalent and interchangeable. More precisely, although the optimal strategies corresponding to the different saddle points are different, it doesn't matter which optimal strategy you choose to play, because the payoff will be the same. However, it is a different story for non-zero-sum games.

Example 7. (Dating game) Rose prefers to have a drink rather than watching football, and Colin prefers to watch football than a drink. Yet, they both like to do something together. If we can measure happiness by numbers, we probably end up with the following game:

		Colin	
		football	drink
Rose	football	$\{(2,5)\}$	$(-1,-1)$
	drink	$(0,0)$	$\{(5,2)\}$

We see that there are two pure Nash equilibria, with different payoffs: $(2,5)$ and $(5,2)$. Thus the payoffs are not equivalent. Since the payoffs are not the same, the corresponding Nash equilibrium strategies are not interchangeable. For example, it would be better for Rose to use the strategy $(0,1)$ than $(1,0)$ and get 5 units of happiness. Similarly, for Colin, it would be better for him to use $(1,0)$ than $(0,1)$ and get 5 units of happiness himself. However, if both chooses to use their favorite equilibrium strategy, then they end up with $(0,0)$ which is worse than both Nash equilibria.

In this game, disclosure of one's intentions may actually be advantageous. If Rose just makes the announcement: "We will go for a drink. End of discussion!" and if they have not yet married, then Colin may be forced (or happy) to follow along and goes for a drink. However, if Colin thinks Rose is just bluffing (or nagging if they are husband and wife), then he may choose to ignore it and ends up in the worse case for both.

Thus we see that Nash equilibrium may not be the answer to our questions as to how the game should be played.

In zero-sum games, the interests of Rose and Colin are directly opposite. Thus whatever Rose wins, Colin loses; and there is no way to get a win-win solution. But in non-zero-sum game, this situation may arise. Consider

$$\begin{bmatrix} (1,4) & \{(5,6)\} \\ \{(4,1) & (2,3)\} \end{bmatrix}.$$

In this case, the optimal strategies of $(1,0)$ for Rose and $(0,1)$ for Colin will give a win-win solution to both. However, most of the time, we end up in a lose-lose situation.

Example 8. (Prisoners' Dilemma) Consider the Prisoner's Dilemma game in §3.3:

ATENȚIE!

Matricea conține anii de închisoare, utilitățile ar trebui să fie cu minus!

		Colin	
		confess	not confess
Rose	confess	{(10,10)}	{(0,20)
	not confess	(20,0)}	(1,1)

The payoffs in the table are the numbers of years they will have to serve in the prison. Thus both players will try to minimize the payoffs. We see that they end up with the Nash equilibrium at $(10,10)$ with both players confessing. This outcome is worse than if neither one confesses. Thus instead of a win-win solution, the players opt for a lose-lose solution—the Nash equilibrium.

In economic terminology, we say that the Nash equilibrium in Example 8 is not Pareto-optimal. **Vilfredo Pareto** is an Italian economist who proposed that one shouldn't accept an outcome as our solution if there is another outcome that can make everyone better off. More precisely, we have the following definition.

Definition 6.1. An outcome of a game is *non-Pareto-optimal* if there is another outcome which would give both players higher payoffs, or would give one player the same payoff but the other player a higher payoff. An outcome is *Pareto-optimal* if there is no such other outcome.

In general, a game may have many Pareto-optimal outcomes. In fact the other three outcomes in Example 8 are all Pareto-optimal. Only the Nash equilibrium is not! For example, if we are already at the outcome $(20,0)$, then changing to any one of the other three outcomes will either lead to increase in years for Colin. As a matter of fact, in a zero-sum game, every outcome is Pareto-optimal. It is because given that outcome, if one of the player changes the strategy, then because it is zero-sum, either Rose or Colin will get less payoff.

Pareto's idea was that we should try to get a solution that is Pareto-optimal. His idea is a cogent principle of *group rationality*. Unfortunately, in game theory, this group rationality is put in the second priority after *individual rationality*. Nash equilibrium is obtained by fulfilling the individual desire more than the group desire, though the selfish decisions may come back to haunt the individuals. Pollution, arms race, and exhaustive fishing are all good examples of these "good" game decisions.

6.3 Mixed Nash Equilibrium

When a game does not have a pure Nash equilibrium, we can use the method in §4.3 to locate its *mixed Nash equilibrium*. Don't worry that you can't find it. Nash has guaranteed you that at least one such equilibrium exists, and we are going to find them in this section. To find them, let us recall the method in §4.3 first. Consider a zero-sum game with game matrix G. Let the mixed minimax equilibrium strategies for Rose and Colin be $\mathbf{y} = (y_1, y_2, \ldots, y_m)$ and $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ respectively. Then we have

$$\mathbf{y} \cdot G = [y_1, y_2, \ldots, y_m] \cdot G = [v, v, \ldots, v] \quad \text{and} \quad G \cdot \mathbf{x}^T = G \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} v \\ v \\ \vdots \\ v \end{bmatrix}, \quad (6.5)$$

where v is the value of the game. More precisely, the minimax equilibrium strategy is one such that when it is multiplied to G, every entry in the product is the same, and this common value is the value of the game.

The mixed Nash equilibrium strategies for bi-matrix games are obtained in exactly the same fashion. The only complication is that we now have two matrices R (Rose's payoff matrix) and C (Colin's payoff matrix) instead of just one matrix. For each matrix, we can obtain the corresponding minimax solution: $\mathbf{y}$ and $\mathbf{x}$. So which pair gives the Nash equilibrium strategies? Before answering this question, let's recall how to find them.

Example 9. Consider the game:

$$\begin{bmatrix} (-1, 4) & (5, 0) \\ (2, -10) & (-5, 5) \end{bmatrix}.$$

It has no pure Nash equilibria. The R and C matrices are:

$$R = \begin{bmatrix} -1 & 5 \\ 2 & -5 \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} 4 & 0 \\ -10 & 5 \end{bmatrix}.$$

For the R matrix, we can obtain the mixed minimax strategies and the value of the game by the oddment method. They are $\mathbf{y}_R = (7/13, 6/13)$, $\mathbf{x}_R = (10/13, 3/13)$, and $v_R = 5/13$. As a check, let's see if (6.5) holds. We have

$$\mathbf{y}_R \cdot R = \begin{bmatrix} \frac{7}{13}, \frac{6}{13} \end{bmatrix} \begin{bmatrix} -1 & 5 \\ 2 & -5 \end{bmatrix} = \begin{bmatrix} \frac{5}{13}, \frac{5}{13} \end{bmatrix} = [v_R, v_R] \quad (6.6)$$

and

$$R \cdot \mathbf{x}_R^T = \begin{bmatrix} -1 & 5 \\ 2 & -5 \end{bmatrix} \begin{bmatrix} \frac{10}{13} \\ \frac{3}{13} \end{bmatrix} = \begin{bmatrix} \frac{5}{13} \\ \frac{5}{13} \end{bmatrix} = \begin{bmatrix} v_R \\ v_R \end{bmatrix}.$$

Thus (6.5) holds.

Similarly, for the matrix C, we can obtain the corresponding minimax solution: $\mathbf{y}_C = (15/19, 4/19)$, $\mathbf{x}_C = (5/19, 14/19)$, and $v_C = 20/19$. We can check if (6.5) holds too:

$$\mathbf{y}_C \cdot C = \begin{bmatrix} \frac{15}{19}, \frac{4}{19} \end{bmatrix} \begin{bmatrix} 4 & 0 \\ -10 & 5 \end{bmatrix} = \begin{bmatrix} \frac{20}{19}, \frac{20}{19} \end{bmatrix} = [v_C, v_C]$$

and

$$C \cdot \mathbf{x}_C^T = \begin{bmatrix} 4 & 0 \\ -10 & 5 \end{bmatrix} \begin{bmatrix} \frac{5}{19} \\ \frac{14}{19} \end{bmatrix} = \begin{bmatrix} \frac{20}{19} \\ \frac{20}{19} \end{bmatrix} = \begin{bmatrix} v_C \\ v_C \end{bmatrix}. \tag{6.7}$$

So now we have four strategies: $\mathbf{y}_R$, $\mathbf{y}_C$, $\mathbf{x}_R$, and $\mathbf{x}_C$. Which two of them are the Nash equilibrium strategies?

To answer this, we have to recall the situation in zero-sum games. What properties does the minimax equilibrium solution have there? We know that if Colin plays the minimax equilibrium strategy, he can guarantee what Rose will be getting (she will be getting the value of the game), independent of what Rose plays. Since the game is zero-sum, what Rose is getting is what he will be paying. Thus by playing the minimax equilibrium strategy, Colin can guarantee what Rose and he himself are getting. Similarly, if Rose plays the minimax equilibrium strategy, she is guaranteeing her payoffs and that of Colin, independent of what Colin will play. Any derivations from the minimax equilibrium strategy risk the chance of being exploited by the opponent, in the sense that the opponent will then have another strategy that can offer him/her better payoff than the value of the game.

Now we claim that Rose's strategy $\mathbf{y}_C$ and Colin's strategy $\mathbf{x}_R$ will have precisely the same properties as mentioned above. Thus this pair of strategies $\mathbf{y}_C$ and $\mathbf{x}_R$ constitutes the mixed Nash equilibrium. Instead of proving it, we illustrate this with a simple example.

Example 10. We now demonstrate that $\mathbf{y}_C = (15/19, 4/19)$ and $\mathbf{x}_R = (10/13, 3/13)$ is a mixed Nash equilibrium for Example 9 and the payoffs to Rose and Colin are $v_R = 5/13$ and $v_C = 20/19$ respectively.

1. If Colin plays $\mathbf{x}_R$, then Rose's payoff is v_R independent of what Rose plays.
   Reason: Let's say Rose plays $(y, 1-y)$ for some y between 0 and 1. Then Rose's expected payoff is given by

$$\begin{aligned} [y, 1-y] \cdot R \cdot \mathbf{x}_R^T &= [y, 1-y] \begin{bmatrix} -1 & 5 \\ 2 & -5 \end{bmatrix} \begin{bmatrix} \frac{10}{13} \\ \frac{3}{13} \end{bmatrix} \\ &= [y, 1-y] \begin{bmatrix} \frac{5}{13} \\ \frac{5}{13} \end{bmatrix} = \frac{5}{13} = v_R. \end{aligned}$$

2. If Rose plays $\mathbf{y}_C$, then Colin's payoff is v_C independent of what Colin plays.
   Reason: Let's say Colin plays $(x, 1-x)$ for some x between 0 and 1. Then Colin's expected payoff is given by

$$\begin{aligned} \mathbf{y}_C \cdot C \cdot \begin{bmatrix} x \\ 1-x \end{bmatrix} &= \begin{bmatrix} \frac{15}{19}, \frac{4}{19} \end{bmatrix} \begin{bmatrix} 4 & 0 \\ -10 & 5 \end{bmatrix} \begin{bmatrix} x \\ 1-x \end{bmatrix} \\ &= \begin{bmatrix} \frac{20}{19}, \frac{20}{19} \end{bmatrix} \begin{bmatrix} x \\ 1-x \end{bmatrix} = \frac{20}{19} = v_C. \end{aligned}$$

3. If Colin does not play $\mathbf{x}_R$, then Rose can get a better payoff than v_R.
   Reason: Let Colin's and Rose's strategies be $(x, 1-x)$ and $(y, 1-y)$ respectively. Then Rose's expected payoff is

$$\begin{aligned} [y, 1-y] \cdot R \cdot \begin{bmatrix} x \\ 1-x \end{bmatrix} &= [y, 1-y] \cdot \begin{bmatrix} -1 & 5 \\ 2 & -5 \end{bmatrix} \cdot \begin{bmatrix} x \\ 1-x \end{bmatrix} \\ &= [y, 1-y] \begin{bmatrix} 5-6x \\ 7x-5 \end{bmatrix}. \end{aligned} \tag{6.8}$$

As in Chapter 4, we plot the lines $5-6x$ and $7x-5$ for x between 0 and 1, see Figure 6.1. We see that the lines intersect at $x = 10/13$ which corresponds to $\mathbf{x}_R$. The height of the lines at $x = 10/13$ is precisely $v_R = 5/13$. If Colin chooses his strategy such that $x < 10/13$, then $5-6x > 7x-5$. From (6.8) or from the figure, we see that Rose can simply play $(1,0)$ (i.e. $y = 1$) to get a payoff better than $5/13$. If Colin choose $x > 10/13$, then $5-6x < 7x-5$, and Rose can simply play $(0,1)$ (i.e. $y = 0$) to get a payoff better than $5/13$.

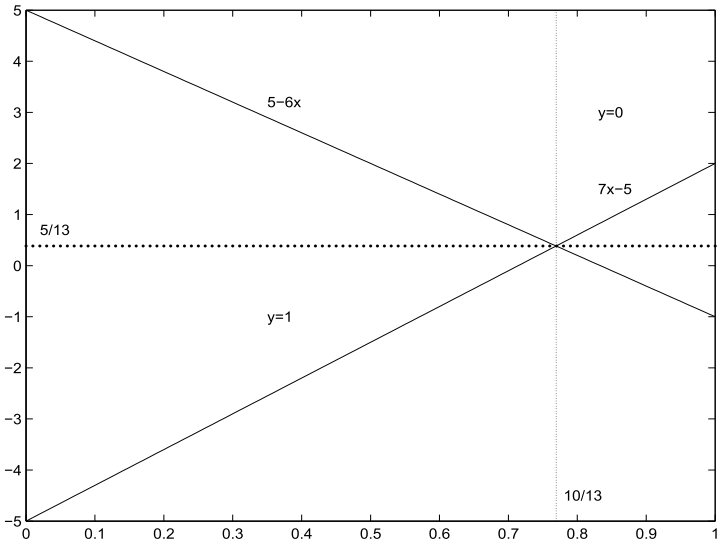


Figure 6.1. Rose's Payoff for Example 10.

4. If Rose does not play $\mathbf{y}_C$, then Colin can get a better payoff than v_C.
   Reason: The reason is similar to that in Item 3, and we will not repeat it. Simply say, if Rose plays stupidly, then Colin can play either $(1,0)$ or $(0,1)$ (depending on Rose's stupid strategy) to get a better payoff than v_C.

In conclusion, the pair of strategies $\mathbf{y}_C$ and $\mathbf{x}_R$ has all the properties we want for an equilibrium solution. Once the players play these strategies, they will not want to move to other strategies. No more! We term these strategies the *mixed Nash equilibrium strategies*. The payoff (v_R, v_C) is the *mixed Nash equilibrium*.

For easy reference later, let us summarize the results in the following table:

Rose's strategy	Colin's strategy	Rose's payoff	Colin's payoff
$\mathbf{y}_C$	whatever		v_C
whatever	$\mathbf{x}_R$	v_R	
$\neq \mathbf{y}_C$	for some		$\geq v_C$
for some	$\neq \mathbf{x}_R$	$\geq v_R$	

Nash equilibrium strategies

Let's try another example.

Example 11. Consider the dating game in Example 7. It already has two unpleasant pure Nash equilibria. Now we claim that it has a mixed Nash equilibrium too. First

the R and C matrices are given by:

$$R = \begin{bmatrix} 2 & -1 \\ 0 & 5 \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} 5 & -1 \\ 0 & 2 \end{bmatrix}.$$

By the oddment method, we find that $\mathbf{y}_R = (5/8, 3/8)$, $\mathbf{x}_R = (3/4, 1/4)$, $v_R = 5/4$, $\mathbf{y}_C = (1/4, 3/4)$, $\mathbf{x}_C = (3/8, 5/8)$, and $v_C = 5/4$. Thus the mixed Nash equilibrium strategies are: $\mathbf{y}_C = (1/4, 3/4)$ for Rose and $\mathbf{x}_R = (3/4, 1/4)$ for Colin. The expected payoffs to Rose and Colin are the same: 5/4. When playing these Nash strategies, the players not only control what the others are gaining, but as we have seen in Example 10, it is to their own interest to play these strategies. For if not, their opponent will take advantage of it and wins more.

We note again that the Nash equilibria are not interchangeable, be they mixed or pure. In fact, the mixed Nash equilibrium in this case is far worse than either one of the pure Nash equilibria we obtain in Example 7.

1.1 Probability and Optimization

Consider a 3-person game for players A, B, and C. Each can either show 1 finger or 2 fingers. If you are the only one showing 1 finger, you won HK$1 from the house. If you are the only one showing 2 fingers, then you won HK$2 from the house. Otherwise, everyone gets nothing. What should the mixed Nash equilibrium be?

Since the game is symmetric with respect to each player, we can safely assume that the optimal strategy for each one of them is the same. In fact, if A has a strategy that wins more than the other two, then B and C will definitely follow suit, and sooner or later, they will play the same strategies (mixed or pure).

Can the Nash equilibrium be consisted only of pure strategies? That is: all players play either only 1 finger or 2 fingers and will not change the strategy from game to game. Obviously, this cannot be an equilibrium. For if B and C play 1 finger constantly, a rational A will change strategy and play 2 fingers instead. Similarly for the case if B and C play 2 fingers. Thus there is no *pure Nash equilibrium*, and we have to look for *mixed Nash equilibrium*.

Let the optimal strategy for A, B and C be $(x, 1-x)$ where x is the fraction of time that the player will play 1 finger and $1-x$ is the fraction of time that the player will play 2 fingers. (Of course $0 \leq x \leq 1$.) We will have the following table:

Moves			Payoff			
A	B	C	A	B	C	Percentage of Time
1	1	1	0	0	0	x^3
1	1	2	0	0	2	$x^2(1-x)$
1	2	1	0	2	0	$x^2(1-x)$
1	2	2	1	0	0	$x(1-x)^2$
2	1	1	2	0	0	$x^2(1-x)$
2	1	2	0	1	0	$x(1-x)^2$
2	2	1	0	0	1	$x(1-x)^2$
2	2	2	0	0	0	$(1-x)^3$

We can see that the expected payoff $P_A(x)$ for A is:

$$P_A(x) = x(1-x)^2 + 2x^2(1-x) = x - x^3.$$

The maximum of $P_A(x)$ can be obtained by differentiation:

$$0 = \frac{dP_A(x)}{dx} = \frac{d(x - x^3)}{dx} = 1 - 3x^2 \quad \Longrightarrow x = \frac{1}{\sqrt{3}}.$$

Thus the optimal x_* is $x_* = 1/\sqrt{3} \approx 0.58$.

One may safely conclude that the "optimal strategy" is $(x_*, 1 - x_*) \approx (0.58, 0.42)$, i.e. 58% of the time you play 1 finger and 42% of the time you play 2 fingers. Then everyone is expected to have:

$$P_A\left(\frac{1}{\sqrt{3}}\right) = x - x^3\big|_{x_*=\frac{1}{\sqrt{3}}} = \frac{2\sqrt{3}}{9} = 0.385, \tag{1.1}$$

i.e. HK$0.385 per game for each player. This is the result one would get if one only uses calculus to maximize the expected payoff.

But is this the happy ending of the story? Sir Winston Churchill once said: *"However beautiful the strategy, you should occasionally look at the results"*. So look again! If you are as rational as A, you will reason as follows. Now that I know B and C will play $(x_*, 1 - x_*) \approx (0.58, 0.42)$, is there another strategy where I can earn more? (That's what game theory comes in. It transcends calculus.)

Let's say A does the problem again, but assuming that now he will play $(y, 1 - y)$ instead of $(x_*, 1 - x_*)$, where y may not be equal to x_*. Then we have the following situation:

Moves			Payoff			Percentage of Time
A	B	C	A	B	C	
1	1	1	0	0	0	yx_*^2
1	1	2	0	0	2	$yx_*(1 - x_*)$
1	2	1	0	2	0	$yx_*(1 - x_*)$
1	2	2	1	0	0	$y(1 - x_*)^2$
2	1	1	2	0	0	$(1 - y)x_*^2$
2	1	2	0	1	0	$(1 - y)x_*(1 - x_*)$
2	2	1	0	0	1	$(1 - y)x_*(1 - x_*)$
2	2	2	0	0	0	$(1 - y)(1 - x_*)^2$

Hence the expected payoff for A will be:

$$P_A(x_*) = y(1 - x_*)^2 + 2(1 - y)x_*^2 = y(1 - 2x_* - x_*^2) + 2x_*^2. \tag{1.2}$$

Since $x_* = 1/\sqrt{3}$, then

$$P_A\left(\frac{1}{\sqrt{3}}\right) = y\left(1 - 2\frac{1}{\sqrt{3}} - \frac{1}{3}\right) + \frac{2}{3}. \tag{1.3}$$

The important point to note here is that

$$1 - 2\frac{1}{\sqrt{3}} - \frac{1}{3} \approx -0.49 < 0.$$

Thus if A chooses $y = 0$ in (1.3), he has

$$P_A\left(\frac{1}{\sqrt{3}}\right) = \frac{2}{3} = 0.66,$$

which is larger than the payoff 0.38 he gets in (1.1).

Bingo! We see that if B and C are *smart* enough to keep to their "optimal strategy", then A can simply play his *dumb strategy* of $(y, 1-y) = (0, 1)$, (i.e. A play 2 fingers constantly). Then A can get more than his mathematically smart companions.

Thus the strategy $(x_*, 1-x_*) \approx (0.58, 0.42)$ cannot be a mixed Nash equilibrium, because if the others stick to it, some players may defect from it. In fact, if C is as smart as I claim he is, he will reason as follows. Now that I know A will play (0,1) and B will play $(x_*, 1-x_*)$, what should I play? If he uses the same arguments and the tables above, he will come up with the strategy of $(1, 0)$ and reap a higher profit of $1 - 1/\sqrt{3} \approx 0.42$. (Can you derive this result?) The situation is rapidly decaying for B, because now he is losing in every game with A playing (0,1) and C playing (1,0). It is before long that B will realize this and changes his strategy according. Thus the 3 players will fall into a circular reasoning and get nowhere.

1.2 Mixed Nash Equilibrium

So what should the mixed Nash equilibrium be? Let us consider the problem again, assuming that A will play $(y, 1-y)$ and B and C will play $(x, 1-x)$ where x and y are not fixed yet and are to be determined. Then the expected payoff to A will be a function of x and y is of the form similar to (1.2):

$$P_A(x, y) = y(1-x)^2 + 2(1-y)x^2 = y(1 - 2x - x^2) + 2x^2. \tag{1.4}$$

The expected payoff of A depends on the factor $1 - 2x - x^2$. Look at the graph of $f(x) = 1 - 2x - x^2$ in Figure 1.1. We see that it has two roots at

$$x = \frac{-2 \pm \sqrt{4+4}}{2} = -1 \pm \sqrt{2}.$$

There is one root that is in between 0 and 1, namely $x_0 = \sqrt{2} - 1 \approx 0.41$.

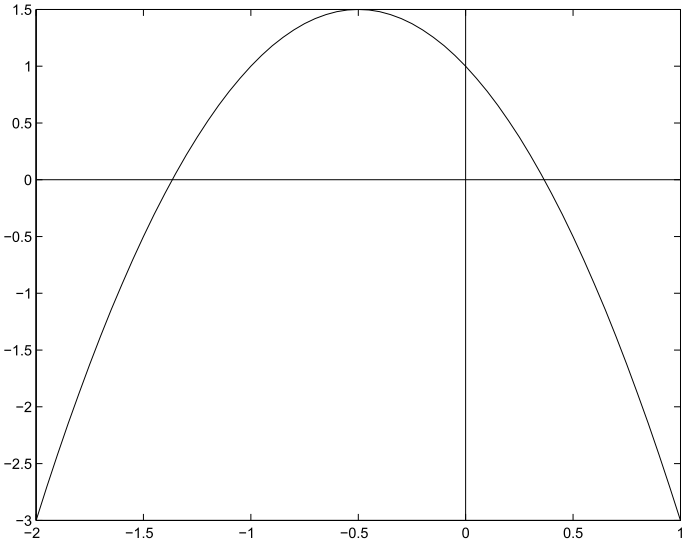


Figure 1.1. The function $f(x) = 1 - 2x - x^2$.

If $x > x_0$, then $1 - 2x - x^2 < 0$. Hence to maximize his expected payoff, A should choose $y = 0$ in (1.4). Then he has

$$P_A(x, 0) = 0 \cdot (1 - 2x - x^2) + 2x^2 = 2x^2.$$

On the other hand, if $x < x_0$, then $1 - 2x - x^2 > 0$. Hence A should choose $y = 1$ in (1.4), and he will have

$$P_A(x, 1) = 1 \cdot (1 - 2x - x^2) + 2x^2 = 1 - 2x + x^2 = (1 - x)^2.$$

Let us plot $P_A(x, y)$ against x in these two cases, see Figure 1.2.

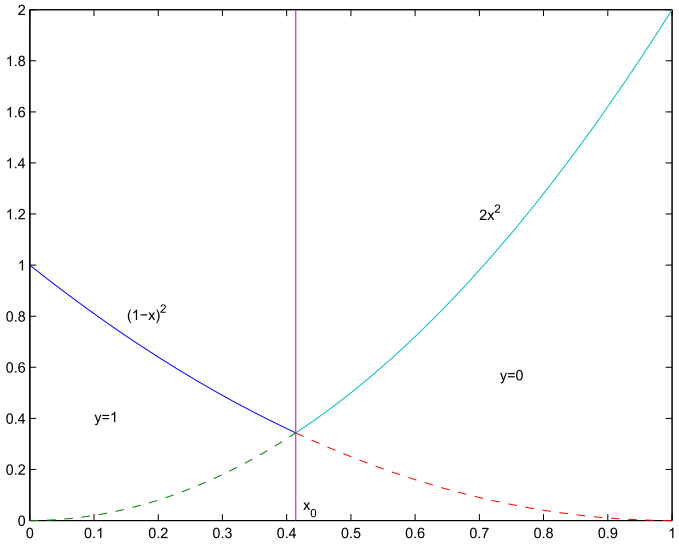


Figure 1.2. Payoff $P_A(x)$ versus x.

We see that the minimum $P_A(x, y)$ is obtained at $x = x_0 \approx 0.41$. At this point, the payoff for A is

$$P_A(x, y) = 2x_0^2 \approx 0.343,$$

and this is independent of what y is.

Notice that for all the other $x \neq x_0$, A can either choose $y = 0$ (when $x > x_0$) or $y = 1$ (when $x < x_0$) to reap a higher payoff than HK\$0.343. Thus the strategy $(x, 1-x) = (x_0, 1-x_0) \approx (0.41, 0.59)$ is the *optimal mixed strategy* for B and C. More precisely, if both B and C play 1 finger 41% of the time and 2 fingers 59% of the time, they can assure that A gets only HK\$0.343 no matter what A does. Any other deviation from this optimal mixed strategy by B and C will lead to advantage for A.

By symmetry, Player A should also play this optimal mixed strategy $(x_0, 1 - x_0)$, so that B and C will only get HK\$0.343 no matter what they do.

In conclusion, we see that it is advantageous for all three players to play the optimal mixed strategy $(x_0, 1 - x_0)$. Hence this is the Nash equilibrium.

1.3 Questions and Answers

There are many questions raised by the students after I finished the example. Here I try to answer some of them.

Why do the players prefer a smaller payoff of HK$0.343 (which is obtained by Game Theory) to a higher payoff of HK$0.385 (which is obtained by Calculus)? Are they being irrational? The answer is no. It's because these players are highly rational. By playing $(x_0, 1-x_0)$, they can be sure that no one can get higher than HK$0.343, if the other two stick to it. (Recall that if two stick with $(x_0, 1-x_0)$, the third will get HK$0.343 no matter what he plays.) Thus no one has an incentive to defect from this strategy. And this is the basic idea behind Nash equilibrium. On the contrary, if two of them play the optimal solution by calculus, i.e. $(x_*, 1-x_*)$, the third has all the incentive to play $(0, 1)$ to get a higher payoff of HK$0.667.

Yes, if B and C stick to $(x_0, 1-x_0)$, then A will get HK$0.343 no matter what he plays. But let's say he still prefer to play $(0, 1)$. Then the payoff to B and C will be $x_0 \cdot (1-x_0) = 0.242$ (can you derive this?), which is less than 0.343 as promised by Nash. So will A play this just to hurt B and C? Again the answer is no if A is rational. For if he plays (0,1), then B and C, after knowing this fact, will certainly change their strategies. Then A can no longer guarantee himself that he can get HK$0.343 per game. In Nash's terminology, the strategies $(0, 1)$ for A, $(x_0, 1-x_0)$ for B, and $(x_0, 1-x_0)$ for C is not a Nash equilibrium. I guess the moral of the story is: *if there's no gain, don't cause the pain.*

What if B uses $(1, 0)$ and C uses $(0, 1)$? Then A will always lose. Should he play? Yes, as long as B and C are not cooperating to get A. For A can play $(0, 1)$ continuously. Then C will know that he gets nothing by playing $(0, 1)$. If C is rational, he will change his strategy.

1.4 A Useful Trick

Having computed the Nash equilibrium $(x_0, 1-x_0)$ in §1.2, let me tell you a faster way of finding it.

Look at (1.4) again. We have

$$P_A(x, y) = y(1-x)^2 + (1-y)(2x^2), \tag{1.5}$$

where $(y, 1-y)$ is the strategy for A, and $(x, 1-x)$ is the strategy for both B and C. From (1.5), we see that the payoff for A is a *combination* of the two functions $(1-x)^2$ and $2x^2$. By combination, we mean that the payoff $P_A(x, y)$ is a fraction of $(1-x)^2$ plus a fraction of $2x^2$. (Note that because $0 \le y \le 1$, so $y(1-x)^2$ is a fraction of $(1-x)^2$, and $(1-y)(2x^2)$ is a fraction of $2x^2$.) I have drawn these two functions in Figure 1.2. We see that they intersect exactly at x_0.

To the right of x_0, i.e. when $x > x_0$, we see that the function $2x^2$ is higher than $(1-x)^2$. Thus if A wants to maximize his payoff $P_A(x, y)$, he will want more fraction of $2x^2$ than more fraction of $(1-x)^2$. In fact, he would want 100% of $2x^2$ and 0% of $(1-x)^2$. By (1.1), that means he wants $y = 0$. This is exactly what we have in §1.2. See Figure 1.2 too.

To the left of x_0, i.e. when $x < x_0$, we see that the function $(1-x)^2$ is higher than $2x^2$. Thus if A wants to maximize his payoff $P_A(x, y)$, he will want more fraction of $(1-x)^2$ than more fraction of $2x^2$. In fact, he would want 100% of $(1-x)^2$ and 0% of $2x^2$. By (1.1), that means he wants $y = 1$. This is again exactly what we have in §1.2.

For B and C, after they see the figure, they will know that if they choose $x > x_0$ or $x < x_0$, A will get higher profit than if they choose $x = x_0$. Thus, they will settle with $x = x_0$. By symmetry argument (that the game is symmetric with respect to all three players), we deduce that A should also choose $y = x_0$.

Thus we see that we can simply draw the graphs of $2x^2$ and $(1-x)^2$, and then all conclusions fall into place. We will see the power of this derivation again when we talk about 2-person zero-sum games. For the time being, let us use this to solve for a similar 4-person game.

Example 1. Consider a 4-person game for players A, B, C, and D as follows. Each can either show 1 finger or 2 fingers. If you are the only one showing 1 finger, you won HK$1 from the house. If you are the only one showing 2 fingers, then you won HK$2 from the house. Otherwise, everyone gets nothing. The mixed Nash equilibrium can be found using the trick above. In fact, the payoff for A is

$$P_A(x,y) = y(1-x)^3 + (1-y)(2x^3),$$

where $(y, 1-y)$ is the strategy for A, and $(x, 1-x)$ is the strategy for B, C, and D. The graphs for $(1-x)^3$ and $2x^3$ are given in Figure 1.3.

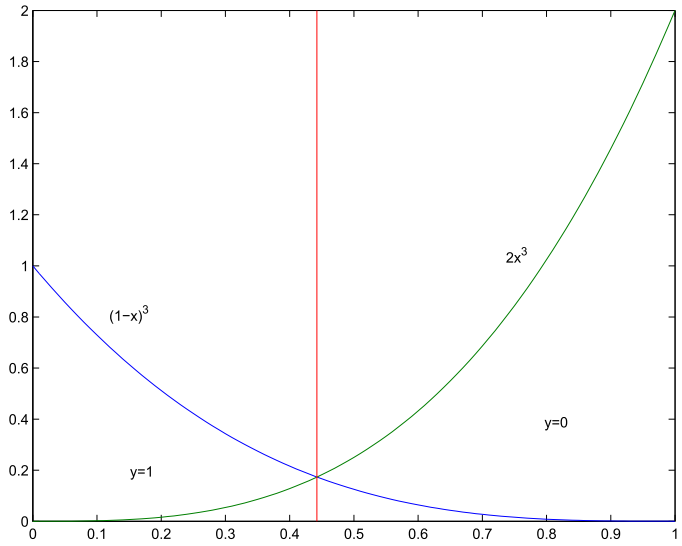


Figure 1.3. Function $(1-x)^3$ and $2x^3$.

They intersect at the point where $(1-x)^3 = 2x^3$, or equivalently, $1-3x+3x^2-3x^3 = 0$. It's a cubic polynomial and is not easy to solve. The root between 0 and 1 is found by computer (e.g. EXCEL) to be 0.443. Thus the mixed Nash equilibrium is $(0.443, 0.557)$ for all players. The payoff for each player will be $2 \cdot (0.443)^3 \approx 0.173$.

6.6 Two-Person Cooperative Games

Up to this point, we have only dealt with non-cooperative games, where such things as pre-play communications, contracts, side payments, and such are forbidden. As opposed to these, we may consider cooperative games. In the simplest of cooperative two-person games, the two players can obtain a certain amount of payoffs if they can come to a agreement. The main problem then is to decide how this payoff is to be divided between the two players.

To take a trivial example, suppose for their service in this course, we offer to give Rose and Colin HK$20,000 to divide any way they wish, if only they can agree on this division. The "even split" of HK$10,000 each seems eminently fair. But there is no way to enforce this if one of them were to insist, say, on HK$15,000. In other words, every division of the money is in some way possible: if Rose needs HK$9,500 very badly, and if she envisions that the bargaining with Colin will take forever, she may find herself forced to accept HK$9,500 rather than arguing endlessly, and let the money depreciate.

The search in such games is for a *fair* solution— an outcome that will adequately represent the players' bargaining *positions*, though not their bargaining *abilities*. It may be thought of as an arbitrated outcome: the outcome that a fair and impartial arbitrator might give. Let us consolidate the idea by an example.

Example 14. (Sum Matrix) Renault and Peugeot both make small cars (Mégane for Renault and 206 for Peugeot). Neither can produce enough to saturate the French market. But, if both produce as much as possible, they can over-saturate the market. If one company opts for full production (F) while the other produces only a small amount (M), then the former makes a large profit, while the other makes only a small profit. If both produce to full capacity, there is a glut on the market and they both lose money. If both produce small amounts, then there is a danger of losing market shares (and hence money) to foreign companies.

The situation is asymmetrical because Renault has renovated its factory which enables it to produce at a somewhat lower cost but must, on the other hand, amortize the new infrastructures. Hence Renault suffers a large loss if there is an underproduction, but only a small loss if there is an overproduction, while Peugeot's loss is large for overproduction and small for underproduction. Their profits are expressed as follows:

		Peugeot	
		F	M
Renault	F	$(-10,-40)$	$(40,10)$
	M	$(10,40)$	$(-40,-10)$

where the payoffs are in millions of euros. Their individual payoff matrices are:

$$R = \begin{bmatrix} -10 & 40 \\ 10 & -40 \end{bmatrix} \quad \text{and} \quad P = \begin{bmatrix} -40 & 10 \\ 40 & -10 \end{bmatrix}.$$

If there is no cooperation between the two companies, then we see that the game has two pure Nash equilibria: $(10, 40)$ and $(40, 10)$. With a little bit patience, you can find that it also has a mixed Nash equilibrium: $(0.5, 0.5)$ for Renault, $(0.8, 0.2)$ for Peugeot with payoffs 0 euros for both companies.

What happens if the two companies can sit down and work out the production plan? Clearly they can choose either one of the pure Nash equilibria, and get 50 millions of euros in total for the two companies. But is it the best they can get by working together? Will there be a pair of mixed strategies where they can get more in total? The answer is no. We can easily see this by summing their payoff matrices:

$$R + P = \begin{bmatrix} -50 & 50 \\ 50 & -50 \end{bmatrix}$$

which is called the *sum matrix* of the game. Now assuming that the mixed strategies for Renault and Peugeot are (y_1, y_2) and (x_1, x_2) respectively with y_1, y_2, x_1, and x_2 nonnegative, $y_1 + y_2 = 1$, and $x_1 + x_2 = 1$. The total payoff they can receive together will be:

$$[y_1, y_2] \begin{bmatrix} -50 & 50 \\ 50 & -50 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -50x_1y_1 + 50x_1y_2 + 50x_2y_1 - 50x_2y_2.$$

Since x_1, x_2, y_1, and y_2 are nonnegative, and since 50 is the maximum coefficient here, we have

$$\begin{aligned} -50x_1y_1 + 50x_1y_2 + 50x_2y_1 - 50x_2y_2 &\le 50x_1y_1 + 50x_1y_2 + 50x_2y_1 + 50x_2y_2 \\ &= 50[x_1(y_1 + y_2) + x_2(y_1 + y_2)] \\ &= 50[x_1 + x_2] = 50. \end{aligned}$$

What we have seen is that no matter how the companies negotiate, they can't get more than the maximum entry in the sum matrix. In effect, the payoff is Pareto-optimal — group-wise, the players have achieved the optimal solution for the group. Moreover, they can achieve this Pareto-optimal payoff simply by playing only pure strategies. In our case, it's either F for Renault and M for Peugeot, or M for Renault and M for Peugeot. Both choices earn 50 million euros for the two companies.

We have shown that there is always a way to achieve a Pareto-optimal solution by looking at the sum matrix. After the players know how much they together can earn as a group, now it's time for them to fight amongst themselves to see how this trove is to be divided. To see how this can be done, we have to quantitatively describe the negotiating powers of the players. This is done by a *threat matrix*.

Example 15. (Threat Matrix) At first glance, the game in Example 14 seems to be quite symmetric, which suggests that both companies should divide the total of 50 million euros amongst themselves: each would then receive 25 million euros of profit. But is it fair to both companies? If not, how would they negotiate a better deal for themselves?

Let us start with Renault. It can argue it like this: "We don't agree with the even-break of 25 million euros. If we don't get more, we will simply opt for F." If you are Peugeot, how would you response? You can say we then go for F too. But then you lose 30 million euros more than Renault does. If you say you go for M, still you earn 30 million euros less than Renault does. In fact, no matter what mixed strategy you play (recall that mixed strategies are just weighted proportions of your two pure strategies), you will earn 30 millions less than Renault. In any case, you are cornered. You have a *weaker bargaining position* than Renault has. You may really have to give more to Renault.

But wait a minute. The game is symmetric (so you think) and you may be able to threaten Renault by the same trick. Can you? So you said to Renault that you want more than 25 million euros and if not, you will simply opt for F. Then Renault will counter-threaten you and say if you want more, I will simply play the mixed strategy $(0.5, 0.5)$. In that case, both of you earn 0 euros (you can easily verify this). You will get the same cold response if you threaten Renault with the pure strategy M. You see that you are NOT in a negotiating position to ask for more. The question for you now will be how much should I yield to Renault. What is the fair division of the 50 million euros?

The answer can be obtained easily by forming the *threat matrix*, which is simply the difference between the payoff matrices of the two players:

$$R - P = \begin{bmatrix} 30 & 30 \\ -30 & -30 \end{bmatrix}. \tag{6.9}$$

From the threat matrix, we can now understand why Renault has a strong negotiating position. The first row is all positive and the minimum value is 30. Thus by playing the first pure strategy F, Renault has a 30 million euros advantage over Peugeot. From the threat matrix, we see that Peugeot has no negotiating power because the threat of playing any pure strategies (F or M) can be counter-balanced by Renault by using the mixed strategies $(0.5, 0.5)$. Obviously, Renault can't threat with the pure strategy M, for this strategy will always give advantage instead of threat to Peugeot.

In general, the value of the game for the threat matrix is called the *threat differential*. It is easy to check that the threat matrix in (6.9) has a pure minimax equilibrium

(i.e. saddle point) with the value of the game equals to 30. Thus the threat differential for the game is 30. It's time to decide how much each player gets.

Definition 6.2. The *arbitrated threat solution* to a 2-person cooperative game is one where

1. the total payoffs of the 2 players equals to the maximum value of the sum matrix, and
2. the difference of the payoffs to the players equals to the threat differential.

For our car makers, the total they get is 50 million euros and the threat differential is 30 million euros. Thus, the payoffs should be 40 millions to Renault and 10 millions to Peugeot. Then the sum will be 50 million and the difference is 30 millions as required.

Notice that we do not specify how the players can achieve these payoffs. Clearly with Renault opts for F and Peugeot opts for M, then we achieve the required payoffs. But it can be that Renault opts for M and Peugeot for F, and then after getting their payoffs of $(10, 40)$, Peugeot makes a *side payment* of 30 million euros to Renault to finish the cooperative agreement. Let us consider one more example.

Example 16. Consider the bi-matrix cooperative game:

$$R = \begin{bmatrix} -1 & 2 \\ 1 & -4 \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} -4 & 1 \\ 4 & -1 \end{bmatrix}.$$

The maximum entry in the sum matrix is clearly 5, so together the 2 players can expect to win 5 units. The threat matrix is:

$$R - C = \begin{bmatrix} 3 & 1 \\ -3 & -3 \end{bmatrix}.$$

It has a saddle point at $(1, 2)$-th entry. Thus the value of the game is 1, and so is the threat differential. In fact, in this game, Rose can threaten Colin by playing the first pure strategy. She will then always have 1 unit advantage over Colin. The arbitrated threat solution is simply: 3 for Rose and 2 for Colin (so that the sum of the payoffs is 5 and the difference is 1 as required). In this example, their strategies will be: $(0, 1)$ for Rose and $(1, 0)$ for Colin. It will give Rose 1 unit and Colin 4 units. But then Colin has to give back 2 units to Rose to finish the deal. Of course, it is better to write down everything in black and white before the game begins, if Rose trusts money more than Colin.

Example 17. Consider the bi-matrix game:

$$\begin{bmatrix} (5, 4) & (6, 3) \\ (3, 1) & (1, 3) \end{bmatrix}.$$

If there are no communication, then both Rose and Colin will play the first strategy (the pure Nash equilibrium) and together they earn 9 units. If cooperation is allowed, then the maximum entry in the sum matrix is still 9. How do they share the 9 units? The "equal share" will lead to a smaller payoff for Rose. However, the threat matrix is:

$$\begin{bmatrix} 1 & 3 \\ 2 & -2 \end{bmatrix}.$$

It has a mixed minimax equilibrium with strategies $(2/3, 1/3)$ for Rose and $(5/6, 1/6)$ for Colin, and the value of the game is $4/3$. Thus Rose is in the position to ask for $4/3$ units more from Colin. (How should Rose make her threat now?) The cooperative payoffs are therefore: $31/6$ for Rose and $23/6$ for Colin. Thus Rose gets $1/6$ unit more by pretending to cooperate but actually making the threat. It is interesting to see that Colin earns $1/6$ less just for being able to talk with Rose. Sometimes, it doesn't pay to listen.

N-Person Cooperative Games

When there are only two players in the game, the players can either play the game cooperatively or play against each other. However, when there are more than two players in the game, some players may choose to join together to form a *coalition* to play against other players or other groups of players. It is this process of coalition-formation, and the consequent bargaining possibilities that we are going to study in this chapter. We start with a new form to describe n-person games. Then we give one solution method for such games. We end the chapter by considering one business application — cost allocation.

7.1 Characteristic Form

For n-person games, the matrix form or the tree form representations of the games will be large and not be easy to work with. Von Neumann therefore had introduced another form of representation called the *characteristic form* to facilitate the solution of the games. Basically, the characteristic form lists the maximum payoffs that each possible coalition can get if such coalition is formed.

We shall label the players $P_1, P_2, \cdots, P_n$. Any collection of players in this group of players is called a *coalition*. In particular, the full collection $\mathcal{G} = \{P_1, P_2, \cdots, P_n\}$ is a coalition, called the *grand coalition*. For example, in a 3-person games, the possible coalitions are: $\{P_1\}$, $\{P_2\}$, $\{P_3\}$, $\{P_1, P_2\}$, $\{P_1, P_3\}$, $\{P_2, P_3\}$, and the grand coalition $\{P_1, P_2, P_3\}$.

If a coalition $\mathcal{S} \neq \mathcal{G}$ is formed, it is natural to think of the game as a contest between two coalitions: the coalition $\mathcal{S}$ and the coalition of all the players not in $\mathcal{S}$ (called the counter-coalition $\mathcal{S}^c$). This 2-coalition game is non-cooperative. Indeed, if the coalition and the counter-coalition cooperated, the grand coalition would form. When played against its counter-coalition, each coalition can use the joint strategies of some or all of its members. We assume that the coalition would know how best to use these strategies, so as to maximize the amount of payoffs received by all of its members. More precisely, we consider the 2-coalition game between $\mathcal{S}$ and $\mathcal{S}^c$ as a 2-person non-cooperative game and the coalition will try to maximize its profit by using Nash's strategies. The *characteristic function* ν is a function that records this maximum payoff for each coalition, which can be computed by solving the 2-coalition non-cooperative game.

Example 1. Consider the following three-person game. Each of the three players can choose either heads (H) or tails (T). If two players choose the same, but the remaining

player chooses differently, then this last player must give HK$1 to each of the other two players. In any other cases, there will be no payoff. For this game, $\nu(\{P_1, P_2, P_3\}) = 0$ since the game is zero-sum: the three players together can gain nothing as all payments must take place amongst them.

Suppose that the coalition $\mathcal{S} = \{P_2, P_3\}$ is formed. It will be opposed by the counter-coalition $\mathcal{S}^c = \{P_1\}$, giving what is essentially a two-person game between the two coalitions: $\mathcal{S}$ and $\mathcal{S}^c$. The coalition $\{P_1\}$ has two strategies H and T, while $\{P_2, P_3\}$ has four strategies, HH, HT, TH, and TT. This gives us a 2×4 zero-sum game:

		$\mathcal{S}$			
		HH	HT	TH	TT
$\mathcal{S}^c$	H	0	1	1	-2
	T	-2	1	1	0

For this matrix, the second and third columns are dominated, and it is easy to see that the value of the game is HK$$-1$. This is what P_1 expects to gain. The coalition $\{P_2, P_3\}$ then expects to gain HK$1 per game. Thus

$$\nu(\{P_1\}) = -1 \quad \text{and} \quad \nu(\{P_2, P_3\}) = 1.$$

By the symmetry of this game, we conclude that the characteristic function of this game is

$$\nu(\mathcal{S}) = \begin{cases} -1, & \text{if } \mathcal{S} = \{P_1\}, \{P_2\}, \text{or } \{P_3\}, \\ +1, & \text{if } \mathcal{S} = \{P_1, P_2\}, \{P_2, P_3\}, \text{or } \{P_3, P_1\}, \\ 0, & \text{if } \mathcal{S} = \{P_1, P_2, P_3\}. \end{cases}$$

where $|\mathcal{S}|$ is the number of players in coalition $\mathcal{S}$.

Example 2. Consider another 3-person cooperative game with payoff table:

Strategy	Payoff
(H,H,H)	$(-2, 1, 2)$
(H,H,T)	$(1, 1, -1)$
(H,T,H)	$(0, -1, 2)$
(H,T,T)	$(-1, 2, 0)$
(T,H,H)	$(1, -1, 1)$
(T,H,T)	$(0, 0, 1)$
(T,T,H)	$(1, 0, 0)$
(T,T,T)	$(1, 2, -2)$

What is $\nu(\{P_1, P_2, P_3\})$? It is the largest total payoff that all the players can get when they act together cooperatively. From the table, we see that they have eight pure strategies, and the maximum payoff is 1. Thus $\nu(\{P_1, P_2, P_3\}) = 1$.

Suppose that coalition $\mathcal{S} = \{P_1, P_3\}$ is formed. Then the counter-coalition is $\mathcal{S}^c = \{P_2\}$. Similar to Example 1, the coalition $\{P_2\}$ has two strategies H and T, while $\{P_1, P_3\}$ has four strategies, HH, HT, TH, and TT. This gives us a 2×4 non-zero-sum game:

		$\mathcal{S}$			
		HH	HT	TH	TT
$\mathcal{S}^c$	H	$(1, 0)$	$(1, 0)$	$(-1, 2)$	$(0, 1)$
	T	$(-1, 2)$	$(2, -1)$	$(0, 1)$	$(2, -1)$

The mixed Nash equilibrium for this game is $\mathbf{y}_C = (1/3, 2/3)$ for $\mathcal{S}^c$ and $\mathbf{x}_R = (1/3, 0, 2/3, 0)$ for $\mathcal{S}$. (This can be found easily by noting that in $\mathcal{S}$'s payoff matrix C, the second and the fourth columns are dominated, and hence the game is essentially a 2×2 bi-matrix game). Using the Nash equilibrium strategies, we find that $\nu(\mathcal{S}) = \nu(\{P_1, P_3\}) = 4/3$ and $\nu(\mathcal{S}^c) = \nu(\{P_2\}) = -1/3$. Computing in a similar way, we have $\nu(\{P_1, P_2\}) = 1$, $\nu(\{P_3\}) = 0$, $\nu(\{P_2, P_3\}) = 3/4$, and $\nu(\{P_1\}) = 1/4$.

Examples 1–2 show us how the characteristic function of a game may be obtained from its matrix forms. We see that for each coalition $\mathcal{S}$, we have to solve the 2-coalition non-cooperative game to obtain the characteristic value $\nu(\mathcal{S})$, and it may be a very tedious job. Nonetheless, the characteristic forms of n-person games do offer some conceptual advantages for analyzing the games, and it's time for us to define it.

Definition 7.1. The *characteristic function* of an n-person game is a function ν that is defined for every possible coalition and satisfies

$$\nu(\text{when } \mathcal{S} \text{ merges with } \mathcal{T}) \geq \nu(\mathcal{S}) + \nu(\mathcal{T}) \tag{7.1}$$

for any two coalitions $\mathcal{S}$ and $\mathcal{T}$ with no common members. A game with the characteristic function given is said to be in the *characteristic form.*

Condition (7.1) is known as *super-additivity* condition and represents the fact that coalitions $\mathcal{S}$ and $\mathcal{T}$, acting together, should obtain at least as much payoffs for their members as when they act separately. For if not, there is no point for them to merge. Note that by (7.1), $\nu(\mathcal{G})$ should be larger than $\nu(\mathcal{S})$ for all other $\mathcal{S}$.

Example 3. Consider a 3-person game with

$$\nu(\{P_1\}) = -\frac{1}{2},\ \nu(\{P_2\}) = 0,\ \nu(\{P_3\}) = -\frac{1}{2},$$

$$\nu(\{P_1, P_2\}) = \frac{1}{4},\ \nu(\{P_1, P_3\}) = 0,\ \nu(\{P_2, P_3\}) = \frac{1}{2},\ \nu(\{P_1, P_2, P_3\}) = 1.$$

We show that it is a characteristic function, i.e. it satisfies (7.1). Indeed,

$$\begin{aligned}
\frac{1}{4} = \nu(\{P_1, P_2\}) &\geq \nu(\{P_1\}) + \nu(\{P_2\}) = -\frac{1}{2}, \\
0 = \nu(\{P_1, P_3\}) &\geq \nu(\{P_1\}) + \nu(\{P_3\}) = -1, \\
\frac{1}{2} = \nu(\{P_2, P_3\}) &\geq \nu(\{P_2\}) + \nu(\{P_3\}) = -\frac{1}{2}, \\
1 = \nu(\{P_1, P_2, P_3\}) &\geq \nu(\{P_1\}) + \nu(\{P_2, P_3\}) = 0, \\
1 = \nu(\{P_1, P_2, P_3\}) &\geq \nu(\{P_2\}) + \nu(\{P_1, P_3\}) = 0, \\
1 = \nu(\{P_1, P_2, P_3\}) &\geq \nu(\{P_3\}) + \nu(\{P_1, P_2\}) = -\frac{1}{4}.
\end{aligned}$$

Notice that the characteristic function just tells us what the maximum payoffs that a coalition is expected to get, but it doesn't say anything about how the payoffs should be distributed amongst its members. This will be discussed later. The characteristic form of a game releases us from the burden of analyzing the competition and cooperation amongst individuals, and allows us to concentrate on what happens after the coalitions are formed. Also very often, games are given directly in characteristic function forms. Let us illustrate this by two examples.

Example 4. Three students in the class UGB253NA need to buy the textbook by Straffin. The list price is HK$110. There is a discount of HK$10 (respectively HK$20) on each book if it is bought in two (respectively three). We can express this 3-person "saving game", where the savings $\nu(\mathcal{S})$ for the coalitions are given as follows:

$$\nu(\{P_1\}) = \nu(\{P_2\}) = \nu(\{P_3\}) = 0,$$

$$\nu(\{P_1, P_2\}) = \nu(\{P_1, P_3\}) = \nu(\{P_2, P_3\}) = 20,\ \nu(\{P_1, P_2, P_3\}) = 60.$$

Example 5. A seller S owns a horse which is worthless to him. A farmer F and a butcher B would both like to buy the horse. The farmer feels that the horse is worth $1000, while the butcher feels that the horse is worth $500. In this game, the general idea is that, by transferring ownership of the horse from the seller to one of the two buyers, utility is created. Thus, coalition $\{S, F\}$ can create 1000 dollars of utility, which the two players can divide in any way they choose. For example, if they decide on a price of $300, then S gains $300 (he has exchanged a worthless horse for $300), while F gains $700 (he has obtained a $1000 horse for $300). Similarly, coalition $\{S, B\}$ can gain $500. Single-player coalitions obtain no utility; nor can $\{F, B\}$ obtain any utility.

It is also possible that coalition $\{S, F, B\}$ forms. If so, the best they can do is to give the horse to F, and then a total of $1000 will be obtained. We thus have the characteristic function:

$$\nu(\{S\}) = \nu(\{F\}) = \nu(\{B\}) = \nu(\{F, B\}) = 0,$$

$$\nu(\{S, B\}) = 500 \quad \text{and} \quad \nu(\{S, F\}) = \nu(\{S, F, B\}) = 1000.$$

Let us suppose that a game ν is played. What payoffs can the players expect from this game? This is not an easy question to answer — there is no generally accepted theory to determine a unique *solution* to such a game. We may assume, however, that no player will accept less than what he could obtain for himself when acting alone. We will also assume that the players are able to come to an agreement so that they can share all the available payoffs. This leads us to the concept of an *imputation.*

Definition 7.2. Let ν be the characteristic function of an n-person game. An imputation is a vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ satisfying

$$\sum_{i=1}^{n} x_i = \nu(\mathcal{G}), \tag{7.2}$$

$$x_i \geq \nu(\{P_i\}) \quad \text{for all } i. \tag{7.3}$$

By super-additivity (7.1), we know that $\nu(\mathcal{G})$ is the maximum payoff amongst all payoffs $\nu(\mathcal{S})$. Hence the players will cooperate to share this maximum payoff. The x_i in the theorem can be considered as the payoff requested by player P_i. Conditions (7.2) and (7.3) are known respectively as *group rationality* and *individual rationality.* In essence, an imputation is one of the possible distributions of the profits (payoffs) that might be obtained if the game ν is played.

In Example 4, the imputations are vectors $\mathbf{x} = (x_1, x_2, x_3)$ satisfying $x_1 + x_2 + x_3 = 60$, and $x_i \geq 0$ for $i = 1, 2, 3$. You may consider the x_i as the saving that the ith player asks for. In Example 1, the imputations x_i satisfies $x_i \geq -1$ for $i = 1, 2, 3$ and $x_1 + x_2 + x_3 = 0$. In Example 2, they satisfy $x_1 \geq 1/4$, $x_2 \geq -1/3$, $x_3 \geq 0$, and

$x_1+x_2+x_3=1$. In Example 5, they satisfy $x_i \geq 0$ for $i=S,F,B$ and $x_S+x_F+x_B=1000$.

Unfortunately, most games have an infinity number of imputations, and it is not clear how to distinguish one of these imputations as being the most "reasonable". Several "solution concepts" have been suggested for n-person games. We shall only consider one here, namely the *core*.

7.2 The Core

There are many different concepts of "a solution" to an n-person game. One idea is that a solution in the form of imputation should not be dominated by another imputation. For if so, then certainly some members would prefer the dominating imputation rather than a dominated one. The collection of non-dominated imputations is called a *core*. Any imputations in the core can be considered as a solution to the given n-person cooperative game.

Definition 7.3. The *core* of an n-person game ν is the collection of all imputations $\mathbf{x}=(x_1,x_2,\cdots,x_n)$ such that, for every coalition $\mathcal{S}=\{P_{i_1},P_{i_2},\ldots,P_{i_\ell}\}$,

$$x_{i_1}+x_{i_2}+\ldots+x_{i_\ell}\geq \nu(\mathcal{S}). \tag{7.4}$$

Note what this means: if an imputation belongs to the core, then every potential or actual coalition is obtaining as much as it possibly could obtain for its members if it were to form. In a way, the imputation is not dominated by another imputation. Thus a core exhibits a great deal of stability. If, on the other hand, an imputation is not in the core, then there is at least one coalition $\mathcal{S}$ whose members are not getting as much as they could. There will then be strong pressure on these members to change to another imputation. Let us look at some examples to consolidate the ideas.

Example 6. Consider Example 4. Let $\mathbf{x}=(x_1,x_2,x_3)$ be an imputation in the core. By (7.2)–(7.4), we have

$$x_1\geq \nu(\{P_1\})=0,\ x_2\geq \nu(\{P_2\})=0,\ x_3\geq \nu(\{P_3\})=0,$$

$$x_1+x_2\geq \nu(\{P_1,P_2\})=20,\ x_1+x_3\geq \nu(\{P_1,P_3\})=20,$$

$$x_2+x_3\geq \nu(\{P_2,P_3\})=20,\ \text{and } x_1+x_2+x_3=\nu(\{P_1,P_2,P_3\})=60.$$

Hence $0\leq x_3=60-(x_1+x_2)\leq 60-20\leq 40$. Similarly we get $0\leq x_1,x_2\leq 40$. Thus the core consists of all vectors $\mathbf{x}=(x_1,x_2,x_3)$ such that $0\leq x_1,x_2,x_3\leq 40$ with $x_1+x_2+x_3=60$. In particular, vectors like $(20,20,20)$ and $(0,20,40)$ are in the core.

Example 7. Consider the game in Example 1. We claim that its core must be empty. We show that by contradiction. That is we suppose that an imputation $\mathbf{x}$ is in the core. Then, we try to show that it is impossible.

Let $\mathbf{x}=(x_1,x_2,x_3)$ be an imputation in the core. By (7.4), we must have

$$x_1+x_2\geq \nu(\{P_1,P_2\})=1.$$

Since $\mathbf{x}$ is an imputation, by (7.2),

$$x_1+x_2+x_3=\nu(\{P_1,P_2,P_3\})=0.$$

Hence $x_3=-(x_1+x_2)\leq -1$. But by a similar argument, we find that $x_1\leq -1$ and $x_2\leq -1$. But this contradicts $x_1+x_2+x_3=0$. In other words, $\mathbf{x}$ cannot be an imputation. We conclude that there are no imputations in the core.

The result in Example 7 may be interpreted as meaning that the game is unstable. There are no imputations (i.e. distribution of payoffs) that everyone is satisfied with. Someone will always feel that he/she can gain more by giving up some and forming a new coalition.

In Example 7, consider the imputation $(1/2, 1/2, -1)$. You may think of it as a result of P_1 and P_2 forming a coalition playing against P_3. Then P_1 and P_2 split the payoff of HK\$1. However, this imputation is unstable in the sense that P_3 can certainly offer P_2 HK\$1 for forming a coalition with P_2 and playing against P_1. That corresponds to the imputation $(-1, 1, 0)$. This imputation clearly dominates $(1/2, 1/2, -1)$ for P_2 and P_3, so both players will prefer this, and hence form a new coalition of $\{P_2, P_3\}$. Of course, it doesn't stop P_1 in lowering his bid to lure P_2 or P_3 back into a coalition. Then the vicious cycle begins. In essence, if an imputation is not in the core, it is not stable. You shouldn't accept an offer that is not in the core. You should look for someone with better offer.

You can verify that Example 2 also has an empty core. Let's try an example with a non-empty core.

Example 8. Consider the game in Example 5. If $\mathbf{x} = (x_S, x_F, x_B)$ is an imputation, then by (7.2)

$$x_S + x_F + x_B = 1000. \tag{7.5}$$

If $\mathbf{x}$ is in the core, by (7.4), we must have

$$x_S + x_F \geq \nu(\{S, F\}) = 1000.$$

Hence

$$x_B = 1000 - (x_S + x_F) \leq 1000 - 1000 \leq 0.$$

Since $x_B \geq \nu(\{B\}) = 0$ by (7.3), we conclude that $x_B = 0$. Then by (7.5), $x_S + x_F = 1000$.

By the core condition (7.4) again, we also have

$$x_S + x_B \geq \nu(\{S, B\}) = 500.$$

Since $x_B = 0$, it follows that $x_S \geq 500$. On the other hand, $x_F \geq \nu(\{F\}) = 0$, and so $x_S = 1000 - x_F \leq 1000$. We conclude that an imputation in the core must be of the form

$$\mathbf{x} = (x_S, 1000 - x_S, 0) \quad \text{with } 500 \leq x_S \leq 1000.$$

We can give an interesting interpretation to this. Essentially, the farmer will buy the horse for a price somewhere between \$500 and \$1000. The price cannot be greater than \$1000 since the horse is only worth that much to him. On the other hand, the price must be at least \$500 since, if the farmer offers anything less than \$500, then the butcher can make a competing offer. Note that the butcher will get nothing from the game. Nevertheless, his presence is important in "bidding up" the price. In effect, the butcher gives the seller a choice of coalition partner while the farmer has no such alternative coalition to join. This is reflected in the seller's better bargaining position and consequent greater gain in the game.

Example 9. Consider Example 3. For an imputation $\mathbf{x} = (x_1, x_2, x_3)$ to be in the core, we need

$$\begin{aligned} x_1 &\geq \nu(\{P_1\}) &&= -\tfrac{1}{2}, \\ x_2 &\geq \nu(\{P_2\}) &&= 0, \\ x_3 &\geq \nu(\{P_3\}) &&= -\tfrac{1}{2}, \\ x_1 + x_2 &\geq \nu(\{P_1, P_2\}) &&= \tfrac{1}{4}, \\ x_1 + x_3 &\geq \nu(\{P_1, P_3\}) &&= 0, \\ x_2 + x_3 &\geq \nu(\{P_2, P_3\}) &&= \tfrac{1}{2}, \\ x_1 + x_2 + x_3 &= \nu(\{P_1, P_2, P_3\}) &&= 1. \end{aligned}$$

Thus we have

$$\begin{aligned} -\frac{1}{2} \leq x_1 = 1 - (x_2 + x_3) \leq 1 - \frac{1}{2} = \frac{1}{2}, \\ 0 \leq x_2 = 1 - (x_1 + x_3) \leq 1 - 0 = 1, \\ -\frac{1}{2} \leq x_3 = 1 - (x_1 + x_2) \leq 1 - \frac{1}{4} = \frac{3}{4}, \end{aligned}$$

i.e.

$$\begin{aligned} -\frac{1}{2} \leq x_1 \leq \frac{1}{2}, \\ 0 \leq x_2 \leq 1, \\ -\frac{1}{2} \leq x_3 \leq \frac{3}{4}. \end{aligned}$$

We see that the core is nonempty and there are many imputations inside the core. One example is $(1/3, 1/3, 1/3)$, i.e. everyone gets $1/3$, and everyone should be happy.

7.3 Cost Allocation

We end the chapter with another business application — cost allocation. In many business deals, we have to cooperate with others in order to achieve a common goal. However, different people or groups may contribute to the task in different ways so that they may have different bargaining power. Therefore it may be unfair to share the cost equally. How to allocate cost to each is the subject of this section. We begin with a simple example similar to Example 4.

Example 11. Albert, Bobbie, and Colin each has to buy a copy of Straffin's book on Game Theory. The list price is HK$110. There is a discount if you buy in batch: one book for HK$110, two for HK$160 and three for HK$250. Albert has a discount card that allows him to save extra HK$10 for each book he buys in batch. That is:

Number of Books	1	2	3
Without discount card	110	160	250
With discount card	100	140	220

Should they buy the book alone, or with some other(s), and what price should they pay? Obviously, we expect Albert to pay less. But how much less?

First we compute the price they have to pay if they buy alone or together:

$$P(\{A\}) = 100,\ P(\{B\}) = 110,\ P(\{C\}) = 110,$$

$$P(\{A,B\}) = 140,\ P(\{A,C\}) = 140,\ P(\{B,C\}) = 160,\ P(\{A,B,C\}) = 220.$$

From these, we can compute the saving ν each coalition has over buying alone. For example: $\nu(\{A,B\}) = P(\{A\}) + P(\{B\}) - P(\{A,B\})$. Then we have

$$\nu(\{A\}) = \nu(\{B\}) = \nu(\{C\}) = 0,$$

$$\nu(\{A,B\}) = 70,\ \nu(\{A,C\}) = 70,\ \nu(\{B,C\}) = 60,\ \nu(\{A,B,C\}) = 100.$$

Since the saving is largest for $\{A,B,C\}$. They should buy the book together. The next question is how to divide the saving of HK$100. For an imputation (i.e. saving in this case) $\mathbf{x} = (x_1, x_2, x_3)$ to be in the core, we need

$$\begin{aligned}
x_1 &\geq \nu(\{A\}) &&= 0,\\
x_2 &\geq \nu(\{B\}) &&= 0,\\
x_3 &\geq \nu(\{C\}) &&= 0,\\
x_1 + x_2 &\geq \nu(\{A,B\}) &&= 70,\\
x_1 + x_3 &\geq \nu(\{A,C\}) &&= 70,\\
x_2 + x_3 &\geq \nu(\{B,C\}) &&= 60,\\
x_1 + x_2 + x_3 &= \nu(\{A,B,C\}) &&= 100.
\end{aligned}$$

The fourth and the last inequalities will imply that $x_3 \leq 30$. Similarly, we get $x_2 \leq 30$ and $x_1 \leq 40$. But then the last equality implies that $x_1 = 40$, $x_2 = 30$, and $x_3 = 30$. Notice that this core has only one solution. Thus the savings each has over buying individually are: HK$40 for Albert, HK$30 for Bobbie and Colin. Recall that Albert can buy the book for HK$100, while the other can buy it for HK$110. Thus the prices they should pay for the book are: HK$60 for Albert, and HK$80 for Bobbie and Colin.

How to interpret the results? Obviously, Albert cannot ask either Bobbie or Colin to pay more than HK$80 each. Because Bobbie and Colin can buy the book themselves for

HK$160, i.e. HK$80 each. But can Bobbie and Colin threaten to pay less than HK$80 in a 3-person coalition? If Bobbie and Colin together ask Albert to pay HK$70 and they each pays HK$75, what should Albert do? Then Albert should counter-threaten Colin and Bobbie. But how?

The core solution indicates the way. If Albert pays HK$70 and Bobbie and Colin each pays HK$75, then the savings are $x_1 = 30$, $x_2 = 35$, and $x_3 = 35$. We see that this saving (imputaton) does not satisfy the fourth and the five inequalities above. This indicates that Albert can fight back and pays less by forming a 2-person coalition either with Bobbie or with Colin. Let's say Albert will use Bobbie to achieve his secret agenda. Then he will declare to Colin that he will buy the book with Bobbie only and let Bobbie pay only HK$70. That is x_1 will still be 30, but $x_2 = 40$. This pair of savings satisfies the fourth inequality. Colin will know that he will be paying full price for the book, and he has no option but to give in (i.e. rescind his request to ask Albert to pay HK$70). Once Colin gives in, it is time to fry Bobbie using the same trick. In essence, Albert is in a better bargaining position, because he can use the in-fighting between Bobbie and Colin to his own advantage. The nice thing about core is that it tells us how one should argue to his own advantage, and for how much.

We see from the examples that the core is a very useful solution concept for n-person games. In many cases it corresponds to the results of classical economic analysis. Unfortunately, it is very frequently empty, as can be seen from Example 7 above. Thus the core is not a totally satisfactory solution concept for n-person cooperative games.

Of course, the core may be the empty set of no imputations at all, and indeed it is the empty set for any essential constant-sum game. For an example of a three-person game with non-empty core, consider Game 23.1:

$$v(1) = v(2) = v(3) = 0$$
$$v(12) = \tfrac{1}{4} \qquad v(13) = \tfrac{1}{2} \qquad v(23) = \tfrac{3}{4}$$
$$v(123) = 1$$

An imputation (x_1, x_2, x_3) is in the core of this game if

$$x_1 + x_2 \geqslant v(12) = \tfrac{1}{4}$$
$$x_1 + x_3 \geqslant v(13) = \tfrac{1}{2}$$
$$x_2 + x_3 \geqslant v(23) = \tfrac{3}{4}.$$

For example, $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$ meets these conditions, so is in the core. To find all imputations in the core, we can use the imputation diagram, as in Figure 25.1. Notice that

$$x_1 + x_2 \geqslant \tfrac{1}{4} \text{ if and only if } x_3 \leqslant \tfrac{3}{4}$$
$$x_1 + x_3 \geqslant \tfrac{1}{2} \text{ if and only if } x_2 \leqslant \tfrac{1}{2}$$
$$x_2 + x_3 \geqslant \tfrac{3}{4} \text{ if and only if } x_1 \leqslant \tfrac{1}{4}.$$

When we graph these inequalities, they have a non-empty intersection in the imputation triangle. The core is a trapezoid with vertices at $(0, \frac{1}{4}, \frac{3}{4})$, $(0, \frac{1}{2}, \frac{1}{2})$,

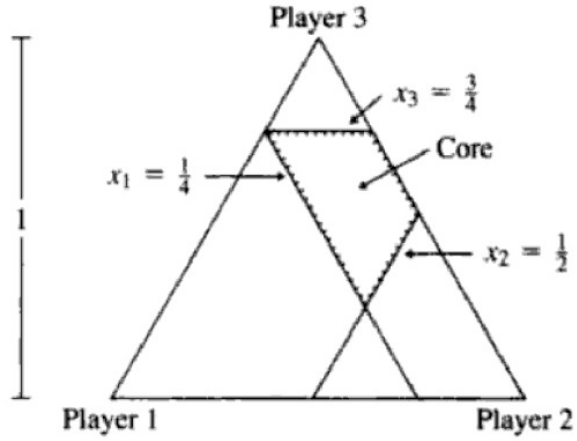


Figure 25.1: The core of Game 23.1.

$(\frac{1}{4}, 0, \frac{3}{4})$, and $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$. Any imputation in this trapezoid is not dominated by any other imputation in the triangle.

GAME 19.3: THE COMMUNICATIONS SATELLITE GAME [McDonald, 1977]. Western Union (W), Hughes Aircraft (H), and General Telephone (G) can put up individual communications satellites, or share satellites jointly in different combinations. McDonald calculates the values of the coalitions (in millions of dollars) as

$$v(\phi) = 0 \qquad v(W) = 3 \qquad v(H) = 2 \qquad v(G) = 1$$
$$v(WH) = 8 \qquad v(WG) = 6.5 \qquad v(HG) = 8.2 \qquad v(WGH) = 11.2$$

If we normalize this game, we get

$$v(G) = v(H) = v(W) = 0$$
$$v(GH) = 5.2 \qquad v(GW) = 2.5 \qquad v(HW) = 3$$
$$v(GHW) = 5.2$$

Game 25.2

When we draw the core (Figure 25.3), it turns out to be empty, but the core conditions are only slightly inconsistent. In this situation, William Lucas suggested to McDonald that a reasonable outcome would be (2.3, 2.8, .1), the point in the center of the little triangle where the core just barely isn't. This would correspond to GTE and Hughes sharing a satellite, but giving a small sidepayment to Western Union to keep it from trying to disrupt their coalition.

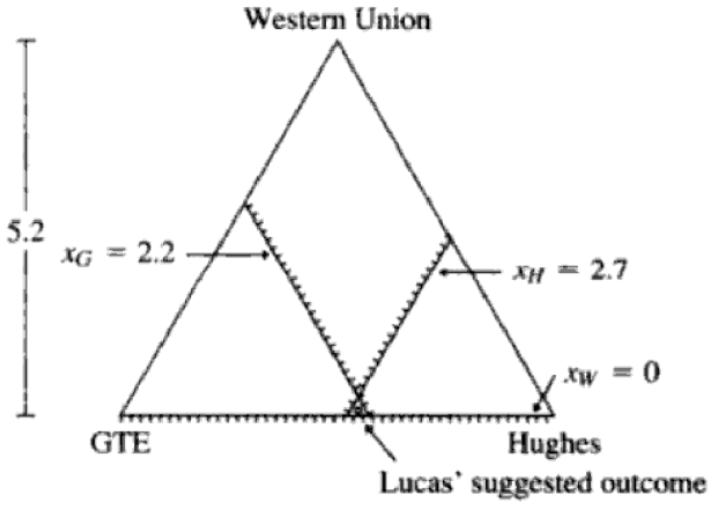


Figure 25.3: The communications satellite game 25.2 has an empty core.

S	$v(S)$
()	0
(1)	1
(2)	3
(12)	6

Figure 4.4: Example game. If the agents form coalition (12) then how much utility should each one get?

The Shapley Value

While the core gives us one possible solution, it suffers from the fact that many games don't have any solutions in the core and from its lack of guidance in fairly distributing the payments from a coalition to its members. The Shapley value solves these problems by giving us one specific set of payments for coalition members, which are deemed fair.

The problem with identifying fairness in characteristic form games is best illustrated by an example. Figure 4.4 shows a game for two players. Clearly, we should choose the coalition (12) as it has the highest value. Now we must decide how much each agent should get. The simplest solution is to divide the total of 6 evenly amongst the coalition members, so that each agent gets 3. This seems unfair to agent 2 because agent 2 could have gotten 3 by simply staying on its own coalition (2). It seems like the fair thing to do is to give each agent a payment that is proportional to the value it contributes to the coalition, that is, the amount that value increases by having the agent in the coalition. But, how do we extend this idea to cases with more than 2 agents?

Shapley was able to extend this idea by realizing that each agent should get a payment that corresponds to its marginal contribution to the final value. An agent's marginal contribution to a coalition is the difference between the value before the agent joins the coalition and after he joined. For example, if before you join Initech their annual profits are \$10M but after you are there for a year they increase to \$11M then you can claim that your marginal contribution is to Initech is \$1M assuming, of course, that everything else stays the same during that year.

The one remaining problem is that there are many different orderings in which n agents could have joined the coalition, namely, there are $n!$ orderings of n elements. The **Shapley value** simply averages over all possible orderings. That is, the Shapley value gives each agent a utility proportional to its average marginal contribution to every possible coalition, in every possible order it could have been formed. More formally, we define the Shapley value as:

Definition 4.4 (Shapley Value)**.** *Let $B(\pi, i)$ be the set of agents in the agent ordering π which appear before agent i. The* Shapley value *for agent i given A agents is given by*

$$\phi(A, i) = \frac{1}{A!} \sum_{\pi \in \Pi_A} v(B(\pi, i) \cup i) - v(B(\pi, i)),$$

where Π_A is the set of all possible orderings of the set A. Another way to express the same formula is

$$\phi(A, i) = \sum_{S \subseteq A} \frac{(|A| - |S|)!\,(|S| - i)!}{|A|!} [v(S) - v(S - \{i\})].$$

Notice that the Shapley values are calculated for a particular coalition A in the definition above. They are not meant as a way of determining which is the best coalition structure. They can only be used to distribute the payments of a coalition once it is formed.

Lets calculate the Shapley values for the game in figure 4.4 and the grand coalition (12). Since there are only two agents it means that there are only two possible orderings: (12) and (21). As such we have that

$$\begin{aligned}
\phi(\{1,2\},1) &= \frac{1}{2} \cdot (v(1) - v() + v(21) - v(2)) \\
&= \frac{1}{2} \cdot (1 - 0 + 6 - 3) = 2 \\
\phi(\{1,2\},2) &= \frac{1}{2} \cdot (v(12) - v(1) + v(2) - v()) \\
&= \frac{1}{2} \cdot (6 - 1 + 3 - 0) = 4
\end{aligned}$$

A somewhat surprising and extremely useful characteristic of the Shapley value is that it is always feasible. In our example the payments of 4 and 2 add up to 6 which is the same value we get in the grand coalition (12). Another nice feature of the Shapley value is that it always exists and is unique. Thus, we do not have to worry about coordination mechanism to choose among different payments. A final interesting result is that the Shapley value might not be in the core, even for cases where the core exists. This is a potential problem as it means that the resulting payments might not be stable and some agents might choose to leave the coalition in order to receive a higher payment on a different coalition.

Unfortunately, while the Shapley value has some very attractive theoretical properties, it does have some serious drawbacks when we try to use it for building multiagent systems. The biggest problem is computational. The Shapley value requires us to calculate at least $2^{|A|}$ orderings, this is only possible for very small sets A. It also requires that we know the value of v for every single subset S. In many real-world applications the calculation of v is complex. For example, it might require simulating how a particular coalition of agents would work together. These complex calculations could dramatically increase the total time. Finally, the Shapley value does not give us the actual coalition structure. Thus, it only solves the second part of the coalition formation problem. We must still determine which coalition the agents will form and how they will do it.

We will illustrate his method on the game

$$v(A) = v(B) = v(C) = 0$$
$$v(AB) = 2 \qquad v(AC) = 4 \qquad v(BC) = 6$$
$$v(ABC) = 7,$$

Game 26.1

The calculation is as follows:

	Value added by		
Order	A	B	C
ABC	0	2	5
ACB	0	3	4
BAC	2	0	5
BCA	1	0	6
CAB	4	3	0
CBA	1	6	0
	8	14	20

$$\varphi = \tfrac{1}{6}(8, 14, 20) = (1\tfrac{1}{3}, 2\tfrac{1}{3}, 3\tfrac{1}{3}).$$

For an example of how to arrive at the numbers in the table, consider the order BCA. The value added by each player is calculated as follows:

$$\text{B:} \quad v(\text{B}) - v(\phi) = 0 - 0 = 0$$
$$\text{C:} \quad v(\text{BC}) - v(\text{B}) = 6 - 0 = 6$$
$$\text{A:} \quad v(\text{ABC}) - v(\text{BC}) = 7 - 6 = 1.$$

We can think of the Shapley value of player i in a game as the average amount that player i contributes when the grand coalition forms, given that all orders of coalition formation are equally likely. It would seem fair to give player i this average contribution. Of course, the real argument for the Shapley value as a fair imputation is based on Shapley's theorem.

The calculation of φ based on orders is conceptually simple, but for larger games, it becomes infeasible, since the number $n!$ of orders grows very rapidly as n increases. For example, consider the 4-person constant-sum game

$$v(A) = v(B) = v(C) = v(D) = 0$$
$$v(AB) = 50 \qquad v(CD) = 70 \qquad v(AC) = 30$$
$$v(BD) = 90 \qquad v(AD) = 30 \qquad v(BC) = 90$$
$$v(ABC) = v(ABD) = v(ACD) = v(BCD) = v(ABCD) = 120.$$

Game 26.2

The Shapley value, calculated in Table 26.1, is

$$\varphi = \tfrac{1}{24}(440, 920, 760, 760) = (18\tfrac{1}{3}, 38\tfrac{1}{3}, 31\tfrac{2}{3}, 31\tfrac{2}{3}).$$

Notice how B's strong bargaining position and A's weak one are mirrored in the Shapley value.

	Value added by			
Order	A	B	C	D
ABCD	0	50	70	0
ABDC	0	50	0	70
ACBD	0	90	30	0
ACDB	0	0	30	90
ADBC	0	90	0	30
ADCB	0	0	90	30
BACD	50	0	70	0
BADC	50	0	0	70
BCAD	30	0	90	0
BCDA	0	0	90	30
BDAC	30	0	0	90
BDCA	0	0	30	90
CABD	30	90	0	0
CADB	30	0	0	90
CBAD	30	90	0	0
CBDA	0	90	0	30
CDAB	50	0	0	70
CDBA	0	50	0	70
DABC	30	90	0	0
DACB	30	0	90	0
DBAC	30	90	0	0
DBCA	0	90	30	0
DCAB	50	0	70	0
DCBA	0	50	70	0
	440	920	760	760

Table 26.1: Shapley value calculation for Game 26.2.

This calculation makes it clear that we would not want to write out the table for calculating the Shapley value for a 5-person game! Fortunately, the calculation can be simplified by focussing on an individual player and asking how often he contributes how much to the forming grand coalition. When player i joins the forming grand coalition, he and the players who have already joined make up some coalition S, of size s, which contains player i. The amount i contributes is $v(S) - v(S - i)$. Furthermore, this contribution occurs for exactly those orderings in which i is preceded by the $s - 1$ other players in S, and followed by the $n - s$ players not in S. The number of orderings in which this happens is $(s - 1)!\,(n - s)!$. Hence we get the following expression for the Shapley value of player i:

$$(26.3)\quad \varphi_i = \frac{1}{n!}\sum_{i\in S}(s-1)!(n-s)!\,[v(S) - v(S - i)] \qquad (s = \text{the size of } S)$$

The summation is over all coalitions S which contain i. There are 2^{n-1} such coalitions, which is considerably fewer than the $n!$ terms we would add in the table calculation. For example, the calculation of φ_A in Game 26.2 could be done as follows:

Coalition S	$(s-1)!(n-s)!$	$v(S) - v(S - i)$	Product
A	$1 \times 6 = 6$	$0 - 0 = 0$	0
AB	$1 \times 2 = 2$	$50 - 0 = 50$	100
AC	$1 \times 2 = 2$	$30 - 0 = 30$	60
AD	$1 \times 2 = 2$	$30 - 0 = 30$	60
ABC	$2 \times 1 = 2$	$120 - 90 = 30$	60
ABD	$2 \times 1 = 2$	$120 - 90 = 30$	60
ACD	$2 \times 1 = 2$	$120 - 70 = 50$	100
ABCD	$6 \times 1 = 6$	$120 - 120 = 0$	0
			440

$$\varphi_A = \tfrac{1}{24}440 = 18\tfrac{1}{3}.$$