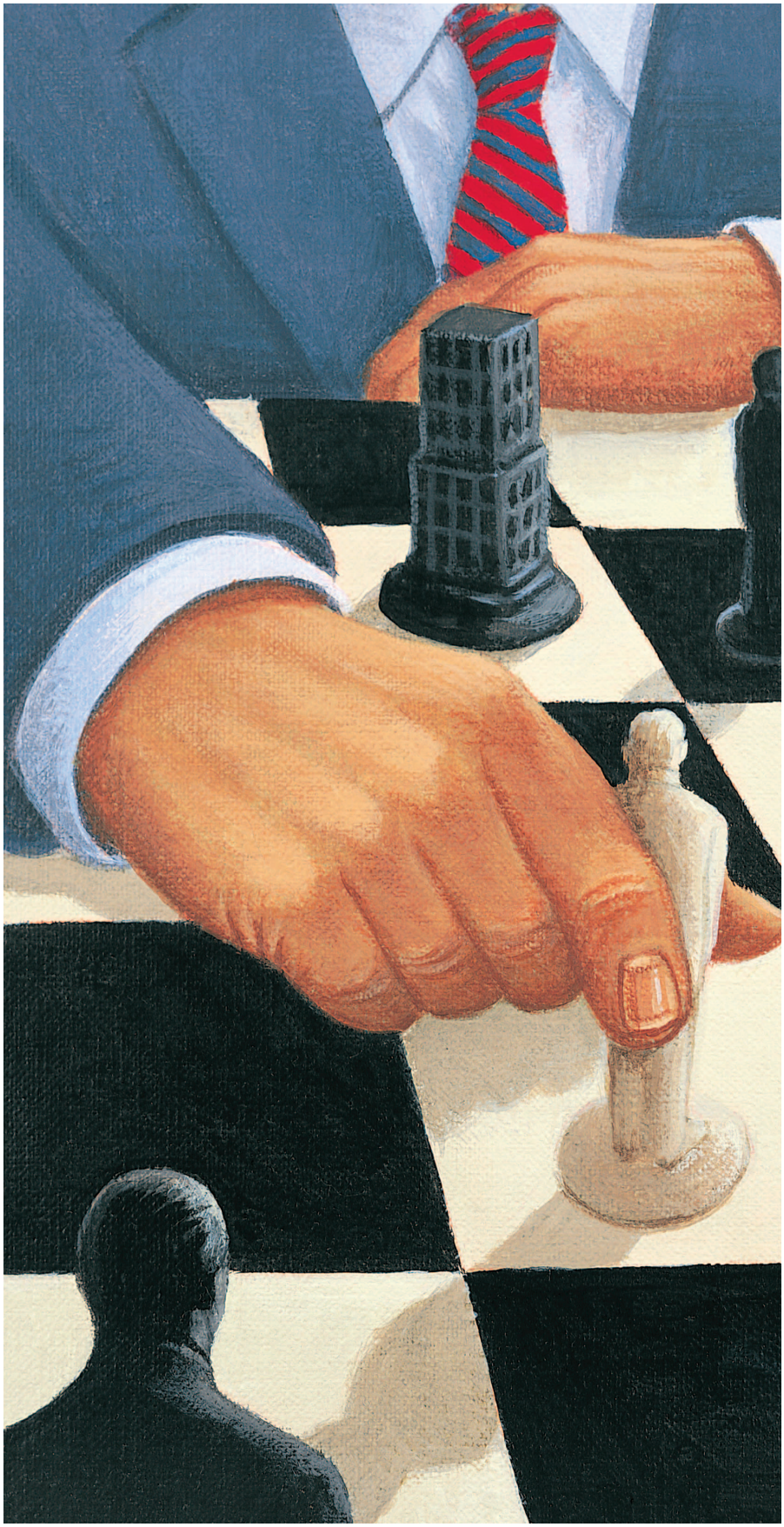
Simon M. Lucas
University of Essex, UK

Graham Kendall
University of Nottingham, UK

Evolutionary Computation and Games



Games provide competitive, dynamic environments that make ideal test beds for computational intelligence theories, architectures, and algorithms. Natural evolution can be considered to be a game in which the rewards for an organism that plays a good game of life are the propagation of its genetic material to its successors and its continued survival. In natural evolution, the fitness of an individual is defined with respect to its competitors and collaborators, as well as to the environment. Within the evolutionary computation (EC) literature, this is known as co-evolution and within this paradigm, expert game-playing strategies have been evolved without the need for human expertise. The "Evolving Game Strategies" sidebar discusses the main design decisions involved when applying evolution in this way.

Much of the early work on computational intelligence and games was directed toward classic board games, such as tic-tac-toe (noughts and crosses) [1], chess, and checkers [15]). Board games can now, in most cases, be played by a computer at a higher level than the best humans. There is an interesting table in [2, page 350], which shows various games and their status regarding human or machine superiority. The best computer checkers player, for example, is considered to be better than the world champion (this is discussed further below). The best computer chess players, despite Deep Blue beating Kasparov, are still not rated as highly as the best (i.e., the world champion) human players, though the day cannot be too far away when the best computer players can *consistently*

1556-603X/06/$20.00©2006IEEE

beat the best human players. Board games usually succumb to *brute force*[1] methods of search (mini-max search, alpha-beta pruning, parallel architectures, etc.) to produce the very best players. Go is an exception, and has so far resisted machine attack. The best Go computer players now play at the level of a good novice (see [3], [4] for review papers and [5]–[8] for some recent research). Go strategy seems to rely as much on pattern recognition as it does on logical analysis, and the large branching factor severely restricts the look-ahead that can be used within a game-tree search.

Games also provide interesting abstractions of real-world situations, a classic example being Axelrod's Prisoner's Dilemma [9]. Of particular interest to the computational intelligence community, is the iterated version of this game (IPD), where players can devise strategies that depend upon previous behavior. An updated competition [10], celebrating the 20th anniversary of Axelrod's competition, was held at the 2004 IEEE Congress on Evolutionary Computation (Portland, Oregon, June 2004) and at the IEEE Symposium on Computational Intelligence and Games (Essex, UK, April 2005), and this still remains an extremely active area of research in areas as diverse as biology, economics and bargaining, as well as EC.

In recent years, researchers have been applying EC methods to evolve all kinds of game-players, including real-time arcade and console games (e.g., Quake, Pac-Man). There are many goals of this research, and one emerging theme is using EC to generate opponents that are more interesting and fun to play against, rather than being necessarily superior.

Before discussing possible future research directions, it is interesting to note some of the achievements during the past 50 years or so, during which time games have held a fascination for researchers.

Games of Perfect Information

Games of perfect information are those in which all the available information is known by all the players at all times. Chess is the best-known example and has received particular interest culminating with Deep Blue beating Kasparov in 1997, albeit with specialized hardware [11] and brute force search, rather than with AI/EC techniques. However, chess still receives research interest as scientists turn to learning techniques that allow a computer to 'learn' to play chess, rather than being 'told' how it should play (e.g., [12]–[14]). Learning techniques were being used for checkers as far back as the 1950s with Samuel's seminal work ([15], which was reproduced in [16]). This would ultimately lead to Jonathan Schaeffer developing Chinook, which won the world checkers title in 1994 [17], [18]. As was the case with Deep Blue, the question of whether Chinook used AI techniques is open to debate. Chinook had an opening and end game database. In certain games, it was able to play the entire game from these two databases. If this could not be achieved, then a form of mini-max search with alpha-beta pruning and a parallel architecture was used. Chinook is still the recognized world champion, a situation that is likely to remain for the foreseeable future. If Chinook is finally defeated, then it is almost certain that it will be by another computer. Even this is unlikely. On the Chinook Web site [19], there is a report of a tentative proof that the *White Doctor* opening is a draw. This means that any program using this opening, whether playing black or white, will never lose. Of course, if this proof is shown to be incorrect, then it is possible that Chinook can be beaten; but the team at the University of Alberta has just produced (May 14, 2005) a 10-piece endgame database that, combined with its opening game database, makes it a formidable opponent. Despite the undoubted success of Chinook, the search has continued for a checkers player that is built using "true" AI techniques (e.g., [20]–[25]), where the playing strategy is learned through experience rather than being pre-programmed.

In natural evolution, the fitness of an individual is defined with respect to its competitors and collaborators, as well as to the environment.

Chellapilla and Fogel [20]–[22] developed Anaconda, named due to the strangle hold it places on its opponent. It is also known as Blondie24 [22], which is the name it uses when playing on the Internet. This name was chosen in a successful attempt to attract players on the assumption they were playing against a blonde 24-year-old female. Blondie24 utilizes an artificial neural network with 5,046 weights, which are evolved by an evolutionary strategy. The inputs to the network are the current

[1]Although the term *brute force* is widely used in this context, we should point out that making this kind of search work well in practice typically requires very elegant and finely-tuned programming.

Evolving Game Strategies

There are several issues to consider when evolving a game strategy:

Fitness Function

Depending on the nature of the game and the objectives of the research, the fitness function can be based upon game-playing ability against a fixed (non-evolved) set of opponents, or against a population of evolved agents; the latter case is known as co-evolution and has the advantage of not requiring any human-designed playing strategies to be implemented. Instead, natural selection leads to an improving population of players. When using a fixed opponent, the problem of choosing the correct level of difficulty arises; for if the opponent is either too strong or too weak, then randomly constructed agents will populate regions of the space with insufficient gradient. A useful technique here, when possible, is to choose a strong fixed player, but have it make random moves with a probability that is tuned to give an even chance of a member of the current population winning.

Exploration

It is important to encourage the evolving agents to make sufficient exploration of game strategy space. This can be done by ensuring the population of players is sufficiently large, and/or adding noise to the player selections. Otherwise, deterministic players may play too limited a range of games.

Implementation

An important aspect of evolving game playing agents, and one that often receives scant attention in research papers, is that of efficient implementation of the game. This is important since the quality of the evolved strategies, and of the complexity of architecture that can be evolved, is critically dependent upon the number of fitness evaluations or in other words, upon the number of game players, or the time steps executed in the case of a real-time game. In recent work on evolving Othello strategies, one of the authors (Lucas) made an initial Java prototype implementation of the system that was able to play only five games per second (when evolving a weighted piece counter at 1-ply). However, through careful re-engineering of the software, he improved this to more than one thousand games per second for the same setup. The tricks used included the use of a profiler to observe which operations were taking the most time, replacing any Java Collections used with simpler, more restricted custom-designed collection classes, removing all unnecessary dynamic memory management, and performing incremental evaluation of the board (i.e., evaluating the effects of making a move given a particular weighted piece counter, without actually making the move). Also, using 1-dimensional instead of 2-dimensional arrays, with a blank border around the board to detect off-board cases, was observed to make a significant difference; this latter trick being borrowed from Thomas Runarsson's Othello implementation in C.

Game Interface

This defines how the evolved agent will interact with the game. The main choices are state evaluator or action selector. The state evaluator's task is to evaluate the desirability of any given state from the evaluator's perspective. This is a very flexible interface and allows it to be used directly with mini-max search, where the leaves of the game tree are passed to the state evaluation function. The alternative is the action selector, which gives the agent more control as to which move to make but is typically harder to learn. The action selector is given the current board state, and asked to choose an action. Given this setup, it is challenging for a neural network to even select a legal move, and this interface is often modified to select the highest-rated legal move chosen by the agent, rather than the highest-rated move (which may be illegal).

Input Representation and Agent Architecture

We consider these together, as they are intimately related. Whether we view a particular feature to be part of the input coding scheme or the first layer of a neural network agent, it is to some extent an arbitrary distinction. When designing neural networks for board games, it has been common to use a convolutional style of neural network, which naturally exploits any translation-invariant features, or alternatively, to use sliding windows of various sizes to construct extra input features. More advanced schemes also incorporate invariance to symmetry and may also use higher-level features based upon graph collapsing in the game of Go, for example.

Regarding the choice of agent architecture, most prevalent is the neural network; and these have proven to be evolvable across a wide range of games. However, other architectures may be even better suited, and researchers are urged to consider alternatives such as GP-style expression trees or fuzzy systems. Much more comparison-based work is needed before a general sense of which architectures are best suited to which style(s) of game is achieved.

Continued...

Evolving Game Strategies *Continue*

Ply Depth

Each game has its unique space of strategies, and the game dynamics should influence the setup of the system. For two-player perfect information games, an important consideration is the ply-depth to use during evolution. Chellapilla and Fogel, and Hughes independently settled on 4-ply when evolving checkers players, as a compromise between good quality evaluation and CPU time. For other games, a different ply-depth may be appropriate. Chong et al. [57] used 2-ply when evolving Othello players, for example.

Evolutionary Algorithm

For some games, it seems that the setup of the EA may be non-critical. Runarsson and Lucas, however, found that when evolving piece counters for small-board Go, many of the details were very important [58]. To be get good performance with co-evolution, they found it essential to use parent/child weighted averaging, and a sufficiently large population of 30 players (10 was not sufficient).

board position, and it outputs a value that is used in a mini-max search. During the training period, the program is given no information other than whether it won or lost (it is not even told by how much). Blondie24 is not provided with a strategy and contains no database of opening and ending game positions. Co-evolution is used to develop Blondie24 by playing games against itself. Once it is able to play at a suitable level, it often searches to a depth of 10; but depths of 6 and 8 are common in play. This program was available to the delegates of the Congress on Evolutionary Computing (CEC) conference for two years (CEC'00 San Diego and CEC'01 Seoul) with Fogel offering a prize of $100 (CEC'00) and $200 (CEC'01) to anybody who could defeat it. The prize remained unclaimed.

Hughes has shown that good playing performance experimented could be achieved by evolving position-weighted piece counters for Checkers [23], with his *Brunette* player competing closely with Blondie when allowed to search to a deeper ply, commensurate with the reduced evaluation time per board that results from the simpler evaluation function. Hughes has also investigated both the use of co-evolution and Monte-Carlo methods for position evaluation as alternatives to mini-max search [24].

There are many goals of this research, and one emerging theme is using EC to generate opponents that are more interesting and fun to play against, rather than being necessarily superior.

Monte-Carlo simulation can be applied to evaluate a game state simply by playing out a game to its conclusion by making a succession of random moves, and repeating this process a large number of times to estimate the probability of winning from that state. Although the probability is actually estimated for purely random players, this method has, nonetheless, proven to give surprisingly good estimates. It has been used to good effect in a number of different games, including Go [26] and Real-Time Strategy games [27].

As well as chess and checkers, which tend to receive the most media interest, other games have also made significant contributions. Research into Backgammon [28], [29] has made advances in reinforcement learning and other machine-learning techniques. Go [3], which remains a massive challenge, has led to advances in knowledge representation and search techniques. Bridge [30] has inspired areas such as partition search, the practical application of Monte-Carlo techniques to realistic problems, and the use of squeaky wheel optimization [31] in game playing. Othello has also been the subject of significant research, culminating in Logistello's 6-0 defeat of the current world champion [32]. Furthermore, there is an Othello competition associated with CEC 2006, which aims to find the best Othello position evaluation function when restricted to 1-ply search.

Games of Imperfect Information

Games of imperfect information are classified by the fact that some of the information is hidden from some (or all) of the players. Card games typically fall into this category with, perhaps, poker attracting the most recent research interest. As far back as 1944, game theory (developed by von Neumann and Morgensten [33] to model and study the economic environment) was using a simplified version of poker as a test bed. They recognized that accomplished poker players regularly adopt bluffing as part of their game, which would have to be accounted for in any automated poker player.

Findler [34] studied automated poker during a 20-year period. He also worked on a simplified game that was based on 5-card draw poker with no ante and no consideration of betting position due to the computer always playing last. He concluded that dynamic and adaptive algorithms are required for successful play, and static mathematical models were unsuccessful and easily beaten. Other than von Neumann's and

Findler's works, research has been conducted by a limited number of research groups, all relatively recent.

One of the groups is led by Jonathan Schaeffer. He and members of his research team have developed ideas that have led to the release of Poki, which is the strongest automated poker-playing program to date. It is still a long way from being able to compete in the World Series of Poker, an annual event held in Las Vegas, but their results show promise. Schaeffer's work concentrates on two main areas [35], [43]. The first is to make betting decisions using probabilistic knowledge [36] to determine what action to take (fold, call or raise) given the current game state. Billings et al. also use real-time simulation of the remainder of the game that allows the program to determine a statistically significant result in the program's decision-making process. Schaeffer's group is also looking at opponent modelling [37]. This allows Poki to maintain a model of an opponent that it uses to decide which betting decisions to make. This is done on the assumption that players adopt differing styles and any automated player has to be able to exploit this. For the interested reader, perhaps, the best starting paper for poker is [43], with a later paper [38] presenting more recent work.

The Gala system developed by Koller and Pfeffer [39] allows games of imperfect information to be specified and solved, using a tree-based approach. However, due to the size of the trees, they state, "*. . . we are nowhere close to being able to solve huge games such as full-scale poker, and it is unlikely that we will ever be able to do so.*"

Luigi Barone and Lyndon While have carried out research into the automation of poker [40]–[42] using EC techniques. They recognize that there are four main types of poker players: Loose Passive, Loose Aggressive, Tight Passive and Tight Aggressive [41], [42] players. They suggest [40] using evolutionary strategies as a way of modelling an adaptive poker player. They use a simple poker variant where each player has two private cards and access to five community cards, and there is only one round of betting. This initial work incorporates three main areas of analysis: hand strength, betting position and risk management. The work demonstrates how a player that has evolved using evolutionary strategies can adapt its style to two types of tables (loose and tight). In [41] they develop this work by introducing a hypercube, which is an n dimensional vector used to store candidate solutions. The hypercube has one dimension for the betting position (early, middle and late) and another dimension for the risk management (selected from the interval 0.3). At each stage of the game, the relevant candidate solutions are selected from the hypercube (e.g., middle betting position and risk management) and the decision is made whether to fold, call or raise. To make this decision, the hypercube entry holds seven real numbers that are used, via suitably defined functions, to produce probabilities for each choice. It is the seven real values that are evolved, depending on whether the player won the hand. Barone and While report that this player is able to adapt to different playing styles. Their 2000 work [42] extends the dimensions of the hypercube to include four betting rounds (pre-flop, post-flop, post-turn and post-river) and an opponent dimension, so that the evolved player can choose from the hypercube depending on which type of player it is playing against. The authors report that this player outperforms a competent static player.

Poker is particularly interesting, from an EC research point of view, as it is a game of imperfect information and difficult to tackle with brute-force search. Poker also contains other unknowns such as the playing styles of the other players who may use bluffing (and double bluffing) during the course of the game. Indeed, Poker presents a range of challenging research goals that have not yet been conquered by the EC/AI community. Darse Billings et al. [43] have stated,

> *"Poker is an interesting testbed for artificial intelligence research. It is a game of imperfect information, where multiple competing agents must deal with probabilistic knowledge, risk assessment, and possible deception, not unlike decisions made in the real world."*
>
> and concluded,
>
> *". . . but there remains considerable research to be done to play [poker] at a world-class level."*

Unlike *complete information* games in which the techniques to solve the games (computational power allowing) have been known and understood for a long time (such as mini-max search and alpha-beta pruning), games of imperfect information have not received the same sort of analysis and, in doing so, could prove relevant to many other areas, including economics, online auctions and negotiating. Another game in this vein is Monopoly (although admittedly, there is only a limited amount of hidden information consisting of the ordering of the *Chance* and *Community Chest* cards), and Frayn [44] showed how evolution could be used to learn successful strategies for this game, based upon several factors that include good evolved estimates of the true value of each property. Interestingly, in addition to evolving strong players, Frayn's work is informative for human players wishing to improve their game, and offers a dramatic improvement to previous more simplistic estimates of property values that were based solely upon the probability of a player visiting them.

Video Games

The above discussion of perfect and imperfect information games focused on computer agents developed to play human-centered games. Human-centered games are limited by what can easily be manipulated given human mental capacity and dexterity. Video games, on the other hand, operate under no such constraints and typically have a vastly more complex state space than even the most complex of human-centered games. This richer complexity encourages the development or evolution of more general purpose AI agents than are required for playing board or card games, and successful game-play may

involve a rich variety of skills. Laird [45] emphasized the challenge posed by real-time video games to the AI community, and this is now a rapidly growing field of research. Of the 38 papers presented at IEEE CIG 2005 [46], 16 were concerned with real-time video or real-time strategy games, making this the dominant category.

Currently, the vast majority of Game-AI agents are controlled through scripts, developed at significant expense by game programmers. The use of computational intelligence techniques offers an interesting alternative to scripted approaches, whereby the agent behavior can be controlled using an evolved neural network, for example, rather than being programmed. This has two very interesting implications. Firstly, programming the scripts can be laborious and time consuming (and, therefore, expensive). Evolving the agent behaviors is, therefore, financially attractive. Second, the fact that the behavior is evolved rather than programmed by a human means that the evolved agent may create novel strategies that game players may find especially entertaining to compete with. Furthermore, the evolved agents tend to be excellent at exploiting loopholes in a game. Identifying and removing these loopholes is an important part of the game development life cycle, and one in which evolutionary computation is just starting to be used [47]. These are aspects of the game, whereby a player finds a fast way to win or clear a level. The effect of this is that the player tends to lose respect for the game. Currently, game developers devote many man hours of game playing to find these.

Online Learning and Adaptive Characters

A major challenge for the academic community is to develop effective learning algorithms that operate without explicit learning signals, and can adapt in real time within complex environments. This is a major research topic in many disciplines, and includes concepts such as self-organizing neural networks and reinforcement learning (in particular Temporal Difference learning, see "Co-Evolution Versus Temporal Difference Learning" sidebar). To have game characters that effectively exhibit such characteristics would make many games much more playable, and cause characters to invest in complex, long-term dynamics. For example, if a computer opponent learns from playing against the human player, then the human player may benefit in the long term by making deliberate mistakes early on. We do not know of any existing games that use this type of learning, but we believe it has a great deal to contribute. Yannakakis [48] has already used an objective measure of *interestingness* to evolve more engaging ghost behaviors for a simplified version of Pac-Man and other prey-predator games.

New Game Genres

When computational intelligence techniques become widespread in games, the possibility of opening new genres arises. This has already been seen in the work of Stanley and Miikkulainen [49] in their NERO video game, which won the best paper prize at IEEE CIG 2005. In NERO, the object of the game is to train an army unit to fight as a cohesive team and, thereby, achieve the game goals. Each soldier has a neural network that is able to learn through experience, a feature on which the game relies. Figure 1 shows a training maze set up by a human player. Soldiers who make more progress through the maze have more offspring than those who make less progress, and eventually good maze-solving behaviors can be evolved. Here, the novel use of evolutionary algorithms has led to a new game, and one that fosters some understanding of evolutionary processes.

While the major recent trend has been toward designing NPCs (non-playing characters) for highly complex 3D games such as Quake, the challenges of older 2D arcade classics such as Pac-Man should not be underestimated; and Lucas [50], in his initial attempts, was only able to evolve novice to intermediate level neural network players (see Figure 2). Previous work on Pac-Man (e.g., [51], [52]) had used much less demanding simulations of the game. The advantage of working with older-style arcade games is that they offer a sufficiently interesting challenge, while being much easier to implement and faster to simulate. While there exists open-source first-person shooters, these can be quite technical to interface to, and the problem of lengthy simulation time still remains.

Real-World Games

In complex multi-task, multi-objective research areas such as robotics, measuring progress can be a significant problem. One solution is to use competitive games to rank the ability of a robot or team of robots. A great success in this area has been the robotic football tournaments in the form of RoboCup [53] and the Robot Soccer World cup series. This has stimulated many research groups to work with a

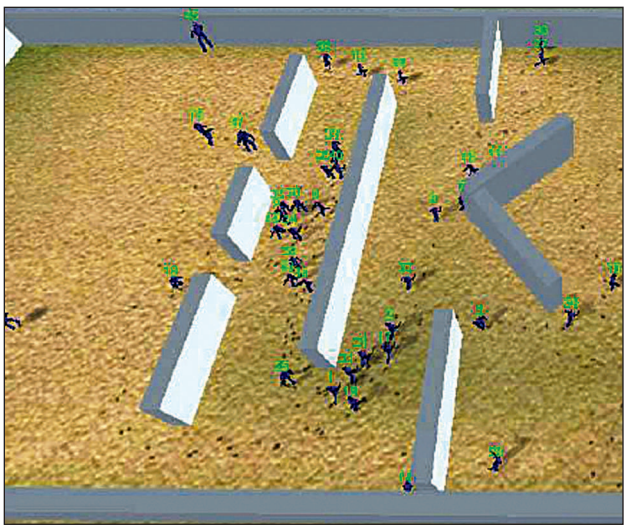


FIGURE 1 Screen shot from NERO video game. Here, the human player has set up a 'navigate through maze' exercise to train the soldiers to acquire this type of behavior.

common focus, and progress can now be measured by the performance of current teams against previous teams. Real-world robotic games are always much harder than they first appear and involve dealing with noisy sensors and imperfect actuators, together with all the strategic issues of how best to play the game in question.

Monte-Carlo simulation can be applied to evaluate a game state simply by playing out a game to its conclusion by making a succession of random moves, and repeating this process a large number of times to estimate the probability of winning from that state.

Robotic football is a very complex game, with a significant investment of time and money required to begin research. On a smaller scale, but still offering an immense challenge, has been the computer-controlled car racing series organized as competitions associated with CEC (Congress on Evolutionary Computation). This has a start-up cost of a few tens of dollars for a Web cam and a simple bang-bang type remote control car. By using the real-time video feed from the Web cam, the aim of the controller is to race the car around the track as fast as possible. This is hard enough in itself, but much harder and more interesting still when racing two or more cars against each other at the same time, as with Formula-1 or Indy car racing. The beauty of racing model cars, however, is that deliberate obstruction and crashing can be allowed (as it is in stock-car racing), with the cars usually surviving collisions with only cosmetic damage (wing-mirrors have a short life expectancy!).

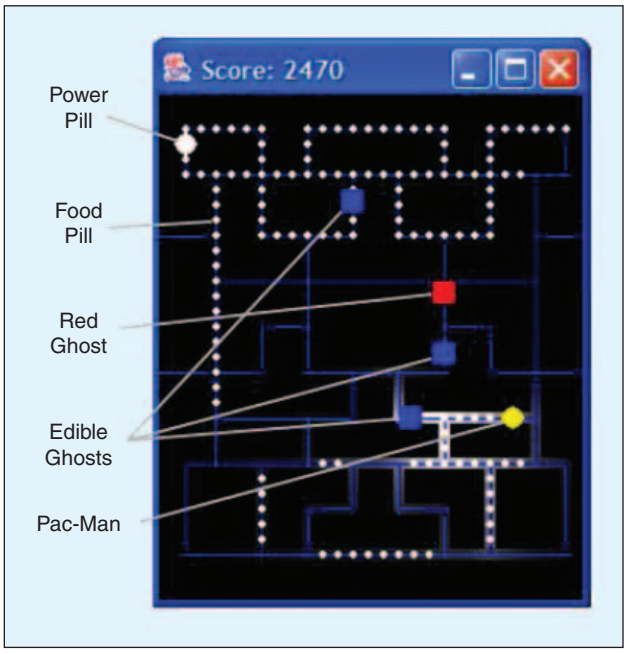


FIGURE 2 Pac-Man: The gray overlay shows the desirability rating (whiter = more desirable) provided by the evolved neural network for each location in the maze, given the current state of the pills, power pills, and the ghost positions.

For the CEC car racing series, the best entry has been the evolved car racing system of Ivan Tanev [54]. The approach taken by Ivan was to hand design a car control algorithm, then optimize the parameters with an EA. Ivan's entry was sufficient to win the CEC 2005 car racing competition, but got stuck several times while completing the 10-lap time trial. One the other hand, Togelius and Lucas [55] were able to evolve a high-performance neural-network controller for a simulated version of the track, while performing poorly on the real-world track. This provides further evidence that real-world problems tend to be much harder than their simulated counterparts, but the rewards for solving them are greater. On a similar theme, it would be interesting to see whether the method of Floreano et al. [56] to evolve an active vision system for driving a car in a 3D racing game could be applied to driving a car in the real world.

Evolutionary Computation in Commercial Games

Clearly, there have already been some examples of evolutionary algorithms deployed in commercial games. David Fogel's Blondie program is a famous example, and the work of Joerg Denziger and co-workers (on using evolutionary algorithms to test soccer games) has already been mentioned. However, the authors believe that there is much more scope for the utilization of EAs and other CI techniques within commercial games, and if appropriately deployed, could lead to not only reduced development costs but also more importantly, better gameplay. This latter point seems to be a contentious issue, with some industry experts saying that what users want is predictable gameplay, so that when a player has learned a set of actions to complete a level, then these actions will work on future occasions. Similarly, there is a widely held belief that players do not want AI opponents to be too smart, or too hard to beat. Our view is that more intelligent opponents could potentially lead to gameplay that would be more varied and more enjoyable.

Conclusions

Games have always held a natural fascination for humans, and have long been used as a test bed for artificial intelligence and machine-learning algorithms. Traditionally, most successful game playing strategies have been achieved through clever programming and effective search strategies, but machine-learning techniques such as evolutionary algorithms are now often competitive with and, in some cases, superior to hand-coded systems.

There are many games still waiting to be explored from an evolutionary perspective, and many more waiting to be invented. This research area, despite its long history, has a very

fresh and innovative feel to it at the moment, and it is clear that video and robotic-type games offer a variety of significant challenges for computer science and AI researchers. Academic research about evolving or, otherwise, learning non-player character strategies for first-person shooter games is now a respectable area of computer science research, as indeed it should be. There is also much more to be done with ball games, both team and individual.

In a recent interview Ken Katagari, inventor of the Playstation, commented that soon games will become indistinguishable

Co-Evolution Versus Temporal Difference Learning

Much is made of the fact that co-evolution can be used to learn game strategies simply by having a population of players play many games against each other, and allowing the more successful players to have more offspring than the less successful players. However, temporal difference learning (TDL) is also able to achieve the same remarkable kind of feat, of learning simply by playing.

A key difference between the two approaches is that temporal difference learning attempts to learn a value function by observing the progression of game states during game-play. It learns from the sequence of game states in an entirely unsupervised way; there is no expert knowledge used to train the system. In the simplest form of TDL, called TD(0), the learner adjusts the value function (this could involve using back-propagation to update the weights of an MLP, for example) to make the value of the current state more like the value of the next state. It may seem surprising that this works, since changes in an incorrect value function would seem meaningless, but there are two reasons why this is not the case. Firstly, the true values of board positions are available at the end of the game, so the information is fed back directly and is guaranteed to be correct. Secondly, as the value function starts to become meaningful, the changes in the value function during game-play will also have some statistical importance, in the sense that they should be better than a purely random estimate. As this happens, the information available to the learner (albeit partially self-generated) begins to improve dramatically. To give a chess-based example, consider a weighted piece counter style of evaluation function. Assume that initially the learner knows nothing of the value of the pieces (they are set to random values, or to zero). After playing many games, the learner notes that 80 percent of the time, the winner has one more queen than the loser. Now, during game-play, the loss of a queen will have a negative effect on the player's value estimate, and it can potentially learn that board positions (i.e., game states) that lead to such a loss should have a low value. This is a more direct form of credit assignment than is usually performed with co-evolutionary learning, in which only win/lose/draw information is fed back at the end of a set of games (see Figure 3).

There have been very few direct comparisons between TDL and co-evolutionary learning, where the algorithms have been used to learn the parameters of the same architecture applied to the same game. So far, only two such studies are known. In a study using neural networks to play Gin Rummy, Kotnik and Kalita [59] found that co-evolution significantly outperformed TDL. Runarsson and Lucas compared TDL with co-evolution for learning strategies for small-board Go [58] and found that, under most experimental settings, TDL was able to learn better strategies than co-evolution and to learn them more quickly. They also found, however, that the very best strategies were learned by a carefully designed co-evolutionary algorithm, with special attention paid to the weight sharing scheme that was necessary to smooth the noise inherent in the fitness function. A slightly less direct (in the sense that he did not perform all of the experiments) comparison was made by Darwen for Backgammon [60] in which he found that co-evolution outperformed TDL for training a perceptron, but found the opposite to be true when learning the parameters of an MLP. An interesting exercise would be to construct (or evolve!) new games that could clearly show cases where one method outperformed the other. In a similar way, it is noted that GP has already been used to evolve simple test functions to illustrate cases in which differential evolution outperforms particle swarm optimization and vice versa [61].

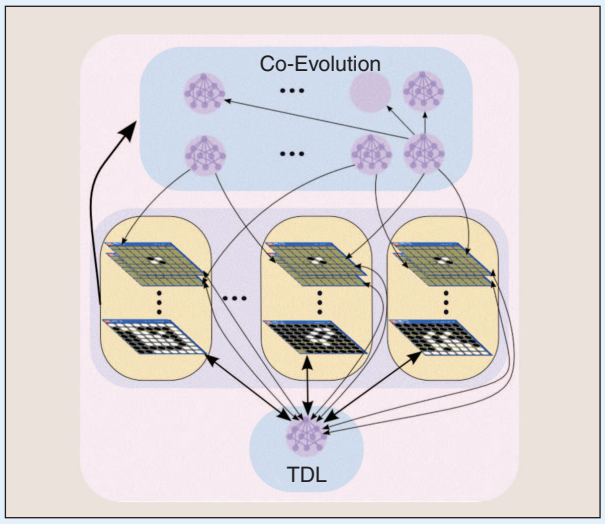


FIGURE 3 Illustration of game/learner interactions for TDL and co-evolution. A co-evolutionary learner gets feedback on the number of games won after playing a set of games while the TDL learner gets feedback after every move of every game. Note also that co-evolution involves a population of players (depicted as neural networks here) whereas TDL typically uses a single player, playing against itself. The bold arrows indicate the reliable feedback that TDL receives at the end of each game on the value of a final position.

from movies and, subsequently, from real life. Graphically this looks like a safe bet, but developing the computational intelligence behind the glossy shells is a more open-ended challenge, and one in which evolution has a big part to play.

Acknowledgments

We thank Thomas Runarsson for discussions related to this article.

We also acknowledge the support of the IEEE and the Engineering and Physical Sciences Research Council (EP/C546024/1). This support was instrumental in establishing the IEEE Symposium on Computational Intelligence and Games conference series. The first conference (co-chaired by the authors) took place in April 2005 and led to this article.

References

[1] D. Fogel, "Using evolutionary programming to create networks that are capable of playing tic-tac-toe," in Proceedings of IEEE International Conference on Neural Networks, San Francisco: IEEE, pp. 875–880, ,1993.

[2] Chips Challenging Champions. J. Schaeffer and J van den Herik (eds), Elsevier, 2002.

[3] M. Muller, "Computer Go," Artificial Intelligence, vol. 134, pp. 145–179, 2002.

[4] B. Bouzy and T. Cazenave "Computer go: An ai-oriented survey," Artificial Intelligence Journal, pp. 39-103, 2001.

[5] K. Stanley, and R. Miikkulainen "Evolving a roving eye for go," In proceedings of Genetic and Evolutionary Computation Conference, pp. 1226–1238, 2004.

[6] N. Richards, D. Moriarty, and R. Miikkulainen, "Evolving neural networks to play go," Applied Intelligence, vol. 8, pp. 85–96, 1998.

[7] T. Graepel, M. Goutrie, M. Krüger, and R. Herbrich, "Learning on graphs in the game of go," in Proceedings of International Conference on Artificial Neural Networks (ICANN), Springer, pp. 347–352, 2001.

[8] M. Enzenberger, "Evaluation in Go by a neural network using soft segmentation," In10th Advances in Computer Games conference, 2003, pp. 97–108, 2003.

[9] R.M. Axelrod, "The evolution of cooperation," BASIC Books, New York, 1984.

[10] http://www.prisoners-dilemma.com/

[11] S. Hamilton and L. Garber, "Deep Blue's hardware-software synergy," IEEE Computer, vol. 30, no. 10, pp 29–35, Oct. 1997.

[12] D.B Fogel, T.J. Hays, S.L. Hahn, and J. Quon, "Further evolution of a self-learning chess program, IEEE 2005 Symposium on Computational Intelligence & Games, G. Kendall and S. Lucas (eds.), Essex, UK, pp. 73–77, Apr. 4–6, 2005.

[13] D.B. Fogel, T.J. Hays, S.L. Hahn, and J. Quon, "A self-learning evolutionary chess program," Proceedings of the IEEE, vol. 92, no. 12, pp. 1947–1954.

[14] G. Kendall and G. Whitwell, "An evolutionary approach for the tuning of a chess evaluation function using population dynamics," Proc. of CEC'01, pp. 995–1002, 2001.

[15] A.L. Samuel, "Some studies in machine learning using the game of checkers," IBM Journal of Research and Development, vol. 3, no. 3, pp 210–229, 1959.

[16] A.L. Samuel, "Some studies in machine learning using the game of checkers," IBM Journal of Research and Development, vol. 44, no. 1/2, Jan/Mar, pp. 207–226, 2000.

[17] J. Culberson, N. Treloar, B. Knight, P. Lu, and D.A Szafron, "World championship caliber checkers program, jonathan schaeffer," Artificial Intelligence, vol. 53, no. 2-3, pp. 273–290, 1992.

[18] J. Schaeffer, One jump ahead : Challenging human supremacy in checkers, Springer-Verlag, ISBN:0387949305, 1997.

[19] http://www.cs.ualberta.ca/~chinook/

[20] K. Chellipilla and D.B. Fogel, "Evolving an expert checkers playing program without using human expertise," IEEE Transactions on Evolutionary Computation, vol. 5, pp. 422–428, 2001.

[21] K. Chellipilla and D.B. Fogel, "Evolution, neural networks, games, and intelligence," Proc. IEEE, vol 87, no. 9, pp 1471–1498, 1999.

[22] D. Fogel, "Blondie24:Playing at the Edge of AI," Morgan Kaufmann, ISBN: 1558607838, 2001.

[23] E. Hughes, "Piece difference: Simple to evolve?," In Proceedings of Congress on Evolutionary Computation, pp. 2470–2473, 2003.

[24] E. Hughes, "Checkers using a co-evolutionary on-line evolutionary algorithm," Proceedings of the 2005 IEEE Congress on Evolutionary Computation, pp 1899–1905.

[25] K.-J. Kim and S.-B. Cho, "Evolving speciated checkers players with crowding algorithm, in Proceedings of the 2002 Congress on Evolutionary Computation, vol. 1, pp. 407–412, 2002.

[26] C. Tristan and B. Helmstetter, "Combining Tactical Search and Monte-Carlo in the Game of Go," Proceedings of IEEE Symposium on Computational Intelligence and Games 2005, pp 171–175.

[27] M. Chung, M. Buro, and J. Schaeffer, "Monte carlo planning in RTS games," Proceedings of IEEE Symposium on Computational Intelligence and Games," pp. 117–124, 2005.

[28] G. Tesauro, "Temporal difference learning and TD-gammon," Communications of the ACM, vol. 38, no. 3, pp. 58–68, 1995.

[29] G. Tesauro, "Comments on co-evolution in the successful learning of backgammon strategy," Machine Learning, vol. 32, no. 3, pp. 241–243, 1998.

[30] M. Ginsberg, "GIB: Imperfect information in a computationally challenging game," Journal of Artificial Intelligence Research, vol. 14, pp. 303–358, 2001.

[31] D.E. Joslin and D.P. Clements, "Squeaky Wheel Optimization," vol. 10, pp. 353–373, 1999.

[32] M. Buro, "The othello match of the year: Takeshi murakami vs. logistello," ICCA Journal vol. 20, no. 3, pp. 189–193, 1997.

[33] J. von Neumann and O. Morgenstern, Theory of games and economic behavior, Princeton NJ: Princeton Univ. Press, 1944.

[34] N. Findler, "Studies in machine cognition using the game of poker," CACM 20(4), pp 230–245, 1977.

[35] D. Billings, D. Papp, J. Schaeffer, and D. Szafron, "Poker as a testbed for machine intelligence research," Advances in Artificial Intelligence, Springer-Verlag, pp. 1-15, 1998.

[36] D. Billings, L. Peña, J. Schaeffer, and D. Szafron, "Using probabilistic knowledge and simulation to play poker," Proc. of AAAI-99, pp 697–703, 1999.

[37] D. Billings, D. Papp, J. Schaeffer, and D. Szafron, "Opponent modeling in poker," Proc. of AAAI-98, pp 493–499, 1998.

[38] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron, "Approximating game-theoretic optimal strategies for full-scale poker," Proceedings of IJCAI-03, (Eighteenth International Joint Conference on Artificial Intelligence), 2003.

[39] D. Koller and A. Pfeffer, "Representations and solutions for game-theoretic problems," Artificial Intelligence, vol. 94, no. 1-2, pp 167–215, 1997.

[40] L. Barone and L. While, "Evolving adaptive play for simplified poker," Proceedings of IEE Intl. Conf. on Computational Intelligence (ICEC-98), pp. 108–113, 1998.

[41] L. Barone and L. While, "An adaptive learning model for simplified poker using evolutionary algorithms," Proc. of Congress on Evol. Comp. (CEC'99), pp. 153–160, 1999.

[42] L. Barone and L. While, "Adaptive learning for poker," In proceedings of the Genetic and Evolutionary Computation Conference, pp. 566–573, 2000.

[43] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron, "The challenge of poker," Artificial Intelligence Journal, vol. 134, no. 1-2, pp 201–240.

[44] C.M. Frayn, "An evolutionary approach to strategies for the game of monopoly®, Proceedings of IEEE Symposium on Computational Intelligence and Games, pp 66–67, 2005.

[45] J. Laird, "Using a computer game to develop advanced AI," Computer, vol. 34, pp. 70–75, 2001.

[46] G. Kendall and S.M. Lucas (eds), Proceedings of IEEE Symposium on Computational Intelligence and Games, IEEE Symposium on Computational Intelligence and Games, 2005.

[47] J. Denzinger, et al, "Dealing with parameterized actions in behavior testing of commercial computer games," Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 51–58, 2005.

[48] N. Georgios Yannakakis and J. Hallam, "A generic approach for generating interesting interactive Pac-Man, Proceedings of IEEE Symposium on Computational Intelligence and Games, pp 94–101, 2005.

[49] K.O. Stanley, B.D. Bryant, and R. Miikkulainen, "Evolving neural network agents in the NERO video game," Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 182–189, 2005.

[50] S.M. Lucas, "Evolving a neural network location evaluator to play ms. pac-man," Proceedings of the IEEE Symposium on Computational Intelligence and Games, pp. 203–210, 2005.

[51] J. Koza, "Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, MA, 1992.

[52] M. Gallagher and A. Ryan, "Learning to play pac-man: An evolutionary," Rule–based Approach In Proceedings of IEEE Congress on Evolutionary Computation, pp. 2462–2469, 2003.

[53] http://www.robocup.org/

[54] I. Tanev, "Evolution of the driving styles of anticipatory agent remotely operating a scaled model of racing car," Proceedings of Congress on Evolutionary Computation, pp. 1891–1898, 2005.

[55] J. Togelius and M. Lucas Simon, "Evolving controllers for simulated car racing," Proceedings of the Congress on Evolutionary Computation 2005, pp 1906–1913.

[56] D. Floreano, T. Kato, D. Marocco, and E. Sauser, "Coevolution of active vision and feature selection," Biological Cybernetics, vol. 90 pp. 218–228, 2004.

[57] S.Y. Chong, M.K. Tan, and J.D. White, "Observing the Evolution of Neural Networks Learning to Play the Game of Othello," IEEE Transactions on Evolutionary Computation, vol. 9, no. 3, pp. 240–251, June 2005.

[58] T.P. Runarsson and S.M. Lucas, "Co-evolution versus self-play temporal difference learning for acquiring position evaluation in small-board Go," Accepted for IEEE Transactions on Evolutionary Computation—Special Issue on Evolutionary Computation and Games, Summer 2005.

[59] C. Kotnik and J. Kalita, "The significance of temporal-difference learning in self-play training: TD-rummy versus EVO-rummy," In Proceedings of the International Conference on Machine Learning (ICML), pp. 369–375, 2003.

[60] P.J. Darwen, "Why co-evolution beats temporal difference learning at backgammon for a linear architecture, but not a non-linear architecture," in Proceedings of the 2001 Congress on Evolutionary Computation CEC2001. IEEE Press, pp. 1003–1010, 2001.

[61] W.B. Langdon and R. Poli, "Evolving Problems to Learn about Particle Swarm and other Optimisers," Proceedings of the IEEE Congress on Evolutionary Computation, pp 81–88, 2005.