Available online at www.sciencedirect.com

ScienceDirect

European Journal of Operational Research 176 (2007) 1299–1318

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

Invited Review

Dynamic programming and board games: A survey

David K. Smith *

Department of Mathematical Sciences, University of Exeter, North Park Road, Exeter EX4 4QE, UK

Received 9 August 2004; accepted 3 October 2005
Available online 6 December 2005

Abstract

In several of the earliest papers on dynamic programming (DP), reference was made to the possibility that the DP approach might be used to advise players on the optimal strategy for board games such as chess. Since these papers in the 1950s, there have been many attempts to develop such strategies, drawing on ideas from DP and other branches of mathematics. This paper presents a survey of those where a dynamic programming approach has been useful, or where such a formulation of the problem will allow further insight into the optimal mode of play.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Dynamic programming; Games; Leisure

1. Introduction

In the book, "Fights, Games and Debates" [34], the three areas of "conflict" are distinguished: Fights—where you harm your opponent; Debates—where you convince your opponent; Games—where you outwit your opponent. This paper considers the third, looking at games played on a board, possibly with one or more dice or with playing cards or with pencil and paper. Many examples are also available in electronic form. The theme of games means that there is no physical contact (as in a fight), or verbal skill (as in a debate), but logical skill, in outwitting the opponent or in successfully solving a puzzle set by "nature".

Such games have been studied, more or less seriously, for many years, and there is a wide-ranging literature about them, drawing on ideas from pure mathematics, logic and computer science. In some cases, such as computer chess programs, and the exact method of Little et al. [25] for the travelling salesman problem, study of a game or puzzle has contributed to a broader research area. This paper concentrates on the

* Tel.: +44 1392 264478; fax: +44 1392 264460.
E-mail address: d.k.smith@exeter.ac.uk

0377-2217/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2005.10.026

approach of dynamic programming (DP), and the contribution (direct or indirect) that DP has made to the investigation of the logical skills needed to solve a puzzle or defeat an opponent in a board game.

2. A structure for DP studies

There are numerous ways in which one might study the use of dynamic programming in "board" games. In what follows, the classification "board" is a generous one. It is intended as a way of distinguishing the games from sports and other physical games. In the paper, we consider a range of situations where the "game" is played by one person against another, or by one person against a "position" that has been set up by chance. Success is based on intellect, mental agility and logic, rather than fitness.

Throughout the paper, we use the traditional terms which are always used in dynamic programming formulations, including: stage, state, policy, decision, recurrence relation; and the expression: curse of dimensionality. Formal definitions will be found in most references to dynamic programming.

Informally, in the general context of board games, the stage is a measure of how many decisions or "moves" remain for a player. In some of the games considered, there are potentially an infinite number of stages remaining, and in such circumstances, the functional recurrence of DP is independent of the stage. The state throughout the paper is the position of the playing pieces (pieces includes playing cards or similar) on the board in front of the player. The decision is what to do next; this will mean an action (usually a move) which changes the state of the board. The policy and recurrence relation imply a value which may be associated with the state and stage. The dimensionality depends on the game, and on how one chooses to describe the state.

The paper is structured to examine games classified in according to the number of players (one or two) and whether or not there is randomness. Sections 3 and 4 consider games played by one person. Section 3 focuses on deterministic games, Section 4 on those where a move has a stochastic element. Then the paper examines two player games, again with a focus on deterministic games (Section 5) and stochastic games (Section 6). Each section is reasonably self-contained.

3. One player, deterministic games

We start with the simplest of all board games. There is one player, who has complete knowledge of the state of the board, and about the consequence of any of his (or her) decisions. Each move produces a known change in the state. (From now on the players will be considered as male.)

Games in this category include:

- Solving the Towers of Hanoi from a given position.
- Unscrambling a Rubik's cube.
- Rearranging the 15-puzzle.
- Solving Freecell (game included in Windows).
- Solving some forms of the family of card games, Patience.
- Solitaire (the board game) problems.

The first common feature of these games is that the player is trying to achieve a given target, and wants to make moves which will achieve that goal. In the towers of Hanoi, the aim is to pile the discs into a specified order. Given a scrambled Rubik's cube, one wants to be able to restore it correctly. With a scrambled 15-puzzle, one wants to achieve the sequential pattern. In the card games, the aim is to collect the cards into particular "stacks".

The second common feature is that the starting position is not chosen by the player; he is competing against "nature", in the sense that the player does not have control over the position.

There are two particular problems with such games. First, is there a solution? Second, how can the solution be reached as quickly as possible? Dynamic programming offers scope for each one.

For the problem of identifying if a given position can be solved or not, a dynamic programming approach will define an objective function valued at +1 if so, and 0, or −1 if not. This is of the form known as "final value" problems. Then, theoretically, any position can be evaluated by a recurrence relationship such as

$$f(p) = \max_{m \in M(p)} (f(T(p,m))). \tag{1}$$

p is the current position, and $f(p)$ is the value of the position. The player's moves (m) form the set $M(p)$, and a move changes the position to $T(p,m)$. The target position, "out", has value +1, and positions which are known to be impossible to solve have value 0 (or −1).

Unfortunately, this may not always work, because one may not be able to identify a sufficiently large set of positions whose outcome is known. For Rubik's cube and the 15-puzzle, group theory allows the identification of such positions, but when one has used that approach, the dynamic programming relation is more or less redundant. Generally this recurrence relation is at the heart of studies of game trees in computer science and it is associated with ideas of decision trees in Operational Research. The curse of dimensionality may also strike. The work of Allis et al. [4] and others in Artificial Intelligence has shown that it is possible to work with extremely large databases of game positions.

3.1. Dynamic programming applied to "Freecell"

Freecell is a form of card patience that is included with most versions of the Microsoft Windows operating system. The 52 cards of a standard pack are arranged face-up, in 8 columns, with 6 or 7 cards in each. The exposed card at the foot of a column may be moved and placed on another exposed card, whose value is one higher, and whose colour is the opposite. If a column is empty, any card may be moved there. Four cells (freecells) are available to store cards temporarily, allowing several cards forming an ascending sequence to be moved between columns. There are also four stacks for the suits. The aim is to move the four suits into these stacks with the ace at the bottom, in ascending order. Once moved to a stack, a card cannot be returned to play. The position of every card is known, removing any randomness from the game once the deck has been dealt. However the state space is huge, even allowing for the interchangeability of the suits and columns of cards.

When Microsoft launched this game in the Windows 95 operating system, a group of game players started the "Freecell project" [5] to solve every one of the 32,000 games or to show that a game cannot be solved. They found, and proved exhaustively, that there is only one impossible starting position in the set provided by Microsoft. (This is game 11,982. Details are at [5].) Other impossible initial positions can be generated. The systematic proof of impossibility is based on the basic recurrence relationship (Eq. (1)). However, the dimensionality of the problem meant that while the recurrence relationship of DP could be used, evaluation of the objective function for all states was too time-consuming. The analysis was carried out by a forward-looking algorithm in the game tree, using restricted dynamic programming. A computer program using such an algorithm is available from [16].

3.2. Other objectives

If one knows that a position is soluble, then it becomes sensible to start thinking of the best way to solve it. In some cases, this may be the way which minimizes the number of moves that are needed, however the word "move" is defined. This has led to a different kind of dynamic programming recurrence, as illustrated

$$f(p) = 1 + \min_{m \in M(p)} (f(T(p, m))), \tag{2}$$

where the notation for p, m, $M(p)$, T is as in Eq. (1), and $f(p)$ is now the number of moves from the position to the solution under an optimal policy. Such an approach has been considered in various contexts. There have been competitions to solve scrambled Rubik's cubes as quickly as possible, including the "World Championships" in 2003 [47]. For over a century, recreational mathematicians have searched for minimal move solutions in the board game Peg Solitaire ([12, Chapter 24], [7] with computational analysis in [45]). There are assorted decision problems and objectives with Freecell [5]. (e.g. Can a position be solved using fewer than 4 freecells? What is the minimum number of cards that must be moved to the suit stacks?) Once again, dimensionality is a serious problem. In most such situations, forward dynamic programming is immensely superior to backward dynamic programming since it restricts the states to a subset of the vast number which may occur in a general game. (Instead of evaluating every terminal state of the game, and then working backwards towards the initial state, a forward approach considers the states which can be reached from the initial state, then those which can be reached from these, and so on, and only evaluates the terminal states which can be reached. There is a very close relationship between the dynamic programming concept of forward analysis and the computer scientist's analysis of a game tree.)

4. One player, stochastic games

The next level of complication is when the single player's move transforms the state or position in a random way. Randomness enters through the use of one or more dice or playing cards, such as in games of patience with hidden cards. There are still at least two possible objective functions, mirroring the deterministic case. First, one may want to maximize the chance of solving the game. Second, one may want to minimize the expected number of moves that are needed to solve it. These objectives are illustrated in Eqs. (3) and (4)

$$f(p) = \max_{m \in M(p)} \sum_q \pi^m_{pq} f(q), \tag{3}$$

where p is the current position, and $f(p)$ is here the *probability* of moving from position p to the solution under an optimal policy. The player's moves again form the set $M(p)$ and corresponding to each m in $M(p)$, there is a transformation probability π^m_{pq}. Similarly, one has the recurrence

$$f(p) = 1 + \min_{m \in M(p)} \sum_q \pi^m_{pq} f(q), \tag{4}$$

where p is the current position, and $f(p)$ is here the *expected number* of moves from the position to the solution under an optimal policy. The notation is as before.

4.1. An example: Klondike patience (Solitaire)

A simple example of the sort of game where this applies is in Fig. 1. This shows the opening position in the form of patience supplied with the Windows software, which is generally known as Klondike patience. (The family of card games known in the UK as "patience" are referred to as "solitaire" in North America, so Klondike patience is described as "Solitaire" by Microsoft. The American term is slightly misleading, as there are some two-player patience games.) The cards have been dealt into seven columns, with one exposed card in each and from none to six cards hidden underneath. The remaining cards in the deck are turned over (revealed) either singly or in threes, depending on the version being played, and the aim is to build descending series in alternate colours in each column, and ascending series in suits in the four spaces for

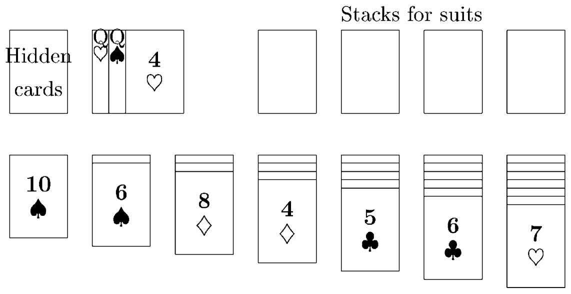

Fig. 1. The state of a game of 'Solitaire' (Klondike Patience). What is the best move?

the stacks. The (single) player may only have one move available to him, or there may be several. As one or more of the cards are hidden, then the outcome is stochastic. In the position pictured, the player has a choice of moves; either the ♡ 4 or ♢ 4 can be moved to the ♣ 5; either the ♣ 6 or the ♠ 6 can be moved to the ♡ 7. Heuristics which aim to maximise the expected number of different cards seen in the future, a surrogate measure related to the probability of winning, dictate that the ♡ 4 and ♣ 6 be moved.

The curse of dimensionality applies for the state description, and most wise players use heuristics. (Recent versions of the software, e.g. Windows XP, incorporate an option to automatically move all permitted cards into the suit stacks. This greedy heuristic can be shown to be non-optimal.) In spite of this reservation about the dimensionality, in principle this is a board game which is amenable to a dynamic programming approach. The only accessible analysis of Klondike is by a US undergraduate (Rabb [33], quoted by Kuykendall [24]) who programmed a computer with some heuristic rules to play the game, and these allowed about 8% of the randomly generated games to be completed. These rules do not directly evaluate the position in terms of a winning probability, but use an objective function which calculates a score which is correlated with that probability, and then apply DP to this objective. Playing the same games by hand, Rabb reported a success rate of 15%. Aldous and Diaconis [1] report an estimated success rate of 37% when playing Klondike with perfect information. It is likely that Rabb's computer program could be improved (using more detailed strategies and scoring), but there are situations where the human brain is superior in grasping a multi-step strategy.

4.2. Dropping eggs

A puzzle which has appeared in several collections is as follows: freshly laid eggs break if they are dropped from a window on the Cth floor of a skyscraper, or higher. They do not break if they are dropped from any floor below the Cth. How can you measure C (known to be less than or equal to some limit H) given N eggs with as few experiments as possible?

This problem appears trivial, and is easy to solve with pencil and paper for $N = 1$ and $N = 2$. Sneidovich [41] presents a dynamic programming model for all values of N, which has useful parallels for optimal play in some board games. The functional equation for the minimum number of trials needed to find C in the range $1 \leqslant C \leqslant H$ is presented as

$$f(n,k) = 1 + \min_{x\in(1,2,\dots,k)}(\max(f(n-1,x-1), f(n,k-x))) \quad n = 1,\dots,N;\; k = 2,3,4,\dots,H, \tag{5}$$

where there are n eggs available, and the analysis to date shows that there are still k possible values of C, observing that by definition $f(n,1) = 1$. x represents the choice of which height in the range of possible values to drop the next egg from.

4.3. The game of "Flip"

Trick [44] describes a one player dice game, "Flip" and its analysis by dynamic programming. The paper is concerned with both the analysis of how to play the game and with the possibility of modifying the rules to make the game more appealing to players. In this way, it became a case study for MBA students to see what makes a game attractive, while there is sufficient complexity in the game that the solution is not trivial. The rules of "Flip" are simple, requiring a set of twelve cards or markers numbered from 1 to 12, and two dice. Initially, all 12 markers show their numbers. In his turn, the player rolls the dice and must then choose one or more of the visible markers whose sum equals the total shown on the two dice. These markers are turned face down and become unavailable for future turns of the game. Play continues until either all the markers are face down (a win for the player) or there is no combination of face-up values that equal the roll (a loss).

In the dynamic programming formulation, there are $2^{12} = 4096$ possible states, corresponding to the possible positions of the 12 markers. A decision is made when the value of the total of the dice is known. There may be several ways of choosing markers to move; a total of 5 could be made up of (5), $(1, 4)$, $(2, 3)$. The player selects from the possibilities and this decision changes the state of the set of markers. The objective is to maximise the probability of completing the game. The paper shows that in its basic form, the optimal strategy gives a player a 1 in 276 chance of a win. Such a low success rate is inherently unappealing. Accordingly, three variants of the rules are introduced, all of which are analysed by DP in the paper. One, that a player is allowed a quota of "failing" rolls (i.e. rolls which cannot be matched by the markers). Two, the player is permitted to "pass" and claim a "failure", even when there is a roll which can be matched by the markers. The third option converted the game to "Flip–Flop"; each of the markers was numbered on each side. Initially, all the markers showed the same (light) numbered side, and the objective was to reach a state in which all showed the same (dark) numbered side. At any stage, markers could be flipped from light to dark or flopped back from dark to light (hence the name). So markers could be used one time, three times, five times,... (As an afterword, Trick suggests that the game has two-player variants which are amenable to dynamic programming.)

4.4. Farmer Klaus and the Mouse and Obstgarten

According to Campbell [14], the board game "Farmer Klaus and the Mouse" is one of the most popular board games in some community libraries in Germany. It is a cooperative dice game for children aged 3 and up, in which the aim is to reach a "winning" position rather than a "losing" one. A related game is "Obstgarten" (orchard). Both games are for one or more players; having two or more participants does not affect a strategy, but makes the game more "fun". At the start of the game, there are 4 sets of 6 pieces representing sacks of grain on the board, and six mice off the board. The cubical die used has five faces marked with mice or types of grain, and the sixth is "wild" and can be used for any sort of grain. Throwing the die moves the corresponding item off (grain) or on (mice), if possible. The player wins if the board is clear of grain before all six mice have appeared. The only decision is which grain should be chosen when the die shows the "wild" face. (Obstgarten is similar; there are 4 sets of 10 baskets of fruit, and a 9-piece jigsaw of a hungry bird to be assembled. A player may take two baskets off the board at any throw of the die.)

The optimal strategy for these games can be seen to be common sense, taking one sack of the most numerous type of grain. (The worst strategy takes one of the least numerous.) Campbell's paper shows how the dynamic programming recurrence evaluates the probability of success using both the best and the worst policies (50% and 40% respectively for Farmer Klaus, 68% and 53% for Obstgarten). Since these figures ensure that the result is not a foregone conclusion, one wonders whether the game designers used any mathematical analysis in their work.

4.5. Tetris

The computer game *Tetris* was invented by the mathematician Alexey Pazhitnov in the 1980s. It rapidly became a best-seller, and it is estimated that over 50 million copies have been sold, in addition to the variants which are available free. In the game, a player is presented with a sequence of tetromino pieces which must be rotated and/or moved sideways before dropping into a rectangular gameboard whose cells may be empty or occupied. Any completed row of this gameboard is cleared and the pieces (now regarded as individual filled cells) above it drop by one row. Points are scored by the action of clearing one or more rows of the gameboard by appropriate placement of the tetromino. Kostreva and Hartman [23] have used a dynamic programming formulation to analyse best play in the game. They stress that there are several possible objectives for the player when presented by a fresh tetromino to be placed on the board. Their objective is a weighted sum of four:

(i) minimise the number of empty cells on the gameboard underneath the piece being played;
(ii) maximise the number of *contacting cell walls* when the piece has been played;
(iii) minimise the maximum height of the boundary between the filled and empty cells;
(iv) maximise the number of rows that are completed by placing the tetromino,

where the weights are chosen in value and sign to create an objective which is maximised. The authors note that the values of the weights may depend on the state, but this option is ignored in their implementation.

4.6. Television gameshows

Recent television gameshows, in which contestants face a succession of choices or questions, have inspired some studies using dynamic programming.

"The Weakest Link" involves each player in a choice about whether or not to "bank" the current amount of prize money. The contestant then answers a general knowledge question; a correct answer adds to the prize money, a wrong answer clears any prize money that is not in the bank. Thomas [43] presents a dynamic programming model for when to bank prizes, based on an assumed probability of getting answers right, and an upper limit on the amount that may be in the bank in one round.

Although there is less strategy in the game, Rump [36] has described some features of "Who wants to be a Millionaire?" as dynamic programming. The paper offers several ideas for teachers relating to operational research concepts.

The public broadcasting network of US television in the 1980s included a programme *Square One*, which was orientated to teaching mathematical ideas. A part of this was a mock gameshow, *But Who's Counting*; competing teams were presented with a digit uniformly distributed in the range 0–9. They had to choose where to place this digit in a five-digit number. After five rounds each team's number was complete and the team with the largest number was declared the winner. Ignoring the competitive setting, each team faced a similar decision problem; how to maximise the five-digit number obtained, using an optimal strategy. This is a variation of the secretary problem [39] or sequential assignment problem and a solution by dynamic programming is presented by Puterman [32]. In this the decision where to place the digit depends on the number of unoccupied positions remaining, and not on the position of these in the number.

5. Two players: Deterministic games

When two people are playing against each other, one enters an area where pure mathematicians have made serious strides in analysis.

5.1. Combinatorial games—A formal definition

Guy [18] defines combinatorial games as having the following properties:

(i) There are just two players, who are given distinct names, such as Left and Right, or Black and White. (So: no coalitions.)
(ii) The game has a well-defined set of positions, which is usually a finite set. Generally, there is a starting position.
(iii) There are rules which define the two sets of moves that Left and Right can make from each position to new positions.
(iv) Players move alternately and there are no chance or random moves.
(v) A player who cannot move is the loser.
(vi) Games always end because some player cannot move.
(vii) Both players know the state of the game; this is complete information. (So one player cannot bluff about the position.)

Within the games that are so defined, a subset of particular interest is the set of *impartial* games, which are those where the two players have the same possible moves. Games with Black and White pieces, such as Chess and Draughts (Checkers) are therefore *partisan*, i.e. not impartial.

Guy's definition means that many familiar games are not strictly combinatorial games, since one or more of the conditions listed above will be violated. For instance

Game	Violates
Ludo; Snakes and Ladders; Backgammon	Chance moves
Scissors-Paper-Stone	Incomplete information; players move simultaneously
Battleships	Incomplete information
Bridge (viewing N–S and E–W as two players)	Incomplete information (players lack details of their own cards)
Noughts and Crosses	Can be tied
Chess	Partisan and it can be tied (e.g. by stalemate)
Mastermind	Asymmetric, incomplete, and—strictly speaking—a one-player game once a position has been established
Monopoly	Chance moves; asymmetric

The mathematics community recognizes a variant, the set of *misère* games, in which the last player to play is the loser, not the winner. In most cases, the misère versions of games are harder to analyze than the normal ones.

Since it uses a description of the current state of the game, and since the objective can be defined in terms of the final state and objective, the dynamic programming approach can be used to study a large range of kinds of games including those which are partisan, as well as normal and misère variants of games.

5.2. Nim and other related games

The starting point for much of the analysis of such games is in the game of Nim. The rules are:

(i) Start with one or more heaps of beans.

(ii) The player who has a move selects one of the heaps and removes as many beans as he/she likes from it (but must take at least one bean); repeat this step until there are no beans left—the last player to have taken any beans is the winner.

There is a family of "take-away" games which is derived from the idea of Nim. They are in the family group of impartial games described earlier. Some limit the number of beans that may be taken away from a heap, or state that the beans must be taken from more than one heap, or that the beans taken away must be put into a new heap. Comprehensive discussions of such games are given by Schuh [38, Chapter 6] and Berlekamp et al. [12, Chapters 14 and 15]. Smith [40] describes the dynamic programming approach to one such game, that devised by Dudeney.

From a single heap [of points or tokens, such as the matchsticks used in Nim] either player may subtract any number from 1 to Y, except that the immediately previous deduction may not be repeated, and you win if you can always move but at some stage your opponent cannot.

Dudeney published the game with a heap of 37, and $Y = 5$, which is the smallest value of Y to give interesting (and counter-intuitive) results.

5.3. Bellman's contribution to game playing

There are several references in Richard Bellman's work to his ideas about using dynamic programming for playing games in an optimal way. In one of his early papers [11], he looked at a board game, and the best way to play it. He assumed that it would be possible to know exactly the value of every position that might occur in the course of a game. He defined a function $f(p)$ as the value of a game for the player (White) about to move with the game in position p. $f(p) = 1$ if p is a winning position, $f(p) = -1$ if it is a losing position and $f(p) = 0$ if a draw is the best result.

Bellman presented a recurrence relationship for White; White has a set of permitted moves available, and in response to each of these, Black has a set of responses. Black wishes to place White in a losing position, so will try and respond so as to present White with a position whose value is as small as possible. White, knowing this, examines the worst response that Black can make to each of his (White's) moves, and tries to protect himself from a response which leads to a draw or loss if Black plays optimally. This leads to the recurrence:

$$f(p) = \max_{w \in W(p)} \min_{b \in B(p') | p' = T_w(p,w)} f(T_b(p', b)), \tag{6}$$

where $W(p)$ is the set of moves available to White when the game is in position p. Similarly $B(p')$ is the set of moves available to Black in position p'. The position generated by White making move w in position p is $p' = T_w(p, w)$ and Black's move b at position p' generates position $T_b(p', b)$.

Bellman used chess as an example, and commented:

As it stands, however, the equation is useless computationally because of the dimensionality barrier, and it is useless analytically because of our lack of knowledge of the intrinsic structure of chess. (Bellman in [11])

Bellman's expression of a recurrence relation for a board game between two players has been used, in one form or another, in the three disciplines of pure mathematics, computer science and dynamic programming.

5.4. Two ways to solve Nim

5.4.1. By dynamic programming

Consider a game of Nim. The *state* of the game is fully described by the size of the heaps of beans on the board. So, with Bellman's notation, a state near the end of a game might be

$p = \{1, 2, 2\}$.

White's move could transform this to one of

$$T_{W1}(p) = \{2, 2\}, \quad T_{W2}(p) = \{1, 1, 2\}, \quad T_{W3}(p) = \{1, 2\}$$

and then Black's response would transform these states further, giving

From: $\{2, 2\}$ to: $\{1, 2\}$ $\{2\}$,
From: $\{1, 1, 2\}$ to: $\{1, 1\}$ $\{1, 2\}$ $\{1, 1, 1\}$,
From: $\{1, 2\}$ to: $\{1, 1\}$ $\{2\}$.

As it stands, one does not know the value of any of the three positions that White's move would lead to. Therefore, in order to apply dynamic programming successfully, it is necessary to give values for $f(p)$ for terminal or boundary conditions.

One can observe that for any state where White has only one heap, then that must be a winning position. White can take all the beans, and Black cannot move. So, for any integer $n \geqslant 1, f(\{n\}) = +1$. Then using the recurrence relationship, it is possible to evaluate positions with two heaps, such as those mentioned above. One can then construct the game tree shown in Fig. 2 in which values of the objective for all of White's positions are also given.

5.4.2. By binary arithmetic

The more common approach to analyzing Nim uses the observation that any position in Nim is exactly equivalent to the same position with a common number of pieces taken away from two heaps. (Ignoring consideration of optimality, if White takes away s pieces from one heap, and Black takes away s pieces from another heap, neither player can have gained any advantage.) This means that the value of the position $p = \{1, 2, 2\}$ is the same as the value of the positions $p_a = \{1, 2\}$ and $p_b = \{1, 1, 1\}$ (taking away 1 from

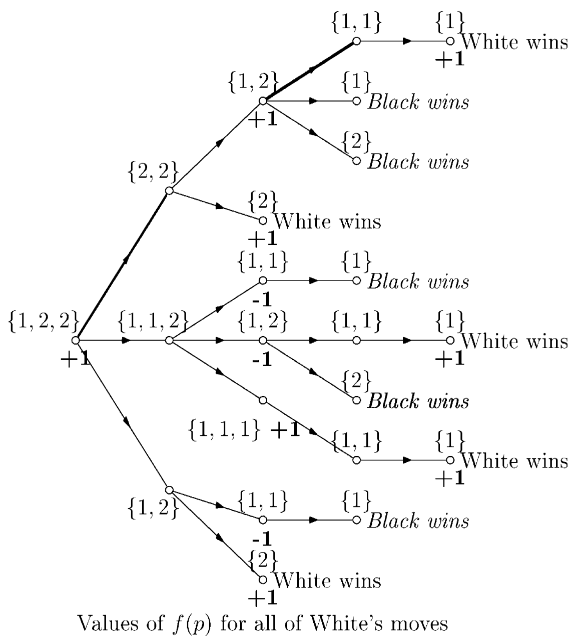

Fig. 2. The values of the tree in for a game of Nim.

two of the heaps) or $p_c = \{1\}$ (taking away 2 from two heaps). This obviously simplifies the analysis a great deal, and it gives rise to the simple rule for finding what is a winning move to make:

> Express the number of counters in each heap as a binary number, and add the resulting binary numbers together, ignoring any carrying between columns. When you move, move to a position in which all the "columns" have even numbers in them.

(This result was first published by Bouton [13], although it is not certain who discovered the result.)

There are associated strategy rules for other variants of Nim. These are discussed in several places, with Moore's paper [27] the earliest, and a recent paper by Chlond and Akyol [15] gives an overview from an operational research perspective, with a short bibliography. In all of this work one can trace a strong link to the ideas of Bellman, in the need to establish the value of a position as simply as possible. For Nim, it is possible to formulate a dynamic programming state description which uses the reduction rule to simplify the states, but the binary approach provides the optimal strategy without recourse to dynamic programming.

5.5. *Games of position*

Bell and Cornelius [9] and other writers describe some of the most familiar pencil and paper games under the heading "Games of position". It is likely that some of them have been around for 3000 years or so. Within the group there is a sub-division into two-player games: three-in-a-row games, five-in-a-row games and blocking games; and single player games such as solitaire (the English or French board game played with pegs/balls on a board).

5.6. *Three-in-a-row games: Noughts and Crosses and the Morris family*

Noughts and Crosses (Tic-tac-toe) is one of the simplest games to describe and analyze, and the optimal strategy leads to a draw. It can be analyzed using Bellman's recurrence relation. This game has the distinction of being the first interactive computer game, in 1949. The first software program designed to play Noughts and Crosses was written by Douglas as part of his Ph.D. dissertation on Human–Computer interaction. The computer was the EDSAC machine built at Cambridge University (UK).

Winning Ways [12] shows some ways in which the game can be successfully disguised, such as:

- Players take turns to select a number from 1 to 9 inclusive.
- Once a number has been chosen, that player "keeps" it.
- The winner is the first player to have three numbers whose sum is 15.

or

- There are nine cards, labelled: SPIT, NOT, SO, FAT, FOP, AS, IF, IN, PAN.
- Players take turns to take a card.
- The winner is the first player to have collected three cards whose words share a common letter.

Ovid, in his book *Ars Amatoria* written about 1 BC, suggested games for young ladies to play to entertain their lovers. Ovid's game is another name for the simplest of the Morris family of games. It is played on a 3×3 grid, and players have 3 stones each. They play in turn, until either all the stones are on the grid, or one player has three stones in a row (vertically, horizontally or diagonally). Once all the stones are down, play continues by players moving one of their stones along the grid lines to an adjacent cell, until a position is reached in which one player is unable to move. Again, Bellman's relation becomes useful; the first player can win by choosing the central square, and the game is drawn—though there are many traps—if the player starts anywhere else.

5.7. Games which have been solved by DP and total enumeration

“Connect-4” (or the “Captain’s Mistress”) has been solved using a game tree and total enumeration [2]. The game tree uses the basic DP relationship, in the same way as the example of Nim above. Connect-4 is a two player game which uses a rectangular board (usually 7 columns and 6 rows) placed vertically between the players. The players have equal number of pieces (21 pieces for a 7×6 board). Each player can drop a piece (usually a round disc) at the top of the board in one of the columns; the piece falls down and fills the lowest unoccupied space. A player cannot drop a piece into a full column. The object of the game is to connect four pieces vertically, horizontally or diagonally. If the board is filled and no one has aligned four pieces then the game ends in a draw.

Several other games have also been solved by total enumeration, an approach which has only become possible with the advent of the electronic computer. These include Qubic, Go-Moku and Nine Men’s Morris [17]. A list of “solved” games up to the early 1990s is given by Allis in his Ph.D. thesis [3]. The term “total enumeration” is a slight exaggeration, as many of the searches use directed enumeration, in which only certain positions are examined. This technique, known as “proof-number search” (or pn–search) uses a forward dynamic programming methodology to select branches in a game tree which are to be examined [4].

Total enumeration is covered thoroughly by Nievergelt et al. [29] who point out that this approach to problem solving now means that chess-playing computers can outperform all but the best few hundred players in the world.

5.8. Endgames

Endgames are of especial interest for many researchers. These are the positions when the number of pieces that the players possess has become small, so that the number of states is relatively small and the curse of dimensionality is not so serious.

Work is in progress for solving endgames in draughts/checkers. All 34,778,882,769,216 endgames in checkers with ten pieces or less have been evaluated [37].

In chess, several endgame positions have been exhaustively explored. To the outsider, these may seem very trivial (typically, 2 kings, and one or two pawns on each side). There are a huge number, millions or billions, of possible states.

5.9. Awari

The African game of Awari[1] has become very popular in the Artificial Intelligence community, because it has simple rules, a limited range of choice of moves, but the number of possible positions is considerable, depending on the choice of board in use. Irving et al. [19] describe a tree-search algorithm based on forward dynamic programming for Kalah, a slightly simplified form of Awari. The authors use many recent ideas from artificial intelligence to optimize the search for a solution. Rules for Awari in the form used by the AI community are given by Romein and Bal [35]; this paper records the complete analysis of the 848 billion

[1] There are numerous other names for the game, including adi, congkak, mokaotan, maggaleceng, aggalacang, nogarata, owari, mancala, awele, oware, bantumi, bao, gebeta, omweso, wari, congklak, warri, round-and-round, adji-boto, abapa, gabata, nam-nam, tampoduo, omweso, marabout, mandoli, ba-awa, uri, tampoudo, ayo, ayoayo, jerin-jerin, ayo-j’duo, eson-xorgol, toguz-xorgol, pallanguli, and sungka. In addition, there are variant rules.

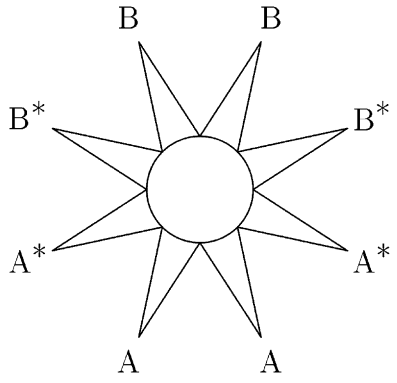

Fig. 3. The board for the Maori game *Mu Torere*.

states of the game, using a parallel-processor computer, and an algorithm which is based on dynamic programming recursion. Under optimal play Awari is drawn.

A computer version has been implemented on some Nokia mobile telephones as "Bantumi".

5.10. Blocking games: Mu Torere

There are many games whose objective is to position pieces on a board so that the opponent cannot move. One of the simplest of these is a Maori game called Mu Torere, shown in Fig. 3. It has the following rules:

- The two players each have four distinct markers.
- The game is played on an eight-pointed star design.
- Markers may be placed on the points or the centre.
- Players move one marker at a time, alternately, moving either to the centre (if empty) or to an adjacent empty point.
- For the first two moves, only the 'outer' markers may be moved. (Those with an asterisk.)
- The game is won when one player is blocked and cannot move.

The game has been analyzed by Ascher [6] who has made a specialist study of the way that various cultures use mathematics. Her analysis does not expressly refer to dynamic programming, but uses the principles of recursion and the analysis of states. Ascher shows that under an optimal strategy for the players, the game continues indefinitely, but points out that if either player deviates from this strategy, then the opponent can generally find a winning strategy. (In this, the game is like Nim, in that a deviation from optimality by one player can immediately be turned to the opponent's advantage.)

There is no interest in using boards with different numbers of points. Less than eight leads to a trivial game, and more than eight generates a situation where quick wins are possible.

5.11. A strategy game between two players

The following game was presented as a puzzle for readers of the American Mathematical Monthly. It is unclear whether this was an original problem.

At the start of the game, the dealer (Black) places $n + 1$ green balls and n red balls into a bowl. The balls are to be drawn one at a time from the bowl without replacement. The game ends when the bowl is empty.

The gambler begins the game with a bankroll of one unit of (infinitely divisible) money, the dealer has as much money as necessary. Before each ball is drawn, the gambler (White) declares how much he bets; he may choose to bet any amount from 0 up to his entire bankroll at that point. After the gambler declares the size of his wager, the dealer chooses a ball from the bowl (not necessarily at random). If a green ball is drawn, the gambler wins an amount equal to his bet; if a red ball is drawn, he loses his bet. The gambler seeks to maximize his bankroll at the end of the game, while the dealer seeks to minimize the gambler's final bankroll. (Since the players have deterministic choice, the final bankrolls are deterministic.) What is the gambler's final bankroll, assuming optimal play by both gambler and dealer? [31].

To solve this problem, we consider the more general game where the gambler starts with 1 unit and the dealer has m green balls and n red balls. Let $f(m,n)$ denote the optimal final bankroll in this game. If this game started with x units then the optimal final bankroll would be $xf(m,n)$. There will be exactly $m+n$ plays of the game, so the stage can be indexed by the number of balls. The state is defined by the amount of money held by the gambler, and the number of each type of ball. The decision is how much to gamble, given the state, and this describes the policy. Since the outcome is measured in terms of return, the decision will be relative to the money currently held, and so $f(m,n)$ may be used to measure the policy.

The dealer's choice is simple: to play red or green. Assuming that the gambler wishes to maximize his final bankroll, the recurrence relation will be expressed using the outcomes for each decision of the dealer

$$f(m,n) = \max_{0\leqslant x\leqslant 1}(\min((1-x)f(m,n-1),(1+x)f(m-1,n))) \tag{7}$$

with boundary condition: $f(0,0)=1$. Since the gambler makes a choice before the dealer, then the gambler should choose x so that the two expected outcomes have the same value.

It will be apparent that this recurrence is a special case of the two-player recurrence, identified by Bellman, in Eq. (6). White's set of moves, is the set of all real numbers $\{0\leqslant x\leqslant 1\}$, and Black has a choice of a red or green ball. When all the balls are the same colour, then further boundary conditions follow:

$$f(0,n)=1 \quad f(m,0)=2^m, \tag{8}$$

from which one deduces that

$$f(1,1)=\max_x \min(2-2x,1+x)=1.333, \tag{9}$$

$$f(2,1)=\max_x \min(4-4x,1.333(1+x))=2 \tag{10}$$

and it is possible to show that $f(n+1,n)=2$ for all values of n.

5.12. Two player games with incomplete information

Mastermind is a game which has been extensively studied within the mixture of disciplines mentioned earlier. van Hentenryck [46] describes how a suboptimal strategy can be derived and presents an implementation using a constraint handling language. His approach is for the game in its numerical form "Bulls and Cows" where the secret code is a set of four distinct digits in the range $\{0,\ldots,9\}$. It is essentially a dynamic programming formulation which has three rules:

(i) find a guess which is consistent with the previous guesses;
(ii) collect the feedback from the setter;
(iii) stop if the code is correct, otherwise add the feedback information to the knowledge of the state.

Other authors have tackled this problem in similar ways. The optimal strategy, when the objective is to find the secret code as quickly as possible, involves making some inconsistent guesses, which van Hentenryk's three-rule formulation does not permit.

6. Two player, stochastic games

Many one-and two-player games have been used by British and American school mathematics teachers to introduce mathematical recreations. Some enthusiastic teachers have gradually pooled their ideas, and this has given rise to several books of such "end-of-term treats", with the idea of relating them to the mathematics curriculum. Hence a book by Bell and Cornelius [9], which carries the subtitle: "a resource book for mathematical investigations". As dynamic programming does not figure largely in the curricula, most of the investigations are simple counting and discussion of rules of thumb, but there are several board games which can be investigated by students as projects using dynamic programming.

In this section, we consider games between two players, where the outcome of a move by one or both is stochastic. This may be because the state of one player is not known to the other, or because there is a stochastic feature which controls the moves. This stochastic feature may be based on the throw of one or more dice, or the effect of shuffling a pack or packs of cards. An optimal global strategy may not be possible, in these games, because of the curse of dimensionality, but locally optimal strategies can be found.

6.1. The board game "Risk"

The board game of "Risk" is a strategy game where two or more players assume the roles of commanders of armies, and battle to dominate a world map with forty-two territories. The whole game, even for two players, is too complex for analysis. However, a small part of the strategy, concerned with the conduct of a "battle" across the border between territories, is described by Maliphant and Smith [26]. The aim was to find the optimal choice of die-rolling options for different positions, once a decision had been taken to play a battle on a particular border. The strategy was a locally optimal one, insofar as it sought the best rules to follow in the short term, leaving the long term outcome of the game to more heuristic policies. This is typical of the contributions of dynamic programming in several areas. (Ger Koole has written two papers [21] and [22] on strategies in the Dutch form of Risk, which has slightly different rules from the version studied in [26]. Tan [42] has also published analysis of strategies in Risk.)

6.2. Race and chase games

A particular subset of the two-player stochastic games is that where players alternate to move one or more pieces around a board, marked with cells, from a given starting cell to a destination, and then take them off the board. The moves are determined by a stochastic process, usually the throw of a fair die. Race games (such as "snakes and ladders") are those in which the aim is for one player to remove all his/her pieces before the other. Generally there is little opportunity for strategy in such games; the exception is when a player has two or more pieces and the rules require that a player should reach the destination precisely. The probability of winning will depend on which piece is moved; there may be significant difference in this probability when the pieces are close to the destination. In most forms of race game, the player who starts has an advantage; accordingly there are many strategies designed to reduce this advantage, such as requiring a "six to start", or imposing barriers ("throw a six to pass this cell") or devices to retard progress or speed it ("snakes" and "ladders"). Such games are discussed in Bell [10] and Parlett [30].

Backgammon and other similar games involve the players chasing one another round and round a board. Pieces are removed by capture of one type or another, and progress is determined in a stochastic way, such as with two dice. This family of games has been the subject of dynamic programming-related analysis. The parent game is again too complex for complete analysis, although there are many heuristic strategies designed to give working rules for human players, or as the basis of programs in game-playing computers.

The game of “Ludo” is a simple game which combines features of race and chase. Each player (of two, three or four) has four playing pieces which move around a board. The players follow the same circular course, but enter and leave it at different points. If one player catches a piece belonging to an opponent, then the caught piece is returned to its starting position. When a player has two or more pieces on the board, there is a choice of which to move, and this leads to a game of strategy which is amenable to analysis using dynamic programming. Unpublished work [8] indicates that there are positions where the best move is counter-intuitive, with a player choosing to move his piece one space in front of the opponent, so as to retain the chance of catching the latter.

6.3. Darts

Kohler’s paper [20] on the game of darts is an example of the power of dynamic programming applied to stochastic games. However, it falls outside the definition of “board” games that is being used.

6.4. Jeopardy games with dice and cards

The family known as “Jeopardy games” has numerous members; essentially, two or more players take it in turn and use a randomizing device to obtain a score. Depending on the value, they have a choice of increasing the score by the same method or stopping and adding the score to a running total. Some scores from the device automatically end the player’s turn, and impose a score of zero for the turn. In some jeopardy games (“jeopardy race”) the objective is to be the first to exceed some predetermined total, in others (“jeopardy approach”) it is to be the closest player to such a total.

Neller and Presser [28] describe the use of dynamic programming in solving the jeopardy race game “Pig”, and present a wide-ranging discussion of ways of solving this and related games. They describe the rules as follows:

The object of the jeopardy dice game Pig is to be the first player to reach 100 points. Each player’s turn consists of repeatedly rolling a die. After each roll, the player is faced with two choices: roll again, or hold (decline to roll again).

- If the player rolls a 1, the player scores nothing and it becomes the opponent’s turn.
- If the player rolls a number other than 1, the number is added to the player’s turn total and the player’s turn continues.
- If the player holds, the turn total, the sum of the rolls during the turn, is added to the player’s score, and it becomes the opponent’s turn.

Thus, a player faces a binary choice, to roll or to hold. The state of the game when making this choice is described by the opponent’s score, the player’s score, and the current turn total. Straightforward dynamic programming is inadequate, since states can recur (appear on both sides of the DP recurrence relation, such as when both players roll a “1”). The authors use value iteration in a subtle way. There are 505,000 states, so they iterate to find the probability of a win when the scores total 198, then 197, and so on down to the starting position.

The paper gives the following formulation. Let $f(i,j,k)$ be the player’s probability of winning if his score is i, the opponent’s score is j, and the player’s turn total is k. In the case where $i+k \geqslant 100$, we have $f(i,j,k) = 1$ because the player can simply hold and win. In the general case where $0 \leqslant i, j < 100$ and $k < 100 - i$, the probability of a player who plays optimally winning is

$$f(i,j,k) = \max(f(i,j,k,\text{roll}), f(i,j,k,\text{hold})), \tag{11}$$

where $f(i,j,k,\text{roll})$ and $f(i,j,k,\text{hold})$ are the probabilities of winning for rolling or holding, respectively. These probabilities are

$$\begin{aligned} f(i,j,k,\text{roll}) &= \frac{1}{6}[1 - f(j,i,0) + f(i,j,k+2) + f(i,j,k+3) \\ &\quad + f(i,j,k+4) + f(i,j,k+5) + f(i,j,k+6)], \qquad (12) \\ f(i,j,k,\text{hold}) &= 1 - f(j,i+k,0). \qquad (13) \end{aligned}$$

The probability of winning after rolling a 1 or after holding is the probability that the other player will not win beginning with the next turn.

6.5. The memory game

Dynamic programming has been used to solve a widely-played board game, which is variously known as "Pelmanism", "The Memory game" or "Concentration". This is a game played with a number of pairs of cards, which are arranged randomly on the table, face down. A player's turn consists of turning two cards over, so that he and the other player(s) can see which cards are showing. If they match, then the player takes the pair, and has another turn. This game is often used to try to improve a person's memory—but if we assume that the player has perfect memory and therefore knows the values of each card which has appeared, then the mathematician's interest shifts to finding the best strategy. The objective is to maximize the expected surplus of pairs that the winning player has over the other.

There are essentially three types of action that a player can make on a turn. In Fig. 4 a situation has arisen in which there are four pairs on the table, and two of the cards are "known" to players with perfect memories. There are three possible moves, selecting at random from the two groups "Old" and "New".

- To turn up a pair of Old cards. ("Old–Old" or O–O.)
- To turn up a New card, and if it does not match an old card, select another New card. ("New–New" or N–N.)
- To turn up a New card, and if it does not match an old card, select an Old card. '("New–Old" or N–O.)

An "Old–Old" strategy is not really of interest since if that is optimal for one player, it will also be optimal for the other. (In other words, to be first player in such a state will lead to a negative expected surplus of pairs.)

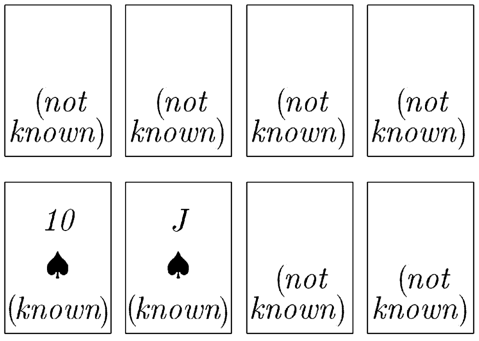

Fig. 4. The memory game: there are three possible decisions, in this state (4, 2): "Old, Old", "New, Old", "New, New".

Interest in the decision-making process centres around the situations where the best strategy is to select "New–Old". With such a move, the player has a chance of selecting a known card, but after a failure to match cards, on the second turn of the cards, does not collect any more information.

The dynamic programming formulation of this problem is reasonably straightforward, since there are only three options. The objective is to maximize the number of pairs expected, so one defines an appropriate objective function, and then works out the expected consequences of each of the three possible moves. The full theory is published by Zwick and Paterson [48] and they present the dynamic programming relationship as

$$\begin{aligned} f(n,k) = \max\Bigg\{ & \left\{\frac{k}{2n-k}(1+f(n-1,k-1)) - \frac{2(n-k)}{2n-k}f(n,k+1)\right\}, \\ & \left\{\frac{k}{2n-k}(1+f(n-1,k-1)) - \frac{2(n-k)}{2n-k}\left(\frac{k-1}{2n-k-1}(1+f(n-1,k)) \right.\right. \\ & \left.\left. + \frac{2(n-k-1)}{2n-k-1}f(n,k+2)\right)\right\}\Bigg\}, \end{aligned} \tag{14}$$

where $f(n,k)$ is the expected number of pairs collected following an optimal policy, from a state where there are n pairs on the table (i.e. $2n$ cards) of which k are known. (Clearly $0 \leqslant k \leqslant n$ and the k cards are all different.) The optimal rule which they derive and prove is as follows:

- Use "Old–Old" when $3k \geqslant 2(n+1)$ and $n+k$ is odd.
- Use "New–Old" when $k \geqslant 1$ and $n+k$ is even or in position $(6,1)$.
- Use "New–New" otherwise.

Following these rules, one finds:

$$f(n,0) = \frac{1}{4n+2} + O\left(\frac{1}{n^3}\right)(n \text{ odd}) = -\frac{1}{4n-2} + O\left(\frac{1}{n^3}\right)(n \text{ even}). \tag{15}$$

They admit that the simplicity of the game is misleading; the proof of their results is a long, complex one. They also leave several strands of research open. (This game has also been implemented on some Nokia mobile telephones.)

7. Summary

This paper has tried to present a succession of results and ideas about the use of dynamic programming in board games, pointing to the variety and complexity of the problems that currently exist.

Increasing computer power is making it possible to evaluate more and more positions in board games. Modelers in dynamic programming need to work with the computer scientists and the experts in combinatorial games, to try to solve particular examples of games, and to derive results for general examples. The experiences of people in AI, who are developing ways of limiting the forward search in dynamic programming models seems especially relevant.

Above all, this is an area of study which has many problems, but is interesting and—dare one admit it—fun!

Acknowledgements

This paper is based on several seminars including that given at the IFORS conference in Beijing in 1999. Thanks are due to many people, including Roger Hill, Jenny Bell, Peter Vamos, John Norman, Lyn Thomas and two anonymous referees.

References

[1] D. Aldous, P. Diaconis, Longest increasing subsequences: From patience sorting to the Baik–Deift–Johansson theorem, Bulletin of the American Mathematical Society 36 (4) (1999) 413–432.

[2] J.D. Allen, A note on the computer solution of Connect-Four, in: D.N.L. Levy, D.F. Beal (Eds.), Heuristic Programming in Artificial Intelligence, vol. 1, Ellis Horwood, Chichester, UK, 1989, pp. 134–135.

[3] L. Victor Allis, Searching for solutions in games and artificial intelligence, Ph.D. Thesis, University of Limburg, Maastricht, Netherlands, 1994.

[4] L. Victor Allis, M. van der Meulen, H.J. van den Herik, Proof-number search, Artificial Intelligence 66 (1) (1994) 91–124.

[5] Anon, The Freecell Project, This was documented on the Web, but all links to the proof now appear to be missing. There is a useful web-site about Freecell and its history at: <http://member.aol.com/wgreview/fcfaq.html>.

[6] M. Ascher, Mu Torere: An analysis of a Maori game, Mathematics Magazine 60 (2) (1987) 90–100.

[7] J.D. Beasley, The Ins and Outs of Peg Solitaire, Oxford Paperbacks, 1992.

[8] J. Bell, D.K. Smith, Iterative dynamic programming and the end of some board games, in: Presented at the EURO Conference, Istanbul July 2003. Available from: <http://www.maths.ex.ac.uk/DKSmith/eu03pape.pdf>, July 2003.

[9] R. Bell, M. Cornelius, Board Games Around the world: A Resource Book for Mathematical Investigations, Cambridge University Press, Cambridge, UK, 1988.

[10] R.C. Bell, Board and Table Games from Many Civilizations, Dover Publications, 1980.

[11] R. Bellman, On the application of dynamic programming to the determination of optimal play in chess and checkers, Proceedings of the National Academy of Sciences 53 (1965) 244–247.

[12] E.R. Berlekamp, J.H. Conway, R.K. Guy, Winning Ways for your Mathematical Plays, vols. 1&2, Academic Press, London, 1982.

[13] C.L. Bouton, Nim, a game with a complete mathematical theory, Annals of Mathematics 3 (1902) 35–39.

[14] P.J. Campbell, Farmer klaus and the mouse, The UMAP Journal 23 (2) (2002) 121–134.

[15] M.J. Chlond, O. Akyol, A Nimatron, INFORMS Transactions on Education 3 (3) (2003). Available from: <http://ite.pubs.informs.org/Vol3No3/ChlondAkyol/>.

[16] S. Fish, Freecell solver. A program in C and discussion. Available from: <http://vipe.technion.ac.il/shlomif/lecture/Freecell-Solver>.

[17] R. Gasser, Solving nine men's Morris, in: Games of No Chance 29, Cambridge University Press for MSRI, 1999, pp. 101–113. Available from: <www.msri.org/publications/books/Book29/files/gasser.pdf>.

[18] R.K. Guy, What is a game? in: R.J. Nowakowski (Ed.), Games of No Chance, Cambridge University Press, Cambridge, UK, 1996, pp. 43–60.

[19] G. Irving, J. Donkers, J.W.H.M. Uiterwijk, Solving kalah, International Computer Games Association Journal 23 (4) (2000) 139–148.

[20] D. Kohler, Optimal strategies for the game of darts, Journal of the Operational Research Society 33 (10) (1982) 871–884.

[21] G. Koole, An optimal dice rolling policy for Risk, Nieuw Archief voor Wiskunde 12 (1–2) (1994) 49–52.

[22] G. Koole, Stochastisch dynamisch programmeren, Nieuwe Wiskrant. Tijdschrift voor Nederlands Wiskundeonderwijs. 21 (3) (2002) 23–26.

[23] M.M. Kostreva, R. Hartman, Multiple objective solutions for Tetris, Journal of Recreational Mathematics 32 (3) (2003–2004) 214–225.

[24] C. Kuykendall, Analyzing solitaire, Science 283 (5403) (1999) 794–795.

[25] J.D.C. Little, K.G. Murty, D.W. Sweeney, C. Karel, An algorithm for the traveling salesman problem, Operations Research 11 (1963) 972–989.

[26] S.A. Maliphant, D.K. Smith, Mini-Risk: Strategies for a simplified board game, Journal of the Operational Research Society 41 (1) (1990) 9–16.

[27] E.H. Moore, A generalization of the game called Nim, Annals of Mathematics 11 (1910) 93–94.

[28] T.W. Neller, C.G.M. Presser, Optimal play of the dice game pig, The UMAP Journal 25 (1) (2004) 25–47.

[29] J. Nievergelt, R. Gasser, F. Maser, C. Wirth, All the needles in a haystack: Can exhaustive search overcome combinatorial chaos? Lecture Notes in Computer Science, vol. 1000, Springer-Verlag, 1995, pp. 254–274. Available from: <nobi.ethz.ch/febi/ex-search-paper/paper.html>.

[30] D. Parlett, Oxford History of Board Games, Oxford University Press, 1999.

[31] P.R Pudaile, Problem 10801, American Mathematical Monthly 107 (4) (2000) 368.

[32] M.L. Puterman, Markov Decision Processes, Wiley, 1994.

[33] A. Rabb, A probabilistic analysis of the game of Solitaire, Undergraduate honors thesis, Harvard University, 1988.

[34] A. Rapoport, Fights, Games and Debates, University of Michigan Press, Ann Arbor, Michigan, 1960.

[35] J.W. Romein, H.E. Bal, Notes: Awari is solved, Journal of the International Computer Games Association 25 (September) (2002) 162–165.

[36] C.M. Rump, Who wants to see a $ million error, INFORMS Transactions on Education 1 (3) (2001) 102–111.
[37] J. Schaeffer, One Jump Ahead, Springer-Verlag, 1997.
[38] F. Schuh, The Master Book of Mathematical Recreations, Dover, 1968.
[39] D.K. Smith, Dynamic Programming: A Practical Introduction, Ellis Horwood, Chichester, Sussex, UK, 1991.
[40] D.K Smith, Dynamic programming and two problems of Dudeney, Mathematical Spectrum (May) (1996).
[41] M. Sneidovich, OR/MS Games: 4. The joy of egg-dropping in Braunschweig and Hong Kong, INFORMS Transactions on Education 4 (1) (2003) 48–64.
[42] B. Tan, Markov chains and the RISK board game, Mathematics Magazine 70 (5) (1997) 349–357.
[43] L.C. Thomas, The best banking strategy when playing The Weakest Link, Journal of the Operational Research Society 54 (7) (2003) 747–750.
[44] M. Trick, Building a better game through dynamic programming: A flip analysis, INFORMS Transactions on Education 2 (1) (2001). Available from: <http://ite.informs.org/vol2no1/trick/>.
[45] R. Uehara, S. Iwata, Generalized hi-q is np-complete, Transactions of the IEICE 73 (1990) 270–273.
[46] P. van Hentenryck, Constraint Satisfaction in Logic Programming, The MIT Press, Cambridge, Mass, USA, 1989.
[47] Various. Rubik's official online site. Available from: <http://www.rubikschamps.com>, 2003.
[48] U. Zwick, M.S. Paterson, The Memory game, Theoretical Computer Science 110 (1) (1993) 169–196.