Problema 8-*Puzzle*: Análise da solução usando Backtracking e Algoritmos Genéticos

Nelson Florêncio Junior, Frederico Gadelha Guimarães
PPGCC - Programa de Pós-Graduação em Ciência da Computação
UFOP - Universidade Federal de Ouro Preto
Ouro Preto, Minas Gerais, Brasil
email: sidnelson21@gmail.com, fredericoguimaraes@ufmg.br

***Resumo*—Com jogos de tabuleiro, como o 8-*Puzzle*, onde existem regras bem definidas podemos gerar um espaço de busca bem definido, porém algumas soluções podem ser custosas para o desempenho de um determinado computador. Dado o problema podemos resolvê-lo com uma busca de estados em um grafo, onde cada nó do grafo representa um estado (configuração) do *Puzzle*. Deve-se então encontrar um caminho num grafo que leve ao estado final, no caso um puzzle ordenado. Os Algoritmos Genéticos, pertencentes a Computação Evolutiva, proporcionam uma busca inteligente, evoluindo baseando-se nos resultados obtidos de testes anteriores, diferentimente do algoritmo baseado no paradigma *Backtracking* (Tentativa e Erro), onde testamos todos os caminhos possíveis num grafo até encontrar a solução. Desenvolvendo os algoritmos e comparando seus resultados por meio de uma ánálise de compexidade podemos chegar a uma conclusão de qual paradigma se encaixa melhor ao problema ou em que situação cada um se encaixa melhor.**

***Keywords*-8-Puzzle, Algoritimos Genéticos, *Backtracking* e complexidade.**

I. Introdução

A. 8-Puzzle

O jogo do 8-*Puzzle* é um jogo de tabuleiro de blocos deslizáveis. O problema é bastante abordado na disciplina de Inteligência Artifial, segundo [1], além do seu apelo intelectual inerente, os jogos de tabuleiro tem certas propriedades que os tornaram objetos de estudo ideal para trabalhos inicias.

O objetivo do jogo é mover as peças a partir de um estado inicial até encontrar seu estado final, quando o *Puzzle* está ordenado de forma crescente, como na Figura 1. As regras do jogo são bastante simples, a peça vazia é a única que pode movimentar-se, dependendo da situção pode haver de dois a quatro movimentos possíveis (cima, baixo, direita e esquerda). Estes movimentos geram novos estados até econtrar o estado final. O Puzzle possui um espaço de estados no valor de 9!. Segundo [2] a solução ótima para este problema pertence a classe NP-Completo.

B. Busca em Espaço de Estados

Existem diversas maneiras de solucioná-lo, tais como algoritmos de combinação, busca em largura, profundidade, técnicas de busca direcional, entre diversas outras. Uma forma de solucionar o problema é usando a teoria de busca

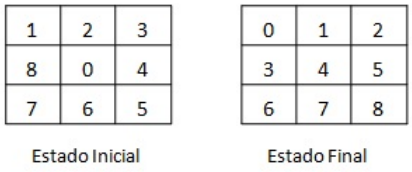

Figura 1. Estados do Puzzle

em espaço de estados, representando o problema em forma de um grafo. No entanto para modelar este problema deve-se levar em consideração quatro características básicas:

- **Estado Inicial:** consiste na representação inicial do probrema. Neste caso pode-se considera um *Puzzle* com as peças todas desordenadas.
- **Estado Final ou objetivo:** refere-se a solução do problema, neste caso quando encontra-se o *Puzzle* totalmente ordenado de forma crescente.
- **Operadores:** são as operações que podem ser realizadas em cada estado, neste caso são os movimentos (cima, baixo direita e esquerda).
- **Custo do Caminho:** corresponde a uma função que atribui um custo para um caminho, neste caso o valor será um para todos os movimentos.

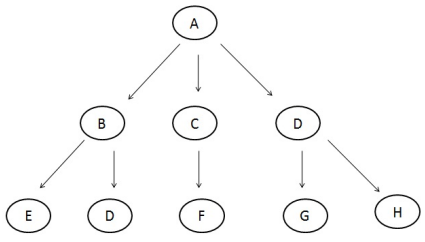

Figura 2. Grafo

Segundo [1], os grafos de espaço de estados para jogos são normalmente grafos radicados, ou seja, possui um nó raiz que alcança quaquer outro nó filho por um caminho no grafo. Para grafos radicados, incluem-se como relação entre

nós as denominações pai, filho e irmãos. Na Figura 2, pode exemplificar esta relação, onde A é o nó pai de (B, C, D), que são irmãos e filhos de A.

C. Backtracking

Também conhecido como Tentativa e erro ou mesmo regressão, caracteriza-se como um Paradigma de Projeto de Algoritmos, que segundo [3] busca decompor o processo em um número finito de sub-tarefas parciais que devem ser exploradas exaustivamente. Pode ser visto como um processo de pesquisa ou de tentativa que gradualmente constrói e percorre uma árvore de subtarefas. A busca em profundidade recursiva apresenta estas características, por isto servirá como parâmetro de comparação com os Algoritmos Genéticos.

É uma solução clássica para problemas de busca, como o proposto neste artigo, o problema é que quanto maior o espaço de busca, menor as chances de se encontrar uma resposta em tempo de execução.

D. Algoritmos Genéticos

Fundamentado principalmente pelo americano John Henry Holland, os Algoritmos Genéticos pertencem a classe dos algoritmos bio-inspirados, ou seja, inspirados na natureza eles imitam o comportamento evolutivo das espécies. O comportamento evolutivo foi definido por Darwin, que afirma que em uma população, os indivíduos mais aptos sobrevivem e passam sua características para seus descendentes.

Segundo [4], eles empregam uma estratégia de busca paralela e estruturada, direcionada à busca de pontos de "alta aptidão", ou seja, pontos nos quais a função a ser minimizada ou maximizada tem valores relativamente baixos ou altos. Algoritmos Genéticos não são buscas aleatórias não-direcionadas, pois exploram informações históricas para encontrar novos pontos de busca onde são esperados melhores desempenhos.

Para caracterizar os Algoritmos Genéticos deve-se levar em consideração:

- **Representação das Soluções de Problema:** Algoritmos Genéticos processam populações de indivíduos ou cromossomos. Cromossomos são estruturas de dados, geralmente vetores ou cadeias de valores binárias que representam uma possível solução do problema. Estes vetores são formados por genes. O conjunto de todas as configurações que o cromossomo pode assumir forma o espaço de busca.
- **Inicialização da População** determina o início do ciclo do algoritmo, geralmente é preenchida de forma aleatória.
- **Avaliação:** a cada cromossomo é atribuído um valor de aptidão definido por uma função. Pode ser visto como uma nota que avalia o quão boa é a solução codificada por um cromossomo.
- **Seleção:** a partir do valor de avaliação de cada cromossomo podemos selecionar quais são mais aptos para continuar na próxima geração. Existem vários métodos de seleção como: o Elitismo, que privilegia os indivíduos mais aptos e o da Roleta, onde os individuos são sorteados para próxima geração, a probabilidade de escolha de um indivíduo é diretamente proporcional a sua aptidão.
- **Operadores Genéticos:** existem dois operadores genéticos. Eles garantem que a nova geração herde características das gerações anteriores. O primeiro é chamado de *crossover* ou cruzamento, ele recombina as características dos pais gerando dois novos filhos, como visto na Figura 3. O outro operador é o de mutação, onde um ou mais genes de um individuo são alterados aleatoriamente, garantindo que o espaço de busca não fixe num máximo ou mínimo local.

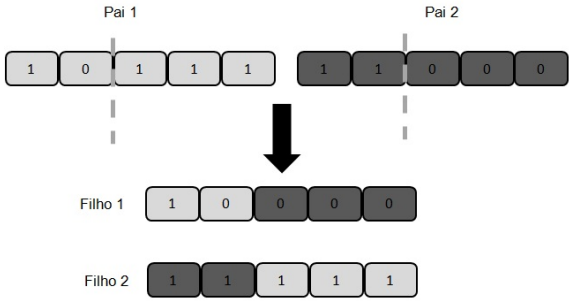

Figura 3. Operador de *Crossover*

II. Justificativa

Os jogos de tabuleiros facilitam o aprendizado em Inteligência Artificial, isso devido a suas regras bem estabelecidas possibilitando melhor entendimendo do espaço de busca se compararmos com problemas mais complexos da vida real. Sendo assim para o início de uma pesquiva em Computação Evolutiva seria interessante desenvolver uma aplicação envolvendo Algoritmos Genéticos e comparar com algum método convencional, no caso o *Backtracking*. Segundo [4], as técnicas convencionais de busca possuem limitação de sua natureza serial. Com a expansão do processamento paralelo, técnicas de otimização paralelas serão largamente beneficiadas, o que é o caso dos Algoritmos Genéticos.

III. Objetivo

O objetivo deste trabalho é desenvolver dois algoritmos que resolvam o problema do 8-*Puzzle*, um baseado no paradgma de *Backtracking* e outro em Computação Evolutiva, especificamente Algoritimos Genéticos. Fazer uma análise de complexidade entre os dois algoritmos e apresentar os resultados.

IV. Trabalhos Relacionados

Nos Estados Unidos, [5] realizou um trabalho parecido onde obteve resultados interessantes. Existem casos em que para facilitar a forma de representação dos cromossomos autores afrouxaram as regras do Puzzle, permitindo que todas as peças possam ser trocadas entre si limitando-se apenas as trocas em vertical, como em [6] e [7], este obteve um grande exito aplicando Algoritmos Genéticos e a metaheurística GRASP nas suas aplicações com *Puzzle*, o que não seria interessante neste caso.

V. Métodos

Ambos os métodos foram implementados na linguagem de programação C++, no ambiente de desenvolvimento Code-Blocks 10.05.

A. Backtracking

Baseado nos conceitos de Tentativa e erro, foi implementado um algoritmo de busca em profundidade recursiva. A busca acontece expandindo o grafo de espaço de estados, como visto na Figura 4, por meio dos possíveis movimentos do estado atual, gerando filhos a serem visitados.

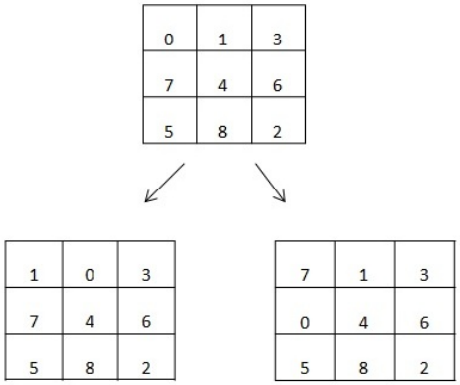

Figura 4. Expansão do Grafo de espaço de estados

Esta busca foi baseada numa estrutura de busca de espaço de estados. Caso encontre o objetivo, a busca é encerrada retornando sucesso, caso encontre um limite, que pode de ser um limite de profundidade do grafo ou um estado já visitado, a busca retrocede para o nó mais recente sobre o caminho, que tenha irmãos ainda não examinados. Veja no algorimo seguinte:

BuscaEmProfundidade($EstadoAtual$)

```
1  if EstadoAtual = Objetivo
2      return SUCESSO
3  Fechado[] := EstadoAtual
4  while Filhos[EstadoAtual] <> 0
5      Filho := ProximoFilho
6      if Filho <> Fechado[]
7          BuscaEmProfundidade(Filho) = SUCESSO
8      return SUCESSO
9  Fim-While
10 return FALHA
11 Fim-função
```

Nota-se que há um vetor de estados denominado Fechado, este vetor é uma variavel global, ela armazena todos os estados visitados no espaço de busca. Na linha 6, depois de gerados os filhos, de acordo com os possíveis movimentos do estado do *Puzzle*, verifica-se o filho já havia sido visitado, caso não o algoritmo chama a função de busca novamente passando o filho encontrado como estado atual, caso sim ele retorna falha e volta para nó anterior e verifica se há irmãos a serem visitados.

B. Algoritmos Genéticos

Os Agoritmos Genéticos podem apresentar soluções próximas as soluções ótimas, segundo [4], uma de suas vantagens é que as técnicas de busca convencionais iniciam seu processamento em um único candidato, por outro lado as técnicas de Computação Evolutiva operam sobre uma população de candidatos em paralelo, assim elas podem fazer a busca em diferentes áreas do espaço de solução, alocando um número de membros apropriado para a busca em várias regiões. O algoritmo é expresso da seguinte forma, onde t representa cada geração:

AlgoritmoGenetico()

```
1  t = 0
2  criar população P(t);
3  for cada indiviudo i de P(t)
4      avaliar aptidao individuo(i)
5  Fim-para
6  while condição de parada não satisfeita
7      t := t + 1;
8      Selecionar população P(t) de P(t − 1);
9      Aplicar operadores de cruzamento sobre P(t);
10     Aplicar operadores de mutação sobre P(t);
11     Avaliar P(t);
12 Fim-enquanto
```

Inicialmente é gerada uma população formada por um conjunto aleatório de indivíduos que podem ser vistos como possível solução do problema. A Figura 5 demonstra a população inicial gerada com genes formados com valores variando de 0 a 3, (0 - cima, 1 - baixo, 2 - esquerda e 3 - direita), referentes aos movimentos da posição vazia do

Puzzle. Para não sortear movimentos inválidos foi criada uma função que verifica a posição da peça vazia e retorna os movimentos válidos, depois é sorteado um valor dentre estes movimentos.

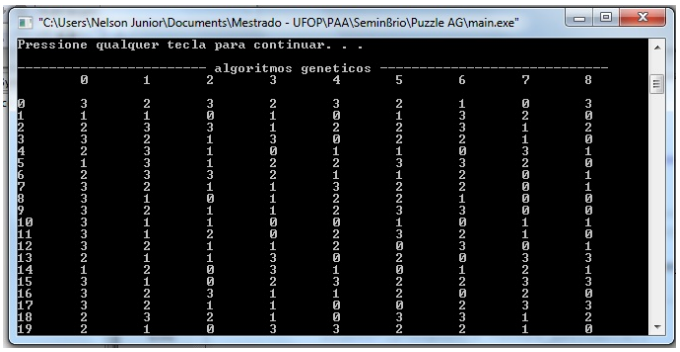

Figura 5. População Inicial

Para avaliar cada indivíduo foi criada uma função baseada em três heurísticas:

- **peças fora do lugar:** soma do número de peças fora do luga em cada estado.
- **distância de Manhatan:** soma das distâncias de cada peça a sua posição original
- **número de inversões:** a inversão acontece quando uma peça tem como antecessor uma peça com valor inferior ao dela. Note que quando este caso acontece com as peças de valor 0 e 1, esta heurística não influência tanto, porém quando envolvem outras peças aumenta significativavente o número de movimentos para resolução do Puzzle.

A função foi desenvolvida baseada na idéia de [5] e [1]. Ela é expressa da seguinte forma:

$$f(n) = 36 * NumPecasTrocadas + 18 * DistaciaManhattan + 2 * NumInversoes$$

Para o processo de seleção de indivíduos para a próxima geração foi adotado o método do Elitismo, que mantém uma porcentagem dos indivíduos mais aptos, neste caso foi adotada uma porcentagem de 10% da população.

Alguns membros mantidos pela seleção sofreram alterações gerando novos indivíduos. Estas modificações foram devido aos operadores genéticos de *crossover* e mutação. O operador de *crossover* foi aplicado com uma taxa de 30% e o de mutação foi aplicado uma taxa de 60%. Geralmente o operador de *crossover* predomina, porém neste caso devido a possibilidade de um gene não ser válido em determinado estado ele perde um pouco da qualidade. Exemplo se um gene possui o valor 0 referente ao movimento para cima e a posição vazia estiver na primeira linha do *Puzzle*, este gene será inválido. Isto não acontece com o operador de mutação, pois os valores sorteados sempre serão válidos. É importante ressaltar que se a taxa de mutação for muito alta a busca se torna essencialmente aleatória.

A cada geração a aptidão dos cromossomos é minimizada, consequentemente o número de movimentos para resolver o Puzzle também diminue. A condição de parada adotada foi o número de gerações, pois o objetivo em questão é achar a melhor solução possível.

VI. Análise de Complexidade

A. Backtracking

Para análise de complexidade do algoritmo de *Backtracking* baseado em busca em profundidade recursiva, devemos levar em consideração dois termos: o número em média de filhos gerados por cada nó (B) e a profundidade máxima do grafo de espaço de estados (m). Por não precisar manter todos os nós do grafo na memória, esta busca possui uma boa complexidade de espaço. Como exemplificado na Figura 6, os nós brancos ainda não foram visitados, portanto não gastam espaço na memória. Logo a função de complexidade de espaço é linear e pode ser definada como: $f(n) = O(B*m)$. No 8-Puzzle, a profundidade do grafo de espaço de estados se refere ao número de movimentos.

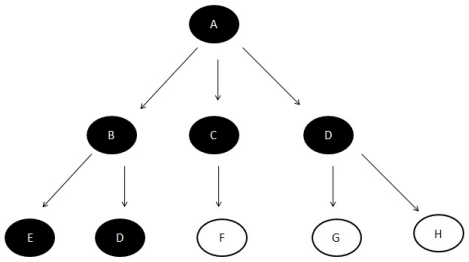

Figura 6. Busca em Profundidade

Já a complexidade de tempo demonstra a principal falha deste método, pois sua complexidade é exponecial, representada por: $f(n) = O(B^m)$, ou seja, além de não garantir uma solução ótima, dependendo da instância o *Backtracking* levará um enorme tempo para encontrar uma solução. Para exemplificar, levando em conta o tempo em minutos, com um espaço de busca de profundidade 100, este algoritmo gastaria 2^{100} minutos para resolver o problema no pior caso.

B. Algoritmos Genéticos

Segundo [8], a complexidade de um Algoritmo Genético não depende somente do tamanho de entrada n de um problema, mas também de seus parâmetros como número de gerações, número de indivíduos e a taxa dos operadores genéticos. Numa forma abstrata podemos considerar uma função geral da complexidade como:

$$f(n) = O(Comp_{Inic} + [numGer * (Comp_{Reprod} + CompTerm)]) + Comp_{Recupera})$$

Onde $numGer$ refere-se ao número de gerações. A complexidade de $Comp_{Inic}$ refere-se a complexidada do procedimento da população inicial, $Comp_{Term}$ é referente a complexidade de calcular se a condição de parada foi atingida, $Comp_{Recupera}$ é a função que retorna a solução. Estas são complexidades simples que dependem da implementação. Já a $Comp_{Reprod}$ é totalmente relacionada com o número de cromossomos e é definida por:

$Comp_Reprod = L * \sum(1/k * (Comp_{Escolher Individuo} + Comp_{mod})$

Onde L é o número de vezes que a geração é executada, k é o número de operadores genéticos existentes e/ou utilizados no problema. $Comp_{Escolher Individuo}$ é a complexidade de escolher um indivíduo e $Comp_{mod}$ é a complexidade dos procedimentos de mutação e/ou *crossover*.

No algoritmo desenvolvido neste trabalho, a função de complexidade possui algumas particularidades devido a estrutura dos códigos, por exemplo , em uma única função foram feitas as operações de seleção, mutação e *crossover*. Como foi utilizado o algoritimo de Inserção para ordenar os indivíduos, esta função possui complexidade $O(n^2)$, onde n é o número de indivíduos. Como esta complexidade domina as outras assintoticamente, pode-se considerar a complexidade de tempo e espaço deste algoritmo como: $f = num_{ger} * O(n^2)$, onde n é o número de indivíduos.

VII. Experimentos

Como os Algoritmos Genéticos são algoritmos estocásticos, ou seja, são processos não determinísticos, com origem em eventos aleatórios, por isso cada instância foi testada 10 vezes tirando-se uma média das soluções encontradas. Para realização dos experimentos foram testadas cinco instâncias, todas elas com 4000 gerações e 1000 indivíduos ou cromossomos.

A Tabela I demonstra como os resultados encontrados usando os Algoritmos Genéticos são eficientes, comparando-os com *Backtracking* até mesmo a média dos resultados alcançados mantém uma grande diferença entre os métodos.

Tabela I
Análise dos métodos

	Número de Movimentos		
Instância	*Backtracking*	Média(AG)	Melhor Resultado(AG)
1	94	46,7	37
2	137	45,8	38
3	84	35	25
4	300	41	33
5	316	31,1	21

O gráfico da Figura 7, demonstra a variação dos resultados encontrados durante os testes dos Algoritmos Genéticos. Note que o pior resultado encontrado dentre as instâncias aconteceu no segundo teste da instância 2, mesmo assim ainda é menor que a metade do valor encontrado pelo método *Backtracking*. Em todos os testes o algoritmo encontrou uma solução, em alguns casos como na segunda e na quinta instância há uma grande possibilidade de ser a melhor solução, devido ao pequeno número de movimentos.

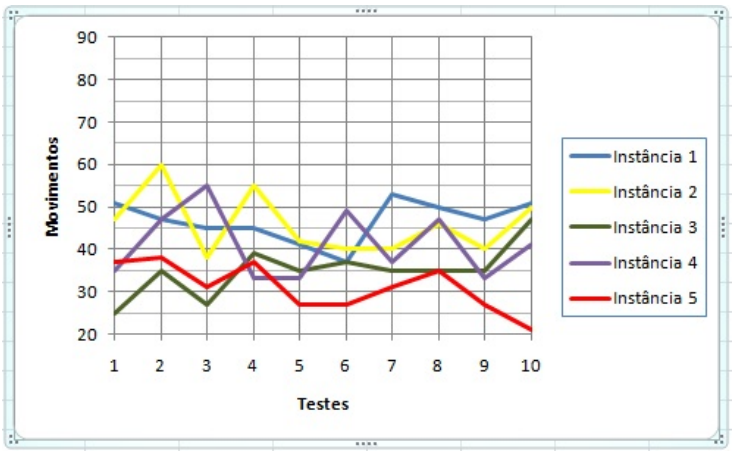

Figura 7. Testes com os Algoritmos Genéticos

O comportamento do Algoritmo Genético é expresso pela Figura 8. Note que os movimentos são minimizados a medida que as gerações aumentam. Este fato acontece devido a aptidão dos cromossomos também diminuirem, demonstrando que a função de aptidão avalia de forma correta cada cromossomo. Assim o comportamento evolutivo do algoritmo é realizado encontrando o menor número de movimentos possível.

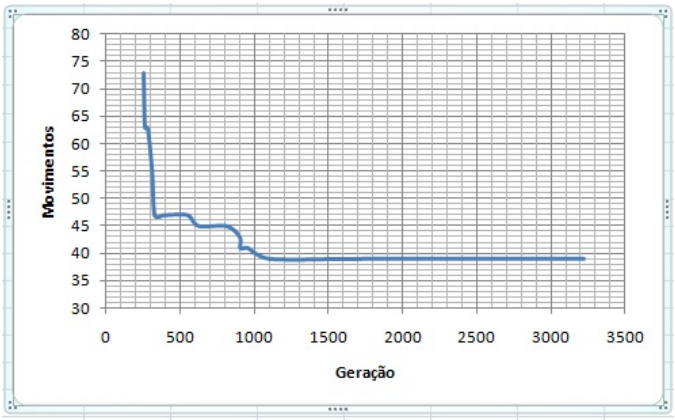

Figura 8. Gráfico: Movimentos sendo minimizados

VIII. Conclusões

Os resultados comprovam que comparado com o *Backtracking*, os Algoritmos Genéticos possibilitam encontrar soluções muito mais interessantes. Os operadores genéticos garantem que a busca não se torne puramente aleatória aproveitando os melhores indivíduos de gerações anteriores.

Os algoritmos Genéticos posuem uma enorme vantagem se comparado a outros métodos, pois realizam uma busca

de otimização global, ou seja, possui uma busca paralela e estruturada que visa uma população não um único indivíduo.

Em trabalhos futuros seria interessante melhorar a representação dos cromosomos de modo que permita um melhor aproveitamento do operador de *crossover*, possibilitando encontrar soluções com um menor número de gerações, consequentemente diminuindo o tempo de execução.

REFERÊNCIAS

[1] G. F. Luger, *Inteligência Artificial: estruturas e estratégias para a solução de problemas complexos*, 4th ed. Bookman, 2004, 85-363-0396-4.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Prentice-Hall, Englewood Cliffs, NJ, 2003.

[3] N. Ziviani, *Projeto de Algoritmos com Implementações em Java e C++*. Cengage Learning, 2006, 10: 8522105251.

[4] S. O. Rezende, *Sistemas Inteligentes: Fundamentos e Aplicações*, 1st ed. RECOPE-IA Rede Cooperativa de Pesquisa em Inteligência Artifial, 2005, 9788520416839.

[5] T. Qian, "Using genetic algorithm to solve sliding tile puzzles," 2007, http://www.cs.oswego.edu/~qian/csc466/Ting%20-%20GA.pdf visitado em 01/06/2011.

[6] P. T. Hong and E. AL, "Applying genetic algorithms to game search trees," *Soft Computing*, 2001.

[7] A. F. V. Machado, E. W. G. Clua, C. R. B. Bonin, G. M. Novaes, and M. L. R. A. Filho, "Estudo e implementação de heurísticas para otimizar os algoritmos aplicados a puzzle games," Centro Federal de Educação Tecnológica, Dep. Informática Industrial, MG, Brasil, Technical Report, 2008. [Online]. Available: http://www.inf.pucminas.br/sbgames08/EBooks/Proceedings-SBGames-Posters-2008-Final-EB.pdf

[8] M. S. Neubert, "Análise formal da complexidade de algoritmos genéticos," Master's thesis, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Abril 1998.