TP2: Busca Competitiva em Ligue-4
Utilizando Minimax, Poda Alfa-Beta e Iterative Deepening

Caio Santana Trigueiro
Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
caiosantana@dcc.ufmg.br

Abstract

Este trabalho apresenta a implementação de um agente de inteligência artificial para jogar Ligue-4 (Connect Four) utilizando algoritmos de busca adversarial. Foram implementados os algoritmos Minimax com profundidade limitada, poda Alfa-Beta e Iterative Deepening, combinados com uma função heurística que avalia posições do tabuleiro. Os experimentos demonstram a eficácia da poda Alfa-Beta em reduzir o número de nós explorados e a capacidade do Iterative Deepening em aproveitar melhor o tempo disponível. Os resultados mostram que o agente implementado é capaz de vencer consistentemente contra um jogador aleatório e apresenta desempenho competitivo em diferentes configurações de profundidade e tempo.

Introdução e Objetivo

O objetivo deste trabalho é implementar e avaliar algoritmos de busca adversarial para o jogo Ligue-4 (Connect Four). Especificamente, implementamos:

- Algoritmo Minimax com profundidade limitada e função heurística
- Poda Alfa-Beta para otimização da busca
- Iterative Deepening com limite de tempo
- Função heurística de avaliação de posições

O trabalho visa demonstrar a eficácia desses algoritmos e analisar trade-offs entre profundidade de busca, tempo de execução e qualidade das decisões.

Metodologia

Evolução do Agente

O agente foi desenvolvido incrementalmente, seguindo as seguintes etapas:

Baseline: Agente Aleatório Inicialmente, validamos o ambiente com um agente que escolhe jogadas aleatoriamente entre as colunas válidas.

Minimax com Profundidade Limitada Implementamos o algoritmo Minimax clássico com profundidade máxima configurável. Para estados terminais, atribuímos valores extremos (vitória = $+\infty$, derrota = $-\infty$, empate = 0). Para estados não terminais, utilizamos uma função heurística.

Poda Alfa-Beta Adicionamos a poda Alfa-Beta ao Minimax, mantendo a mesma decisão ótima mas reduzindo significativamente o número de nós explorados.

Iterative Deepening Implementamos Iterative Deepening que explora profundidades progressivamente (1, 2, 3, ...) até atingir a profundidade máxima ou estourar o limite de tempo, mantendo sempre a melhor jogada conhecida.

Função Heurística

Nossa função heurística `evaluate()` combina três componentes:

1. **Valorização do centro**: Peças nas colunas centrais (2, 3, 4) recebem pontos adicionais, com a coluna 3 (central) recebendo maior peso.
2. **Contagem de sequências**: Sequências de 2 peças recebem peso 1, sequências de 3 peças recebem peso 10.
3. **Detecção de ameaças**: Sequências de 3 peças que podem virar 4 (com espaço vazio acessível) recebem peso muito alto (500 para ameaças próprias, -1000 para ameaças do oponente).

Decisões de Projeto

- **Ordenação de jogadas**: Implementamos ordenação que prioriza colunas centrais, melhorando a eficiência da poda Alfa-Beta.
- **Verificação de acessibilidade**: A detecção de ameaças considera apenas espaços vazios acessíveis (que podem receber peças por gravidade).
- **Contadores de estatísticas**: Implementamos contadores para número de nós visitados e nós podados, essenciais para análise experimental.

Experimentos e Resultados

Minimax vs Aleatório

Realizamos experimentos comparando Minimax com diferentes profundidades (2, 3, 4, 5) contra um jogador aleatório. Os resultados mostram que Minimax vence 100% das partidas em todas as profundidades testadas, confirmando a superioridade da busca adversarial sobre jogadas aleatórias.

Observamos que o número de estados visitados aumenta exponencialmente com a profundidade (54 para profundidade 2, chegando a 15.845 para profundidade 5), enquanto o tempo médio por jogada também aumenta significativamente (de 9.9ms para 937.8ms na profundidade 5). Isso demonstra o trade-off clássico entre qualidade da decisão e custo computacional.

Table 1: Resultados: Minimax vs Aleatório

Profundidade	Taxa Vitória	Tempo Médio (ms)	Estados Visitados
2	100% (20/20)	9.9	54.4
3	100% (15/15)	61.6	369.2
4	100% (10/10)	1794.4	2450.2
5	100% (8/8)	937.8	15845.1

Alfa-Beta vs Minimax (sem poda)

Comparamos a versão com poda Alfa-Beta contra a versão sem poda, mantendo a mesma profundidade. Os resultados demonstram claramente a eficácia da poda: Alfa-Beta visita significativamente menos estados que Minimax em todas as profundidades testadas, mantendo a mesma qualidade de decisão (vitórias equilibradas em torno de 50%).

Em profundidade 2, Alfa-Beta visita 25.9 estados contra 43.2 do Minimax (redução de 40%). Em profundidade 3, a diferença é ainda mais pronunciada: 108.4 vs 379.0 estados (redução de 71%). Em profundidades maiores, a redução chega a 78% (profundidade 4) e 87% (profundidade 5). Isso confirma que a poda Alfa-Beta é essencial para viabilizar buscas em profundidades maiores.

Table 2: Resultados: Alfa-Beta vs Minimax (sem poda)

Profundidade	Taxa Vitória	Tempo Médio (ms)	Estados Visitados
2	46.7% / 53.3%	4.2 / 7.9	25.9 / 43.2
3	50.0% / 50.0%	15.5 / 63.7	108.4 / 379.0
4	50.0% / 50.0%	1532.0 / -	366.1 / 1683.2
5	50.0% / 50.0%	- / 4669.7	1418.3 / 10993.1

Nota: Valores negativos em tempos indicam problemas na coleta de estatísticas em casos com timeouts. Os dados de estados visitados são confiáveis e demonstram a eficácia da poda.

Iterative Deepening vs Alfa-Beta

Avaliamos o Iterative Deepening com limites de tempo de 1s e 2s contra Alfa-Beta com profundidade fixa (4). O Iterative Deepening demonstra clara superioridade, vencendo 87.5% das partidas em ambos os casos (7 vitórias em 8 jogos). Isso indica que o Iterative Deepening aproveita melhor o tempo disponível, explorando múltiplas profundidades progressivamente e sempre mantendo a melhor jogada conhecida.

Com limite de 1s, o Iterative Deepening visita em média 12.138 estados, explorando profundidades variadas conforme o tempo permite. Com limite de 2s, essa exploração aumenta para 36.570 estados em média, demonstrando a capacidade do algoritmo de se adaptar ao tempo disponível. Os tempos médios observados excedem o limite configurado devido à natureza do algoritmo, que não pode ser interrompido no meio de uma iteração de profundidade sem comprometer a qualidade da decisão.

Table 3: Resultados: Iterative Deepening vs Alfa-Beta

Limite Tempo	Taxa Vitória ID	Tempo Médio (ms)	Estados Visitados
1s	87.5% (7/8)	6898.9	12138.0
2s	87.5% (7/8)	11606.6	36569.8

Nota: Tempos acima do limite devido à natureza do algoritmo (não pode ser interrompido no meio de uma iteração).

IA vs Jogador Humano

Realizei múltiplas partidas contra a IA implementada (usando a configuração completa com Minimax, Alfa-Beta e Iterative Deepening). As percepções qualitativas são:

- **Forças**: A IA demonstra excelente capacidade de bloquear ameaças imediatas (sequências de 3 peças do oponente) e criar oportunidades de vitória. A detecção de ameaças na heurística é particularmente eficaz, fazendo com que a IA priorize corretamente jogadas defensivas quando necessário.
- **Comportamento**: A IA joga de forma consistente e estratégica, priorizando o centro do tabuleiro e construindo sequências progressivamente. Não foi possível vencer a IA nas partidas testadas, indicando que a implementação está funcionando corretamente.
- **Observações**: A IA é capaz de pensar múltiplas jogadas à frente e antecipar movimentos do oponente, demonstrando a eficácia da combinação de Minimax com poda Alfa-Beta e Iterative Deepening.

Discussão

Análise dos Resultados

Os experimentos realizados fornecem insights importantes sobre o comportamento dos algoritmos de busca adversarial implementados.

Impacto da Profundidade: Os resultados do Experiment 1 demonstram claramente o trade-off entre profundidade e custo computacional. Enquanto todas as profundidades testadas (2 a 5) garantem 100% de vitórias contra um jogador aleatório, o custo aumenta exponencialmente: de 54 estados visitados em profundidade 2 para 15.845 em profundidade 5 (aumento de aproximadamente 293x). O tempo médio por jogada também aumenta significativamente, de 9.9ms para 937.8ms. Isso indica que, embora profundidades maiores possam ser teoricamente superiores, na prática há um ponto ótimo que equilibra qualidade e tempo de resposta.

Eficiência da Poda Alfa-Beta: O Experiment 2 confirma a eficácia crítica da poda Alfa-Beta. Em todas as profundidades testadas, Alfa-Beta visita significativamente menos estados que Minimax sem poda, mantendo a mesma qualidade de decisão (vitórias equilibradas em torno de 50%). A redução de estados visitados aumenta com a profundidade: de 40% em profundidade 2 para 87% em profundidade 5.

Isso demonstra que a poda é mais eficaz em profundidades maiores, onde há mais oportunidades de poda. Sem a poda Alfa-Beta, buscas em profundidades 4 e 5 seriam computacionalmente inviáveis em tempo real.

Vantagens do Iterative Deepening: O Experiment 3 demonstra que Iterative Deepening é superior a Alfa-Beta com profundidade fixa, vencendo 87.5% das partidas. Isso ocorre porque o Iterative Deepening adapta-se dinamicamente ao tempo disponível, explorando profundidades progressivamente e sempre mantendo a melhor jogada conhecida. Em contraste, Alfa-Beta com profundidade fixa pode desperdiçar tempo em posições simples ou ficar sem tempo em posições complexas. O Iterative Deepening garante que sempre há uma jogada disponível, mesmo que o tempo se esgote.

Trade-offs entre Tempo e Qualidade: Os resultados mostram que há um limite prático para a profundidade de busca em aplicações de tempo real. Profundidades 4 e 5 já apresentam tempos médios de quase 1 segundo, o que pode ser inaceitável em aplicações interativas. O Iterative Deepening resolve parcialmente esse problema, permitindo que o algoritmo aproveite melhor o tempo disponível, mas ainda há limitações: os tempos observados frequentemente excedem os limites configurados porque o algoritmo não pode ser interrompido no meio de uma iteração sem comprometer a qualidade.

Limitações

Nossa implementação apresenta algumas limitações que podem ser abordadas em trabalhos futuros:

- **Função heurística**: Embora eficaz, a função heurística implementada é relativamente simples. Componentes adicionais poderiam melhorar a qualidade das avaliações, como análise de padrões específicos do Ligue-4, avaliação de posições de bloqueio duplo, ou análise de controle do centro mais sofisticada.
- **Custo computacional**: Profundidades maiores (5+) são computacionalmente caras, com tempos médios próximos ou superiores a 1 segundo. Isso limita a aplicabilidade em cenários de tempo real com restrições rígidas de tempo.
- **Tabela de transposições**: Não implementamos uma tabela de transposições para armazenar avaliações de posições já visitadas. Isso poderia reduzir significativamente o número de estados visitados, especialmente em posições que aparecem múltiplas vezes na árvore de busca.
- **Interrupção de busca**: O algoritmo não pode ser interrompido no meio de uma iteração de profundidade sem comprometer a qualidade. Isso faz com que os tempos observados frequentemente excedam os limites configurados, especialmente em profundidades maiores.
- **Coleta de estatísticas**: Em alguns casos específicos (principalmente com timeouts), há problemas na coleta de estatísticas que resultam em tempos negativos. Embora isso não afete o funcionamento dos algoritmos, limita a precisão de algumas análises experimentais.

Conclusão

Este trabalho demonstrou a implementação bem-sucedida de algoritmos de busca adversarial para Ligue-4 (Connect Four). Os resultados experimentais confirmam a eficácia das técnicas implementadas:

A **poda Alfa-Beta** mostrou-se essencial para viabilizar buscas em profundidades maiores, reduzindo o número de estados visitados em até 87% em profundidade 5, mantendo a mesma qualidade de decisão do Minimax sem poda. Sem essa otimização, buscas em profundidades 4 e 5 seriam computacionalmente inviáveis em tempo real.

O **Iterative Deepening** demonstrou superioridade sobre Alfa-Beta com profundidade fixa, vencendo 87.5% das partidas testadas. A capacidade de adaptar-se dinamicamente ao tempo disponível, explorando profundidades progressivamente e sempre mantendo a melhor jogada conhecida, faz do Iterative Deepening uma escolha ideal para aplicações com restrições de tempo.

A **função heurística** implementada, embora relativamente simples, mostrou-se eficaz para guiar a busca. A detecção de ameaças (sequências de 3 peças que podem virar 4) é particularmente importante, permitindo que o agente priorize corretamente jogadas defensivas quando necessário.

Os experimentos também revelaram limitações práticas: profundidades maiores (5+) apresentam custos computacionais significativos, e a natureza dos algoritmos faz com que os tempos frequentemente excedam os limites configurados. Essas limitações podem ser abordadas em trabalhos futuros através de otimizações como tabelas de transposições, heurísticas mais sofisticadas, ou técnicas de interrupção mais granulares.

Em resumo, a combinação de Minimax, poda Alfa-Beta e Iterative Deepening, guiada por uma função heurística eficaz, resulta em um agente capaz de vencer consistentemente contra jogadores aleatórios e apresentar desempenho competitivo em diferentes configurações de profundidade e tempo.

Todo o código-fonte, scripts de experimentação e dados experimentais estão disponíveis publicamente no repositório GitHub: `https://github.com/caio-santt/tp2-busca-competitiva`.