An Adaptation of Social Learning in Evolutionary Computation for Tic-Tac-Toe

Razali Yaakob[1] and Graham Kendall[2]

[1]Faculty of Computer Science and IT, University Putra Malaysia,
43400 UPM Serdang, Selangor, Malaysia.
[2]School of Computer Science, The University of Nottingham,
Wollaton Road, Nottingham NG8 1BB, United Kingdom

Summary
This paper investigates an integration of individual and social learning, utilising evolutionary neural networks, in order to evolve game playing strategies. Individual learning enables players to create their own strategies. Then, we allow the use of social learning to allow poor performing players to learn from players which are playing at a higher level. The feed forward neural networks are evolved via evolution strategies. The evolved neural network players play first and compete against a nearly perfect player. At the end of each game, the evolved players receive a score based on whether they won, lost or drew. Our results demonstrate that the use of social learning helps players learn strategies, which are superior to those evolved when social learning is not utilised.

Key words:
Evolutionary strategies, neural network, individual learning, social learning, Tic-Tac-Toe

1. Introduction

When playing a game, humans use a variety of techniques in order to develop strategies to defeat other humans. Humans can improve their strategy by themselves or through the experience of competing against others. Humans can also copy strategies from better players and develop their own strategy based on this copy. Therefore, humans are able to learn through a mixture of individual and social learning.

Games have been used as a test bed for artificial intelligence since the 1950's where, among others, pioneering work was carried out by Alan Turing [1], Claude Shannon [2] and Arthur Samuel [3]. Several games have now reached the stage where a computer implementation is able to play at a higher level than most (if not all) human players (see, for example, Deep Blue [4], Chinook [5, 6, 7], Logistello [8] and TD-Gammon [9]).

Deep Blue is an automated Chess player which defeated Gary Kasporov in May 1997. It exploited a database of over 700,000 Grandmaster chess games from previous matches (including the matches against Kasparov in 1996 and 1997) [4], which included an extended opening book database.

Chinook, developed by Jonathan Schaeffer's team at The University of Alberta, won the world checkers title in 1994 [6]. It is another good example of an automated game that incorporates human knowledge. There are four aspects that contributed to the success of Chinook; these being search, an evaluation function, a database of endgame positions and an opening book.

The focus of this work is to evolve an automated player that learns to play a game without incorporating any knowledge. We draw motivation from Blondie24 [10], which learnt strategies for playing Checkers by using a combination of an evolutionary algorithm and artificial neural networks. In this paper, we will integrate social learning within a Blondie24 style framework and investigate its effectiveness using the game of Tic-Tac-Toe. Unlike Blondie24, the strategies do not compete against each other, instead the strategies play against a nearly perfect player, drawing inspiration from [11], where the evolved player against a nearly perfect player. Our work is also motivated by [12, 13], where a simulated stock market used co-evolving neural networks, which evolved through a process individual and social learning.

Further reading about the literature evolutionary computation and games can be found in [14].

2. Social Learning

Social learning research has been largely based in the context of agent-based computational economics [12], [15], and according to [15], in individual learning, the agents learn exclusively from their own experience and in social learning, the agents learn from the experience of other agents. In this work, the player will learn individually based on experience by playing against a nearly perfect player[1] and after a period of time, we allow the player to "learn" from each other in a process we call social learning.

[1] is a beatable player which will move randomly 10% of the time

Manuscript received September 5, 2009
Manuscript revised September 20, 2009

In a context of automated game playing, individual learning is defined as a player which learns, and develops, their own strategy through the experience of playing against other players. In this type of learning, the players never copy a strategy from other players and never replace its strategy with a new (perhaps random) strategy. In contrast, the idea of social learning is to give the player the opportunity to copy or create a new strategy, replacing their current strategy. The player still has the opportunity to evolve its own strategy through individual learning, but if the strategy is not good enough, or the player is not happy with their current strategy, the player can choose to either request a superior strategy or create a new random strategy.

We create a pool to retain the best strategy at a certain period. We call this the social pool and this represents the best strategies from the population. We then make the social pool available to those players which are not performing well. This concept is quite similar to the concept of hall of fame [16], where all the best evolved players at every generation are maintained and can be used in the future. During social learning, the player can choose to replace their current strategy with a (hopefully) better strategy, drawn from the social pool. All strategies in the social pool maintain a score, which is updated through time. Social learning proceeds as follows:

1. Rank the players from the highest to the lowest.
2. Copy the best player (or players, if more than one) to the social pool.
3. For the remaining players, there are two possibilities,
   a) If the players are satisfied with their current strategy (based on the current score), keep that strategy, else
   b) If the players are not satisfied with their current strategy, three alternatives are available,
      i) Copy a strategy from the pool, or
      ii) Create a new random strategy, or
      iii) Retain the current strategy

Social learning in this work is not the same as an island model in evolutionary computation. In an island model [17], there are several populations, each of which is evolved independently. Migration between the sub-populations occurs where the best player from the sub-population is migrated to another sub-population. A sub-population might only receive a better strategy from another sub-population and may not create any new strategies. In social learning, the individual players have a chance to receive a better strategy, retain their current strategy or create a new random strategy.

3. Design Of Learning Techniques

There are two experimental methodologies in this paper, i.e. with and without social learning. We hypothesise that social learning provides a superior environment in which strategies can evolve. The players outside of the social pool are called individual players, and these players attempt to develop their own strategies, drawing from the social pool at certain times.

3.1 Design Without Social Learning

The experimental setup *without social learning* mimics that in [18], with some minor modifications. This experiment will be used as a comparison for the experiment which includes social learning. The difference between [18] and our setup is as follows:

- The revised neural network architecture consists of nine input nodes, nine hidden nodes and an output node (see Fig.1). All hidden nodes are connected with bias (previously we did not have bias, but found through experimentation that it is necessary). In [18], the neural networks have nine input and output nodes, and the number of hidden nodes depends on mutation during the experiment.
- The architecture in [18] is mutated during the experiment, whereas, there is no mutation on the network architecture in this work.
- In [18], the neural network chooses a best move based on the highest value from one of the nine output nodes from the neural network. However, in this work, the evolved neural networks will search for the best move using 1-ply search using the following algorithm:
-

1. Read the current state of the game.
   a. Put a marker 'X' on any available square.
   b. Pass the new current state to the neural network.
   c. Store the output value.
   d. Remove the marker that was placed in step 1a.
   e. Repeat steps 1a to 1d until all available squares have been evaluated by the neural network and.
2. Select the highest output as the best move.

The *without social learning* algorithm is as follows:

1. Initialise a random population of neural networks, P_i, where P is a neural network player, $i = 1, ..., N$, and N is a total number of neural networks in the population.
2. Each strategy has its associated self-adaptive parameter, s_i, $i = 1, ..., N$, and are initialised randomly.
3. Generate an offspring for each parent, where for each P_i, $i = 1, ..., N$, an offspring, P'_i is created by using Eq. 1a and 1b. The total neural networks now is $N+N$:

$$s'_j = s_j.exp(\tau.N_j(0, 1)) \quad (1a)$$

$$w'_j = w_j + s'_j.N_j(0, 1); \quad (1b)$$

4. Each neural network players play against the nearly perfect player (see [18] for this algorithm).
5. In each game, the neural network players receive a score based on whether they win, lose or draw. This score is used as a fitness and is used for selection.
6. Repeat steps 4 and 5, four times.
7. At the end of each generation, the *N* strategies (neural networks) with the highest scores are selected as parents and retained for the next generation. These parents are then mutated to create another *N* offspring using Eq. 1a and 1b.
8. Repeat steps 4 to 7 until the stopping criterion is reached (i.e. generations = 1,000).

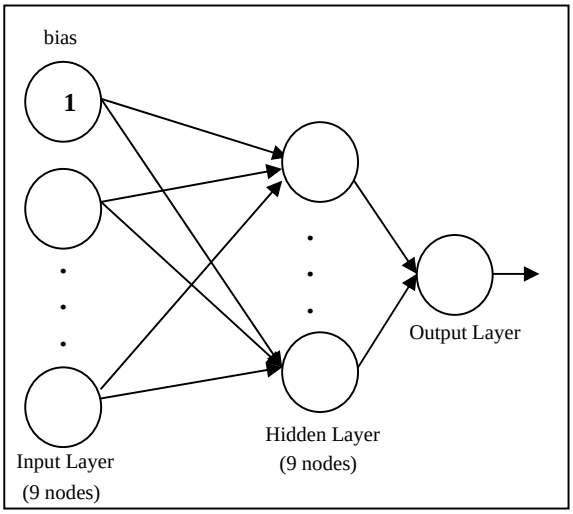

Fig.1: The new structure of the neural network

We choose four iterations in step 6 so as to make sure the experiment will not take too long to run, but at the same time it gives an opportunity for the player to play several times. In this paper, the neural network players receive +1 for win, -1 for lose and 0 for draw. Since each player will play against the nearly perfect player 32 times (eight second possible moves and four trials), the highest score for each player, at each generation, is 32 points.

3.2 Design With Social Learning

The design is similar to the algorithm without social learning, including the structure of the networks, the move selection and calculating the fitness. However, in the experiment with social learning, social learning activities occur at every k^{th} generation. The fitness calculated before the social activity is called the individual fitness.

There are several issues that we need to address for this algorithm. The first is how to avoid copying the same strategy too many times, especially when the social pool does not have many entries. To solve this problem, we limit the number of times a strategy can be copied from the social pool to be four, at a given generation.

The other issue is how to maintain variation in the population and also have a significant number of better strategies in the population? We propose two phases of social learning, which we call minor and major social learning. The objective of minor social learning is to copy the best player at a certain generation to the social pool without carrying out any other social activities. The algorithm for this technique is as follows:

1. Follow the steps 1 until 6 as in Section III-A
2. If (minor social learning)
   - Keep the best player or players (if more than one) in the social pool.
   - Select *N* strategies (neural networks) with the highest scores as parents and retain for the next generation. These parents are then mutated to create another *N* offspring using Eq. 1a and 1b,
3. If (major social learning),
   a. Keep the best player or players (if more than one) in the social pool.
   b. For the rest of the players, there are two possibilities,
      i) If the players are satisfied with their current strategy, keep the current strategy, else
      ii) If the players are not satisfied with their current strategy, three alternatives are available,
      - Copy a strategy from the pool, or
      - Create a new random strategy, or
      - Retain the current strategy.
4. If (not minor or major social learning)
   - Select *N* strategies (neural networks) with the highest scores as parents and retain for the next generation. These parents are then mutated to create another *N* offspring using Eq. 1a and 1b,
5. Repeat steps 4 until 6 (as in Section III-A) until the stopping criterion is reached (generations = 1,000).

The algorithm for the activities in social learning is as follows:

1. Normalise all the 50 evolved players, *i*, between 0.0 and 1.0 based on their individual fitness, *F*, using Eq. 2, and sort in descending order.

$$V_i = MIN + (MAX - MIN) \times (F_i - d_{min}) / (d_{max} - d_{min}) \quad (2)$$

where

V_i is the normalised value for player i.

MIN and MAX is the lowest and highest value for the range of the normalized value (0.0 and 1.0 respectively).

F_i is the fitness of player i before being normalised.

d_{min} and d_{max} is the lowest and highest score in the current population among all players.

2. If (V_i = 1.0) and the strategy has never been published,
   - Publish the player i into the social pool and assign pool score, P_j , using Eq. 3, where j is an index of the best player in the social pool.
3. If (V_i = 1.0) and the strategy has previously been published to the social pool,
   - Do not publish the player i but update the pool score, P_j using the Eq. 3.
4. If (V_i < 0.9) and (V_i < 1.0),
   - The player i is satisfied with their current strategy and the player stays in the population for the next generation.
5. If (V_i < 0.9),
   - There are three possibilities, one of which is chosen randomly:
     a) Case 1,
        - The player chooses to replace their current strategy with a strategy from the social pool.
        - Based on the scores assigned to each player in the pool, P_j , roulette wheel selection is used to select a strategy from the pool.
     b) Case 2,
        - The player chooses to retain with their current strategy for the next generation.
     c) Case 3,
        - The player chooses to replace their current strategy with a new randomly created strategy.

The pool score in the social pool for each best player, P_j , is updated every time social learning occurs using Eq. 3, where the value of the pool score is based on how long the player has been in the pool and what was its individual score, F_i when it was published into the pool.

$$P_j = \frac{g_j}{\sum_{k=1}^{g} g_k} \times \exp\left(I_j \times C_j\right) \quad (3)$$

Where,

- j is an index of the best player in the social pool,
- g is a generation when published into the social pool,
- g_j is a generation for player j when published into the social pool,
- I is the normalised value of individual score, F, between 1 and 10,
- C is a sum of times player j has been reused.

The objective of this function is to give better players, which have recently been introduced (or updated) to the pool, a higher probability of being selected as replacements for poorly performing individual players. The age of the player is controlled by $g_j/(\sum_{k=1}^{g} g_j)$, where the previous best players will receive less points than the current best players. However, the final calculation for calculating the pool score is also dependent on the individual score of the best player when publishing into the social pool and how many times they have been reused[2] since they were first published. Table 1 shows an example of a social pool for trial number two where social learning occurs at every 100th generation.

Based on Table 1, the player in social pool with j = 0, receives very small value of pool score since this player has the lowest individual score and also has already been in the pool from generation 100. Players j = 11, 12 and 13 have the same individual score, i.e. 30. However, since they have been published into the social pool from different generations, the player recently published into the social pool receives highest pool score.

4. Results

There are two types of experiments in this work, which have a different number of generations, i.e. 1,000 and 5,000.

The objective of running for 5,000 generations is to see what will happen to the experiment with social learning when we run for a longer period. All experiments were run 50 times for 1,000 generations. They were run on a computer Intel(R) Pentium(R) 4, CPU 3.20GHz, and 2.00 GB of RAM.

Table 1: An Example Of Social Pool

Index Player j	Pool Score P	Individual Score F	Generation g	Reused C
0	0.0362438	10	100	0
1	0.279615	13	200	0
2	3.97937	18	300	0
3	3.97937	18	300	0
4	32.0984	22	400	0
5	25.5836	21	500	0
6	25.5836	21	500	0
7	185.726	25	600	0
8	185.726	25	600	0
9	185.726	25	600	0
10	339.822	26	700	0
11	2349.49	30	800	0
12	2643.18	30	900	0
12	2936.86	30	1000	0

[2] The strategy that has been published in the social pool and receives V_i = 1 on the next social learning

4.1 Results for the Experiment with Number of Generations= 1,000

In each trial, we kept the highest score at each generation, $y_1,\ldots, y_{1000}$. Figure 2 shows the mean of the highest score (in percent) from the 50 trials (Eq. 4) for the experiment with (marked WSL on the graph) and without social learning (marked NSL on the graph). Based on the Figure 2, at the end of the experiment without social learning, the highest mean score is about 75%.

$$Y_n = \left(\frac{\sum_{i=1}^{50} y_n}{50}\right) \times 100 \qquad (4)$$

Where,

- n is a generation, n = 1,...,1000,
- y_n is a highest score at the generation n,
- Y_n is an average highest score at the generation n,

In the experiment with social learning, at every 50th generation and 100th generation, the minor and major social learning occur respectively. The experiments (with and without social learning) took about 7 hours to complete 1,000 generations. Fig. 2 shows the mean of the highest score (in percentage) from 50 trials (Eq. 4) on the extended experiment with social learning versus the experiment without social learning. Based on Fig. 2, the experiment with social learning is constantly better than the experiment without social learning from the beginning until the end, where at the end of this experiment, the score for the experiment with social learning is about 80% and 74% for the experiment without social learning.

Looking at Fig. 2, the hypothesis that the experiment with minor and major social learning is shown.

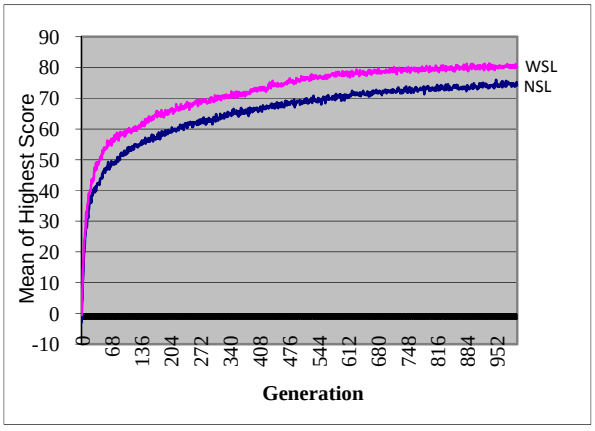

Fig. 2: An experiment with social learning vs. experiment without social learning where number of generations=1,000

4.2 Results for the Experiment with the Number of Generations = 5,000

Fig. 3 shows the mean of a highest score (in percent) from 30 trials (the same formula as Eq. 4 was used but for only 30 trials) for the individual player at each generation for the experiment with and without social learning (marked as WSL and NSL respectively). The objective is to investigate whether social learning still can give better performance than without social learning.

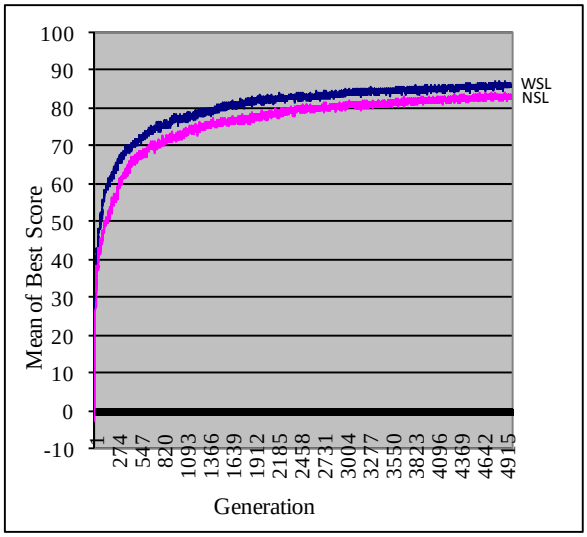

Fig. 3: The mean of best score for 50 trials at each generation (generations = 5000)

Based on Fig. 3, the experiment with social learning produces statistically different and better results when compared to the experiment without social learning, with 95% confidence level.

Table 2 shows the details of social pool at generation 100 for the trial number three of the experiment run 5,000 generations. In this experiment, minor social learning occurs at the first generation and also at every 50th generation. There are two players that have the same score at the first generation and three players at the 50th and 100th generations. These show that the social pool have eight different strategies at the first time major social learning occurs. Therefore, at this level, the number of strategy that can be copied from the social pool is about 32 strategies. However, because of two players have negative values, there are only a maximum of 24 strategies that be copied.

Table 2: An Example Of Social Pool At Generation 100 Of The Experiment Running For 5,000 Generations

Index Player	Pool Score	Individual Score	Generation	Reused
j	*P*	*F*	*g*	*C*
0	-0.018	-3	1	0
1	-0.018	-3	1	0
2	1522.86	15	50	0
3	1522.86	15	50	0
4	1522.86	15	50	0
5	4873.11	16	100	0
6	4873.11	16	100	0
7	4873.11	16	100	0

Table 3 shows the distribution activities that been chosen by the poor strategies at generation 100. There are 14 strategies (or about 28%) that have been chosen for the next generation, from the social pool. With reference to Table 3, the *number of poor strategies* reflects the number of individual players that have $V_i < 0.9$; *number of copies* is number of individual players that copy a strategy from the pool; *number of new random* is number of individual players that replace the current strategy with random strategy; *number of stays* is number of individual players that choose to retain their current strategy.

Table 3 : Distribution Activities Of Poor Strategies For Trial Number Three Of The Experiment Running For 5,000 Generations

Generation #100	
Number of poor strategies:	43
Number of copies:	14
Number of new random:	14
Number of stays:	15
Generation #200	
Number of poor strategies:	49
Number of copies:	17
Number of new random:	19
Number of stays:	13
Generation #300	
Number of poor strategies:	47
Number of copies:	14
Number of new random:	15
Number of stays:	14

Based on the above results, we manage to increase the variety of strategies in the social pool and also increased the number of better strategies in the population. Even though, the end scores for the experiment with social learning are not significantly better than the experiment without social learning, but the experiment with social learning are consistently better than without social learning.

Based on Table 1 and Table 2, we found that minor social learning did help the population to have more superior strategies in the population.

5. Conclusions

In this work, we have run several experiments to investigate the effect of an integration of individual and social learning in the learning process to play the game of Tic-Tac-Toe. We also run an experiment without social learning for comparison. Learning has occurred in all experiments, with the inclusion of social learning producing superior strategies.

As mentioned earlier, one of the issues from our early studies was how to maintain variety of better player in the population pool. We have proposed two phases of social learning (minor and major social learning). The objective is to increase the variety of strategies in the social pool from the very start of the evolution, giving more strategies for the individual players to copy from. Table 2 has shown that the proposed technique has increased the number of strategies at the beginning of the experiment and Fig. 2 and Fig. 3 have shown that the experiment with social learning is consistently better than the experiment without social learning.

In conclusion, two phases of social learning has improved the evolution of game playing strategies. From observation, we must have as many best players in the social pool as possible before the social activities take place. In future work, our focus will be on increasing the number of superior strategies in the social pool, in the hope that it will improve the performance even further. We also plan to use more complex game to ascertain of this evolutionary process is able to scale up.

References

[1] A.M. Turing. Digital computers applied to games. Pages 286–310. Pittman, London, UK, 1953.
[2] C.E. Shannon. Programming a computer for playing chess. Philosophical Magazine, March 1950.
[3] A.L. Samuel. Some studies in machine learning using the game of checkers. IBM Journal of Research and Development, 3(3):210–229, 1959.
[4] M. Campbell, A.J. Hoane, and F.-H. Hsu. Deep blue. Artificial Intelligence, 134:57–83, 2002.
[5] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron. A world championship caliber checkers program. Artificial Intelligence, 53(2-3):273–290, 1992.
[6] J. Schaeffer, R. Lake, and P. Lu. Chinook: The world man-machine checkers champion. AI Magazine, 17(1):21–30, 1996.
[7] J. Schaeffer. The Games Computers (and People) Play, pages 189–266. Academic Press, 2000.
[8] M. Buro. The othello match of the year: Takeshi murakami vs. logistello. International Chess Computer Association Journal, 20(3):189–193, 1997.
[9] G. Tesauro. Temporal difference learning and TD-gammon. Communications of the ACM, 38(3):58–68, 1995.
[10] D.B. Fogel. Blondie24: Playing at the Edge of AI. Morgan Kaufmann, San Florida, California, 2002.
[11] D.B. Fogel. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. IEEE Press, Piscataway, New Jersey, 1995.
[12] G. Kendall and Y. Su. The co-evolution of trading strategies in a multi-agent based simulated stock market through the integration of individual and social learning. In Proceedings of IEEE 2003 Congress on Evolutionary Computation, pp. 2298-2305, 2003.

[13] G. Kendall. Y. Su. Imperfect evolutionary systems. IEEE Transactions on Evolutionary Computation, 11(3):294-307, 2007.
[14] Lucas S.M. and Kendall G. Evolutionary Computation and Games (invited review). IEEE Computational Intelligence Magazine (IEEECIM), 1(1):10-18, 2006.
[15] N. Vriend. An illustration of the essential difference between individual and social learning, and its consequences for computational analysis. Technical report, Queen Mary and Westfield College, University of London, 1998.
[16] C.D. Rosin and R.K. Belew. New methods for competitive coevolution. Evolutionary Computation, 5(1):1–29, 1997.
[17] C. Spieth, F. Streichert, N. Speer, and A. Zell. Utilizing an island model for EA to preserve solution diversity for infering gene regulatory networks. In Congress on Evolutionary Computation, volume 1, pages 146–151, June 2004.
[18] D.B. Fogel. General problem solving: Experiments with tic-tac-toe. In Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, pages 228–248. IEEE Press, 1995.

Razali Yaakob obtained his PhD from University of Nottingham in 2008. Currently, he is a senior lecturer at the Faculty of Computer Science and IT, Universiti Putra Malaysia. His research areas include artificial neural network, pattern recognition, and evolutionary computation in game playing.

Graham Kendall is a member of the Automated Scheduling, Optimisation and Planning Research Group at the University of Nottingham. He is a member of the UK Engineering and Physical Sciences Research Council (EPSRC) Peer Review College and is an associate editor of eight international journals. He also chairs the steering committee of the Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA). Prof Kendall has been a member of the Programme (or refereeing) committees of over 100 international conferences over the last few years. During his career he has edited/authored 10 books and has published over 100 refereed papers. His research interests include: hyper and meta heuristics, evolutionary computation, adaptive learning (with an emphasis on game playing), heuristic development, optimisation, scheduling and artificial intelligence.