Learning to Play a Highly Complex Game from Human Expert Games

Tony Kråkenes and Ole Martin Halck

Norwegian Defence Research Establishment (FFI)
P.O. Box 25, NO-2027 Kjeller, Norway
{tony.krakenes,ole-martin.halck}@ffi.no

Abstract. When the number of possible moves in each state of a game becomes very high, standard methods for computer game playing are no longer feasible. We present an approach for learning to play such a game from human expert games. The high complexity of the action space is dealt with by collapsing the very large set of allowable actions into a small set of categories according to their semantic intent, while the complexity of the state space is handled by representing the states of collections of pieces by a few relevant features in a location-independent way. The state–action mappings implicit in the expert games are then learnt using neural networks. Experiments compare this approach to methods that have previously been applied to this domain.

1 Introduction

This paper describes the application of machine learning techniques to the problem of making a software agent that plays a highly complex stochastic game. The game we consider, *Operation Lucid*, belongs to the class of two-person zero-sum perfect-information stochastic games. It has been designed as a simplified military land combat model, with rules representing central concepts such as movement (and uncertainty in movement), logistics, and of course combat itself, including the asymmetry between attacking and defending a location.

Our studies are concerned with the application of artificial intelligence techniques to decision making in combat models, and in this research Operation Lucid is being used as an environment that captures the important general properties of such models, while allowing us not to get bogged down in unnecessary detail. The insights and results gained in this way can then be used in the development and improvement of full-scale combat models.

The problem of game playing has been extensively studied in machine learning research. A number of papers describing state-of-the-art developments in this field are collected in [1]; this reference also contains a survey of machine learning in games [2]. However, in most of the games studied in this body of research, the main challenges are different to those posed by Operation Lucid, making several of the standard techniques useless for our problem.

T. Elomaa et al. (Eds.): ECML, LNAI 2430, pp. 207-218, 2002.

If we regard Operation Lucid as a decision-making problem in a combat simulation context, related research is somewhat thinner on the ground. Recent work includes [3], in which a genetic algorithm is applied to a force allocation problem not entirely dissimilar to ours, and [4], which describes how a knowledge-intensive agent is used for evaluating military courses of action.

The remainder of the paper is organized as follows: Section 2 describes our problem domain. In Section 3, we describe our approach in dealing with the high complexity of the problem in order to make a game-playing agent; a more detailed description of the implementation of the agent is the subject of Section 4. Section 5 presents some experimental results, and Section 6 concludes the paper.

2 The Game of Operation Lucid

In this section we present our problem environment – the game of Operation Lucid – and describe some of the properties that make it both interesting and very challenging. A fuller description of the game is given in [5].

2.1 Definition and Rules of Operation Lucid

In short Operation Lucid is a two-person stochastic board game where the two players start off with their pieces in opposing ends of the board, as shown in Figure 1. One player, named Blue, is the attacker. Blue starts the game with fifteen pieces; his aim is to cross the board, break through – or evade – his opponent's defence, and move his pieces off the board into the goal node. The defending player, Red, starts with ten pieces; his task is to hinder Blue from succeeding. The result of the game is the number of pieces that Blue manages to get across the board and into the goal node; thus, there is no "winner" or "loser" of a single game. The rest of this section describes the rules of the game.

The game of Operation Lucid is played in 36 turns. At the start of each turn, the right to move pieces is randomly given to either Blue or Red, with equal probabilities. The side winning this draw is allowed to move each piece to one of the neighbouring nodes (that is, to a node that is connected to the piece's current node by an edge) or leave it where it is. The side losing the draw naturally does not get to move any pieces in that turn. The movement of the pieces is subject to two restrictions:

- When the move is finished, no node can have more than three pieces of the same colour.
- Pieces cannot be moved from nodes where the player is defined as the attacker (see below).

Whenever Blue and Red pieces are in the same location at the end of a turn, combat ensues, and one of the pieces in that node is lost and taken out of the game. A weighted random draw decides which side loses a piece. In a node having combat, the player last entering is defined as the *attacker* of that location, while the other part is defined as the *defender* of the location. The weighted random draw is specified by the probability that Blue wins, that is, that Red loses a piece. This probability is given

by the fraction *(Blue strength)/(Blue strength + Red strength),* where a player's *strength* in a node equals the number of own pieces in that node, modified in accordance to two rules. Firstly, the defending player in the node gains one extra point of strength. Secondly, if the Blue player does not have an unbroken path of nodes with only Blue pieces leading from the combat node to one of Blue's starting positions (a *supply line*), Blue loses one point of strength.

The game ends when the 36 turns are completed, or when Blue has no pieces left on the board. The result of the game is the number of Blue pieces that have reached the goal node.

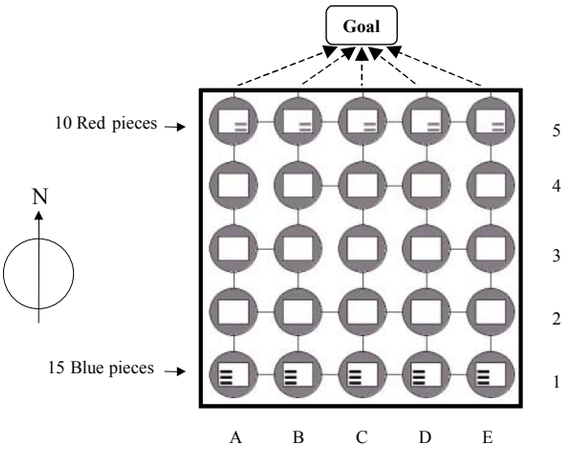


Fig. 1. The board of the game Operation Lucid, with the pieces placed in their initial positions

2.2 A Combat Modelling Interpretation of the Game

Operation Lucid was designed to capture important aspects of military land combat modelling; in particular, it represents a scenario where the goal of one side is to break through enemy defence to reach a certain location. Movement of the pieces on the board naturally represents the movement of force units in a terrain; the stochastic ordering of Blue and Red moves is intended to capture the uncertainty inherent in manoeuvring in possibly unfamiliar territory.

The rules for determining the result of combat naturally take into account the numerical strength of each side in the area of combat. In addition, they represent the advantage of being the defender of a location; this advantage is due to the defender's opportunity to prepare himself and the environs for resisting attacks. The rule regarding Blue supply lines models the effects of logistics; an invading force will need a functioning line of supplies back to his home base to be able to perform well in enemy territory.

2.3 The Complexity of the Problem

A seemingly obvious way of playing Operation Lucid is by evaluating (in some way) each legal move in the current game state, and then choosing the one with the best

evaluation. This would reduce the problem of constructing a player agent to the problem of evaluating moves in given game states. This method is generally not feasible, however, as the number of possible moves in each state tends to be huge. A player may allocate one of at most five actions (stand still or move in either of four directions) to each of at most fifteen pieces, so an upper bound on the number of legal moves is $5^{15} \approx 3 \cdot 10^{10}$. If we assume that a computer generates one thousand possible moves each second (a reasonable assumption according to our experience), it might take up to a year to enumerate all legal moves in one state.

In typical game states the number is usually far lower than this – the player may have fewer than fifteen pieces left, and each of the pieces may not be free to perform all five actions. Also, a lot of the legal moves are equivalent, as all pieces of the same side are interchangeable. From the initial position, for instance, the number of possible non-equivalent moves for Blue is 60,112 (disregarding equivalence by symmetry). In intermediate states of the game the number of legal moves increases quickly, and is generally far too large for an exhaustive enumeration. Thus classical methods based on enumeration and evaluation of moves are infeasible in this domain.

3 Handling the Complexity of the Problem

How can the problem of the game's high complexity be dealt with efficiently? In this section, we briefly mention some previous work on designing player agents for Operation Lucid, and describe the considerations that led to the agent design that is the main subject of this paper.

Our efforts so far have mostly been focused on developing Blue agents, and this is the case in the present work as well. Building good Blue agents tends to be a more challenging task than building Red ones, since the nature of the game requires Blue to be the more creative and active side. However, it is usually a minor task to adjust the algorithms to fit a Red player as well.

3.1 Previous Approaches

As explained above, the usual way of making computers play games, that is, generating all possible moves and evaluating which is the best, is not feasible in Operation Lucid. We therefore looked to the way humans play games in order to create good player agents for the game. Humans generally do not test all available moves; rather, we decide on a goal, and form a plan we believe will help us reach this goal.

Limiting the Set of Evaluated Moves. In one main approach we have followed, we kept part of the evaluative approach to game-play, but limited the number of moves to be evaluated to a tractable level. This was achieved by imposing constraints on the desired number of pieces in various areas of the board – these constraints could be seen as a representation of a plan for how to play. The challenging part of this procedure was deciding on which constraints to impose, so that the problem became tractable in size without eliminating good candidate moves. This approach in general, and in particular a Blue agent that used self-trained neural networks for evaluating moves, is described further in [6]. The constraint-based approach was successful in

that it yielded agents with fairly good performance. The main disadvantage of this method was that the constraints that had to be imposed in order to keep runtime at a reasonable level limited the range of play of the agents. In effect, this approach meant that the main strategy of play was entered by hand through the move constraints, while the move evaluator performed limited tactical optimizations.

Simple Imitation Learning. In our second main approach, we took the idea of playing like a human literally, and designed an agent that played by lazy imitation learning. We made a lookup-table of game states and corresponding expert moves; the agent used this database by trying to imitate the action taken in the stored game state that was most similar to the one at hand. The challenging part of this method was to define a suitable similarity metric between states, and especially to deal with the unwanted side effects arising from states not being exactly equal.

Our work with this kind of pure imitation learning produced rather disappointing results. The main reason for this was that the player very quickly found itself in game states with no very close match in the expert database, so that the move taken in the most similar state was not applicable in the present game state. It was realized that these problems were due to the fact that similarity, both of states and of moves, were seen at what can be called a *syntactical* level. At this level, the actual position and movement of the pieces in each single node were the basis of the agent's behaviour, without any semantical notion of the role of the pieces in the game. The result was that this syntactical imitation would often require some pieces to perform unfeasible moves, while other pieces were left idle.

One obvious fix to this problem would be to increase the size of the expert database, hoping to cover a greater range of board positions. This approach however requires man-hours, and in our case doubling the database showed only marginal improvement during play at the cost of increasing the runtime considerably.

3.2 Current Approach

Our experiments with pure imitation learning showed that the semantics of the problem domain would have to be addressed to some degree in order to achieve good and efficient performance. One way of doing this could be by following the methodology used in case-based reasoning (see e.g. [7]), where retrieval of previous cases from a database is combined with symbolic reasoning based on a semantic model of the domain[1]. Case-based reasoning has previously been applied to games such as chess [9] and Othello [10], as well as many real-world problems. The main disadvantages of this approach is that constructing such a domain model is a difficult and time-consuming task, and that frequent case retrievals – as is the case in game playing – may be costly in terms of runtime.

These considerations led us to the conclusion that we required an agent design that imitates the expert games on a more abstract level than as single positions and movements of pieces, while not depending on a full semantic model of the game. In

[1] We follow the terminology of Mitchell [8], where the term "case-based" is reserved for a subset of the broader set of "instance-based" methods, namely those using richer instance representations than simple feature vectors.

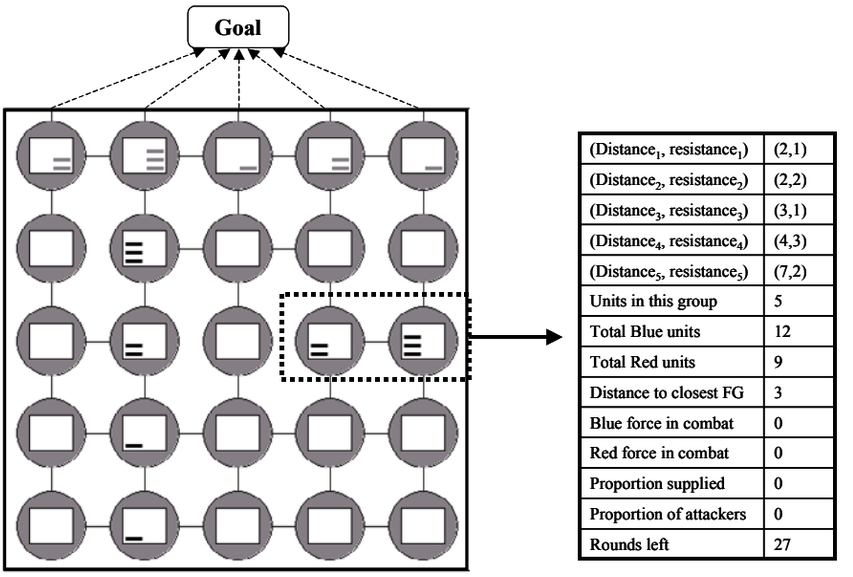
the following, we describe how this was done in the case of moves and game states respectively, and explain how the agent chooses its moves based on the expert games.

Moves. As in the simple imitation learning above, our current approach to handling the complexity of the problem is to have an agent capable of mapping directly from game states into the proper moves to take. In order to make this work well, the action space must be reduced considerably in size. To this end, we collapsed the action space from the syntactical level consisting of all possible combinations of single-piece moves to a semantic level featuring only a few move categories. The move categories are symbols (e.g. *attack, outflank, breakSupply*) describing the overall character or *intent* of the move. The move categories are not disjunctive – they are defined so that a move may be labelled into more than one category.

Game States. Reducing the action space in the manner described above raises another question: which pieces are to perform which types of action? Different pieces in a given expert game may be used for different intentions; this made it desirable to work with sub-collections of pieces sharing common intentions, rather than with all the pieces collectively. We call these sub-collections sharing common intentions *force groups* (FGs). Common intentions usually coincide with co-location of pieces on the board; this led us to define a FG as a collection of pieces of the same colour interconnected with each other but not with other friendly pieces. The pieces in a FG should act together pursuing the intentions of the group, and should differing intentions occur within a FG, it may break up, forming smaller FGs individually regaining conformity of intentions.

The main advantage of introducing the concept of FGs is that it allows collapsing the game state space considerably. Previously, the game state was represented by the number of pieces for each side in each individual node, the number of remaining rounds and which side defended each combat node. We now employ a FG-centric state representation where individual pieces and nodes, even the board geography itself, are no longer of direct interest. What is of interest is a set of aggregated features like path lengths to the goal node, resistance along these paths, the number of own pieces in the FG and in total, the number of enemy pieces, distance to neighbouring FGs (if any), combat strength of the FG (including supply and defender status) and remaining rounds. Each FG in a game state has its individual, location-independent state perception, on which it bases its actions. This means that FGs located on different parts of the board are considered similar if their environs are similar in terms of the FG-centric state representation. The representation we use is illustrated in more detail in Figure 2.

State–Action Mapping. Using the state and move representation just presented, we could repeat the lookup-table method described in Section 3.1 for mapping observed game states into actions to perform. The problem of finding a good distance metric for comparing FG states in order to choose the most similar one would then still remain. Instead, we chose to leave the lazy-learning design, and used the expert games to train a set of neural network classifiers for state–action mapping. In this way, the mappings implicit in the expert games may also generalize better to new FG states; an added advantage is that a full scan through the database at each decision point is no longer necessary, so that runtime is decreased. A more detailed look at the making of these classifiers, along with the rest of the player agent, is the subject of the next section.



(Distance$_1$, resistance$_1$)	(2,1)
(Distance$_2$, resistance$_2$)	(2,2)
(Distance$_3$, resistance$_3$)	(3,1)
(Distance$_4$, resistance$_4$)	(4,3)
(Distance$_5$, resistance$_5$)	(7,2)
Units in this group	5
Total Blue units	12
Total Red units	9
Distance to closest FG	3
Blue force in combat	0
Red force in combat	0
Proportion supplied	0
Proportion of attackers	0
Rounds left	27

Fig. 2. Representation of the game state as seen from the perspective of a force group. The ten first attributes gives the distance to each of the five exit nodes, together with the number of Red pieces on the path to each node. These are sorted in ascending order. The remaining nine attributes describe own and opposing forces, the combat situation, and the number of rounds left

4 Implementing the Agent Design

Our goal is to construct game-playing software agents. Such an agent should be able to get a game state as input, and, from this state and the rules of the game, generate a move as output. The move describes where each of the own pieces should be placed when the turn is finished.

In accordance with the design choices described in the previous section, our agent selects its move as illustrated in Figure 3. Upon receiving the current game state, the agent identifies FGs and gives each FG a self-centric and simplified perception of the game state. Each FG then uses the move type classifiers in conjunction with this state representation in order to select which class (or classes) of moves is appropriate in its current situation. Finally, the agent should of course be able to translate these pieces of semantic move advice into actual board movement of the pieces in a suitable way.

The remainder of this section details the steps involved in building the agent. In the following, *game state* or simply *state* refers to the overall game state (i.e. positioning of pieces, rounds left and attacking sides in the case of combat), while *FG-state* refers to the simplified, egocentric state perception of each FG. Similarly, *move* refers to the collective action of all Blue's pieces in a turn, while *FG-move* refers to the action taken by a single FG only.

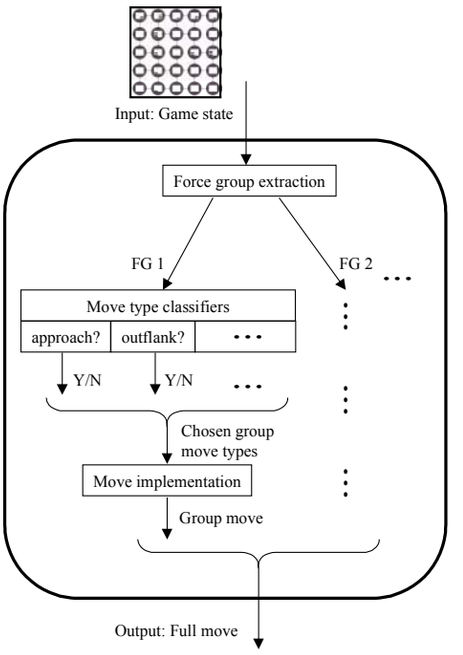


Fig. 3. Agent design

4.1 Database of Expert Games

A database of 20 human expert games for Blue against a fixed Red opponent was compiled. On each of Blue's turns in these games, one or more FGs would be present; the total number of Blue FG-states throughout the game series (not necessarily distinct) was 646. The expert labelled each FG-move into one or more of twelve move categories according to the intent of the move. The move categories are given, along with brief descriptions of their meaning, in Table 1.

The Red opponent employed in the expert games, named *AxesRed*, is an automatic playing agent adopting two main strategies of play. Firstly, it will never advance from the home row (i.e. the northernmost row) into the field, but stay home and wait for Blue to attack. Secondly, it attempts to position its pieces within the home row in a manner that proportionally mirrors the perceived threat on each vertical axis of nodes. For instance, if Blue has 13 pieces left, and 4 of these are located on the B axis (see Figure 1 for references to parts of the board), Red will to the best of his ability attempt to position 4/13 of his remaining pieces on this axis, i.e. in B5. A few simple rules apply to ensure that Red keeps a sound level of play in special cases. Red can of course not have more than 3 pieces in any node, and should its calculations require more than this, it will ensure that backup pieces are kept in the neighbourhood. Red will not reinforce in a combat node – this entails losing the defender's advantage – if this is not more than compensated for by the strength of the extra pieces. Although AxesRed is a rather simple player, it has proved to serve well as a benchmark opponent for evaluating Blue agents.

Table 1. Move categories for Blue force groups

Category	Description
Approach	Move pieces closer to the goal node
EnsureSupply	Keep some pieces back, with the intent of ensuring a supply line for present or future combat
BreakSupply	Break the supply line, i.e. advance pieces previously withheld for supply purposes from the southernmost rows
Attack	Move pieces into a node containing only Red pieces
ReinforceCombat	Move additional pieces into a combat node
ContinueCombat	Neither exit from nor reinforce existing combat
Outflank	Perform an evading manoeuvre, sideways or backwards, aiming at a different exit node
GoToGoal	Move pieces from the northernmost row into the goal node
ConcentrateForces	Move pieces within a FG closer, i.e. occupying fewer nodes
SplitIntoGroups	Divide a FG into two or more FGs
LinkUpGroups	Join a FG to another, creating a larger FG
StayInPosition	Leave all pieces in the FG unmoved

4.2 Neural Network Classifiers for Force Groups

Having assembled the database of FG-states and corresponding expert semantic FG-moves, we trained an ensemble of neural networks (NNs) to serve as FG-state classifiers for the agent. One NN was trained for each label, using the database of FG-states as input data and the presence (0 or 1) of the label in question as target values.

Each network was a standard feedforward NN featuring 19 input nodes, 36 hidden nodes and 1 output node (ranging from 0 to 1), sigmoid activation functions, and weights initially randomised from –0.2 to 0.2. Training was done by back-propagation. The input vector for each FG-state was scaled so that the magnitude ranges of the components were similar. About 1/3 of the data set was initially reserved and used for validating the training procedure, and the NNs were trained by repeatedly picking random examples from the training data.

The learning of the NNs was evaluated by the proportion of correctly classified examples over the validation data set. The ensemble quickly attained a classification performance of 0.9, and after further training reached about 0.95. We noted that performance on the validation set did not start to decrease, even if training was continued. Taking this as an indication that the data set presented little danger of overfitting – even with the large number of network weights used – we restarted training using all available data. The total classification accuracy on the full training set reached about 0.99; the individual label-specific NNs showed minor deviations from this average.

As explained above, the trained NNs are used in the game-playing agent. In a given game state, each FG inputs its FG-state into the twelve nets, each of which answers a number between 0 and 1. Output close to 1 indicates that the FG should perform a move corresponding to the category in question. The output value of 0.5 was used as the limit for choosing move categories.

In this procedure, the decision-making task of the player agent can be regarded as delegated to its constituent FGs, which choose and perform actions based on their own perceptions of the game state. An interpretation of the NNs when this view is adopted is as a shared overall doctrine for how to act in given situations.

4.3 Implementing the Acting Module

With the classification module of our agent properly in place, we turned to the task of actually designing and implementing the acting module. This module receives as input one or more chosen move categories for each FG, and returns the resulting movement of the individual pieces in the FG.

Due to space restrictions, we are unable to go into details of this module here. Instead, we mention a number of difficulties that arose in the implementation of this part of the agent. In particular, the set of chosen move categories may be inconsistent, in which case not all of the move types may be performed, e.g. if both *breakSupply* and *ensureSupply* are chosen. Another inconsistent set of move types is the empty set – since we have specified *stayInPosition* as a category of its own, and this category was not selected, this is not an order to simply stand still.

In cases when more than one move category is specified, we must decide which pieces should move according to which categories, or alternatively which categories should be given precedence. At the current stage, we use the simple strategy of using the numerical outputs of the respective neural nets for ranking the categories. Pieces are moved according to the first category, and if after this some pieces have not been assigned to an action, moves for the next category is implemented, and so on. The problem with empty move sets mentioned above was dealt with by defining the *approach* category as a default action; this category was chosen because advancing across the board is a reasonable baseline course of action for Blue – at any rate, it is almost always better than standing still.

Table 2. Results and approximate runtimes for various Blue agents playing against AxesRed

Player	Average score	Approx. time (s/game)
Human	9.41	–
SimpleBlue	3.88	0.2
OneAxisBlue	5.34	0.4
ConstraintNNBlue	6.60	90
ImitationBlue	4.04	30
Present agent	6.23	12

5 Experiments

We measure the success of a Blue agent by how many pieces it manages to move into the goal node in play against the AxesRed agent. Therefore, we need to have an idea of what constitutes a good result when playing against this particular Red opponent. The result from the expert games is a natural measure of the potential of our agent, since after all it is this expert behaviour we are trying to learn from. What then is a

bad result? This is difficult to say, but we can at least get a notion of a mediocre result by letting some rather naive Blue players challenge the AxesRed player. Two such benchmark Blue players have been designed. The first, *SimpleBlue*, employs the simple strategy of moving all its pieces forward when receiving the turn. This results in a full-breadth simultaneous attack, where three Blue pieces take on two Red pieces in each of the northernmost nodes. The second player, *OneAxisBlue*, initially decides upon an axis of attack, and advances as many of its pieces as possible along this axis for the rest of the game. This results in a focused attack on one of the northernmost nodes. Neither of these two players actively keeps a supply line, although the design of the game ensures that OneAxisBlue's supply line happens to be intact in the first phase of the attack.

Furthermore, it is interesting to compare the performance of our new agent with the best results from the two approaches described in Section 3.1. The agent called *ConstraintNNBlue* – the highest scoring agent we have managed to make during our previous work – is the constraint-based agent with NN move evaluation, while *ImitationBlue* is the rather less successful lazy learner.

The average results obtained by the human expert and the four agents mentioned are given in Table 2, along with the best result obtained by the agent treated in this paper. For each automatic agent, 1000 games were played; the human played 20. The approximate average runtime per game is also reported. As we can see, our present agent outperforms the two benchmark players and the imitating agent, but still has some way to go to reach the human expert – this latter fact is nothing more than could be expected[2]. Comparing the agent to ConstraintNNBlue shows that it fails to set a new record; on the other hand, it is not discouragingly far behind, while being almost an order of magnitude quicker. Moreover, we expect the agent to have considerable potential for improvement within the limits of the current design; a larger database of expert games and better move implementations are two of the more obvious measures that can be taken.

6 Conclusion

We have presented the design and implementation of an agent playing a highly complex stochastic game. The complexity of the game makes it impossible to use standard game-playing methods; instead, the agent uses neural networks to learn to play from a database of human expert games. The high complexity is handled by collapsing the huge action space into a few categories representing the semantic intentions of moves, and representing the game states of subsets of the agent's playing pieces by a few relevant features. An experimental evaluation of this approach shows promising results.

[2] Indeed, our experience with the game leads us to suspect that the human must have been rather lucky in these 20 games to achieve this score.

References

1. Fürnkranz, J., Kubat, M. (eds.): Machines That Learn to Play Games, Nova Science Publishers (2001).
2. Fürnkranz, J.: Machine learning in games: A survey. In: Fürnkranz, J., Kubat, M. (eds.): Machines That Learn to Play Games, Nova Science Publishers (2001) 11–59.
3. Schlabach, J. L., Hayes, C. C., Goldberg, D. E.: FOX-GA: A genetic algorithm for generating and analyzing battlefield courses of action. Evolutionary Computation 7 (1999) 45–68.
4. Boicu, M., Tecuci, G., Marcu, D., Bowman, M., Shyr, P., Ciucu, F., Levcovici, C.: Disciple-COA: From agent programming to agent teaching. In: Langley, P. (ed.): Proceedings of the 17th International Conference on Machine Learning (ICML-2000), Morgan Kaufmann (2000) 73–80.
5. Dahl, F. A., Halck, O. M.: Three games designed for the study of human and automated decision making. Definitions and properties of the games Campaign, Operation Lucid and Operation Opaque. FFI/RAPPORT-98/02799, Norwegian Defence Research Establishment (FFI), Kjeller, Norway (1998).
6. Sendstad, O. J., Halck, O. M., Dahl, F. A.: A constraint-based agent design for playing a highly complex game. In: Proceedings of the 2nd International Conference on the Practical Application of Constraint Technologies and Logic Programming (PACLP 2000), The Practical Application Company Ltd (2000) 93–109.
7. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Communications 7 (1994) 39–59.
8. Mitchell, T. M.: Machine Learning. WCB/McGraw-Hill (1997).
9. Kerner, Y.: Learning strategies for explanation patterns: Basic game patterns with application to chess. In: Veloso, M., Aamodt, A. (eds.): Proceedings of the 1st International Conference on Case-Based Reasoning (ICCBR-95). Lecture Notes in Artificial Intelligence Vol. 1010, Springer-Verlag (1995) 491–500.
10. Callan, J. P., Fawcett, T. E., Rissland, E. L.: CABOT: An adaptive approach to case-based search. In: Proceedings of the 12th International Conference on Artificial Intelligence, Morgan Kaufmann (1991) 803–809.