Tapete Musical de Sachs

Thiago Amendola

Ciência da Computação - Graduação
E-mail: t148062@students.ic.unicamp.br
tico.amendola1@gmail.com

Resumo – Neste trabalho, foi proposto um algoritmo genético para encontrar uma permutação que maximize o encaixe de pecas em um protótipo de tapete musical. O tapete em questão é composto por peças hexagonais cujos encaixes de cada face são representados por feixes de bits pareados de valores equivalentes. Além disso, as peças podem ser customizadas, de modo a rotacionar, inverter bits ou espelhar seu respectivo feixede bits. O algoritmo proposto leva em conta a disposição das peças e suas respectivas configurações para gerar uma solução que maximize a quantidade de bits de encaixe por meio da criação de gerações de soluções e do uso de crossovers e mutações.

Palavras-chave: algoritmos evolutivos, algoritmos genéticos

1. Introdução

Um músico propôs a elaboração de um tapete musical de propósito lúdico e educativo para ser usado por crianças. Este tapete seria composto por diversas peças hexagonais configuráveis, onde cada uma emite um som diferente ao serem pisadas. Essas peças se encaixariam como um quebra-cabeça, de maneira similar aos pedaços de um tatame, de modo a tornarem sua montagem e configurações de construção altamente customizáveis e escaláveis a cada tipo de situação. O intuito principal deste aparato é de usá-lo para o estudo de rítmica para crianças, desenvolvendo noções musicais e lógicas através da interação com o tapete e a reconstrução e reposicionamento do mesmo.

Do ponto de vista tecnológico, as peças do tapete seriam compostas primariamente por um feixe de 32 bits que se estenderia pelas bordas da de seu formato hexagonal. Estes bits se acomodam a cada face do hexágono assim como mostrado na **Figura 1.** Alguns bits do feixe são atribuídos aos vértices do hexágono e contam para ambas as faces que geram o respectivo vértice. Cada face conta com 6 bits no total, exceto as faces superior e inferior, que contam com 7 bits ao todo.

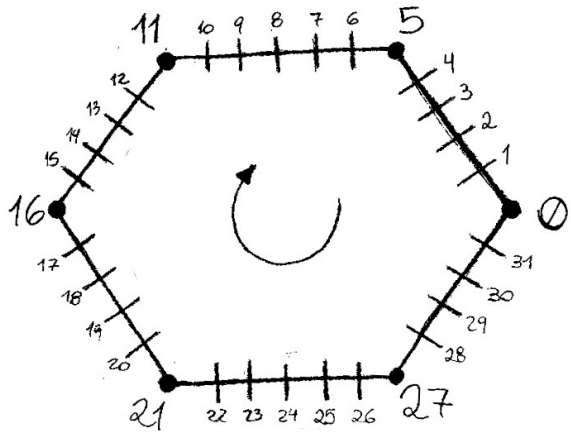

Figura 1*: Representação da peça hexagonal. Os números presentes nas bordas indicam a distribuição de bits em cada face. Os bits de vértice contam para ambos os lados que o geram.*

Os bits presentes em uma face definem o encaixe da mesma. Isso significa que, para encaixar uma peça em cima de outra, a face de baixo da peça de cima deve conter os mesmos valores de bits que os da face de cima da peça de baixo. O mesmo se aplica as outras faces. Além disso, as peças possuem alguns parâmetros de configuração individual: a rotação de bits, a inversão dos bits e o espelhamento do feixe.

Com o intuito de encontrar uma configuração de tapete cuja quantidade de encaixes, ou seja, de bits iguais entre faces adjacentes seja máxima, foi desenvolvido um algoritmo genético capaz de encontrar soluções boas e/ou ótimas. Consideraremos para a solução final um tapete de 32 peças por 16 peças, totalizando 512 peças. Além disso, consideraremos que o tapete respeita uma topologia toroidal, isto é, a peça do topo do tapete deve se encaixar com a respectiva peça do fundo do tapete, sendo a mesma regra aplicada às laterais.

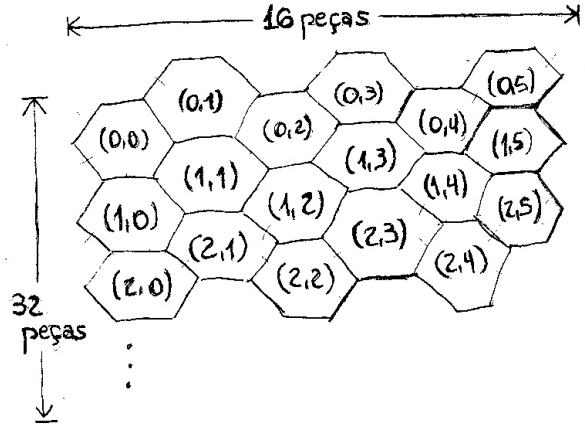

***Figura 2**: Representação da matriz de peças*

2. Trabalho proposto

A representação do tapete foi feita por meio de uma matriz de elementos hexagonais (**Figura 2**). Para detectar as peças vizinhas a uma determinada peça, é avaliada a divisibilidade do valor da coluna por 2, visto que peças de coluna ímpar são mais elevadas e se relacionam de maneira diferente com seus vizinhos. Além disso, o tapete respeita uma topologia toroidal, o que significa que as peças de uma determinada borda se encaixam com as da borda diametralmente oposta. Isso não só garante que todas as faces do tapete sejam consideradas no cálculo do encaixe de bits, como permite a representação de soluções iguais dispostas em permutações diferentes.

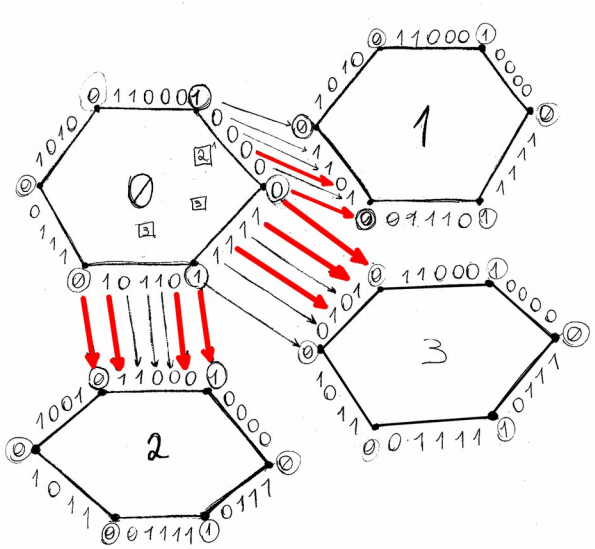

***Figura 3**: Exemplo de checagem de pareamento para os lados superior-direito, inferior-direito e inferior da peça 1. As setas vermelhas representam os bits pareados de valores iguais*

Cada peça ainda contém três parâmetros individuais capazes de customizá-la: a rotação de bits, a inversão dos bits e o espelhamento da peça. O primeiro parâmetro indica em quantos bits o feixe foi rotacionado, considerando esta rotação no sentido horário. O segundo parâmetro é a inversão de bits, isto é, caso seja ativado, todos os valores de bits iguais a 0 se tornam 1 e vice versa. O último parâmetro representa se a peça está espelhada ou não. Praticamente, este valor não é customizável por meio de componentes

nas peças e é considerado apenas para o caso de uma peça ser virada do avesso, espelhando seu feixe.

Será utilizado para o algoritmo genético desenvolvido uma função de fitness equivalente a quantidade de bits de encaixe equivalentes, ou seja, a quantidade de bits pareados de valor equivalente em cada um dos encaixes entre peças. É possível encontrar o valor de fitness checando, para cada peça, três de suas faces não opostas uniformemente por todo o tapete (**Figura 3**). Pela propriedade da topologia toroidal, uma peça cuja face testada esteja na borda do tapete terá a respectiva face pareada com a correspondente face da peça da outra borda do tapete. Foram consideradas nesta solução as faces superior-direita, inferior-direita e inferior. Para a peça 1 representada no exemplo da **Figura 3**, somando todos os bits pareados obtemos um valor de fitness igual a 9. Ao repetir o processo para todas as peças, a soma obtida equivale ao valor de fitness daquela resposta.

3. Métodos utilizados

Para a resolução deste problema, foi desenvolvido um algoritmo genético que capta um conjunto inicial de peças disponíveis. Para a execução do algoritmo deste trabalho, foi utilizada uma entrada com 2048 peças, contendo 512 peças distintas.

Em seguida o algoritmo gera uma população inicial de soluções aleatórias baseadas na permutação destas peças na matriz correspondente ao tapete e nas diferentes configurações de parâmetros de cada peça.

Construída a população inicial, são obtidos os valores de fitness de cada uma das soluções. Elas são então ordenadas pelo valor da fitness e são selecionados pares para a realização de um crossover, gerando um par de novas soluções. As novas soluções são então submetidas a um processo de mutação baseado em um fator previamento fornecido, que pode alterar as configurações de algumas das peças. O processo de encontrar a fitness, gerar novas soluções e mutá-las e repetido por um número finito de gerações.

A linguagem utilizada foi C, visto que é uma linguagem versátil e que dispõe de um bom desempenho, especialmente para o caso de estudo.

4. Resultados obtidos

Por conta de limitações no desenvolvimento, apenas a mutação foi implementada. Os resultados foram obtidos com um conjunto experimental de 4 peças, considerando uma população inicial de 6 soluções, onde 3 novos indivíduos são criados por geração baseados nas três melhores soluções. Elas então sofrem um processo de mutação e ocupam o espaço das três piores soluções da atual geração. Após 100 gerações e algumas execuções, notou-se que a melhor solução varia entre 48 e 56, dependendo da execução. Levando em conta que o limite superior é igual ao acerto total para uma peça (19 bits) multiplicado às proporções do tapete (2x2), o limite superior para o valor de fitness deste tamanho de tapete equivale a 76. Não foi possível concluir se a solução não foi próxima por problemas no algoritmo ou porque a entrada fornecida é incapaz de alcançar dado valor.

5. Conclusão

Foram encontradas dificuldades na implementação do crossover e mutação, visto que o crossover não pode conter uma solução com elementos repetidos. Futuros experimentos serão feitos utilizando Order-

Based Crossover (OBX), Position-Based Crossover (PBX) e Partiallyt Matched Crossover (PMX).

Referências

[1] Columbini, Esther. *Introdução a Inteligência Artificial – Computação Evolutiva parte I.* Acessado em 29/04/16: *http://www.ic.unicamp.br/~esther/teaching/2016s1/mc906/Aula9_s.pdf*

[2] Levine, Lionel. *De Bruijn Sequences*. Acessado em 14/05/16 em: http://www.math.cornell.edu/~levine/18.312/alg-comb-lecture-21.pdf

[3] Lacerda, Estéfane G. M. de. *O problema do Caixeiro Viajante.* Acessado em 14/05/16 em: http://www.dca.ufrn.br/~estefane/metaheuristicas/ag_pcv.pdf