Heurísticas para a Resolução de Problemas NP-Completos no Domínio de Jogos

Alex F. V. Machado Esteban W. G. Clua,
Gustavo M. Novaes* Carla R. B. Bonin* Mauro L. R. A. Filho*

Universidade Federal Fluminense, Dep. Ciência da Computação, RJ / Brasil
* Centro Federal de Educação Tecnológica, Dep. Informática Industrial, MG / Brasil

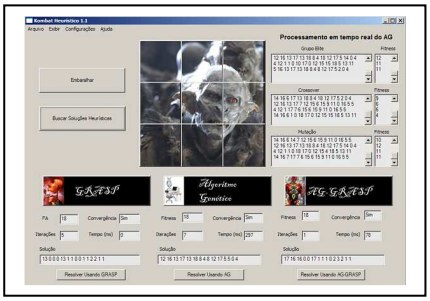

Fig. 1 – Interface do Kombat Heurístico 1.1.
Programa/jogo de avaliação do desempenho entre heurísticas no domínio de jogos de raciocínio baseado em quebra-cabeças.

Resumo

Jogos eletrônicos como quebra-cabeça, xadrez e puzzles necessitam de algoritmos de busca que muitas vezes exigem grande complexidade temporal e gasto de memória para encontrar os melhores lances. Este trabalho tem como objetivo, através da estrutura de busca em árvore no domínio de jogos, comparar a performance e a precisão das heurísticas Algoritmo Genético (AG) [Lacerda e Carvalho 2002], Procedimento de Busca Adaptativa Gulosa e Randomizada (GRASP) [Zanetti e Ochi 2005] e AG com GRASP para encontrar o conjunto solução de movimentos em um jogo de raciocínio (quebra-cabeças). As principais contribuições consistem nas modelagens GRASP e AG-GRASP para este grupo de jogos.

Palavras-chaves: heurísticas, IA, Algoritmo Genético, GRASP, árvore de busca de jogos, jogos de raciocínio, quebra-cabeças.

1. Introdução

Jogos de computadores representam um campo de pesquisa importante da área de inteligência artificial (AI) no qual os resultados podem ser aplicados a diversas outras áreas da ciência. Entre os domínios de jogos que utilizam AI é enfoque deste trabalho o grupo dos jogos de raciocínio (puzzle) que necessitam da solução dos problemas através dos movimentos de peças. Englobando os jogos de tabuleiros (board games) adaptados e criados para PC como xadrez, damas, puzzles, jogo-da-velha e quadrados mágicos, os quais podem ter uma modelagem em árvore de busca.

No jogo de quebra-cabeças abordado neste trabalho (Fig. 1) há um estado inicial indesejável (problema), um conjunto de regras de movimentos que permitem alterar os estados atuais e uma solução ótima a ser atingida. Para a busca desta solução foi reconstruído o algoritmo desenvolvido em [Hong 2001], e proposta duas novas abordagens ao mesmo problema: utilizando o motor GRASP puro e utilizando AG com GRASP (AG-GRASP).

1.1 Árvore de Busca no Domínio de Jogos

Árvore de Busca no Domínio de Jogos ou Game Search Tree (GST) para um jogador na teoria combinatória dos jogos é um grafo direcionado cujos nodos são as configurações de um jogo e os vértices são os movimentos possíveis [Hu e Shing 2002][Wikipedia 2006]. A árvore completa de um jogo começa na posição inicial e contém todos os movimentos possíveis para cada posição. Existem duas classificações de mecanismos para as GSTs: busca livre de um jogador (Exemplo: campo minado) e busca condicionada, em jogos com mais de um jogador onde a definição de um novo estado depende da jogada do oponente (Exemplo: jogo-da-velha). Este trabalho foca a primeira classificação.

1.2 Cálculo da Ordem de Complexidade

Para um algoritmo determinístico tradicional de busca no domínio dos jogos de raciocínio, como Busca em Profundidade, Busca em Largura e Busca Exaustiva, considerando n o número de movimentos possíveis por jogada e, uma vez que para cada movimento advêm outros n movimentos obtemos que para uma pesquisa com profundidade m o número de nós a pesquisar é n_1 x n_2 x n_3 x … x n_m. Desse modo a ordem de complexidade é $O(n^m)$.

Para o jogo do estudo de caso (quebra-cabeças 3x3), a ser demonstrado na seção 2, existem 18 movimentos possíveis, e considerando a profundidade 15 (número de movimentos para se alcançar a solução), seria inviável uma busca em profundidade que poderia levar 18^{15} iterações no pior caso. Para resolver problemas que podem gerar um tempo exponencial de processamento, quer dizer, problemas NP-Completos (Non-deterministic Polynomial Time), optou-se pelas heurísticas Algoritmo Genético, GRASP e uma junção de AG com GRASP para produzir soluções em um tempo polinominal de processamento.

2. Descrição do Jogo

O jogo usado como estudo de caso é uma versão do tradicional quebra-cabeças de tabuleiro, com pequenas modificações para aumentar a jogabilidade no computador. Embora a implementação desenvolvida seja altamente escalável, o número de peças definidas horizontalmente e verticalmente foram de 3x3 (como mostra a estrutura ordenada da Fig. 2a) para acarretar um tempo de processamento para os testes das heurísticas menor.

O procedimento para jogar o quebra-cabeças após embaralhado é alterar os estados do jogo utilizando movimentos regidos por regras até a solução (posições iniciais das peças, antes de serem embaralhadas) ser alcançada. Para estas regras de movimentos definiu-se que é possível trocar peças adjacentes na horizontal e na vertical, e toda peça que se encontra na borda pode ser trocada pela peça da borda oposta (Fig. 2).

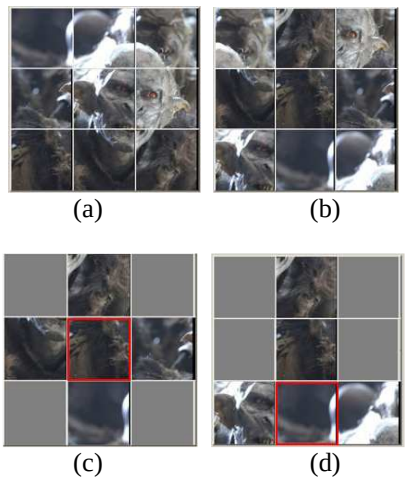

Fig. 2 – Alguns estados do quebra-cabeças: estado inicial(a), imagem embaralhada(b), possíveis movimentos a partir da escolha da peça do centro(c) e possíveis movimentos a partir da peça inferior central (d).

3. Solução Através de Algoritmo Genético

Tomando como base o trabalho de [Hong 2001] é possível aplicar Algoritmos Genéticos para definir os movimentos do jogo de quebra-cabeças. A estrutura de um cromossomos pode ser representada pelo conjunto dos rótulos das associações de uma GST para chegar em determinado nó. Portanto cada gene armazena um movimento possível (definida em uma matriz constante de movimentos). A operação de crossover gera duas novas seqüências de movimentos a partir de partes dos cromossomos. A mutação é usada para trocar elementos (genes) randomicamente de um cromossomo, aumentando a diversificação e tentando sair de ótimos locais. A função de fitness representa o somatório da distância em movimentos de cada peça até sua posição inicial. As variáveis da função (Fig. 3) são: N, a posição atual da peça; R, a posição correta da peça; D uma função para calcular a distância; e, i o índice da peça atual (que pode variar de 1 a 9 peças). Por ser uma função de maximização, quer dizer, o cromossomo que tiver a maior função é melhor, existe a constante 18, pois um cromossomo-solução para o problema deve ter um somatório de 0, o que daria um valor de função de 18.

$$F = 18 - \sum_{i=1}^{i=9} D(R_i, N_i)$$

Fig. 3 – Função de fitness

4. Solução Através de GRASP

A base da solução do problema a partir da heurística GRASP utiliza a GST e a lista de movimentos definidos no AG (seção 3). A estrutura GRASP é composta pela função de avaliação (FA), que é a mesma utilizada pela função de fitness no Algoritmo Genético (Fig. 3). Essa foi definida como sendo também de maximização; a Lista de Candidatos (LC), vetor com todos movimentos possíveis. O tamanho da LC para esse problema será sempre de 18; e a lista restrita de candidatos (LRC), uma lista de E elementos contidas em LC com as melhores funções de avaliação.

As etapas do GRASP modelado a esse problema são (os exemplos são baseados na Fig. 4):

1. Sortear um movimento inicial (entre os 18 movimentos possíveis). Ex.: o M7.
2. Definir a LC. Ex.: sempre serão (para o problema de quebra-cabeças 3x3) os movimentos de M1 até M18.
3. Calcular a FA de cada movimento (nodo) baseado no estado de alteração gerado pelo último movimento. Ex.: M1=0, M2=-1, M3=+1 e M18=-1
4. Definir a LRC. Ex.: Supondo uma lista restrita de 2 elementos, os melhores nodos deste primeiro nível são M1 e M3.
5. Sortear um movimento da LRC. Ex.: M1.
6. Repetir a etapa 2 ate encontrar a FA máxima (neste problema é o valor 18) ou um número de movimentos máximos de 15 (para base de futuras comparações foi definido o mesmo tamanho do AG)
7. Se a solução não for encontrada, repetir a etapa 1 em busca de uma nova outra solução mais R vezes.

8. Ao final, se a solução não for encontrada, comparar a lista de soluções e definir a melhor.

Embora seja altamente recomendado o uso de um método de busca local para obter a garantia de alcançar ótimos locais de boa qualidade [Zanetti e Ochi 2005], foi eliminado esta etapa para poder comparar o verdadeiro motor GRASP com as outras heurísticas.

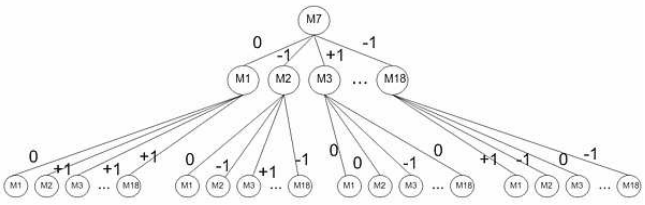

Fig. 4 – Representação da GST do jogo de quebra-cabeças

5. Solução Através de Algoritmo Genético com GRASP

O AG parte de soluções construídas randomicamentes, sem nenhum outro critério de refinamento, já o método de construção do GRASP, embora tenha a preocupação de sempre gerar soluções de boa qualidade, não define uma busca local com garantia de alcançar ótimos locais de boa qualidade.

A modelagem AG-GRASP proposta sugere utilizar o GRASP para a construção de soluções inicias e o AG para a busca local. No Algoritmo Genético proposto em [Hong 2001] 10 soluções iniciais foram geradas randomicamente, em contraposição neste algoritmo é utilizado o GRASP para gerar 10 soluções inicias. Portanto o AG-GRASP aplicado à GST ficaria da seguinte forma:

1. Representação de todas as situações.
2. Definição do tempo limite e do nº de gerações.
3. Definição da profundidade (game tree) e da função de fitness.
4. Definição da taxa de crossover e mutação.
5. Geração da população inicial de cromossomos utilizando GRASP.
6. Execução do crossover.
7. Execução da mutação.
8. Cálculo do valor de fitness de cada offspring.
9. Seleção dos melhores candidates.
10. Finalização ou repetição da etapa 6.

6. Experimentos

A linguagem escolhida para a implementação foi a FreePascal e a ferramenta RAD (Rapid Application Development) Lazarus. Foi desenvolvido um sistema, denominado Kombat Heurístico (Fig. 1, disponível em www.artistasvirtuais.com/jogos), que possibilita o usuário jogar o quebra-cabeças, fazer configurações sobre os parâmetros das heurísticas ou comparar o desempenho das heurísticas. Esta comparação avalia o número de iterações e tempo de processamento entre as aplicações dos algoritmos AG, GRASP e AG-GRASP para a resolução dos movimentos para reorganizar o quebra-cabeça.

6.1 Testes Realizados

Os testes realizados tiveram o objetivo de avaliar o desempenho com relação à performance e a precisão dos algoritmos implementados no software desenvolvido.

Dois parâmetros comuns às três heurísticas que merecem destaques são o número máximo de iterações, que representa, no Algoritmo Genético, o número de gerações e no GRASP o número de soluções geradas; e o número de movimentos máximos para a heurística solucionar o problema, que representa o tamanho da árvore de busca (GST). Quanto menor este parâmetro, mais difícil se torna solucionar o problema. Diminuir este parâmetro equivale a resolver o quebra-cabeças em menos jogadas.

Foram documentados os resultados para 10 jogos variando no número de movimentos máximos (que foram 8, 10, 15 e 25). Quanto menor esse parâmetro maior o nível de dificuldade para a resolução do problema.

6.2 Análise da Performance

A performance foi analisada utilizando os critérios de Percentual de Convergência (Fig. 5), que identifica o total de testes que encontraram um resultado para o determinado jogo, e Tempo para a Convergência (Fig. 6), que é o tempo (em milisegundos) para se alcançar uma solução.

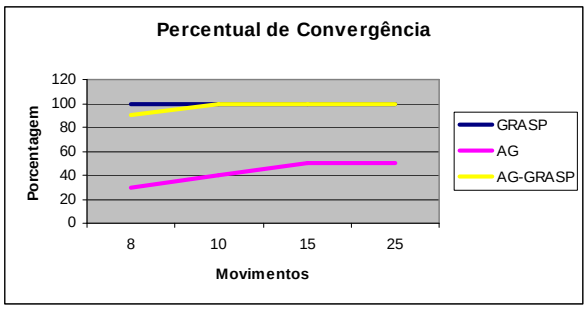

Fig. 5 - Gráfico comparativo entre os percentuais de convergência das três heurísticas

No que diz respeito ao percentual de convergência, observou-se que o motor GRASP puro encontrou uma solução em todos os casos, o AG-GRASP teve dificuldades somente na pior situação (neste caso, resolver através de somente 8 movimentos), e nos quatro grupos de testes, o AG solucionou em média somente menos de 50% dos problemas. Estes resultados são justificados pelo fato de que o AG implementado (e documentado em [Hong 2001]) não é capaz de sair de ótimos locais, repetindo soluções idênticas de cromossomos dentro dos grupos elites gerados. Com relação ao tempo para a convergência, a

performance do AG piora ao se aumentar o número de movimentos máximos, pois como o número de movimentos representa o número de genes de cada cromossomo, as operações de crossover, mutação e seleção do grupo elite gerarão um maior tempo computacional no tempo final. Já o tempo gasto pelo GRASP e pelo AG-GRASP diminuíram até o número de 10 movimentos e mantiveram-se constantes a partir desse momento. Ambos os gráficos demonstram claramente que a performance dos algoritmos AG-GRASP e GRASP puro são superiores ao AG puro.

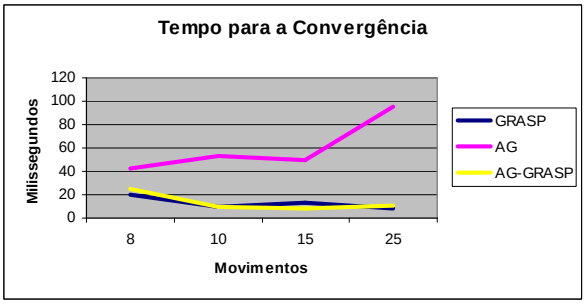

Fig. 6 – Gráfico comparativo entre os tempos de convergência das três heurísticas

6.3 Análise da Precisão

A precisão foi analisada com base no critério de número de iterações para a convergência (Fig. 7), sendo que um valor de iteração igual a 100 (que é o parâmetro de número máximo estipulado) significa que provavelmente o algoritmo não encontrou solução.

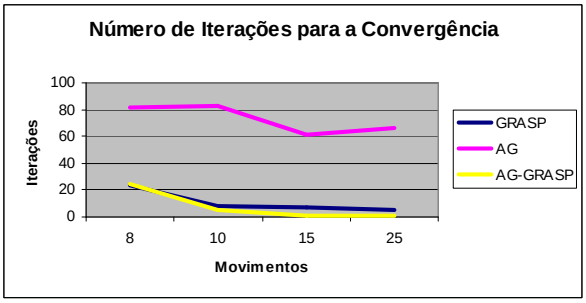

Fig. 7 – Gráfico comparativo entre os números de iterações para a convergência das três heurísticas

O AG mais uma vez demonstrou-se menos eficiente que seus concorrentes. A variação do número de iteração percebido é justificado pelo aumento do tamanho do cromossomo e consequentemente das operações genéticas (descritos na seção anterior). Quanto maior o número de movimentos menor é o número de iterações necessárias tanto para o GRASP quanto para o AG-GRASP, como conseqüência do método de construção aleatório e guloso incluído em ambos os algoritmos, o qual demonstrou-se ainda mais eficiente com a flexibilidade deste aumento da profundidade da árvore.

Neste teste foi observado uma diferença significativa entre os resultados do GRASP puro e do AG-GRASP. A partir do número de 10 movimentos, enquanto o primeiro possui valores altos e baixos de iteração, ocasionando em uma média de 6 iterações, o AG-GRASP tem uma média e uma moda estatística de 1 iteração neste período. Entretanto é importante lembrar que em cada iteração do AG-GRASP é gerado 10 soluções com o GRASP. Em uma análise mais refinada da precisão pode-se concluir que a média do AG-GRASP seria na verdade a constante 10, portanto a heurística mais eficiente neste requisito seria a GRASP.

7. Conclusão

Neste trabalho foram desenvolvidos dois novos algoritmos para busca em árvores de jogos para o domínio de jogos de raciocínio, mais especificadamente quebra-cabeças. O primeiro foi baseado na heurística GRASP pura, apoiado principalmente em suas características randômicas e gulosas de busca. O segundo foi desenvolvido usando o princípio construtivista do GRASP e o Algoritmo Genético como busca local. Uma terceira implementação com Algoritmo Genético puro foi desenvolvida com base em [Hong 2001] para fins de comparação.

Foi criado um software para testar as três heurísticas denominado Kombat Heurístico, que serviu como base para a geração das tabelas comparativas e gráficos, os quais demonstraram que os dois algoritmos propostos têm melhor performance e precisão que o AG puro. Sendo que o algoritmo baseado em GRASP puro teve ligeira vantagem sobre o AG-GRASP, convergindo em 100% dos casos, tendo uma performance média igual e uma precisão de poucas iterações a menos.

Referências

HONG, P. T. et al, 2001. Applying Genetic Algorithms to Game Search Trees. *Soft Computing*.

LACERDA, E. G. M.; CARVALHO, A. C. P. L. F., 2002. Introdução aos Algoritmos Genéticos. *In: Galvão, C.O., Valença, M.J.S. (orgs.) Sistemas inteligentes: aplicações a recursos hídricos e ciências ambientais. UFRGS.*

ZANETTI, MÁRCIA CRISTINA VALLE. OCHI, LUIZ SATORU, 2005. Desenvolvimento e Análise Experimental da Heurística Grasp Aplicada a um Problema de Coleta Seletiva. *XXXVII SBPO, Gramado/RS.*

HU, TE CHIANG; SHING, MAN-TAK, 2002. *Combinatorial Algorithms. Courier Dover.*

WIKIPEDIA, 2007. Definição de game tree. Disponível em: http://en.wikipedia.org/wiki/Game_tree , [Acessado em 08/07].