A Q-Learning Approach to Developing an Automated Neural Computer Player for the Board Game of CLUE®

Chenghui Cai and Silvia Ferrari

***Abstract*— The detective board game of CLUE® can be viewed as a benchmark example of the treasure hunt problem, in which a sensor path is planned based on the expected value of information gathered from targets along the path. The sensor is viewed as an information gathering agent that makes imperfect measurements or observations from the targets, and uses them to infer one or more hidden variables (such as, target features or classification). The treasure hunt problem arises in many modern surveillance systems, such as demining and reconnaissance robotic sensors. Also, it arises in the board game of CLUE®, where pawns must visit the rooms of a mansion to gather information from which the hidden cards can be inferred. In this paper, Q-Learning is used to develop an automated neural computer player that plans the path of its pawn, makes suggestions about the hidden cards, and infers the answer, often winning the game. A neural network is trained to approximate the decision-value function representing the value of information, for which there exists no general closed-form representation. Bayesian inference, test (suggestions), and action (motion) decision making are unified using an MDP framework. The resulting computer player is shown to outperform other computer players implementing Bayesian networks, or constraint satisfaction.**

I. Introduction

THE game of CLUE® is a benchmark example for the treasure hunt, which is a fundamental problem that arises in many modern sensor and surveillance systems comprised of sensors installed on mobile platforms. The treasure hunt problem is a coupled problem of robot path planning and inference that can be approached using pre-posterior decision analysis. Since the sensors are installed on mobile platforms, their path must be planned for the purpose of gathering sensor measurements and infer one or more hidden variables about the targets. Since the targets are distributed throughout the workspace, the path determines what measurements can be obtained by the sensor. Examples of relevant applications include landmine detection and classification by sensors installed on ground robots [1], [2], [3]. These applications constitute a shift with respect to the traditional paradigm of a sensor that is employed in order to enable robot motion planning [4]. Thus, new decision and control techniques need to be developed to optimize the overall system performance by planning sensing or information-gathering strategies that maximize the expected benefit of information of the measurement sequence, while minimizing the cost associated with the use of sensor and platform resources, such as, energy and distance.

The board game of CLUE® is a detective game with the objective of inferring three hidden cards, representing the guilty suspect, the weapon, and the room of an imaginary murder. The pawns navigate the game board in order to visit the rooms of the CLUE® mansion, where they can make suggestions and gather information about the hidden cards, which are viewed as the treasure. Similarly to sensor systems, the information gathered by each player depends on the location of his/her pawn, because a suggestion must always include the room that is presently occupied by the pawn. Also, the distance traveled by the pawn must be minimized in order to minimize the number of turns needed to infer the hidden cards and beat the adversaries. Recently, several authors have demonstrated that computer games are useful for developing and demonstrating computational intelligence methods and algorithms [5], [6], [7], [8], [9], [10]. This paper uses the game of CLUE® to illustrate the key challenges and characteristics of the treasure hunt problem, and develops a methodology that can be extended to sensor path planning research. A graphical-user-interface and an interactive simulation of the game were developed by the authors in [11] to enable a Bayesian network (BN) intelligent computer player to compete against human players, and to test decision-making strategies that have been developed for robotic sensor systems [3], [12], [13].

The approach presented in this paper is based on a Markov decision process (MDP) [14] representation of the game, which can be utilized to obtain a policy that optimizes the expected sum of a discounted reward, representing the value of information minus its cost. The main difficulties associated with MDPs are that the transition and reward functions may be unknown, and that large state and/or decision spaces may be intractable due to the curse of dimensionality [14]. Reinforcement learning is able to obtain a decision-value function by using observed rewards to learn an optimal or nearly optimal policy from data. Function approximation by neural networks (NNs) and sampling techniques may be combined to deal with continuous state spaces [15]. Since the game of CLUE® presents both of these difficulties, in this paper Q-Learning [16] [15] is used to learn the reward function, and a neural network is used to represent the Q function. The approach presented in this paper is related to [17], where a NN is trained by supervised learning to evaluate board positions with hand-labeled data in the game of Backgammon. Also, in [18], an approach is presented for training a NN in Backgammon using time consuming self-

C. Cai is a graduate student of Mechanical Engineering at Duke University, Durham, NC, 27708, USA cc88@duke.edu. S. Ferrari is with the Faculty of Mechanical Engineering at Duke University, Durham, NC, 27708, USA sferrari@duke.edu.

978-1-4244-1821-3/08/$25.00©2008 IEEE

play with temporal difference.

The Q-Learning approach presented in this paper employs next state estimation to avoid learning from a large number of real game trials, and save computation time. The current state estimation is based on probabilistic inference by BNs. The CLUE BN developed by authors in [11] is implemented not only to infer the hidden cards, but also to estimate the MDP state at the current time step, based on the latest available evidence obtained from the game. In fact, the posterior probability mass function (PMF) of the hidden room cards obtained by BN inference is viewed as the MDP state to satisfy the Markov property, and, together with the action (motion) decision, is used as the input to the NN approximation of the Q function. The paper is organized as follows. Section II reviews the background and the CLUE® game rules. Section III presents the methodology used to develop the neural computer player and, in Section IV, the player is tested against other computer players of CLUE® developed by the authors, which, to the best of our knowledge, are the only automated computer games that have been developed for this game.

II. Background

A. The Game of CLUE®

The CLUE® mansion includes nine rooms, the dining room, library, billiard room, hall, kitchen, lounge, ballroom, study and conservatory, and connecting hallways, as illustrated on the game board in Fig. 1. The six suspects, Col. Mustard, Miss Scarlet, Prof. Plum, Mr. Green, Mrs. White and Mrs. Peacock are represented by pawns and can use any of the six weapons, a knife, rope, candlestick, lead pipe, revolver, and wrench, to commit the murder. Since each suspect, weapon, and room is represented by an illustrated card, there are a total of twenty-one cards in the deck. Three hidden cards representing the murder weapon, room, and guilty suspect are selected randomly and removed from the deck at the beginning of the game. The remaining cards are then dealt to the players. During the game, players move their pawns from room to room and, upon entering a room, make a "suggestion" about the hidden cards, which must include the room they presently occupy. There are three ways for a pawn to enter a room: (i) through one of the doors, (ii) via a secret passage (Fig. 1), or (iii) by another player's suggestion. Both (i) and (ii) are decided by the player, whereas (iii) is beyond the player's control. For instance, the player enters the ballroom and makes a "suggestion" represented by a set of {Miss Scarlet, knife, ballroom}, which shows that at that time, he/she believes Miss Scarlet commits the imaginary murder with knife in the ballroom.

Players gather information about the adversaries' cards by making suggestions from which the three hidden cards are inferred. Every suggestion by player i must be disproved by player $(i+1)$ by showing player i one of the suggested cards. If player $(i+1)$ has none of the suggested cards, it is the turn of player $(i+2)$ to disprove the suggestion by player i, and so on, until someone can disprove it using a card from their own deck. Clearly, if nobody can disprove the suggestion, then the suggested cards are the hidden ones. When a player is confident in inferring the hidden cards, he/she can make an accusation. If the accusation is correct, the player wins the game, otherwise he/she loses and exits the game.

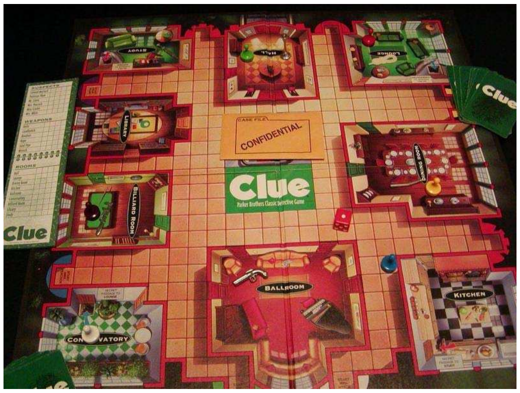

Fig. 1. CLUE® game board. CLUE® & ©2006 Hasbro, Inc. Used with permission.

B. Review of Markov Decision Processes (MDPs)

An MDP is a tuple $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, T, R\}$ representing a sequential decision model. $\mathcal{S} = \{x_1, \ldots, x_n\}$ is a finite set of possible state values, called the *state space*. $\mathcal{A} = \{a_1, \ldots, a_m\}$ is the set of feasible action decisions. T is the transition probability function $T : \mathcal{S} \times \mathcal{A} \to \mathbf{P}(\mathcal{S})$, which describes the MDP state transitions, such that whenever the state at time k has value $X_k = x_i$ and the decision is $U_k = a_j$, there is a probability $\mathbf{P}(X_{k+1} = x_l \mid X_k = x_i, U_k = a_j)$ that the next state value is $X_{k+1} = x_l$. The reward function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, specifies the value of the immediate reward, $r_k = R(\ X_k, U_k)$, received after executing the action decision U_k in state X_k. A policy is a mapping of state values to actions, $\pi : \mathcal{S} \to \mathcal{A}$. Let the value function $V^{\pi}(X_k)$ denote the expected discounted return of policy π

$$V^{\pi}(X_k) = E\left\{\sum_{i=0}^{\infty} \gamma^i r_{k+i} \mid \pi,\ X_k\right\} \qquad (1)$$

Where, r_{k+i} is the reward received at i steps into future, and the discount factor $0 \leq \gamma < 1$ modulates the effect of future rewards on present decisions, with small values emphasizing near-term gain and larger values emphasizing later rewards. Then, an *optimal policy* π^* is one that maximizes $V^{\pi}(X_k)$ for all possible states $X_k \in \mathcal{S}$. The Markov property guarantees that an optimal policy exists, though it may not be unique, and, thus, it is associated with an *optimal value function* $V^*(X_k) = \max_{\pi} V^{\pi}(X_k)$. The optimal policy of an MDP, $\mathcal{M}$, is a fixed point of the Bellman's equation, which can be determined iteratively using policy iteration or value iteration algorithms [19].

In value iteration, the value of a state $V(X_k)$ is the total expected discounted reward accrued by a policy starting at

$X_k \in \mathcal{S}$. The Q function of a state-action pair, $Q(X_k, U_k)$, is the total expected discounted reward accrued by a policy that produces $U_k = \pi(X_k)$ [19]. The Bellman equation can be formulated in terms of the aforementioned functions, such that the state-action value function is

$$Q(X_k, U_k) = E\left\{R(X_k, U_k) + \gamma V(X_{k+1})\right\} \quad (2)$$

$$V(X_{k+1}) = \max_{U_{k+1} \in \mathcal{A}} Q(X_{k+1}, U_{k+1}). \quad (3)$$

If two functions $Q(\cdot)$ and $V(\cdot)$ satisfy the above Bellman equation, then they specify an optimal *greedy* policy

$$\pi^*(X_k) = \arg \max_{U_k \in \mathcal{A}} Q(X_k, U_k) \quad (4)$$

Value-iteration algorithms use eq. (2) to iteratively determine $Q(\cdot)$ and $V(\cdot)$ and, subsequently, determine $\pi^*(\cdot)$.

Value iteration can be used to determine the optimal policy of an MDP, $\mathcal{M}$, provided the transition probability function T is known. If T is unavailable, Q-Learning can be utilized to learn an approximate state-action value function $Q(X_k, U_k)$ that is iteratively updated by the rule,

$$\begin{aligned} Q(X_k, U_k) \;\; &\leftarrow \;\; (1-\alpha) Q(X_k, U_k) \\ &+ \;\; \alpha[r_k + \gamma \max_{U_k \in \mathcal{A}} Q(X_{k+1}, U_k)] \end{aligned} \quad (5)$$

where, α is the learning rate, and $0 < \alpha \leq 1$.

C. Review of Bayesian Network Inference in CLUE®

A Bayesian network is comprised of a directed acyclic graph (DAG) and a parameter structure that can be obtained from expert knowledge or learned from data [20], [21]. The BN nodes $X_C = \{X_1, \ldots, X_N\}$ represent discrete and random variables that each have a finite range $\{x_{i,1}, \ldots, x_{i,n_i}\}$, where $x_{i,j}$ denotes the j^{th} state value or instantiation of variable X_i. The union of the ranges of all nodes in X_C is the state space of the BN. The BN parameters are called conditional probability tables (CPTs). Each CPT is attached to a node, $X_i \in X_C$, and lists in tabular form the conditional probability mass function (PMF), $\mathbf{P}(X_i \mid pa(X_i)]$, where $pa(X_i)$ denotes the set of parents of X_i. If in the BN DAG there is an arc from X_j to X_i, node X_j is said to be a *parent* of X_i, and X_i is said to be a *child* of X_j [20]. After a BN model is determined, it specifies the joint PMF of X_C in terms of the recursive factorization,

$$\mathbf{P}(X_C) \equiv \mathbf{P}(X_1, \ldots, X_N) = \prod_{X_i \in X_C} \mathbf{P}(X_i \mid pa(X_i)) \quad (6)$$

which is represented by the BN architecture. The above BN factorization is convenient for inferring any node(s) in X_C from information or *evidence* about the other nodes in X_C. In particular, a junction-tree BN inference algorithm [22] can be used to efficiently compute the posterior PMF of one or more hidden nodes from the BN CPTs in (6).

In [11], the authors presented a BN approach to automating suggestions in CLUE®. A BN model of the relationships between the cards is obtained by representing every card in the network by a BN node, and by using the outcomes of the suggestions obtained during the game as evidence from which to infer the hidden cards. The architecture of the CLUE® BN is shown in Fig. 2. For simplicity, it is assumed that there are three players in the game, but the same approach can easily be extended to any number of players. The hidden cards are represented by the guilty suspect node S, the murder weapon node W, and the murder room node R. The remaining eighteen cards are randomly dealt to the players, such that every time a card is dealt, it influences the cards that are dealt later. Node $C^{\ell}_{j,i}$ denotes the j^{th} card in the deck, dealt to player i^{th}, and belonging to category $\ell = s, w, r$, where, s denotes the suspect category, w the weapon category, and r the room category. Thus, the possible values of the suspect cards are $\{s_1$ (Col. Mustard), s_2 (Ms. Scarlet), s_3 (Prof. Plum), s_4 (Mr. Green), s_5 (Mrs. White), s_6 (Mrs. Peacock) $\}$, those of the weapon cards are $\{w_1$ (knife), w_2 (rope), w_3 (candlestick), w_4 (lead pipe), w_5 (revolver), w_6 (wrench)$\}$, and those of the room cards are $\{r_1$ (dining room), r_2 (library), r_3 (billiard room), r_4 (hall), r_5 (kitchen), r_6 (lounge), r_7 (ballroom), r_8 (study), r_9 (conservatory)$\}$. Thus, the CPTs of the BN model in Fig. 2 can be obtained using basic probability theory, as explained in [11].

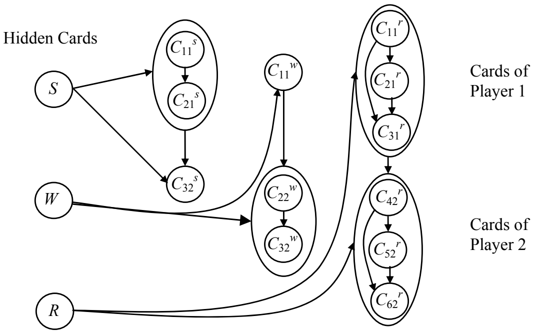

Fig. 2. Architecture of CLUE® BN model, taken from [11]

At every player turn, indexed by a subscript k, new evidence may become available about the adversaries' cards. Information about BN nodes is referred to as *hard evidence* when it refers to exact knowledge of a variable's instantiation. It is referred to as *soft evidence*, when it refers to an observed probability distribution based, for example, on the negation of one or more values in a variable's range. Both types of evidence are obtained during the game of CLUE®, and are organized into an evidence table, E_k, that is updated at every turn. E_k contains the observed probability before turn k for every possible value of every node card $C^{\ell}_{j,i}$ in the CLUE® BN (see [11] for more details). For example, if player 2 shows Mrs. Peacock's card to the neural computer player, then the entry corresponding to the first (uninstantiated) suspect card $C^{s}_{3,2}$ and s_6 is assigned probability one, and all the other values of $C^{s}_{3,2}$, s_1 through s_5, are assigned probability zero. The neural computer player developed in this paper utilizes the CLUE® BN to infer the hidden cards and develop suggestions at every one of its turns, based on the latest evidence table.

The next section presents the design of a neural computer player that uses an MDP representation of the game and Q-Learning to determine an optimal policy for deciding pawn movements and suggestions.

III. Development of MDP-based Neural Computer Player

A. Design Overview

The problem of deciding pawn's movement and suggestions in the game of CLUE® is formulated as an MDP in which the reward to be maximized is the value of information obtained by visiting the mansion's rooms, minus the distance traveled by the pawn. Since the room occupied by the pawn must be included in the suggestion, the pawn's path determines the information that will be gathered over one or more turns by the computer player. To satisfy the Markov property, the MDP state is formulated using posterior probabilities of the hidden room card. The architecture of the MDP-based neural computer player is shown in Fig. 3, where the design blocks are named based on their functionalities.

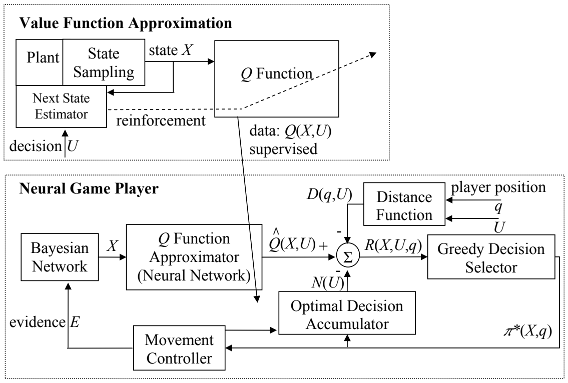

Fig. 3. Architecture of MDP-based Neural Computer Player

Let each player be denoted by P_i, where i represents the turn sequence, and assume the neural computer player is P_1. The set of feasible action decisions for P_1 is $\mathcal{A} = \{a_1, \ldots, a_9\}$, where $a_i \in A$ denotes the decision of entering room r_i. Let the MDP state X_k be defined as the posterior PMF $\mathbf{P}(R \mid E_k)$, such that $X_k = [x_{k,1} \ \cdots \ x_{k,9}]^T$, where $x_{k,i} \equiv \mathbf{P}(R = r_i \mid E_k)$. Then, the Markov property $\mathbf{P}(X_{k+1} \mid X_k, U_k, X_{k-1}, U_{k-1}, \ldots, X_0, U_0) = \mathbf{P}(X_{k+1} \mid X_k, U_k)$ holds, and $\sum_{i=1}^{9} x_{k,i} = \sum_i \mathbf{P}(R = r_i \mid E_k) = 1$ for $\forall k$. Although time is discrete, the MDP state is continuous, with $\mathcal{S} \subset [0,1]^9$. Therefore, as illustrated in Fig. 3, a finite subspace of n possible states, $\mathcal{S}_n$, is first sampled from the continuous state space, i.e., $\mathcal{S}_n \subset \mathcal{S}$. Then, a next-state estimator is developed to estimate the next state X_{k+1} from the executed action U_k and current state X_k, as required by Q-Learning. Equation (5) is utilized to obtain the Q function over the domain $\mathcal{S}_n \times \mathcal{A}$, which can be used to train the corresponding neural network, as explained in Section III-B.

In the neural game player architecture (Fig. 3), the continuous state X_k is inferred from the CLUE® BN using a junction-tree inference algorithm (Section II-C). X_k is fed to the NN approximator to estimate the Q function and output $\hat{Q}(X_k, U_k)$. The MDP state is augmented by the pawn's position $q_k \in \mathbb{R}^2$ in the game board. The distance between q_k and any room r_i is defined as the minimum Manhattan distance [23]. Since the distance traveled to r_i is the cost of the decision a_i, the distance is denoted by $D(q_k, U_k = a_i)$ and is subtracted from the value of information $\hat{Q}(X_k, U_k)$ (Fig. 3). An additional cost, $N(U_k = a_i)$ is used to measure the number of times the player enters room r_i in order to limit visits to the same room. The total reward function $R(X_k, U_k, q_k)$ is defined as a weighted function of $\hat{Q}(X_k, U_k)$, $D(q_k, U_k)$ and $N(U_k)$. Then, the optimal policy $U_k = \pi^*(X_k, q_k)$ determines which room the pawn must move to next, given its position q_k and the posterior PMF of the hidden room card X_k. A simple path planner implements the decision and moves the neural player's pawn to the chosen room. Subsequently, the neural player makes a suggestion that includes the room occupied, and a weapon and suspect chosen by means of the CLUE® BN. The objective is to obtain the best possible evidence about the hidden cards, by moving to rooms that offer the optimal tradeoff between distance traveled and value of information.

B. Value of Information Function Approximation

The value of information is the expected utility of performing a test or observation that may be used to infer hidden variables in a stochastic process. This terminology was first introduced by Howard in [24], who used utilities to guide the test selection in partially-observable MDPs. Although several functions have been proposed for assessing the value of information in dual control [25], and sensor planning [13], the best functional representation often cannot be determined *a priori*. Therefore, in this section, a NN is trained to approximate the Q function associated with the value of information from the data obtained via Q-Learning. The objective is to estimate the reward associated with making a suggestion that includes room r_i prior to visiting r_i. The utility of the suggestion is that its outcome may be used for inference, while its cost is the distance traveled to the room.

The main drawback of Q-Learning is that its convergence speed may be too low due to the large size of the state space created by complex or multiple goal tasks. For instance, learning the Q function for CLUE® may require thousands of learning trials (where, every trial is a game) before convergence to an optimal state-action value function is achieved. Every simulated CLUE® game takes approximately 10 to 20 minutes. Thus, a next-state estimator is presented that estimates X_k by mimicking the outcome of a decision U_k, leading to significant savings in learning times. Suppose P_1 is in state $X_k \in \mathcal{S}_n$ and makes a decision $U_k = a_i$ to enter room r_i and make a suggestion $\mathcal{G}_k = \{S_\mathcal{G}, W_\mathcal{G}, R_\mathcal{G} = r_i\}$, where the subscript $\mathcal{G}$ denotes the suggested card. Then, there are four possible outcomes: (i) $\mathcal{G}_k$ cannot be disproved and the value of X_k indicates that $\mathbf{P}(R = r_i \mid E_k) = 0$, thus

P_1 has r_i and $X_{k+1} = X_k$; (ii) $\mathcal{G}_k$ cannot be disproved and the value of X_k indicates that $\mathbf{P}(R = r_i \mid E_k) > 0$, thus P_1 does not have r_i, and the new state X_{k+1} is $\mathbf{P}(R = r_i \mid E_{k+1}) = 1$ and $\mathbf{P}(R = r_j, j \neq i \mid E_{k+1}) = 0$; (iii) one of the adversaries shows a suspect or weapon card to disprove $\mathcal{G}$, thus $X_{k+1} = X_k$; (iv) one of the adversaries shows the room card r_i to disprove $\mathcal{G}$, thus X_{k+1} changes to include $\mathbf{P}(R = r_i \mid E_{k+1}) = 0$. In this case, the system state is updated by eq. (7). The fact that $R \neq r_i$ is the new evidence obtained at turn k, and then $E_{k+1} = \{E_k, R \neq r_i\}$. Also, $\mathbf{P}(R \neq r_i \mid R = r_j, E_k) = 1, j \neq i$. Eq. (7) can be derived as follows:

$$\begin{aligned} x_{k+1,j} &= \mathbf{P}(R = r_j \mid E_{k+1}) = \mathbf{P}(R = r_j \mid E_k, R \neq r_i) \\ &= \frac{\mathbf{P}(R \neq r_i |\ R = r_j, E_k)\mathbf{P}(R = r_j |\ E_k)}{\mathbf{P}(R \neq r_i |\ E_k)} \\ &= \frac{\mathbf{P}(R \neq r_i |\ R = r_j, E_k)\mathbf{P}(R = r_j |\ E_k)}{\sum_j \mathbf{P}(R \neq r_i |\ R = r_j, E_k)\mathbf{P}(R = r_j |\ E_k)} \\ &= \begin{cases} 0 & \text{if } j = i \\ \frac{\mathbf{P}(R = r_j |\ E_k)}{\sum_{l, l \neq i} \mathbf{P}(R = r_l |\ E_k]} & \text{else} \end{cases} \\ &= \begin{cases} 0 & \text{if } j = i \\ \frac{x_{k,j}}{\sum_{l, l \neq i} x_{k,l}} & \text{else.} \end{cases} \end{aligned} \tag{7}$$

As is seen in eq. (7), if $x_{k,i} = 0$, $X_{k+1} = X_k$. The above four cases of state transitions will be equivalently encoded in the function of "next_state_estimator" in Table I.

Although X_{k+1} can be updated using eq. 7, it is possible that $X_{k+1} \notin \mathcal{S}_n$. In this case, X_{k+1} is approximated by,

$$\hat{X}_{k+1} = \arg \min_{X_k \in \mathcal{S}_n} ||X_{k+1} - X_k|| \tag{8}$$

where, $|| \cdot ||$ is the Eucledian norm. The reward matrix is set by heuristics as follows:

$$R(X_k, U_k) = x_{k,i}, i = \arg U_k, \forall X_k \in \mathcal{S}, U_k \in \mathcal{A}, \tag{9}$$

where the notation $\arg(U_k)$ returns the index i when $U_k = a_i$ meaning the decision of entering room r_i. The Q-Learning-based algorithm in Table I is used to compute the Q function. For a sample number $n = 1300$, used to sample $\mathcal{S}_n$ from $\mathcal{S}$, a discount factor $\gamma = 0.8$, and a learning rate $\alpha = 1$, the implementation of this Q-Learning algorithm takes $1.26 \cdot 10^3$ seconds on a Pentium 4 CPU 3.06 GHz computer, and converges to the optimal Q function in approximately $5 \cdot 10^5$ iterations. Subsequently, the optimal Q function is used to train a two-layer feed-forward sigmoidal NN via backpropagation. The NN has 18 inputs, defined as $I = \{X_k,\ \text{sgn}(U_k)\}$ where sgn$(\cdot)$ denotes the signum function. The NN has 100 hidden nodes and one output $\hat{Q}(X_k, U_k)$. The MATLAB® gradient descent adaptive learning algorithm *traingda* is applied to train the NN to approximate the decision-value function over $\mathcal{S}$.

TABLE I

Q-LEARNING ALGORITHM

```
Inputs: $\mathcal{S}_n, \mathcal{A}, R(\cdot), 0 \leq \gamma < 1, 0 < \alpha \leq 1$
Outputs: $Q(\cdot)$
initialize $Q(\cdot)$ to be a zero matrix
initialize $Q_1(\cdot)$ to be a matrix with elements $\rightarrow \infty$
initialize learning convergence counter $count \leftarrow 0$
for each epoch
    randomly select current state $X_k$ from $\mathcal{S}_n$
    randomly select $U_k$ from $\{U_k \mid R(X_k, U_k) \geq 0\}$
    $\hat{X}_{k+1} \leftarrow$ next_state_estimator$(X_k, U_k, \mathcal{S}_n)$
    update $Q(X_k, U_k)$ using eq. (5)
    /* Check convergence /*
    if $|Q - Q_1| < tol$ and $||Q(X_k, U_k)|| > 0$
        if $counter > threshold$
            break for loop; /* convergence */
        else $count \leftarrow count + 1$
        end; /* if loop */
    else $Q_1 \leftarrow Q$; $count \leftarrow 0$
    end; /* if loop */
end; /* for loop */
normalize $Q$

Function $\hat{X}_{k+1} =$ next_state_estimator$(X_k, U_k, \mathcal{S}_n)$
$i \leftarrow \arg(U_k)$
if $x_{k,i} = 0$ /* $R \neq r_i$ */
    $X_{k+1} \leftarrow X_k$
elseif $x_{k,i} = 1$ /* $R = r_i$ */
    $X_{k+1} \leftarrow X_k$
else /* $R = r_i$, or $r_i$ dealt to $P_2$ or $P_3$ */
    randomly select one of the following cases:
    $(i)$ $\mathcal{G}$ cannot be disproved by $P_2$ and $P_3$ and $R = r_i$
        $j = i, x_{k+1,j} \leftarrow 1$
        $j \neq i, x_{k+1,j} \leftarrow 0$
    $(ii)$ a suspect or weapon card is used to disprove $\mathcal{G}$
        $X_{k+1} \leftarrow X_k$
    $(iii)$ a room card is used to disprove $\mathcal{G}$
        $j = i, x_{k+1,j} \leftarrow 0$
        $j \neq i, x_{k+1,j} \leftarrow \frac{x_{k,j}}{\sum_{l, l \neq i} x_{k,l}}$
end; /* if loop*/
$\hat{X}_{k+1} \leftarrow \arg\min_{X_k \in \mathcal{S}_n} ||X_{k+1} - X_k||$
```

C. Application of MDP-based Neural Computer Player

The MDP-based neural computer player of CLUE® implements the NN described in Section III-B to approximate $\hat{Q}(X_k, U_k)$, and the BN model described in Section II-C to incorporate the evidence gathered by its pawn, and infer the MDP state X_k. Thus, $\hat{Q}(X_k, U_k)$ can be obtained by the NN approximator for every decision $a_i \in \mathcal{A}$. The polygonal decomposition of the pawn's workspace shown in Fig. 4 can be used to evaluate the distance function $D(q_k, a_i)$. When the optimal decision $U_k = \pi^*(X_k, q_k)$ is implemented and the player enters a room r_i to make suggestion, the cost $N(U_k = a_i)$, which is initially set to zero, is increased by one. Then, the total reward function is given by,

$$R(X_k, U_k, q_k) = \hat{Q}(X_k, U_k) - W_d \cdot D(q_k, U_k) - W_n \cdot N(U_k) \tag{10}$$

where W_d and W_n are constant positive weights that are assigned the values $W_d = 0.5$ and $W_n = 1.5$ in this paper.

The optimal decision $\pi^*(X_k, q_k)$ is greedily determined from eq. (11), given current state X_k and player position q_k.

$$\pi^*(X_k, q_k) = \arg \max_{U_k} R(X_k, U_k, q_k). \tag{11}$$

Then, at the neural computer player's turn k, a simple path planner [4] decomposes the pawn's workspace into a convex polygonal decomposition (Fig. 4), and generates a

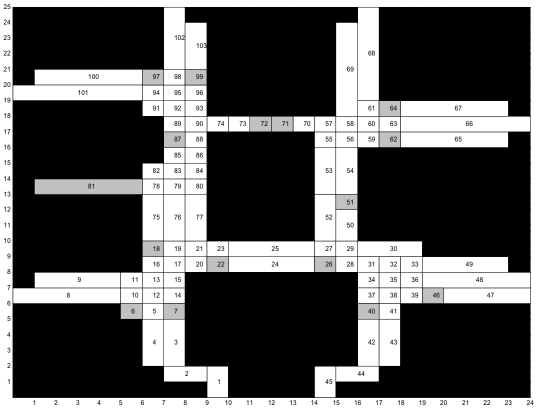

Fig. 4. Convex polygonal decomposition (CPD) of the CLUE® workspace, where cells of a door to room are shown in grey, other cells in white, and obstacles in black.

connectivity graph in which each node represents a polygon that is obstacle free and decomposable into the square bins illustrated on the board. The connectivity graph is searched by the A* algorithm to determine the shortest path from q_k to $r_i, i = \arg \pi^*(X_k, q_k)$. After rolling the dice, the neural computer player moves its pawn along the shortest path, from bin to bin in the two-dimensional grid illustrated on the game board (Fig. 1), until the dice value is met.

Once the neural computer player, P_1, enters room r_i, say at turn $(k + t)$, the BN model described in Section II-C is used to compute the posterior PMFs $\mathbf{P}(S \mid E_{k+t})$ and $\mathbf{P}(W \mid E_{k+t})$ or, in other words, to infer the hidden cards S and W based on the latest available evidence. Then, a suggestion $\mathcal{G} = \{S_\mathcal{G}, W_\mathcal{G}, R_\mathcal{G} = r_i\}$ is produced according to the following heuristic rule: if there is a suspect card s_j with $0 < \mathbf{P}(S = s_j \mid E_{k+t}) < 0.9$, then $S_\mathcal{G} = s_j$, otherwise $S_\mathcal{G}$ is set equal to a suspect card owned by P_1; if there is a weapon card w_l with $0 < \mathbf{P}(W = w_l \mid E_{k+t}) < 0.9$, then $W_\mathcal{G} = w_l$, otherwise $W_\mathcal{G}$ is set equal to a weapon card owned by P_1. At the end of its turn, when $\mathcal{G}$ has been disproved by an adversary, P_1 makes an accusation $\{s_i^*, w_j^*, r_l^*\}$ only if $\mathbf{P}(S = s_i^* \mid E_{k+t+1})$, $\mathbf{P}(W = w_j^* \mid E_{k+t+1})$, and $\mathbf{P}(R = r_l^* \mid E_{k+t+1})$ are all greater than or equal to 0.9, otherwise the game continues.

IV. Game Simulation and Performance Comparison Results

An interactive computer simulation of CLUE® has been developed by the authors in [11] using the MATLAB® Graphical User Interface (GUI) toolbox. The simulated game consists of three phases: the players choose pawns, the computer deals the cards, and the players start the game. For simplicity, in this paper, the neural computer player plays against two adversaries who may be computer or human players. Every player's turn consists of three steps: roll the die, move the pawn a distance no greater than the die number, and transfer the turn to next player. If a player's pawn enters a room during his/her turn, the player makes a suggestion and waits for other players' responses. If the player's pawn is taken into a room by another player, he/she must wait for his/her turn to make a suggestion.

An accusation can be made at the beginning or at the end of a player's turn. Each player can see his/her cards, and his/her decisions are observed and recorded by the other players during the game via the interfaces developed in MATLAB®. He/she can also keep a record of all suggestions that take place during the game, and of how they are disproved, gathering evidence even during the adversaries' turns. The neural computer player is tested by letting it compete against a human player, a BN computer player developed by the authors in [11], a constraint satisfaction player (CSP), and a random computer player.

A. Competing BN and CSP Computer Players

The BN player computer player presented in [11] implements the same BN model used in this paper to infer the posterior PMFs $\mathbf{P}(S \mid E_k)$, $\mathbf{P}(W \mid E_k)$, and $\mathbf{P}(R \mid E_k)$ at every turn k. If any of these posterior probabilities are greater than or equal to 0.9 the BN computer player makes an accusation, otherwise, it rolls the die and moves its pawn. The pawn's path planning is performed based on a heuristic rule, such that if $\mathbf{P}(R = r_j \mid E_k) \geq 0.9$ for some room value r_j, then the pawn enters room r_j as often as possible, and makes a suggestion by the same rule used by the MDP-based neural player (Section III-C).

The constraint satisfaction problem (CSP) player implements the approach reviewed in [19, Chapter 5], with two types of constraints: (C1) $S \neq C^s_{ji}, W \neq C^w_{ji}$ and $R \neq C^r_{ji}$, for $\forall\ i, j$, and (C2) $C^\ell_{ji} \neq C^\ell_{nm}$, for $\ell = s, w, r$, $i, m = 1, 2$, $j, n = 1, 2, 3$, and $i \neq m$ or $j \neq n$. When the CSP player makes a suggestion and the i^{th} player disproves it, an instantiation $C^\ell_{ji} = c^\ell_{ji,l}$ is obtained for some j and included in constraint (C2). Then, in order to make a suggestion, the CSP searches an assignment for S, W, and R that satisfies (C1)-(C2). Based on a heuristic strategy recommended by the CLUE® game rules, the CSP pawn moves to the nearest room to begin making suggestions. Then, it moves back and forth between two adjacent rooms to determine S, W, and subsequently navigates the board to determine R.

The random computer player performs random moves and suggestions and, thus, is used as a place holder that hardly ever wins the game. Since the NN and BN models are trained before the game playing, decision making at every turn by any of NN, BN, CSP and random computer players is very fast with the time less than several seconds on a Pentium 4 CPU 3.06 GHz computer.

B. Simulated Games Results

The efficiency of the BN and CSP players were tested by making both of them compete against an impartial human player who is not familiar with the simulation or the computer players designs. As shown in Table II, the results obtained from 25 games demonstrate that the BN player performs almost as well as the human player, and the former outperform the CSP player, which wins only 20% of the times. As shown in Table III, when the MDP-based neural computer player competes against the CSP and the random computer players it wins 75% of the times. Instead, when the neural player competes against the BN and the random computer player, it wins 65% of the times (Table IV). Thus, it can be concluded that the neural player is the most effective of the computer players. The reason is that the other computer players do not account for the value of information and the game evidence in deciding the pawn's motion. Since the BN player performs almost as well as the human player (Table II) the neural player is likely to perform as well or, perhaps, better than human players. This test will be the subject of future research.

TABLE II

GAME RESULTS OF BN, HUMAN AND CSP PLAYERS PLAYING TOGETHER

Winning Player:	BN Player	Human Player	CSP
Winning Times / Total Playing Times	9 / 25	11 / 25	5 / 25
Winning Rate	36.0%	44.0%	20.0%

TABLE III

GAME RESULTS OF NEURAL PLAYER PLAYING AGAINST CSP PLAYER

Winning Player:	Neural Player	CSP
Winning Times / Total Playing Times	15 / 20	5 / 20
Winning Rate	75.0%	25.0%

TABLE IV

GAME RESULTS OF NEURAL PLAYER PLAYING AGAINST CSP PLAYER

Winning Player:	Neural Player	BN Player
Winning Times / Total Playing Times	12 / 20	8 / 20
Winning Rate	60.0%	40.0%

V. CONCLUSIONS

An MDP-based neural computer player is developed for the board game of CLUE® using a combination of Q-Learning and BN inference to estimate the value of information. The objective of the neural player is to solve a benchmark example of the treasure hunt problem, which arises in several robotic sensor applications, such as demining and reconnaissance. Sampling techniques and NN function approximation are used to address the continuous state space, defined as the posterior PMF of the hidden variables to maintain the Markov property. Thus, Bayesian inference and action (motion) decision making can be unified by the MDP framework. The simulated game statistics show that the neural computer player outperforms existing computer players of CLUE® obtained by Bayesian networks and constraint satisfaction approaches, which do not account for the value of information in deciding the pawn's motion.

VI. ACKNOWLEDGMENTS

CLUE® & ©2006 Hasbro, Inc. Used with permission.

REFERENCES

[1] J. MacDonald, et al., Alternatives for Landmine Detection. RAND, Science and Technology Policy Institute, 2003.

[2] R. Popoli, "The Sensor Management Imperative," Chapter 10 in *Multitarget-Multisensor Tracking: Advanced Applications*, Vol. II, Y. Bar-Shalom, Ed., Artech House, 1992.

[3] S. Ferrari and A. Vaghi, "Demining Sensor Modeling and Feature-level Fusion by Bayesian Networks," *IEEE Sensors Journal*, vol. 6, pp. 471-483, 2006.

[4] J. C. Latombe, *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[5] J.E. Laird, "Using a Computer Game to Develop Advanced AI," *Computer*, vol. 34, no. 7, pp. 70-75, July 2001.

[6] N. Baba and L. C. Jain, *Computational Intelligence in Games.* Heidelberg: Physica-Verlag, 2001.

[7] X. Z. Pang and P. J. Werbos, "Neural Network Design for J Function Approximation in Dynamic Programming," *Math. Modelling and Scientific Computing (a Principia Scientia journal)*, vol. 5, no. 2/3, 1996.

[8] K. Chellapilla and D. Fogel, "Evolution, neural networks, games, and intelligence," *Proc. of the IEEE*, vol. 87, no. 9, pp. 1471-1496, September, 1999.

[9] D. Fogel, T. J. Hays, S. L. Hahn and J. Quon, "A self-learning evolutionary Chess program," *Proc. of the IEEE*, vol. 92, no. 12, pp. 1947-1954, December, 2004.

[10] X. Cai and D. C. Wunsch II, "Computer Go: A grand challenge to AI," in *Studies in Computational Intelligence*, W. Duch and J. Mandziuk Eds. Springer, vol. 63, pp. 443C465, 2007.

[11] C. Cai and S. Ferrari, "On the Development of an Intelligent Computer Player for CLUE®, the Great Detective Boardgame: A Case Study in Preposterior Decision Analysis," *Proc. of the 2006 American Control Conference*, Minneapolis, MN, pp. 4350-4355, June 2006.

[12] S. Ferrari, C. Cai, R. Fierro and B. Perteet, "A Multi-Objective Optimization Approach to Detecting and Tracking Dynamic Targets in Pursuit-Evasion Games," *Proc. of the 2007 American Control Conference*, New York, NY, pp. 5316-5321, July 2007.

[13] C. Cai and S. Ferrari, "Comparison of Information-Theoretic Objective Functions for Decision Support in Sensor Systems," *Proc. of the 2007 American Control Conference*, New York, NY, pp. 63-133, July 2007.

[14] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming.* Wiley, New York, 1994.

[15] S. Rajneesh and G. Madan, "Game-theoretic-reinforcement-adaptive neural controller for nonlinear systems," *Proc. of the 2006 American Control Conference*, Minneapolis, MN, pp. 2975-2980, June 2006.

[16] M. L. Littman, "Markov Games as a framework for Multi-agent Reinforcement Learning", *Proc. of Eleventh International Conference on Machine Learning*, Morgan Kaufman, pp. 157-163, 1994.

[17] G. Tesauro, "eurogammon: a neural network Backgammon program," *Proc. of Int. Joint Conf. On NN*, vol. 3, pages 33–39, 1990.

[18] G. Tesauro, "Temporal difference learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58-68, March 1995.

[19] S. Russell and P. Norvig, Chapter 5, 17 and 21 in *Artificial Intelligence: A Modern Approach.* Prentice Hall, NJ, 2003.

[20] F. V. Jensen, Chapter 1 in *Bayesian Networks and Decision Graphs.* Springer, 2001.

[21] R. Cowell, "Introduction to inference for Bayesian networks," in *Learning in Graphical Models.* M. Jordan, Ed., the MIT press, pp. 9-26, 1998

[22] K. Murphy, *How to Use Bayes Net Toolbox.* [Online]. Available: http://www.cs.ubc.ca/ murphyk/Software/BNT/bnt.html, 2004.

[23] E. F. Krause, *Taxicab geometry: an adventure in non-Euclidean geometry.* Dover, New York, 1986.

[24] R. A. Howard, "Information Value Theory," *IEEE Transactions on Systems Science and Cybernetics*, vol. 2, pp. 22-26, 1966.

[25] Y. Bar-Shalom and E. Tse, "Caution, Probing, and the Value of Information in the Control of Uncertain Systems," *Annals of Economic and Social Measurements*, Vol. 5, No. 2, pp. 323-337, 1976.