Desenvolvimento de um otimizador com base em Algoritmos Genéticos

1st Christopher Renkavieski
Departamento de Ciência da Computação
Universidade do Estado de Santa Catarina
Joinville, Brasil
chris.renka@gmail.com

***Resumo*—Este trabalho apresenta uma ferramenta de otimização baseada em algoritmos genéticos. É explicada a formatação dos arquivos de entrada de parâmetros, e todas as funcionalidades da ferramenta são apresentadas. Esta ferramenta foi aplicada a seis problemas de otimização: maximização de bits alternados, lucro de uma fábrica de rádios, função Ackley com codificação binária e real, maximização de pares alternados par/ímpar, e uma instância do problema do caixeiro viajante. São apresentados, também, os parâmetros utilizados e resultados obtidos para cada um desses problemas, além de uma análise desses resultados.**

***Index Terms*—Computação Evolutiva, Algoritmo Genético, Evolução**

I. Introdução

Algoritmos Genéticos são uma classe de algoritmos bio-inspirados pertencentes a área de Computação Evolutiva. Estes algoritmos utilizam como inspiração a teoria Neo-Darwinista, com o objetivo de ”evoluir”uma solução para um problema, em uma população de possíveis soluções.

Cada indivíduo da população representa uma possível solução a um problema, e, ao longo de um número de gerações, estes indivíduos sofrem mutações e recombinações entre si, gerando novos indivíduos. Estes processos são guiados pela pressão evolutiva, que age por meio do *fitness* e da seleção, que faz com que melhores indivíduos tenham maiores probabilidades de realizar recombinação, também chamada de *crossover*.

Neste trabalho, desenvolvido para a disciplina de Computação Evolucionária, sob orientação do professor Rafael Parpinelli, foi desenvolvida uma ferramenta otimizadora com base nestes princípios.

A seção II deste relatório apresenta os detalhes da ferramenta desenvolvida, incluindo a formatação de seus arquivos de entrada, a geração da população inicial, as técnicas de seleção, *crossover* e mutação, o cálculo da diversidade populacional, o *loop* evolutivo e a função *fitness*.

A seção III detalha os problemas utilizados para avaliar a ferramenta, e a seção IV apresenta os parâmetros e resultados dos testes realizados. A seção V apresenta a análise destes resultados, e a seção VI conclui o relatório com as considerações finais do autor.

II. Ferramenta

A ferramenta foi desenvolvida em linguagem Python 3, e possui as seguintes funcionalidades: leitura dos parâmetros de entrada, criação da população inicial, funções de seleção, *crossover* e mutação, cálculo da diversidade da população, funções para conversão de indivíduos binários em inteiros ou reais, caso necessário, e o *loop* evolutivo [1], sendo que cada uma dessas partes está em um arquivo separado. As funções específicas para cada problema, como a função objetivo e *fitness*, são inseridas juntamente com o *loop* evolutivo.

Todos os valores aleatórios utilizados pela ferramenta são gerados pela biblioteca *random* do Python 3, que utiliza o algoritmo Mersenne Twister [2]. Exceto quando for explicitado o contrário, todos os valores são gerados com distribuição uniforme.

Cada uma das funcionalidades mencionadas é apresentada em detalhes nas subseções seguintes.

A. Entrada de parâmetros

Os parâmetros principais da execução são lidos a partir de dois arquivos de texto: paramPop.txt e paramEvol.txt, sendo que o primeiro guarda os parâmetros referentes à população, e o segundo guarda os parâmetros da evolução.

O arquivo paramPop.txt é composto por até seis linhas, sendo elas:

- tipo: tipo da codificação do indivíduo (BIN, INT ou REAL);
- POP: quantidade de indivíduos;
- D: quantidade de dimensões ou genes do indivíduo. Caso se deseje uma população de indivíduos binários representando valores inteiros ou reais, D continua sendo a quantidade de genes, e não o tamanho do indivíduo;
- Li: lista de limites inferiores para cada dimensão, separadas por espaços. Caso todos as dimensões tenham o mesmo limite inferior, basta escrever este valor uma única vez;
- Ui: lista de limites superiores para cada dimensão, separados por espaços. Como no caso anterior, pode-se escrever apenas um valor, caso todos os limites sejam iguais;
- extra: informação extra sobre a população. Ela pode ser:
  - Caso a codificação seja inteira, extra deve valer 1 quando se deseja permutação, e 0 caso contrário;

  - Caso esteja sendo utilizada codificação binária para se representar valores inteiros ou reais, extra deve ser INT caso a codificação seja inteira, ou REAL x caso a codificação seja real, onde x é o número de casas decimais desejadas.

Caso se esteja trabalhando com codificação inteira com permutação, as linhas Li e Ui podem ser omitidas. Se, por outro lado, elas estiverem presentes, a linha D pode ser omitida. Caso as três linhas estejam presentes, a informação da linha D será ignorada.

Já o segundo arquivo, paramEvol.txt, é composto por sete linhas:

- sel: algoritmo de seleção a ser utilizado. Valor 0 para seleção por torneio, e qualquer valor $0 < k \leq POP$ para seleção por torneio de tamanho k;
- elite: 1 caso se deseje usar elitismo, ou 0 caso contrário;
- cxTipo: tipo de rotina de *crossover* a ser utilizada. Deve ser igual a 1, 2 ou 3, que representam, respectivamente, *crossover* de um ponto, dois pontos e uniforme, para codificações binária ou inteira sem permutação, ou *crossover* BLX, *Uniform Average* e aritmético, para codificação real. Caso a codificação seja inteira com permutação, apenas o *crossover* PMX é utilizado, portanto o valor desta linha é ignorado;
- cxProb: valor real entre 0 e 1, que representa a probabilidade de *crossover*;
- mutProb: valor real entre 0 e 1, que representa a probabilidade de mutação;
- ger: número de gerações de uma execução;
- nExec: número de execuções a ser realizadas. Os gráficos gerados, ao final, são a média de nExec execuções.

B. População inicial

Especificados os parâmetros da população, será gerada aleatoriamente uma população de indivíduos segundo esses parâmetros.

Um ponto que vale ser notado é quanto a geração de indivíduos binários representando valores inteiros ou reais. Nesses casos, o tamanho do indivíduo é calculado com base na quantidade de genes, no intervalo de cada gene, e na quantidade de casas decimais, em caso de valores reais.

As figuras 1, 2, 3 mostram, respectivamente, exemplos de populações geradas com codificação binária, inteira com permutação e real.

```
[False, True, False, True, True, False, True, False, False, True]
[False, False, False, False, True, False, False, True, True, False]
[True, False, True, True, True, False, False, False, True, True]
[True, True, True, True, False, True, True, True, False, False]
[False, True, False, False, True, True, True, False, False, True]
[False, True, False, False, True, True, False, True, True, True]
```

Figura 1. Exemplo de população de indivíduos com codificação binária.

C. Seleção

Para a seleção dos indivíduos da população que poderão realizar *crossover*, foram implementadas duas técnicas: roleta e torneio. Em ambos os casos, a seleção é feita sem reposição,

```
[9, 7, 4, 6, 0, 2, 5, 8, 1, 3]
[6, 0, 3, 9, 5, 1, 7, 4, 2, 8]
[9, 8, 6, 3, 7, 5, 4, 1, 2, 0]
[0, 5, 2, 3, 1, 6, 9, 8, 7, 4]
[8, 2, 9, 0, 7, 3, 1, 4, 6, 5]
[8, 3, 6, 5, 7, 9, 1, 4, 2, 0]
```

Figura 2. Exemplo de população de indivíduos com codificação inteira e com permutação.

```
[-25.512, -13.466, 24.348, 8.67, 7.404]
[14.1, 23.488, -25.673, 16.965, 18.484]
[2.522, 8.403, -22.779, 21.558, 17.751]
[-3.463, 20.255, -1.233, -24.74, 15.678]
[-9.67, -6.852, 21.885, 7.119, 23.914]
[-4.838, -22.125, -27.672, 27.718, 8.428]
```

Figura 3. Exemplo de população de indivíduos com codificação real

ou seja, existe a possibilidade de o segundo indivíduo de um par ser igual ao primeiro.

A seleção por roleta é feita calculando-se o *fitness* relativo de cada indivíduo em relação ao total, e em seguida um indivíduo é selecionado aleatoriamente, com probabilidade proporcional ao seu *fitness* relativo. Caso o objetivo seja minimizar o *fitness*, antes dessa operação, o *fitness* de cada indivíduo é subtraído do maior *fitness* da população multiplicado por 1.15, de modo que o indivíduo que antes possuía o menor valor de *fitness* passa a ter o maior valor, e vice versa. O fator 1.15 é aplicado para que o indivíduo de maior *fitness*, antes do tratamento, não fique com *fitness* zero.

Já na seleção por torneio, a função recebe um valor k como parâmetro. São escolhidos aleatoriamente, da população, k indivíduos, e é selecionado aquele com o maior valor de *fitness* entre estes. Caso se deseje minimizar, o processo é o mesmo, porém é selecionado aquele com menor *fitness* entre os k indivíduos escolhidos aleatoriamente.

D. Crossover

No *crossover*, pares de indivíduos selecionados anteriormente são cruzados, gerando assim dois novos indivíduos cuja carga genética é uma combinação das cargas de seus pais.

Dentre os indivíduos selecionados anteriormente, parte deles farão parte de um par para *crossover*, e os outros serão levados diretamente à próxima etapa, sem cruzamento. Esta decisão é feita a partir da probabilidade de *crossover* informada no arquivo paramEvol.txt, sendo que cada indivíduo tem uma probabilidade cxProb de realizar *crossover*.

Os algoritmos de *crossover* implementados foram:

- Codificações binária e inteira sem permutação:
  - Um ponto;
  - Dois pontos;
  - Uniforme.
- Codificação inteira com permutação:
  - PMX (*Partially-Mapped Crossover*).
- Codificação real:
  - BLX (*Blend Crossover*), com $\alpha = 0.5$;
  - *Uniform Average Crossover*;

– *Crossover* aritmético.

E. Mutação

Os indivíduos gerados na etapa do *crossover*, na sequência, passam pela etapa de mutação, em que os indivíduos podem sofrer alterações aleatórias em seus genes.

Nesta, cada gene de um indivíduo tem uma probabilidade igual a mutProb de sofrer mutação, sendo este valor um dos parâmetros do arquivo paramEvol.txt. Caso um gene sofra mutação, esta é realizada da seguinte forma, de acordo com a codificação do indivíduo:

- Codificação binária: o valor do bit é invertido.
- Codificação inteira sem permutação: um novo valor aleatório é gerado dentro do domínio da variável.
- Codificação inteira com permutação: o gene é trocado de lugar com outro escolhido aleatoriamente.
- Codificação real: o valor do gene é substituído por outro, gerado aleatoriamente, com distribuição gaussiana. A geração é feita com o valor atual do gene no centro da distribuição, e com desvio padrão igual a um décimo do domínio da variável.

F. Diversidade

A diversidade de uma população é calculada de duas formas: par a par, e por momento de inércia.

O cálculo de diversidade par a par é o método tradicional, onde é calculada a média das distâncias entre cada indivíduo e todos os outros. Este valor é, então, normalizado em relação à maior distância possível entre dois indivíduos no espaço de busca. A distância calculada é: Hamming para codificação binária, Manhattan para codificação inteira, e euclideana para codificação real.

Já a segunda forma de se calcular a diversidade foi proposta por Morrison [3], e é equivalente ao cálculo do momento de inércia de n massas pontuais, de mesma massa, em torno do centro de massa do sistema. As coordenadas do centro de massa são calculadas como a média das coordenadas de cada indivíduo da população, e a distância de cada indivíduo ao centro de massa é calculada pela distância de Hamming, pra indivíduos binários, ou pela distância euclideana, para indivíduos inteiros ou reais.

Em seu artigo, Morrison [3] prova a equivalência entre as duas métricas para o caso de indivíduos com codificação binária, mas não para codificação inteira ou real. Neste trabalho, ambas as métricas são utilizadas, para que elas possam ser comparadas nas três codificações. Estando comprovada a equivalência, o cálculo da diversidade pelo centro de massa é vantajoso por ser menos computacionalmente custoso que o método par a par, por ter complexidade linear, enquanto este último tem complexidade quadrática.

G. Conversão binária

Caso se deseje utilizar codificação binária para representar genes inteiros ou reais, os cálculos para o tamanho do indivíduo são realizados automaticamente, bastando ao usuário informar a quantidade de genes do indivíduo, os limites inferior e superior para os genes, e a quantidade de casas decimais desejada, em caso de valores reais.

A função para converter um indivíduo de binário para codificação inteira ou real deve ser chamada juntamente às funções para cálculo do *fitness* ou função objetivo, que são específicas para cada problema.

H. Loop evolutivo e fitness

Cada execução do *loop* evolutivo segue a seguinte ordem: uma população inicial é gerada e tem o *fitness* de cada indivíduo calculado. São guardados o melhor *fitness* e o *fitness* médio dessa população. Também são calculados e guardados os valores da diversidade da população, tanto pelo método par a par, quanto pelo momento de inércia. No *loop* evolutivo, também deve ser definido se o objetivo é maximizar ou minimizar a função *fitness*.

Na sequência, o *loop* é iniciado, sendo repetidas as seguintes etapas pelo número de vezes informado no parâmetro ger do arquivo paramEvol.txt:

- Seleção: em uma população de tamanho n, com n par, são selecionados n indivíduos;
- *Crossover*: é aplicada a rotina de *crossover* aos indivíduos selecionados;
- Mutação: é aplicada a rotina de mutação aos indivíduos gerados na etapa do *crossover*;
- Elitismo: caso a opção de elitismo esteja ativada, o melhor indivíduo da geração anterior é inserido na população gerada pela etapa da mutação. Este indivíduo substituirá outro aleatório da população;
- *Fitness*: o valor da função *fitness* é calculado para todos os indivíduos da população atual;
- Diversidade: a diversidade da população gerada é calculada, por ambos os métodos explicados;
- Guardar informações: são guardados os valores do melhor *fitness* e do *fitness* médio da população, além de ambos os valores da diversidade. Caso haja elitismo, a codificação do melhor indivíduo também é guardada, para ser inserida na próxima geração.

Ao fim deste *loop*, além das listas com os valores de melhor *fitness*, *fitness* médio e das diversidades, também é retornado o melhor indivíduo gerado ao longo da execução, juntamente com o seu valor *fitness*.

Este *loop* é executado um número de vezes igual ao parâmetro nExec, do arquivo paramEvol.txt. Ao fim de nExec execuções, é feita a média, para cada geração, do valor *fitness* médio e do melhor indivíduo, e dos valores de diversidade de cada execução. Estes valores são então salvos em arquivos, que podem ser plotados.

O retorno desta função é a codificação do melhor indivíduo, além do valor da função objetivo deste indivíduo. Também podem ser feitos outros retornos, específicos para o problema, caso necessário.

As funções específicas de cada problema, como a função objetivo, *fitness* ou avaliação de restrições, são colocadas no mesmo arquivo do *loop* evolutivo, e devem ser chamadas den-

tro deste. A seção seguinte apresenta os problemas utilizados para avaliar a ferramenta.

III. Problemas

Foram utilizados seis problemas para testar a ferramenta elaborada, abrangendo todos os tipos de codificação para indivíduos. Estes problemas foram:

- Codificação binária:
  - Maximizar bits alternados;
  - Maximizar lucro de uma fábrica de rádios (valores inteiros codificados em binário);
  - Minimizar função Ackley (valores reais codificados em binário).
- Codificação inteira sem permutação:
  - Maximizar valores alternados par/ímpar ou ímpar/par.
- Codificação inteira com permutação:
  - Problema do caixeiro viajante.
- Codificação real:
  - Minimizar função Ackley.

Cada um destes problemas é apresentado em detalhes nas subseções seguintes.

A. Maximizar bits alternados

Em uma *string* binária, deseja-se maximizar a quantidade de pares True/False ou False/True. Para isto, faz-se com que cada indivíduo seja igual à *string* binária, e o valor da função objetivo é a quantidade de pares de bits alternados dentro dele. Este valor é, então, normalizado em relação à maior quantidade possível de pares alternados, que é igual ao tamanho do indivíduo menos 1.

Assim, por exemplo, com indivíduos de tamanho 5, indivíduos ótimos são:

[True, False, True, False, True],
[False, True, False, True, False]

Enquanto indivíduos de *fitness* zero são:

[True, True, True, True, True],
[False, False, False, False, False]

B. Maximizar lucro de uma fábrica de rádios

Este é um problema de pesquisa operacional, com a seguinte descrição:

Uma fábrica produz dois tipos de rádio, *standard* e luxo. Cada rádio *standard* fornece um lucro de R$30,00, consome um funcionário/dia pra ser produzido, e sua linha de produção suporta um máximo de 24 funcionários. Já um rádio de luxo fornece R$40,00 de lucro, exige dois funcionários/dia pra ser produzido, e sua linha de produção suporta um máximo de 32 funcionários. Sendo que a fábrica possui, no total, 40 funcionários para alocar à produção de rádios, deve-se determinar como eles devem ser alocados para que o lucro diário seja maximizado.

Sendo as variáveis do problema: $st = [0, 24]$ a quantidade de rádios *standard* produzidos, e $lx = [0, 16]$ a quantidade de rádios de luxo, tem-se a função objetivo FO do problema:

$$FO = 30st + 40lx \tag{1}$$

Um indivíduo é codificado como um par (st, lx), sendo que ambos os valores são representados em binário, neste caso.

Além disso, este problema apresenta uma restrição na forma:

$$st + 2lx \leq 40 \tag{2}$$

Para se calcular o *fitness* de um indivíduo para este problema, deve-se levar em consideração tanto a função objetivo quanto a restrição, da seguinte forma:

$$fit = FO + rh \tag{3}$$

Onde fit é o valor da função *fitness*, FO é o valor da função objetivo, r é o coeficiente de penalidade, e h é o valor da penalidade. Como a violação da restrição deve impactar negativamente no *fitness* do indivíduo, faz-se $r = -1$, e h como sendo igual ao quanto a restrição foi violada.

Para que a função objetivo e o valor da penalidade tenham a mesma escala, elas devem ser normalizadas. Assim, a função objetivo normalizada fica:

$$\overline{FO} = \frac{30st + 40lx}{1360} \tag{4}$$

Onde 1360 é o maior lucro possível, ignorando a restrição. Já para a penalidade, tem-se:

$$\overline{h} = max\{0, \frac{st + 2lx - 40}{16}\} \tag{5}$$

Onde 16 é a maior violação possível da restrição.

Assim, tem-se a função *fitness* normalizada:

$$\overline{fit} = \overline{FO} - \overline{h} \tag{6}$$

C. Função Ackley

A função Ackley [4] é uma função de otimização comumente utilizada como *benchmark*. Ela é uma função de minimização, e tem como ponto mínimo a origem.

Esta função é representada pela seguinte fórmula:

$$\begin{aligned} f(x_0...x_n) = -20exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) \\ - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i) + 20 + e) \end{aligned} \tag{7}$$

Sendo que $-32 \leq x_i \leq 32$.

Esta função foi aplicada a indivíduos com codificação real e binária. Foi utilizado o próprio valor da função aplicada ao indivíduo como *fitness*.

D. Maximizar valores alternados par/ímpar ou ímpar/par

Semelhante à maximização de bits alternados, aqui deseja-se maximizar, em uma *string* de valores inteiros, a quantidade de pares de valores par/ímpar ou ímpar/par dentro desta *string*. Assim, cada indivíduo é representado por uma *string* de valores inteiros, e a função objetivo é a quantidade de pares alternados dentro dele. O *fitness* é calculado como o valor da função objetivo, normalizada em relação à maior quantidade de pares alternados possível, que é igual ao tamanho do indivíduo menos um.

Assim, um exemplo de indivíduo ótimo, de tamanho 5, é:

[5, 0, 7, 4, 3]

Enquanto um exemplo de indivíduo de *fitness* zero é:

[2, 8, 4, 6, 0]

E. Problema do caixeiro viajante

O problema do caixeiro viajante é um problema comum de grafos, em que se deseja encontrar o menor caminho que passe por todos os vértices de um grafo uma única vez, e então retorne ao vértice inicial.

Neste trabalho, este problema foi aplicado ao conjunto de 10 cidades cujas coordenadas são mostradas na na figura 4.

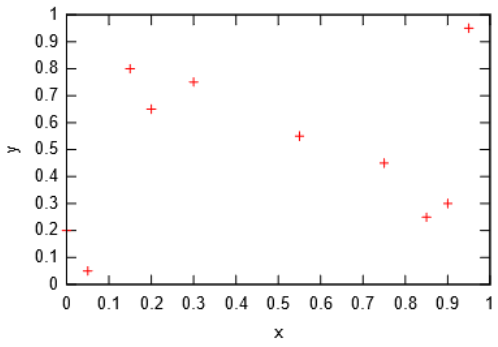

Figura 4. Coordenadas das cidades para o problema do caixeiro viajante.

Foram utilizados indivíduos com codificação inteira com permutação, isto é, não há repetição de valores dentro de um indivíduo. Cada cidade é representada por um número de 0 a 9, e um indivíduo consiste de uma permutação desta lista de valores, e a sequência dos seus genes representa o percurso a ser feito pelas cidades.

É utilizada a distância euclideana para se calcular a distância entre as duas cidades, e a função objetivo é a distância total de um percurso, passando por todas as cidades e voltando à cidade de partida. Como este problema não possui restrições, foi utilizada a própria função objetivo como função *fitness*.

IV. Testes e resultados

Para cada um dos problemas apresentados, as subseções seguintes apresentam os parâmetros utilizados nos testes, e os resultados obtidos. Todos os resultados mostrados são a média de 10 execuções, e com populações de 20 indivíduos.

A. Bits alternados

Para este problema, foram utilizados indivíduos de tamanho 10, e os parâmetros evoltivos foram: seleção por roleta, *crossover* de um ponto, probabilidade de *crossover* de 80%, e probabilidade de mutação de 3%. Foi utilizado elitismo.

A figura 5 mostra a evolução da função *fitness* média e do melhor indivíduo para este problema. As figuras 6 e 7 mostram, respectivamente, a evolução da diversidade da população calculada par e par e por momento de inércia.

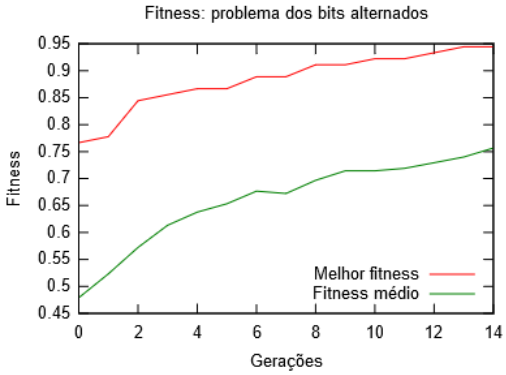

Figura 5. Evolução da função *fitness* média e do melhor indivíduo para o problema dos bits alternados.

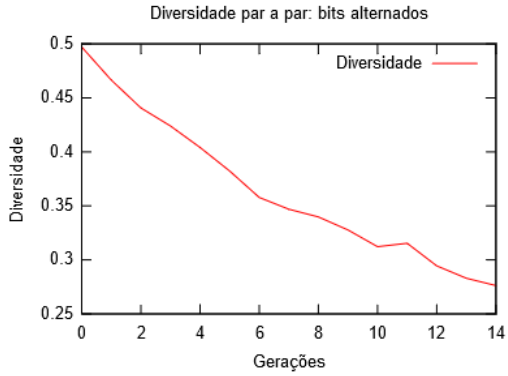

Figura 6. Evolução da diversidade calculada par a par, para o problema dos bits alternados.

O melhor indivíduo encontrado foi: [False, True, False, True, False, True, False, True, False, True], com um total de 9 pares de bits alternados.

B. Fábrica de rádios

Devido às especificações do problema, os indivíduos têm dois genes, que representam as variáveis st e lx. Cada indivíduo é uma *string* binária que representa estes dois valores.

Para este problema foram realizados dois testes. O primeiro dele utilizou os seguintes parâmetros evolutivos: *crossover* uniforme, probabilidade de *crossover* de 80%, probabilidade de mutação de 3%, e elitismo.

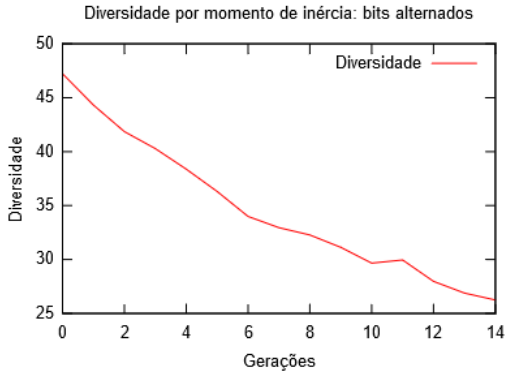

Figura 7. Evolução da diversidade calculada por momento de inércia, para o problema dos bits alternados.

A figura 8 mostra a evolução da função *fitness* para este caso. As figuras 9 e 10 mostram, respectivamente, a evolução da diversidade da população calculada par e par e por momento de inércia.

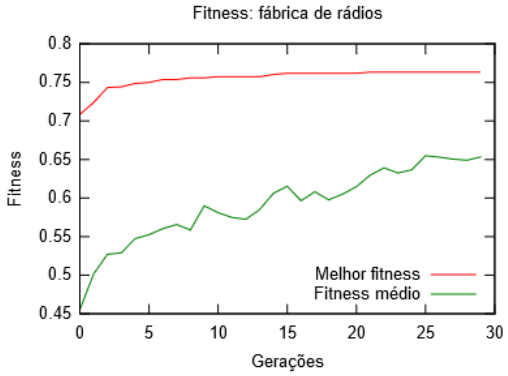

Figura 8. Evolução da função *fitness* média e do melhor indivíduo para o problema da fábrica de rádios.

O melhor indivíduo encontrado para este caso foi: [True, True, True, True, True, True, False, False, False, False], que, convertido para os valores inteiros do domínio, significa 24 rádios *standard* e 8 rádios de luxo, fornecendo um lucro de R$1040,00, e sem violar restrições.

Já o segundo teste modifica a probabilidade de mutação, de 3% para 75%, para avaliar o impacto de uma alta probabilidade de mutação. A figura 11 mostra a evolução da função *fitness* para este caso, enquanto as figuras 12 e 13 mostram a evolução da diversidade, calculada par a par e por momento de inércia, respectivamente.

O melhor indivíduo encontrado neste teste foi o mesmo do teste anterior.

C. Função Ackley com codificação binária

Este problema foi solucionado em duas dimensões, com indivíduos binários representando valores reais de 4 casas

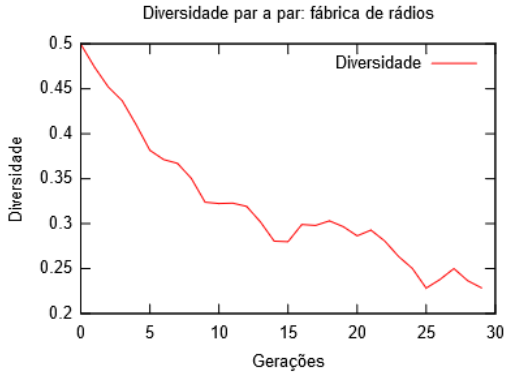

Figura 9. Evolução da diversidade calculada par a par, para o problema da fábrica de rádios.

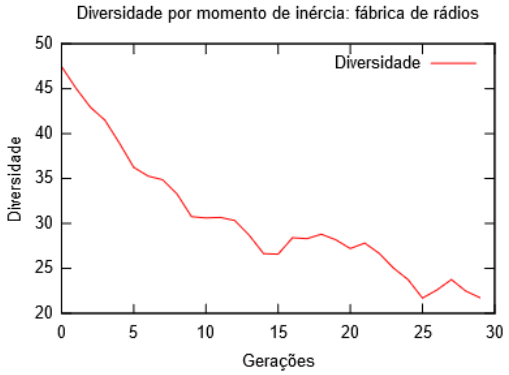

Figura 10. Evolução da diversidade calculada por momento de inércia, para o problema da fábrica de rádios.

decimais.

Foram utilizados os seguintes parâmetros evolutivos: seleção por torneio de tamanho 3, *crossover* de dois pontos, probabilidade de *crossover* de 80%, probabilidade de mutação de 3%, e elitismo.

A figura 14 mostra a evolução da função *fitness* para este problema. As figuras 15 e 16 mostram a evolução da diversidade da população calculada par e par e por momento de inércia, respectivamente.

O melhor indivíduo encontrado nessa execução, em codificação binária, foi: [True, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, True, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False].

A representação real deste indivíduo é (0.0000, 0.0000), e o valor da função Ackley aplicada a ele é $4.4409e-16$.

D. Valores alternados par/ímpar ou ímpar/par

Para este problema, foram utilizados indivíduos de tamanho 10, cujos genes podem entre os valores 0 e 9.

Foram feitos dois testes para este problema. O primeiro deles utilizou seleção por roleta, *crossover* de dois pontos,

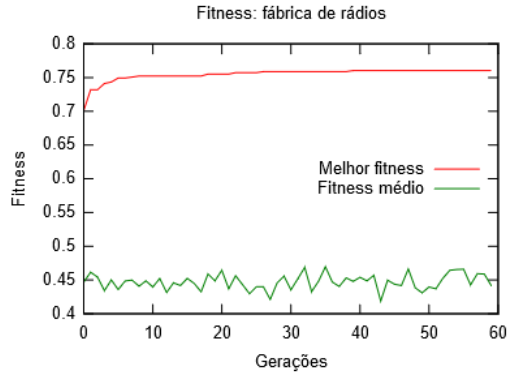

Figura 11. Evolução da função *fitness* média e do melhor indivíduo para o problema da fábrica de rádios com alta probabilidade de mutação.

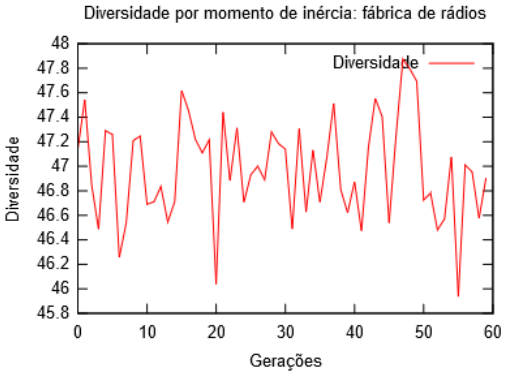

Figura 13. Evolução da diversidade calculada por momento de inércia, para o problema da fábrica de rádios com alta probabilidade de mutação.

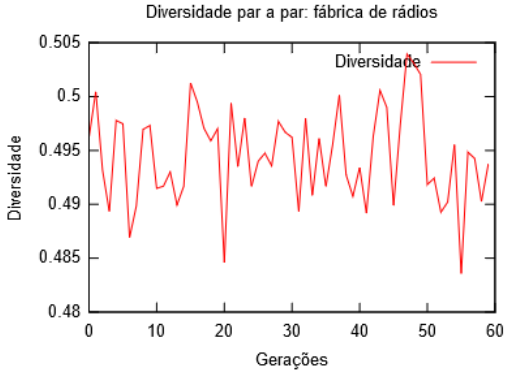

Figura 12. Evolução da diversidade calculada par a par, para o problema da fábrica de rádios com alta probabilidade de mutação.

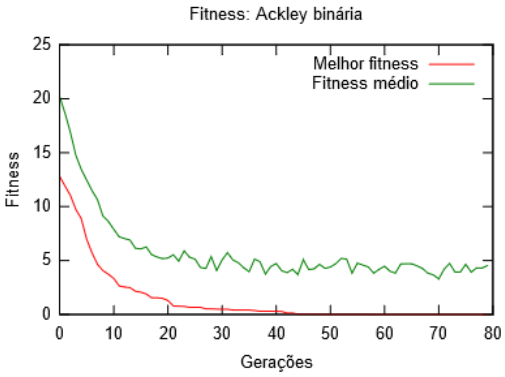

Figura 14. Evolução da função *fitness* média e do melhor indivíduo para a função Ackley com indivíduos binários.

probabilidade de *crossover* de 80%, probabilidade de mutação de 3%, e elitismo.

A figura 17 mostra a evolução da função *fitness* para este problema. As figuras 18 e 19 mostram a evolução da diversidade da população calculada par e par e por momento de inércia, respectivamente.

O melhor indivíduo encontrado neste teste foi [1, 6, 3, 6, 7, 8, 3, 2, 3, 8], que tem 9 pares alternados par/ímpar.

Já no segundo teste, foram utilizados os mesmos parâmetros, com exceção do elitismo, para avaliar o impacto deste parâmetro na evolução da função *fitness*.

A figura 20 mostra a evolução da função *fitness* para este teste. As figuras 21 e 22 mostram a evolução da diversidade da população calculada par e par e por momento de inércia, respectivamente.

Neste teste, o melhor indivíduo encontrado foi [2, 5, 8, 5, 2, 7, 0, 3, 6, 3], com 9 pares alternados par/ímpar.

E. Caixeiro viajante

Este problema foi aplicado ao conjunto de cidades mostrado na figura 4. Foram utilizados: seleção por torneio de tamanho 3, probabilidade de *crossover* de 80%, probabilidade de mutação de 3%, e elitismo.

A figura 23 mostra a evolução da função *fitness* para este problema. As figuras 24 e 25 mostram a evolução da diversidade da população calculada par e par e por momento de inércia, respectivamente.

O melhor caminho encontrado para este problema apresentou tamanho 3.4986, e é mostrado na figura 26.

F. Função Ackley com codificação real

Como no caso anterior, este problema foi solucionado em duas dimensões.

Foram utilizados os seguintes parâmetros evolutivos: seleção por roleta, *crossover* BLX, probabilidade de *crossover* de 80%, probabilidade de mutação de 3%, e elitismo.

A figura 27 mostra a evolução da função *fitness* para este problema. As figuras 28 e 29 mostram a evolução da diversidade da população calculada par e par e por momento de inércia, respectivamente. A diversidade por momento de inércia, neste caso, foi substituída pelo valor de sua raiz quadrada, que apresentou um comportamento melhor, em testes.

O melhor indivíduo encontrado foi ($6.556432571730276e-06$, $8.45467796411823e-06$), e a aplicação da função Ackley sobre ele resulta no valor $3.026436169806246e-05$.

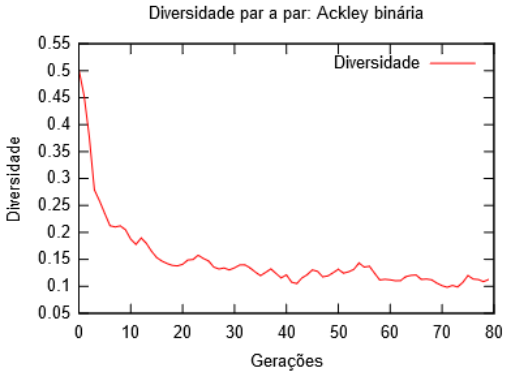

Figura 15. Evolução da diversidade calculada par a par, para a função Ackley com indivíduos binários.

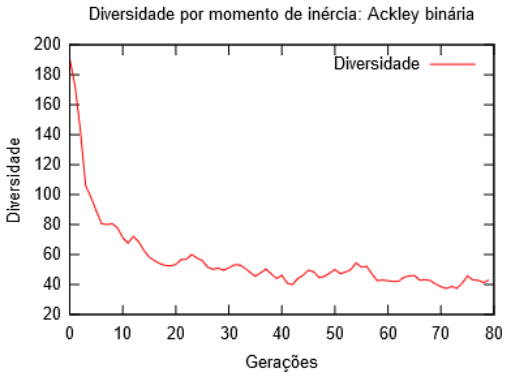

Figura 16. Evolução da diversidade calculada por momento de inércia, para a função Ackley com indivíduos binários.

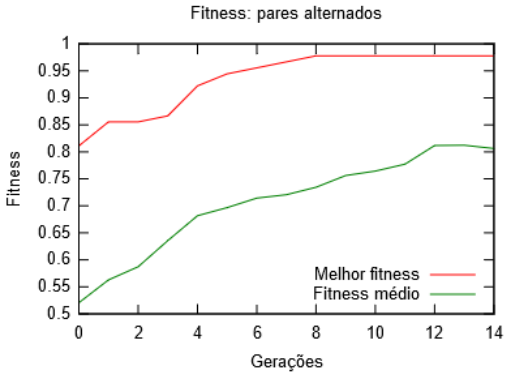

Figura 17. Evolução da função *fitness* média e do melhor indivíduo para o problema dos pares alternados.

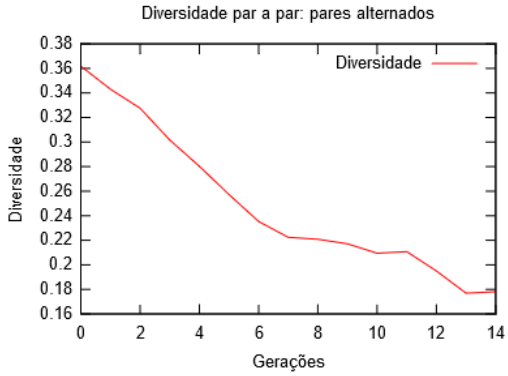

Figura 18. Evolução da diversidade calculada par a par, para o problema dos pares alternados.

V. Análise

Os testes realizados comprovaram a eficácia da ferramenta, atingindo, todas as vezes, resultados ótimos ou muito próximos do ótimo.

No teste realizado para o problema da fábrica de rádios, com alta taxa de mutação, pôde-se observar o impacto deste operador na evolução. Como se pode ver nas figuras 12 e 13, a alta diversidade da população, em comparação com o teste feito com baixa taxa de mutação, indica uma busca próxima à aleatória. Esta hipótese é reforçada pelo *fitness* médio mostrado na figura 11, que é estagnado em um valor baixo, indicando que, em média, a população não está melhorando, e indivíduos com alto *fitness* são apenas gerados ao acaso.

Comparando-se as figuras 20 e 17, pode-se observar o impacto do elitismo na evolução. O teste realizado sem elitismo foi o único em que o valor *fitness* do melhor indivíduo diminuiu entre duas gerações. Em todos os outros testes, o valor *fitness* do melhor indivíduo apenas aumenta ou se mantém igual entre gerações.

Quanto à diversidade da população, pôde-se observar que a métrica baseada em momento de inércia é válida também para codificações inteiras e reais, além da binária, como proposto por Morrison [3], pois apresenta um comportamento semelhante à métrica par a par. A maior diferença observada foi para a codificação real, sendo que, nesses casos, o valor apresentado é a raiz quadrada do momento de inércia, que, em testes, apresentou um comportamento mais semelhante à métrica tradicional que o valor direto do momento de inércia.

VI. Conclusão

Este relatório apresentou uma ferramenta de otimização baseada em algoritmos genéticos, desenvolvida sob orientação do professor Rafael Parpinelli, para a disciplina de Computação Evolucionária. Foram apresentados os diferentes componentes da ferramenta, e os resultados da sua aplicação a seis problemas de otimização.

Com esses resultados, pôde-se comprovar o poder computacional dos algoritmos genéticos, e a sua aplicabilidade a problemas mais complexos, bastando, para isto, que seja determinada a codificação dos indivíduos, a modelagem da função *fitness*, e a escolha dos parâmetros evolutivos.

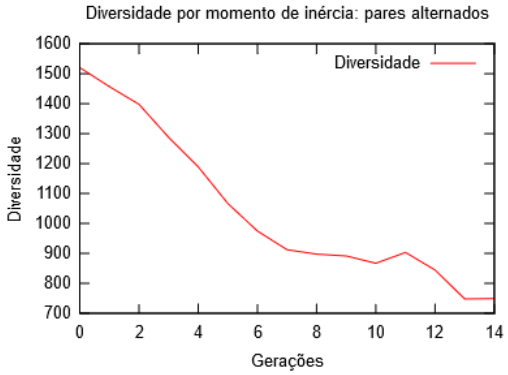

Figura 19. Evolução da diversidade calculada por momento de inércia, para o problema dos pares alternados.

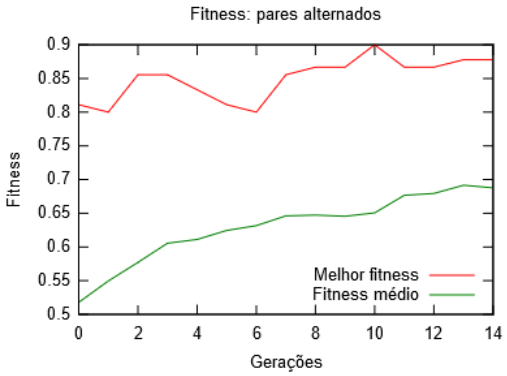

Figura 20. Evolução da função *fitness* média e do melhor indivíduo para o problema dos pares alternados, sem elitismo.

REFERÊNCIAS

[1] Parpinelli. Rafael: "Computação Evolucionária: fundamentos teóricos de AG, DCC / UDESC-Joinville. Disponível em: http://www.joinville.udesc.br/portal/professores/parpinelli/materiais/CE_0403_2018.pdf

[2] Python Software Foundation. 9.6. random - Generate pseudo-random numbers. Disponível em: https://docs.python.org/3/library/random.html

[3] Morrison R.W., De Jong K.A. (2002) Measurement of Population Diversity. In: Collet P., Fonlupt C., Hao JK., Lutton E., Schoenauer M. (eds) Artificial Evolution. EA 2001. Lecture Notes in Computer Science, vol 2310. Springer, Berlin, Heidelberg

[4] Ackley's Function. Disponível em: http://www.cs.unm.edu/ neal.holts/dga/benchmarkFunction/ackley.html

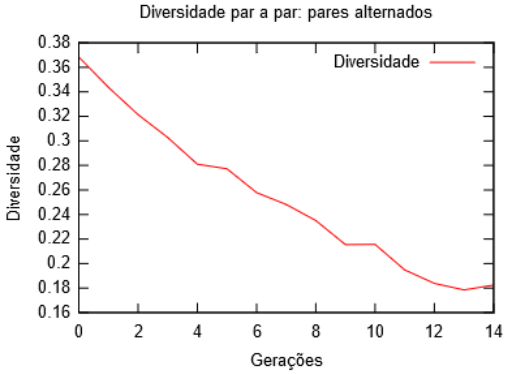

Figura 21. Evolução da diversidade calculada par a par, para o problema dos pares alternados, sem elitismo.

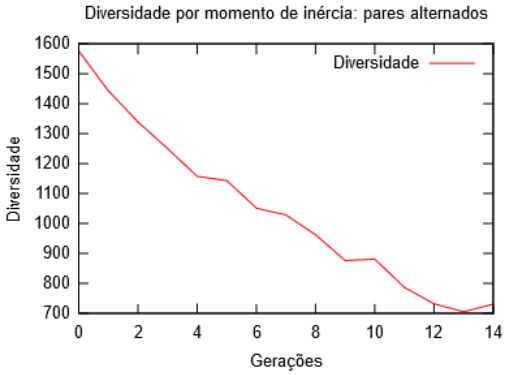

Figura 22. Evolução da diversidade calculada por momento de inércia, para o problema dos pares alternados, sem elitismo.

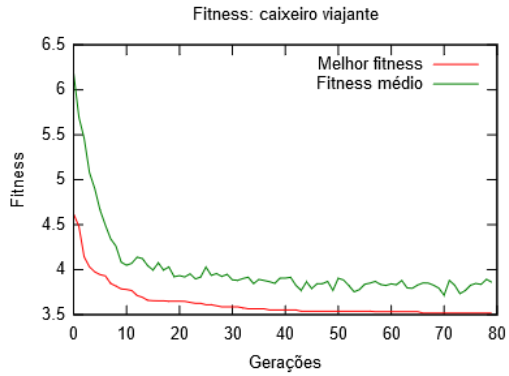

Figura 23. Evolução da função *fitness* média e do melhor indivíduo para o problema do caixeiro viajante.

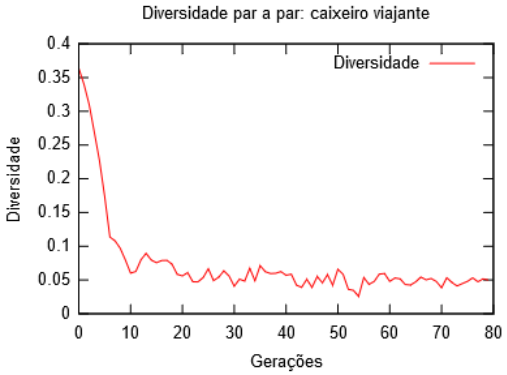

Figura 24. Evolução da diversidade calculada par a par, para o problema do caixeiro viajante.

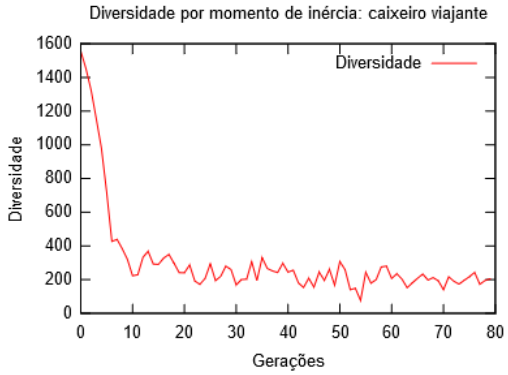

Figura 25. Evolução da diversidade calculada por momento de inércia, para o problema do caixeiro viajante.

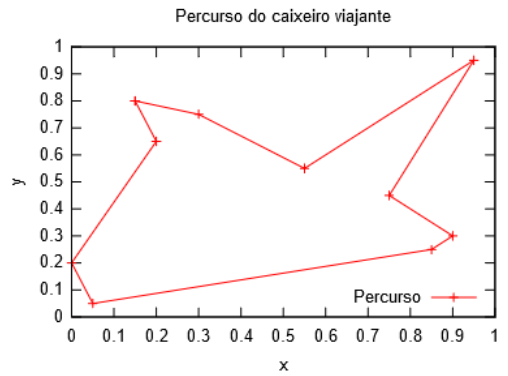

Figura 26. Percurso gerado pelo melhor indivíduo encontrado no problema do caixeiro viajante.

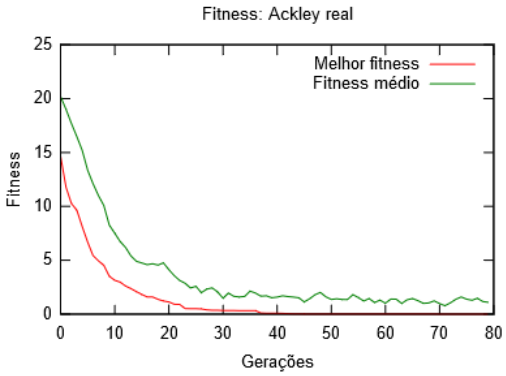

Figura 27. Evolução da função *fitness* média e do melhor indivíduo para a função Ackley com indivíduos reais.

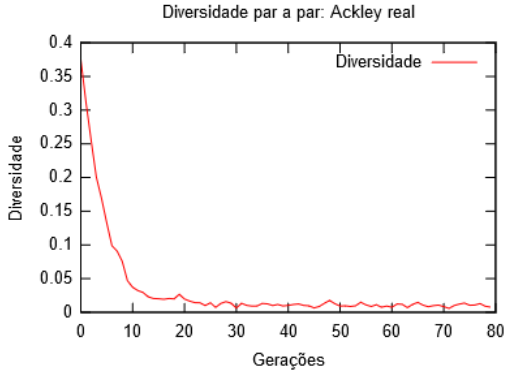

Figura 28. Evolução da diversidade calculada par a par, para a função Ackley com indivíduos reais.

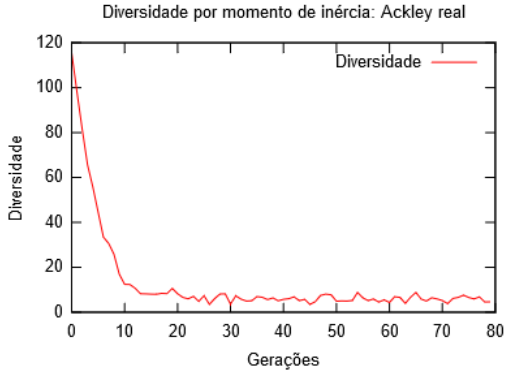

Figura 29. Evolução da diversidade calculada por momento de inércia, para a função Ackley com indivíduos reais.