Relatório do Trabalho para Resolução de Sudoku com Hill Climbing e Algoritmo Genético

Alunos
Daniel Amaral
Deoval Junior
Josué Pereira

Introdução

O objetivo deste trabalho é resolver o quebra-cabeça Sudoku utilizando os algoritmos de Hill Climbing e Genético. Sudoku é um quebra-cabeça baseado na colocação lógica de números. O objetivo do jogo é a colocação de números de 1 a 9 em cada uma das células vazias numa grade de 9x9, constituída por 3x3 subgrades chamadas regiões. O quebra-cabeça contém algumas pistas iniciais, que são números inseridos em algumas células, de maneira a permitir uma indução ou dedução dos números em células que estejam vazias. Cada coluna, linha e região só pode ter um número de cada um dos 1 a 9. Resolver o problema requer apenas raciocínio lógico e algum tempo. Os problemas são normalmente classificados em relação à sua realização. O aspecto do sudoku lembra outros quebra-cabeças de jornal. Foi criado por Howard Garns, um projetista e arquiteto de 74 anos aposentado.

Utilizamos o tabuleiro a seguir rodar os testes.

			7					
1								
			4	3		2		
								6
			5		9			
						4	1	8
				8	1			
		2					5	
	4					3		

Nota: *Todos os testes foram realizados em um processador Intel Core i3 6100 @ 3,70 GHz.*

Algoritmo Hill Climbing

Para resolver o Sudoku utilizando o Hill Climbing precisamos primeiro entender um pouco do funcionamento deste algoritmo. O Hill Climbing é um algoritmo de otimização que busca uma solução local para o problema. É um algoritmo iterativo que começa com uma solução arbitrária para um problema e tenta encontrar uma solução melhor ao alterar gradualmente um único elemento da solução. Se a mudança produz uma solução melhor, uma mudança incremental é feita para a nova solução, repetindo até que nenhuma melhoria adicional possa ser encontrada. Analisando o funcionamento do Hill Climbing podemos ver que ele necessita de 3 condições para funcionar: solução arbitrária inicial, uma forma de alterar gradualmente um elemento da solução e o critério de parada. Além, claro, da função objetivo que queremos maximizar.

Para a resolução do Sudoku utilizamos como:

Solução arbitrária inicial (candidato inicial) - Pegamos cada coluna da grade e distribuímos os números de 1 a 9 na mesma, tomando cuidado com os número que já estão fixados no tabuleiro. Ou seja, temos um vetor com números de 1 a 9, olhamos na coluna em questão quais os números que estão fixados e retiramos do vetor. Agora pegamos cada número desse vetor resultante e adicionamos em uma posição vazia da coluna em questão da grade, que é representada por uma matriz onde os elementos “vazios” do problema inicial estão representados por ‘.’.

Forma de alterar um elemento da solução (criar novos candidatos) - Utilizamos uma função em que pegamos os valores de duas linhas diferentes, sendo que nenhuma das duas pode conter uma valor fixado pela grade inicial, e trocamos os valores de linha. Para escolher as linhas para trocar utilizamos um loop que olhando para uma coluna da grade pegamos o elemento de uma linha e trocamos com todos os elementos das linhas sucessores, ou seja, para a primeira linha iremos trocar este elementos com os elementos da linhas de 2 à 9, para a linha 2 iremos trocar este elemento com os elementos das linhas de 3 à 9. Gerando assim n candidatos a solução, sendo cada candidato representado por uma troca entre a linha inicial e uma única linha sucessora. Fazemos isso para todas as linhas de uma coluna e para todas as colunas.

Critério de parada - Utilizamos como critério de parada um número n de iterações, sendo este número n um número passado como entrada na linha de comando.

Função objetivo - A função objetivo é a parte mais importante do algoritmo, queremos maximizar a mesma. Para isto utilizamos uma função que se baseia em contar quantos números estão repetidos nas linhas e nas subgrades. Quanto menos números repetidos nas linhas e nas subgrades maior será o valor da função objetivo. Não avaliamos as colunas em busca de números repetidos pois na formulação de nossa solução garantimos que não haverá repetição de números nas colunas, pois iniciamos as mesmas com números de 1 a 9 sem repetições e depois fazemos trocas entre os valores de duas linhas. Logo uma linha

L1 que possui valor V1 terá o valor V2 de uma linha L2 após as trocas, esse valor V2 é único naquela coluna, assim como L2 terá o valor V1 que é único naquela coluna.

Resultados e Observações

Para cada teste realizado foi gerado um print do console que exibe o valor do candidato na função objetivo e a tabela preenchida com o candidato.

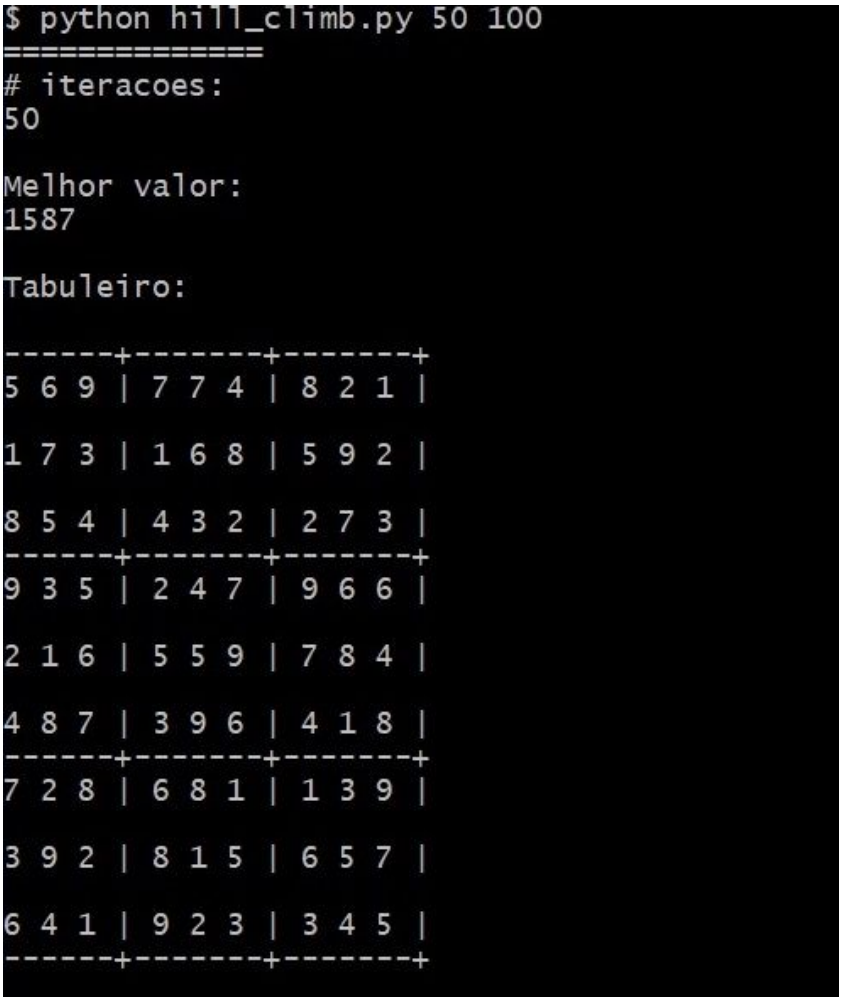
Comparação do número de iterações

# iterações	**Valor do candidato na função objetivo**
5	**1136**
10	1345
15	1488
20	1565
50	**1587**
50000	1587
200000	1587
500000	1565
1000000	1587
5000000	

Esses testes usaram o limite de repetição de valores em 100.

Nesse experimento vemos uma rápida convergência para um padrão cujo o valor da função objetivo é de no máximo 1587, que praticamente se manteve nos experimentos de 50 à 1000000 de iterações definida na entrada. O que é estranho pois o tabuleiro resultante com esse valor contém uma quantidade considerável de repetições em linhas e subgrades.

Tabuleiro resultante da melhor execução:



50 iterações com valor resultante 1587

Algoritmo Genético

Algoritmos Genéticos (AG) são implementados como uma simulação de computador em que uma população de representações abstratas de solução é selecionada em busca de soluções melhores. A evolução geralmente se inicia a partir de um conjunto de soluções criado aleatoriamente e é realizada por meio de gerações. A cada geração, a adaptação de cada solução na população é avaliada, alguns indivíduos são selecionados para a próxima geração, e recombinados ou mutados para formar uma nova população. A nova população então é utilizada como entrada para a próxima iteração do algoritmo. Para o AG temos algumas condições importantes também, como função de avaliação, indivíduo, seleção e reprodução.

Para a resolução do Sudoku utilizamos como:

Representação do indivíduo - É representado por um vetor de 81 posições, onde cada posição representa uma casa na grade do Sudoku. E pode receber valores de 1 a 9.

Função de avaliação - Utilizamos basicamente a mesma função que usamos no Hill Climbing, entretanto agora o objetivo não é maximizar a mesma. Queremos agora dado um indivíduo responder qual a avaliação do mesmo diante do problema, ou seja, o quão bom ele é para a resolução do problema. Iremos contar quantos números estão repetidos nas linhas e nas subgrades. Quanto menos números repetidos nas linhas e nas subgrades melhor será a avaliação do indivíduo. A diferença principal entre a função utilizada no Hill Climbing e esta é que precisamos agora avaliar também as colunas, pois não há garantia que após um cruzamento de indivíduos as colunas não terão valores repetidos.

Resultados e Observações

Nota: *Todas as comparações levarão em conta o melhor indivíduo da última geração do teste realizado. Para cada teste realizado foi gerado um arquivo contendo o melhor indivíduo de cada geração, e gerado um print do console que exibe a tabela preenchida com o indivíduo.*

Comparação de quantidade de indivíduos da população x quantidade de gerações

# indivíduos \ # gerações	1000 gerações	10000 gerações
100 indivíduos	**1638** *(1 min)*	1792 *(12 mins)*
1000 indivíduos	**2089** *(20 mins)*	**2177** *(>120 mins)* *contagem de tempo não havia sido implementada ainda*

Esses testes usaram probabilidades de 80% para crossover e de 1% para mutação.

Pudemos observar com esse experimento, que aumentar a quantidade de gerações não aumenta tanto o ganho quanto aumentar a quantidade de indivíduos. Porém aumenta consideravelmente o tempo.

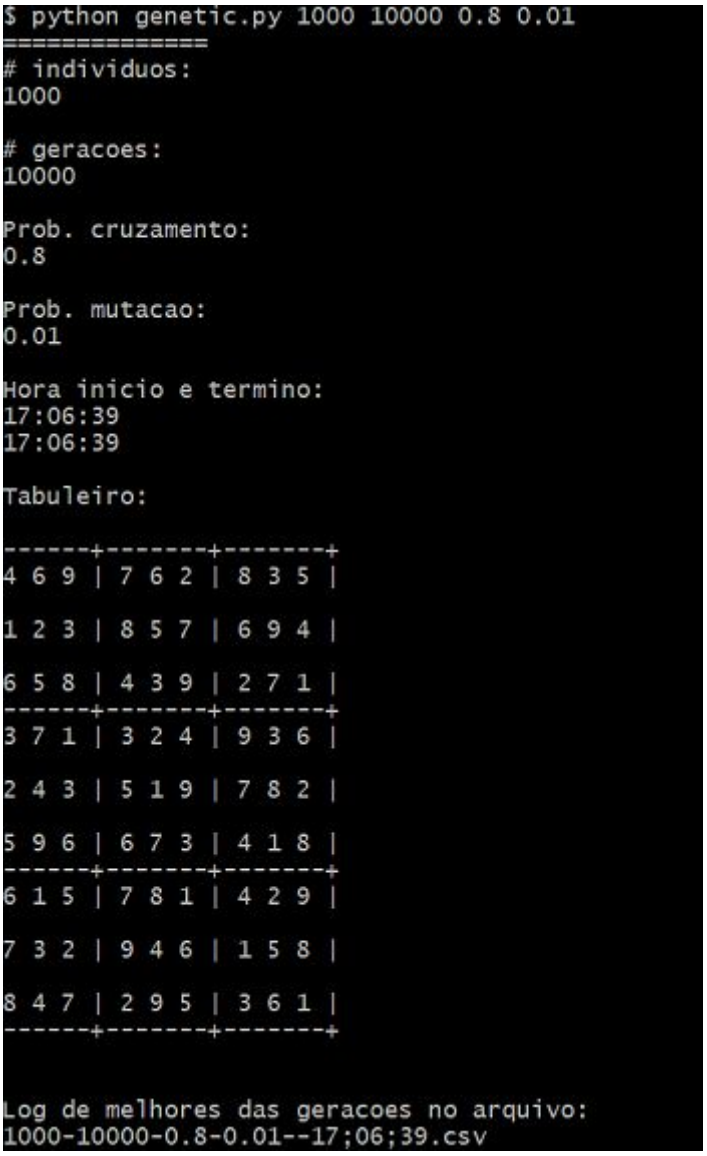
Comparação de probabilidades diferentes de Crossover e Mutação

P(Mutação) \ P(Crossover)	70%	80%	90%
1%	2023	**2089**	2056
5%	2089	2111	**2122**
10%	2111	**1990**	2034

Esses testes usaram 1000 indivíduos e 1000 gerações com tempo médio de 23 min de execução.

Pudemos observar com esse experimento, que aumentar um pouco a probabilidade de mutação causa mais impacto positivo, e que se aumentar muito acaba causando uma queda no resultado junto com um aumento no crossover.

Tabuleiro resultante da melhor execução:



1000 indivíduos, 1000 gerações, 80% prob. cruzamento e 1% prob. mutação

Conclusão

Após rodarmos alguns testes pudemos observar que o Algoritmo Genético resultou em soluções melhores que o Hill Climbing. O Hill Climbing demonstrou um comportamento um pouco estranho independente da quantidade de iterações que o mesmo fez, aumentamos o número de iterações e mesmo assim o valor da função retornado se manteve o mesmo. Achamos que esse é o valor do máximo local para a grade inicial que utilizamos, pois o tabuleiro resultante parece bem errado. Utilizamos um tabuleiro de entrada diferente para o Hill Climbing e mesmo assim o valor retornado foi o mesmo, algo que não sabemos explicar, mas talvez o problema esteja na modelagem dos próximos candidatos. No AG tudo funcionou melhor e não tivemos tantos problemas na modelagem dos próximos indivíduos e obtemos resultados melhores, mas mesmo assim não conseguimos solucionar o problema.