

Estudo e Implementação de Heurísticas para Otimizar os Algoritmos Aplicados a *Puzzle Games*

Alex F. V. Machado Esteban W. G. Clua

Carla R. B. Bonin* Gustavo M. Novaes* Mauro L. R. de A. Filho*

Universidade Federal Fluminense, Dep. Ciência da Computação, RJ, Brasil

*Centro Federal de Educação Tecnológica, Dep. Informática Industrial, MG, Brasil

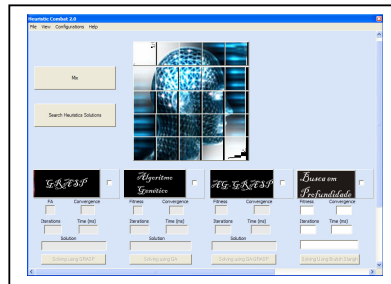


Figura 1 – Interface do *Kombat Heurístico* 2.0.

Programa/jogo de avaliação do desempenho entre heurísticas no domínio de jogos de raciocínio.

Resumo

Este trabalho propõe definir o estado-da-arte no que diz respeito a procedimentos que busquem a solução em jogos eletrônicos de raciocínio (*puzzle games*) com apenas um usuário, nos quais englobam os jogos que usam árvores de busca como modelagem do sistema, tais como os quebra-cabeças e *sudoku*, além de toda gama de jogos que utilizam, mesmo que parcialmente, *path finding*.

Para tanto, é feita uma análise comparativa das heurísticas GRASP, Algoritmo Genético e AG-GRASP, além da técnica tradicional de busca em profundidade, com base em experimentos sobre um jogo de quebra-cabeças de dimensões 5x5.

Palavras-Chave: Otimização de algoritmos, GRASP, algoritmo genético, jogos de raciocínio.

1. Introdução

Este trabalho busca a exploração das heurísticas GRASP (Generic Search Algorithm for the Satisfiability Problem), Algoritmo Genético e AG-GRASP (junção dos conceitos do Algoritmo Genético com o GRASP), em um segmento do domínio de jogos que utiliza predominantemente busca em profundidade e busca exaustiva.

A hipótese tida como base é comprovada através do *software Kombat Heurístico* (Figura 1), que foi desenvolvido durante o projeto. A realização de uma melhor análise da situação/problema existente torna-se possível através de sua interface, permitindo a alteração de parâmetros dos algoritmos para as

análises de convergência e velocidade de resolução através de testes automatizados.

Pretende-se adaptar procedimentos heurísticos a árvore de busca para a otimização de algoritmos aplicados a *puzzle games*. Dessa forma, a implementação de tais heurísticas possibilita a existência de resolvedores completos ou simultâneos (*hints*), que sejam muito mais eficientes do que os registrados na literatura. Outra aplicação indireta a ser observada é o seu uso em jogos que fazem uso da estratégia conhecida como *pathfinding*, que visa traçar a melhor rota para atingir um determinado objetivo [Pozzer 2008].

2. Trabalhos Relacionados

Em [Hong et al. 2001] é feito uma abordagem de algoritmo genético para solucionar os movimentos em um quebra-cabeças que demonstrou que tal algoritmo possui um desempenho melhor que o algoritmo determinístico de busca em profundidade de *Dijkstra*. Neste trabalho foi reconstruído e melhorado o algoritmo desenvolvido em [Hong et al. 2001], e proposta três novas abordagens ao mesmo problema: utilizando o motor GRASP puro, AG com GRASP (AG-GRASP) e busca profundidade para fins de comparação.

3. O Jogo de Quebra-Cabeças Abordado

O jogo usado como estudo de caso é uma versão do tradicional quebra-cabeça de tabuleiro, com pequenas modificações para aumentar a jogabilidade no

computador. Embora a implementação desenvolvida seja altamente escalável, foi criado um ambiente 5x5 composto de 5 peças no eixo vertical e cinco peças no eixo horizontal, totalizando 25 peças (como mostra a estrutura ordenada da Figura 2).



Figura 2 – Estrutura ordenada do quebra-cabeça

O procedimento para jogar o quebra-cabeça proposto é:

- Definir a imagem base do jogo
- Embaralhar as peças para gerar o estado inicial ou o problema
- Alterar os estados do jogo utilizando movimentos regidos por regras até a solução (posições iniciais das peças) ser alcançada

4. Solução Através de Algoritmo Genético

Tomando como base o trabalho de [Hong et al. 2001] é possível aplicar algoritmos genéticos para definir os movimentos do jogo de quebra-cabeças. A estrutura de um cromossomo pode ser representada pelo conjunto dos rótulos das associações de uma *Game Search Tree* para chegar a determinado nodo. Por meio dos cromossomos são então executadas as operações de *crossover* para gerar um grupo de *offspring* foi definida com um ponto fixo para a divisão, e de mutação para trocar elementos (genes) randomicamente de um cromossomo, aumentando a diversificação e tentando sair de ótimos locais. Outra operação realizada é o cálculo da função de *fitness*, que representa o somatório da distância, em movimentos, de cada peça até sua posição inicial.

5. 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. A estrutura GRASP baseia-se em:

- Função de Avaliação (FA): a mesma utilizada pela função de *fitness* no algoritmo genético. Essa foi definida como sendo também de maximização.
- Lista de Candidatos (LC): todos os movimentos possíveis. O tamanho da LC para esse problema será sempre de 50.

- 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:

- Etapa 1 - Sortear um movimento inicial (entre os 50 movimentos possíveis).
- Etapa 2 - Definir a LC.
- Etapa 3 - Calcular a FA de cada movimento (nodo) baseado no estado de alteração gerado pelo último movimento.
- Etapa 4 - Definir a LRC.
- Etapa 5 - Sortear um movimento da LRC.
- Etapa 6 - Repetir a etapa 2 até encontrar a FA máxima ou alcançar o número de movimentos máximos.
- Etapa 7 - Se a solução não for encontrada, repetir a etapa 1 em busca de uma nova outra solução mais R vezes.
- Etapa 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 2005], foi eliminado esta etapa para poder comparar o verdadeiro motor GRASP com as demais heurísticas.

6. Solução Através de Algoritmo Genético com GRASP (AG-GRASP)

O Algoritmo Genético possui um início aleatório no qual os cromossomos são gerados por meio de uma função randômica, sem que haja qualquer critério de refinamento, e possui um método de busca que faz operações buscando otimizar o algoritmo genético, enquanto que o método de construção do GRASP é totalmente elitista tendo a preocupação de sempre gerar soluções de boa qualidade que façam com que o algoritmo inicie de um ótimo local, e não possui um processo de busca associada ao da geração inicial.

Dessa forma, a modelagem AG-GRASP proposta para este projeto tem por objetivo utilizar o GRASP no processo de geração inicial e aplicar o Algoritmo Genético para realizar a busca local. No Algoritmo Genético proposto em [Hong et al. 2001] 10 soluções iniciais foram geradas randomicamente, em contraposição neste algoritmo é utilizado o GRASP para gerar 10 soluções iniciais.

Assim, a modelagem do AG-GRASP aplicado à *Game Search Tree* ficaria da seguinte maneira:

- Etapa 1 - Representação de todas as situações.

- Etapa 2 - Definição do tempo limite e do número de gerações.
- Etapa 3 - Definição da profundidade (game tree) e da função de *fitness*.
- Etapa 4 - Definição da taxa de *crossover* e mutação.
- Etapa 5 - Geração da população inicial de cromossomos utilizando GRASP.
- Etapa 6 - Execução do *crossover*.
- Etapa 7 - Execução da mutação.
- Etapa 8 - Cálculo do valor de *fitness* de cada offspring.
- Etapa 9 - Seleção dos melhores candidatos.
- Etapa 10 - Finalização ou repetição de todo o processo a partir da Etapa 6.

7. Solução Através de Busca em Profundidade (DFS)

A solução através de busca em profundidade foi implementada, baseando-se em explorar todas as possíveis soluções para o jogo e tem como princípio o de explorar cada nodo até que este atinja sua profundidade máxima. Desse modo, o algoritmo desenvolvido consiste em explorar todos os nodos filhos até que atinja um nodo filho que não possua outro nodo filho, também conhecido como nodo folha. Caso não seja encontrada a solução para o problema existente o algoritmo voltará no nodo filho anterior e recomeçará a executar a busca.

8. Implementação e Desenvolvimento do *Software Kombat Heurístico 2.0*

Para a implementação do software foi escolhida a linguagem *FreePascal* na ferramenta RAD (*Rapid Application Development*) *Lazarus*, que foi escolhida devido à sua característica de *cross-compiling*, a facilidade de desenvolvimento, a boa velocidade de resolução, a gratuidade e a disponibilidade da licença de forma a não limitar a expansão dos conceitos a serem observados no projeto [Monteiro et al. 2007] [Piske e Seidel 2007].

Foi então desenvolvido um sistema, o *Kombat Heurístico*, que possibilita o usuário jogar o quebra-cabeça, fazer configurações sobre os parâmetros das heurísticas ou comparar seus desempenhos. O principal objetivo desse programa é comparar o número de iterações, convergência e tempo de processamento entre as aplicações das heurísticas propostas.

8.1 Testes Realizados

Os testes realizados tiveram o objetivo de avaliar o desempenho com relação à convergência e a velocidade de resolução dos algoritmos implementados no *software* desenvolvido. Dois

parâmetros comuns às três heurísticas que merecem destaques são:

- Número máximo de iterações: Representa, no caso do algoritmo genético, o número de gerações e no GRASP o número de soluções geradas.
- Número de movimentos máximos para a heurística solucionar o problema: representa o tamanho da árvore de busca (GST).

Foram documentados os resultados para 20 jogos (ou problemas/situações).

8.2 Análise da Convergência

A convergência foi analisada utilizando o critério de percentual de situações em que foi achada uma solução (Figura 3). Um algoritmo perfeito deveria aumentar seu percentual em proporção ao aumento do número de movimentos possíveis.

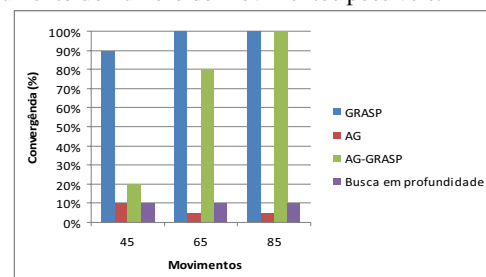


Figura 3 - Gráfico comparativo entre os percentuais de convergência das quatro heurísticas

No que diz respeito ao percentual de convergência, observou-se que a Busca em Profundidade e o Algoritmo Genético solucionaram apenas 10% dos problemas apresentados fazendo uso de 45 movimentos, enquanto que o AG-GRASP solucionou 20% dos casos e o GRASP começou a se destacar solucionando 90% dos problemas gerados. Já com 65 movimentos o motor GRASP puro encontrou uma solução em todos os casos, o AG-GRASP encontrou a solução em 80% dos casos, e o AG solucionou em apenas 5% dos testes realizados, enquanto que a Busca em Profundidade continuou resolvendo o problema em 10% dos casos. Na mais folgada das situações (85 movimentos), o GRASP se manteve resolvendo todos os casos, o AG-GRASP melhorou sua convergência resolvendo também todos os problemas e o AG e a Busca em Profundidade mantiveram seu fraco e insatisfatório desempenho.

8.3 Análise da Velocidade de Resolução

A velocidade foi analisada tendo como base o critério de tempo médio gasto (dos casos que o algoritmo conseguiu convergir) para encontrar a solução para o problema proposto. Onde o algoritmo perfeito deveria diminuir seu tempo em proporção ao aumento de possibilidades de movimento. Na Figura 4 tem-se um gráfico que demonstra o tempo gasto em

segundos para encontrar a solução para os problemas propostos.

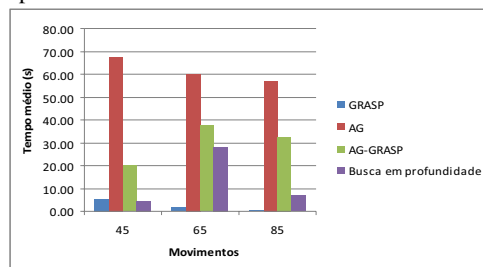


Figura 4 – Gráfico comparativo entre os tempos de convergência das quatro heurísticas

Com relação ao tempo para a convergência, o AG demanda, em alguns casos, mais de um minuto para a resolução dos problemas. A Busca em Profundidade por sua vez gasta aproximadamente 5 segundos para resolver com 45 movimentos, 28 para resolver com 65 e 7 para resolver com 85 movimentos.

Através dos gráficos é possível definir que o GRASP possui também uma maior velocidade pois demanda com 45 movimentos aproximadamente 5 segundos, com 65 movimentos apenas um segundo e em problemas que fazem uso de 85 movimentos para resolução é capaz de resolver em um tempo inferior a um segundo. Já o AG-GRASP, gasta um tempo muito superior devido à suas características herdadas do AG, o que a torna uma heurística imprecisa para os padrões desejados por esta pesquisa.

8.4 Análise do Desempenho

Dessa forma, tem-se que a heurística GRASP foi a que se apresentou melhor adaptada a tal problema proposto, sendo assim, é possível afirmar que esta possui o melhor desempenho aplicada ao domínio de jogos tratado neste estudo. Considerando-se ainda haver uma proporção direta entre o tempo gasto e o número de iterações executadas, deduz-se que o GRASP ainda teria o menor gasto de memória.

9. Conclusão

Neste trabalho foram desenvolvidos dois novos algoritmos para busca em árvores no domínio de jogos, mais especificadamente quebra-cabeças. O primeiro é 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. O AG e a busca em profundidades foram também implementados para dar credibilidade a nossa tese.

Foi criado um *software* para testar as quatro rotinas, denominado *Kombat Heurístico*, o qual serviu como base para a geração das tabelas

comparativas e gráficos, para a avaliação da convergência e velocidade de resolução.

Portanto, após a revisão bibliográfica efetuada, o desenvolvimento do projeto através da implementação do *software Kombat Heurístico*, a realização de testes e a análise dos resultados obtidos foi possível concluir que o GRASP é a heurística que melhor se adaptou ao problema proposto.

O procedimento GRASP sugere duas contribuições de grande relevância para a área de computação em jogos:

- Inovação através de um algoritmo inteligente em uma área que usa predominantemente força bruta.
- Nova oportunidade para a criação de jogos mais confiáveis (por conseguirem analisar a possibilidade de achar a solução), com menos uso de memória e maior velocidade de processamento, portanto tornando possível o desenvolvimento de jogos com melhor utilização do hardware.

Referências

- HONG, P. T. ET AL., 2001. Applying Genetic Algorithms to Game Search Trees. Soft Computing.
- MONTEIRO, FELIPE; BIASA, CESAR; FÉLIX, JOSÉ, 2007. Osciloscópio Digital em Placa ISA. 2007
- PISKE, O. R.; SEIDEL, F. A., 2006. Rapid Application Development, 2007.
- POZZER, C. T., 2008. Teoria dos Grafos. Universidade Federal de Santa Maria, 2008.
- ZANETTI, MÁRCIA CRISTINA VALLE, 2005. Ochi, Luiz Satoru. Desenvolvimento e Análise Experimental da Heurística Grasp Aplicada a um Problema de Coleta Seletiva. XXXVII SBPO 2005, Gramado/RS.