LS-DRAUGHTS - UM SISTEMA DE APRENDIZAGEM PARA DAMAS COM GERAÇÃO AUTOMÁTICA DE CARACTERÍSTICAS

HENRIQUE CASTRO NETO, RITA MARIA SILVA JULIA

Faculdade de Computação, Universidade Federal de Uberlândia
Av. João Naves de Ávila, 2121- Bloco B, sala 1B80, Bairro Santa Mônica, 38400-902, Uberlândia - MG
E-mails: henriquecneto@yahoo.com.br, rita@ufu.br

Abstract— The objective of this work is to propose a Learning System for Draughts – the LS-Draughts, which aims, by using Genetic Algorithms (GAs), to generate, automatically, a concise and efficient set of features which are relevant to represent the game board states and to optimize the training of a draught player agent. This agent consists on an Artificial Neural Network whose weights are updated by the Temporal Differences (TD) Reinforcement Learning methods. The NET-FEATUREMAP mapping is used to represent a game board state in the Network input. The Network output corresponds to a real number (prediction) that indicates how much the input state is favorable to the agent. The agent is trained by the self-play with cloning technique and the best action to be executed considering the game state is chosen by means of Minimax algorithm. Such a learning process is close to that one proposed by Mark Lynch in NeuroDraughts. However, the LS-Draughts expands the NeuroDraughts as it generates, automatically, an effective and concise set of features to be used in the NET-FEATUREMAP mapping, whereas, the last one uses a fixed and manually defined set of features. A tournament was promoted between the best player obtained by the LS-Draughts and the available player of the NeuroDraughts. The tournament was won by the player of the LS-Draughts, which confirms that the GAs can be an important tool to improve the general performance of automatic players.

Keywords— Artificial Neural Network, Machine Learning, Reinforcement Learning, Genetic Algorithms, Temporal Difference, Game Theory.

Resumo— O objetivo deste trabalho é apresentar um Sistema de Aprendizagem para Damas – o LS-Draughts, que visa, por meio da técnica dos Algoritmos Genéticos (AGs), gerar, automaticamente, um conjunto de características mínimas necessárias e essenciais de um jogo de Damas, de forma a otimizar o treino de um agente jogador. Este agente consiste em uma Rede Neural Artificial no qual os pesos são atualizados através do método de Aprendizagem por Reforço TD(λ) - método das Diferenças Temporais. O mapeamento *NET-FEATUREMAP* é utilizado para representar o estado do tabuleiro do jogo na entrada da rede neural. A saída da rede corresponde a um número real (predição) que indica o quanto o estado de entrada da rede é favorável ao agente. O agente é treinado utilizando a técnica de treinamento por *self-play* com clonagem e a melhor ação a ser executada em função do estado do jogo é escolhida por meio do algoritmo de busca *Minimax*. Tal processo de aprendizagem é análogo ao do jogador NeuroDraughts de Mark Lynch. Entretanto, o LS-Draughts expande o NeuroDraughts ao fazer a geração automática de um conjunto eficaz e resumido de características a ser utilizado no mapeamento *NET-FEATUREMAP*, ao passo que, o último, utiliza um conjunto de características fixo e definido manualmente. Foi efetuado um torneio entre o melhor jogador obtido pelo LS-Draughts e o jogador disponível do NeuroDraughts. Os resultados do torneio, vencido pelo jogador do LS-Draughts, evidenciam o fato de o AG representar uma importante ferramenta de melhoria no desempenho geral desses jogadores automáticos.

Palavras-chave— Rede Neural Artificial, Aprendizagem de Máquina, Aprendizagem por Reforço, Algoritmos Genéticos, Diferenças Temporais, Teoria dos Jogos.

1 Introdução

O paradigma da Aprendizagem por Reforço tem sido de grande interesse na área da aprendizagem automática, uma vez que dispensa um "professor" inteligente para o fornecimento de exemplos de treinamento. Este fato o torna particularmente adequado a domínios complexos em que a obtenção destes exemplos seja difícil ou até mesmo impossível [10]. Dentre os métodos de Aprendizagem por Reforço, os métodos TD(λ) se destacam por serem ampla e eficazmente utilizados, inclusive, na construção de agentes capazes de aprender a jogar Damas, Xadrez, Go, Gamão, Othello ou outros jogos [6], [8], [11], [12], [14], [17] e [19]. Tais agentes têm demonstrado que os jogos são, sem dúvida nenhuma, um ótimo domínio para se estudar e checar a eficiência das principais técnicas de aprendizagem automática.

Como exemplo de bons jogadores automáticos, cita-se o jogador de damas de Mark Lynch – NeuroDraughts – que consiste em uma rede neural que utiliza a busca *minimax* e o método TD, juntamente com a estratégia de treinamento por *self-play* com clonagem, como ferramentas para atualização de seus pesos e para o aprendizado de damas [7] e [8]. O NeuroDraughts utiliza o mapeamento *NET-FEATUREMAP* como técnica para representar os estados do tabuleiro do jogo na entrada da rede neural. Para isso, ele utiliza um conjunto de funções – definido como características – que descrevem qualitativa e quantitativamente as posições de peças sobre o tabuleiro. No NeuroDraughts, essas características são selecionadas manualmente e não variam (o jogador é treinado somente para um conjunto fixo de características). O bom desempenho obtido pelo NeuroDraughts em seu processo de aprendizagem mostra a eficiência de se utilizar o mapeamento *NET-FEATUREMAP* como técnica para representar os tabuleiros do jogo. Este trabalho visa estender o NeuroDraughts gerando, automaticamente, através da técnica dos Algoritmos Genéticos (AGs), um conjunto de características mínimas necessárias e essen-

ciais para descrever estados dos tabuleiros de Damas, de forma a tentar otimizar o processo de aprendizagem do agente jogador de Lynch.

A escolha do jogo de Damas como um domínio de aplicação se deve ao fato de que ele apresenta significativas semelhanças com inúmeros problemas práticos. Como exemplos desses problemas práticos, podem-se citar o da interação homem/máquina por meio de diálogo [20] e o do controle de tráfego veicular urbano [21]. Além disso, o jogo de Damas apresenta um nível de complexidade que demanda a utilização de técnicas poderosas como os métodos TD(λ), Busca *Minimax*, Redes Neurais e Algoritmos Genéticos.

2 Métodos das Diferenças Temporais em Jogos

Esta seção explica como os métodos de Aprendizagem por Reforço TD(λ) podem ser utilizados por uma rede neural jogadora. A rede é recompensada positivamente se obtiver um bom desempenho no jogo (em caso de vitória, ela recebe do ambiente um reforço positivo correspondente ao estado de fim de jogo) e ela é punida se obtiver um mau desempenho (em caso de derrota, ela recebe do ambiente um reforço negativo correspondente ao estado de fim de jogo). Para todos os estados de tabuleiro de jogo intermediário (isto é, entre o tabuleiro inicial e o tabuleiro final) representados na camada de entrada da rede, enquanto nenhuma recompensa específica está disponível, o mecanismo TD calcula a predição P de vitória por meio da seguinte equação:

$$P = g(in^{output}), \quad (1)$$

onde g é a função tangente hiperbólica e in^{output} é o campo local induzido sobre o neurônio da camada de saída da rede [4] e [8]. Note que o valor de P depende dos pesos da rede e corresponde a um número real no intervalo [-1,+1] que indica o quanto o estado do tabuleiro do jogo, representado na entrada da rede neural, é favorável ao agente. Cada vez que o agente deve mover uma peça sobre o tabuleiro, o algoritmo de busca *minimax* é utilizado para construir uma árvore com profundidade *n* e cuja raiz S representa o estado corrente do jogo (resultante do último movimento do oponente). Os nós filhos da raiz S correspondem aos estados do tabuleiro que podem ser originados a partir de cada movimento de peça possível para o agente jogador a partir do estado descrito na raiz. Os nós do nível seguinte correspondem a todos os estados do tabuleiro que podem ser originados a partir de cada movimento de peça possível para o jogador oponente a partir dos estados do nível anterior. A mesma estratégia segue em curso até o nível de profundidade *n*. Em seguida, a rede calcula as predições P para cada estado de profundidade *n*. Estes valores são retornados ao algoritmo de busca *minimax* a fim de lhe permitir indicar ao agente qual a melhor ação a ser escolhida e executada em S. Sempre que o agente executar um movimento, os pesos da rede são atualizados de acordo com a equação 2 [13]:

$$\Delta w_t = \alpha (P_t - P_{t-1}) \sum_{k=1}^{t-1} \lambda^{(t-1)-k} \nabla_w P_k, \quad (2)$$

onde P_t é a predição correspondente ao estado do tabuleiro corrente S_t, P_{t-1} é a predição correspondente ao estado do tabuleiro anterior S_{t-1}, cada P_k representa a predição correspondente a um estado de tabuleiro anterior a S_t, α é a taxa de aprendizagem (que define o quão rápido o sistema atualizará os pesos da rede), λ é uma constante que define o quanto o sistema considerará o impacto de estados anteriores a S_t no processo de atualização do pesos da rede e $\nabla_w P_k$ corresponde a derivada parcial de P_k em relação à variável w (peso).

Uma pequena revolução no campo da Aprendizagem por Reforço ocorreu quando Gerald Tesauro apresentou os seus primeiros resultados de treino de uma função de avaliação por meio do método das Diferenças Temporais [16], [17] e [18]. O programa de Tesauro, TD-Gammon, é um jogador de gamão que, inicialmente, apesar de ter pouco conhecimento sobre esse jogo, é capaz de aprender a jogar tão eficientemente quanto os maiores jogadores mundiais [17]. O princípio dos métodos TD(λ) foi primeiramente aplicado por Samuel em seu jogador de damas de 1959, onde ele já utilizava a idéia de atualização de avaliações baseada em predições sucessivas [11]. Um outro trabalho que também obteve sucesso com os métodos TD(λ) é o que foi proposto por Jonathan Schaeffer e outros pesquisadores em [15]. Eles realizaram um estudo detalhado de comparação entre uma função de avaliação treinada manualmente por peritos (que é o caso do atual campeão de damas CHINOOK [14]) e uma função de avaliação treinada pelos métodos TD(λ). Esta análise mostrou que os métodos TD(λ) aliados à estratégia de treinamento por *self-play* tornam-se uma poderosa ferramenta na construção de agentes automáticos capazes de jogar com um alto nível de desempenho.

3 Computação Evolutiva

A Computação Evolutiva é uma área da Ciência da Computação que utiliza idéias da evolução biológica para resolver problemas computacionais que, em sua maioria, requerem busca em um grande espaço de soluções possíveis [9]. Existem diversos métodos ou abordagens para sistemas baseados em evolução no campo da Computação Evolutiva. O termo geral que se utiliza para denominar tais métodos é *algoritmos evolutivos*. A abordagem mais comum e largamente utilizada de algoritmos evolutivos é o Algoritmo Genético (AG) [9], que também é um dos focos importantes neste trabalho.

A aplicação da Computação Evolutiva em jogos tem-se mostrado bastante eficiente na obtenção de

bons agentes jogadores. A base da Computação Evolutiva é o *teorema do esquema*, modelado, matematicamente por Holland [5].

Dentre os trabalhos mais conhecidos e bem sucedidos que utilizam algoritmos evolutivos em jogos, destacam-se os de David Fogel. Em [2] e [3] Fogel utilizou algoritmo evolutivo para evoluir os pesos do seu jogador de damas ANACONDA [2] e do seu jogador de xadrez BLONDIE25 [3].

4 Diferenças Temporais x Computação Evolutiva

Paul Darwen demonstrou em [1] a vantagem de se utilizar Diferenças Temporais no treinamento de redes neurais multicamadas devido a rapidez com que a rede aprende um comportamento não linear sobre um determinado problema. Darwen demonstra esta questão ao discutir o porquê da co-evolução conseguir bater, para uma arquitetura de rede linear, a aprendizagem por Diferença Temporal no jogo do Gamão, mas não conseguir o mesmo feito para uma arquitetura de rede não linear. O autor mostra que, se são necessários bilhões de jogos para que uma arquitetura não-linear treinada por um método co-evolutivo consiga bater uma outra arquitetura não-linear treinada pelo método TD(λ), a qual, por sua vez, requer apenas alguns poucos 100.000 jogos para aprender, então muitos dos bilhões de jogos do método co-evolutivo não estarão, de fato, contribuindo para a aprendizagem. Este fato também parece ser aplicado ao domínio de damas. Por exemplo, o jogador ANACONDA precisou de 126.000 jogos de treinamento para apresentar o mesmo nível de desempenho do CHINOOK, enquanto que o jogador de damas de Schaeffer, treinado pelos métodos TD(λ) [15], precisou de apenas 10.000 jogos de treinamento para obter o mesmo resultado.

Considerando os resultados de Darwen, este trabalho tem por objetivo aliar os benefícios da utilização da técnica de aprendizagem por reforço TD(λ) e dos AGs na construção do sistema LS-Draughts, o que representa uma versão expandida do agente jogador de damas de Mark Lynch. Com os AGs, o LS-Draughts pretende gerar, selecionar e recombinar as características que se perpetuarão ao longo das gerações a fim de otimizar o treinamento de uma rede neural por Diferenças Temporais. Mais detalhes sobre a implementação do LS-Draughts serão visto na próxima seção.

5 O LS-DRAUGHTS

O LS-Draughts é um sistema de aprendizagem de jogos de damas que tem como objetivo principal construir um agente automático capaz de jogar damas com alto nível de desempenho. O agente consiste em uma Rede Neural Artificial cujos pesos são atualizados pelos métodos TD(λ). O mapeamento NET-FEATUREMAP é utilizado para representar os estados do tabuleiro do jogo na entrada da rede. Tais representações se baseiam em conjuntos de características gerados, automaticamente, por AGs. A saída da rede corresponde a um número real (predição) que indica o quanto aquele estado de entrada é favorável ao agente. Além disso, o agente também é treinado utilizando a estratégia de treinamento por *self-play* com clonagem e a melhor ação a ser executada em função do estado do jogo é escolhido por meio da busca *minimax*.

A arquitetura geral do LS-Draughts está indicado na figura abaixo:

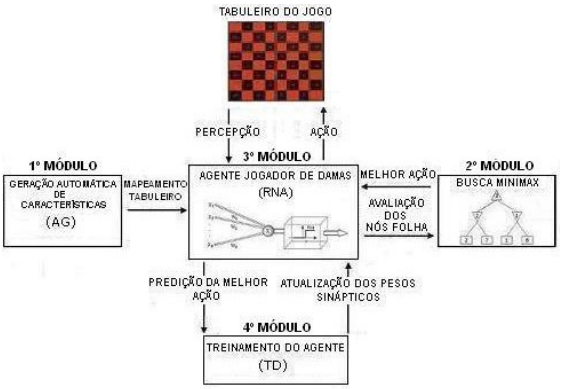

Figura 1. Arquitetura do LS-Draughts.

Conforme a figura 1, o sistema é composto de 4 módulos principais:

- Primeiro Módulo – gerador de características: corresponde ao próprio AG que gerará T_p indivíduos que representam subconjuntos de todas as características disponíveis no mapeamento NET-FEATUREMAP;
- Segundo Módulo – classificador e selecionador de ações: este módulo corresponde ao algoritmo de busca *minimax* cuja função é selecionar a melhor ação a ser executada pelo agente em função do estado do tabuleiro do jogo corrente.
- Terceiro Módulo – agente jogador: corresponde a Rede Neural Artificial cujo processo de aprendizagem é guiado pelo 4º módulo;
- Quarto Módulo – o treinamento: este módulo corresponde ao treinamento do agente jogador através do método das Diferenças Temporais aliado com a estratégia de treinamento por *self-play* com clonagem.

Note que o processo de treinamento do LS-Draughts é similar ao que foi proposto por Lynch no NeuroDraughts (segundo, terceiro e quarto módulos) [8]. Entretanto, o quarto módulo do LS-Draughts modifica o processo de treinamento do NeuroDraughts da seguinte forma: no LS-Draughts, inúmeros indivíduos - isto é, inúmeros conjuntos de caracterís-

ticas - são treinados, enquanto que no Neuro-Draughts, somente um indivíduo é treinado. Além disso, o primeiro módulo estende o NeuroDraughts gerando, automaticamente e por meio dos AGs, as características que representarão os estados do tabuleiro do jogo. O objetivo do processo de treinamento do LS-Draughts é, então, detectar, dentre estes conjuntos de características, aquelas que são necessárias e essenciais para produzir um jogador de damas eficiente. As interações entre o segundo, terceiro e quarto módulos já foram descritos na seção 2. Mais detalhes sobre esta interação podem ser visto em [8]. Na próxima seção, os autores apresentarão as estruturas do primeiro e quarto módulos que caracterizam o LS-Draughts.

5.1 População e codificação dos indivíduos no LS-Draughts

Cada indivíduo na população é codificado como um cromossomo binário de comprimento 15. A representação binária indica se uma determinada característica F_i está presente ou não no gene G_i, onde $i \in \{1,2,3,...,15\}$, conforme é mostrado na figura 2.

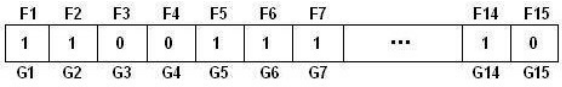

Figura 2. Exemplo de codificação de um indivíduo na população.

A figura 3 mostra as 15 características utilizadas na representação dos 15 genes. Cada número inteiro na coluna BITS correspondente a uma característica F_i indica a quantidade de neurônios que serão alocados para representar F_i na camada de entrada da rede neural.

FEATURES	BITS
F1: PeçasEmVantagens	4
F2: PeçasEmDesvantagens	4
F3: PeçasAmeaçadas	3
F4: PeçasEmAtaque	3
F5: PeçasEmAvanço	3
F6: DiagonalDupla	4
F7: LinhaDePonte	1
F8: ControlePeçasCentral	3
F9: ControlePeçasCentralX	3
F10: TotalMobilidade	4
F11: PeçasExpostas	3
F12: ControleRainhasCentro	3
F13: DiagonalMomentânea	3
F14: Ameaça	3
F15: Ataque	3

Figura 3. Lista das 15 características candidatas para representar o estado do tabuleiro do jogo no LS-Draughts.

Neste trabalho, a população do AG é composta por 50 indivíduos, isto é, $T_P = 50$. Portanto, a população será formada por 50 estruturas cromossômicas (ou indivíduos), onde cada uma delas estará associada a uma rede neural. São esses 50 indivíduos que evoluirão dentro do AG ao longo de 30 gerações.

Os indivíduos do LS-Draughts são gerados de duas formas:

1. Todos os 50 indivíduos da primeira geração GE_0 são gerados como se segue: há uma escolha aleatória de ativação (1 ou 0) do gene G_i correspondente a característica F_i para cada $i \in \{1,2,3,...,15\}$. Depois, cada um desses indivíduos (que representa o estado do jogo) é introduzido na entrada da rede neural que lhe corresponde. As 50 redes neurais produzidas serão treinadas (o processo de treinamento será discutido mais adiante). Depois disso, o LS-Draughts inicia um torneio envolvendo as 50 redes treinadas. Ao fim do torneio, uma avaliação (ou *fitness*) é calculada para cada indivíduo em virtude do seu desempenho durante o torneio (como será detalhado na subseção 5.4). Em seguida, os 50 indivíduos de GE_0 serão repassados como pais para a próxima geração (geração GE_1);
2. Todos os 50 indivíduos das 29 gerações restantes GE_i, onde $1 \le i \le 29$, são gerados como se segue: 50 novos indivíduos são gerados através da aplicação dos operadores genéticos de *crossover* e mutação sobre 25 pares de indivíduos escolhidos pelo torneio estocástico dentre uma população de 50 pais recebidos da geração GE_{i-1}. Depois, as 50 novas redes neurais acopladas a estes 50 novos indivíduos são treinadas. Em seguida, o LS-Draughts inicia um torneio envolvendo as 100 redes treinadas disponíveis (50 correspondente a GE_{i-1} e 50 correspondente a GE_i). Ao fim do torneio, uma avaliação (ou *fitness*) é calculada para cada indivíduo em função do seu desempenho durante o torneio. Os 50 indivíduos que apresentarem os melhores *fitness* serão repassados como pais para a próxima geração GE_{i+1}. Este processo se repete para cada geração, até o fim da geração 29.

5.2 Seleção dos indivíduos e aplicação dos operadores genéticos

O método de seleção utilizado pelo LS-Draughts para selecionar os pais a fim de aplicar os operadores genéticos é o torneio estocástico com $tour = 3$ [9]. Para cada dois pais selecionados pelo torneio, dois novos filhos são gerados. O método de *crossover* utilizado é o cruzamento simples de genes (*crossover* de um único ponto de corte) com probabilidade de 100%. A taxa de probabilidade de mutação utilizada é de 30% por indivíduo.

5.3 *Treinamento das Redes Neurais*

A rede neural multicamada associada a um indivíduo I_i da população, onde $i \in \{1,2,3,...,50\}$, tem N_A neurônios na camada de entrada, onde N_A representa a quantidade de bits associados aos genes ativos (digito 1) em I_i. A camada oculta tem 20 neurônios e a camada de saída é formada por um único neurônio. Os pesos iniciais da rede são gerados aleatoriamente entre -0.2 e +0.2 e o termo *bias* é fixado em 1.

O treinamento de cada rede neural consiste em um grupo de 4 sessões de 400 jogos (nestes jogos de treinamento a rede aprende por reforço conforme foi descrito na seção 2), sendo que metade desses 400 jogos a rede joga como jogador preto (peças pretas) e a outra metade como jogador vermelho (peças vermelhas). Antes do início das 4 sessões de treinamento por *self-play*, é feita uma cópia da rede neural net_i associada ao indivíduo I_i, produzindo a rede clone $cnet_i$. Em seguida, net_i e $cnet_i$ jogam os primeiros 400 jogos correspondente à primeira sessão de treinamento. Durante esses jogos, somente os pesos de net_i são atualizados. Ao fim da primeira sessão, dois jogos-teste são realizados para checar se a nova rede net_i correspondente ao indivíduo I_i tornou-se melhor que seu clone. Caso afirmativo, os pesos da rede net_i são copiados novamente para $cnet_i$ e a próxima sessão de treinamento é iniciada com os jogadores net_i e a rede clone modificada $cnet_i$. Caso contrário, $cnet_i$ não é modificada e a próxima sessão é iniciada com os mesmos jogadores (net_i e $cnet_i$) que terminaram a última sessão. Este processo se repete até o fim da 4ª sessão. Note que, em todos esses jogos, ambas as redes jogadoras usam a mesma estratégia para escolher a melhor ação, conforme foi descrito na seção 2.

Considerando a possibilidade de que nem sempre a rede final net_i obtida ao fim das quatro sessões de treinamento é realmente a melhor (isso ocorre porque a rede net_i pode se especializar em bater apenas o seu último clone durante o processo de treinamento), um pequeno torneio é realizado entre net_i e todos os seus clones gerados nas quatro sessões de treinamento. O vencedor do torneio é considerado como sendo a melhor rede correspondente ao indivíduo I_i que tem sido treinado.

5.4 *Cálculo do Fitness*

No torneio organizado para calcular o *fitness* dos indivíduos de uma dada geração (citado na subseção 5.1), cada indivíduo I_i joga 10 jogos contra o restante dos indivíduos daquela geração. A pontuação utilizada para calcular o *fitness* de I_i é dada da seguinte forma: 2 pontos por cada vitória, 1 ponto por empate e 0 ponto para cada derrota.

6 Resultados Experimentais

A figura 4 mostra os resultados obtidos durante os 4 meses de treinamento do sistema LS-Draughts, onde 30 gerações foram produzidas. Nesta figura, o *fitness* do melhor indivíduo em cada grupo de 5 gerações foi comparado com a média do *fitness* dos 49 indivíduos restantes.

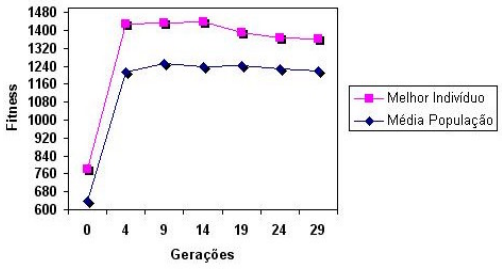

Figura 4. Gráfico de evolução do melhor indivíduo da população em relação à média da população nas gerações 0, 4, 9, 14, 19, 24 e 29.

Com o objetivo de comparar o NeuroDraughts com o LS-Draughts, foi realizado um torneio de 7 jogos entre o melhor indivíduo de cada uma das 30 gerações executadas no LS-Draughts e o jogador de Mark Lynch disponível em seu website. Até a 8ª geração do LS-Draughts, o melhor indivíduo não obteve sucesso. Porém, na 9ª geração, o melhor indivíduo I_{B-9} conseguiu obter uma pontuação favorável comparado com o jogador do NeuroDraughts: 1 vitória e 6 empates. Como I_{B-9} possui 12 genes ativos (isto é, 12 características), ele já pode ser considerado como sendo um bom resultado produzido pelo LS-Draughts, já que ele consegue bater o jogador de Mark Lynch contando com a mesma quantidade de características que este último (12). I_{B-9} permaneceu como melhor indivíduo da população até a 15ª geração. Da 16ª até a 24ª geração, nenhum melhor indivíduo conseguiu bater o jogador de Mark Lynch. Somente na 25ª geração é que o melhor indivíduo I_{B-25} conseguiu bater o jogador de Mark Lynch contando com apenas 7 características: 2 vitórias e 5 empates. O I_{B-25} permaneceu como melhor indivíduo da população até a última geração (30ª geração).

Analisando os 5 empates obtidos pelo indivíduo I_{B-25} contra o NeuroDraughts, foi observado que em 2 deles o I_{B-25} teria vencido se ele estivesse apto a detectar o *loop* de final de jogo.

Um torneio também foi realizado entre os indivíduos I_{B-9} e I_{B-25}. O resultado foi: 2 vitórias para cada lado e 10 empates, o que mostra um desempenho similar para ambos os jogadores. Entretanto, vale lembrar que I_{B-25} tem um conjunto de características menor que I_{B-9}. Assim, dependendo do crité-

rio (desempenho e/ou tamanho do conjunto de características) que alguém pretende otimizar quando escolhe um dos dois jogadores gerados pelo sistema LS-Draughts, I_{B-9} pode ser considerado como o melhor jogador, se somente o critério tempo for utilizado como parâmetro de otimização (visto que I_{B-9} é gerado primeiro). Por outro lado, I_{B-25} pode ser considerado como o melhor jogador do LS-Draughts se ambos os critérios (desempenho e/ou tamanho do conjunto de características) forem considerados como parâmetro de otimização.

7 Conclusões

Foi apresentado o LS-Draughts – um Sistema de Aprendizagem de Damas – que, utilizando AGs, métodos TD(λ), busca *Minimax* e estratégia de aprendizagem por *self-play* com clonagem, gerou agentes jogadores de damas capazes de bater o jogador de Mark Lynch em um torneio de 7 jogos. Os resultados obtidos confirmam o melhor desempenho do LS-Draughts e, conseqüentemente, ratificam a contribuição da inserção do módulo de geração automática de características, por meio dos AGs, no sistema original NeuroDraughts de Mark Lynch. Por outro lado, o *loop* de final de jogo apresentado na seção 6 mostra que o LS-Draughts ainda deve ser trabalhado e aperfeiçoado a fim de melhorar o seu desempenho geral, tornando-se mais competitivo.

Referências Bibliográficas

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

[2] D. B. Fogel e K. Chellapilla. (2002). Verifying anaconda's expert rating by competing against Chinook: experiments in co-evolving a neural checkers player, *Neurocomputing*, v. 42, n.1-4, pp. 69-86.

[3] D. B. Fogel, T. J. Hays, S. L. Hahn e J. Quon . (2004). A self-learning evolutionary chess program, *Proceedings of the IEEE*, v. 92, n. 12, pp. 1947-1954.

[4] S. Haykin. (1998). Neural Networks: A Comprehensive Foundation, Second Edition, Prentice Hall, 1998.

[5] J. H. Holland. (1992). Adaptation in natural and artificial systems, Second Edition, Cambridge, MA, USA, MIT Press.

[6] A. Leuski. (1995). Learning of position evaluation in the game of Othello. Disponível em: http://people.ict.usc.edu/~leuski.

[7] M. Lynch e N. Griffith. (1997). Neurodraughts: the role of representation, search, training regime and architecture in a td draughts player, *Eighth Ireland Conference on Artificial Intelligence*, pp. 64-72. Disponível em: http://iamlynch.com/nd.

[8] M. Lynch. (1997). NeuroDraughts: An application of temporal difference learning to draughts. Disponível em: http://iamlynch.com/nd.

[9] M. Mitchell e C. E. Taylor. (1999). Evolutionary Computation: An Overview, *Annual Review of Ecology and Systematics*, v. 30, pp. 593-616.

[10] S. Russell e P. Norvig. (2003). Artificial Intelligence: A Modern Approach, Second Edition, Prentice Hall.

[11] A. L. Samuel. (1959). Some studies in machine learning using the game of checkers, IBM Journal of Research and Development, v. 3, n. 3, pp. 211-229.

[12] N. N. Schraudolph, P. Dayan e T. J. Sejnowski. (2001). Learning to evaluate go positions via temporal difference methods, *Computational Intelligence in Games Studies in Fuzziness and Soft Computing*, Spring Verlag, v.62.

[13] R. S. Sutton. (1988). Learning to predict by the methods of temporal differences, *Machine Learning*, v. 3, n. 1, pp. 9-44.

[14] J. Schaeffer, R. Lake, P. Lu e M. Bryant. (1996). CHINOOK: The world man-machine checkers champion, *AI Magazine*, v. 17, n.1, pp. 21-29.

[15] J. Schaeffer, M. Hlynka e V. Jussila. (2001). Temporal difference learning applied to a high performance game-playing program, *Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI), pp. 529-534.

[16] G. J. Tesauro. (1992). Practical issues in temporal difference learning, *Machine Learning*, 8.

[17] G. J. Tesauro. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play, *Neural Computation*, v. 6, n. 2, pp. 215-219.

[18] G. J. Tesauro. (1995). Temporal difference learning and td-gammon, *Communications of the ACM*, v. 38, n.3, pp. 19-23.

[19] S. Thrun. (1995). Learning to play the game of chess, *Advances in Neural Information Processing Systems 7*, The MIT Press, pp. 1069-1076.

[20] M. A. Walker. (2000). An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email, *Journal of Artificial Intelligence Research 12*, pp. 387-416.

[21] M. Wiering. (2000). Multi-agent reinforcement learning for traffic light control, *Proceedings of the 17th International Conference on Machine Learning*, pp. 1151-1158.