INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Cássio dos Santos Sousa

Resolução otimizada de problemas com uso de algoritmos evolutivos

Trabalho de Graduação
2016

Computação

Número da CDU (tamanho 10)

Cássio dos Santos Sousa

Resolução Otimizada de Problemas com uso de Algoritmos Evolutivos

Orientador
Prof. Dr. Carlos Henrique Quartucci Forster (ITA)

Engenharia de Computação

SÃO JOSÉ DOS CAMPOS
INSTITUTO TECNOLÓGICO DE AERONÁUTICA

2016

Dados Internacionais de Catalogação-na-Publicação (CIP)
Divisão de Informação e Documentação

Sousa, Cássio dos Santos
Resolução Otimizada de Problemas com uso de Algoritmos Evolutivos / Cássio dos Santos Sousa.
São José dos Campos, 2016.
15f.

Trabalho de Graduação – Divisão de Ciência da Computação –
Instituto Tecnológico de Aeronáutica, 2016. Orientador: Prof. Dr. Carlos Henrique Quartucci Forster.

1. Algoritmo Evolutivo. 2. Inteligência Artificial. I. Instituto Tecnológico de Aeronáutica.
II. Resolução Otimizada de Problemas com uso de Algoritmos Evolutivos.

REFERÊNCIA BIBLIOGRÁFICA

SOUSA, Cássio dos Santos. **Resolução Otimizada de Problemas com uso de Algoritmos Evolutivos**. 2016. 15f. Trabalho de Conclusão de Curso. (Graduação) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

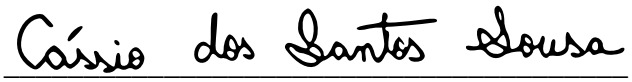
CESSÃO DE DIREITOS

NOME DO AUTOR: Cássio dos Santos Sousa
TÍTULO DO TRABALHO: Resolução Otimizada de Problemas com uso de Algoritmos Evolutivos
TIPO DO TRABALHO/ANO: Graduação / 2016

Cássio dos Santos Sousa
Pça Mal-do-Ar Eduardo Gomes, 50
12228-900 – São José dos Campos – SP

RESOLUÇÃO OTIMIZADA DE PROBLEMAS COM USO DE ALGORITMOS EVOLUTIVOS

Essa publicação foi aceita como Relatório Final de Trabalho de Graduação.



Cássio dos Santos Sousa
Autor

_____
Prof. Dr. Carlos Henrique Quartucci Forster (ITA)
Orientador

_____
Prof. Dr. Juliana de Melo Bezerra
Coordenadora do Curso de Engenharia de Computação

São José dos Campos, _____ de _____ de 2016.

Dedico este trabalho à minha mãe Lucinez, que me trouxe suporte durante todo meu tempo no ITA, e ao Instituto Ismart, oportunidade que muda a minha vida desde 2006.

Agradecimentos

À minha família, que se esforçou tanto para que eu ficasse até o fim no ITA.

Ao meu professor orientador Carlos Forster, que me guiou por caminhos tortuosos até a conclusão deste trabalho de graduação.

Aos meus colegas de turma, que foram uma família a mais enquanto estive no ITA, e que despertaram em mim o desejo de sempre voar mais alto.

Aos demais professores da COMP, que nos acompanham desde antes do Curso Profissional, e que podem nos ver hoje como verdadeiros engenheiros.

E ao Instituto Ismart, que investe em minha formação acadêmica, profissional e pessoal desde 2006 com muito amor e carinho.

When you are inspired by some great purpose, some extraordinary project, all your thoughts break their bounds. Your mind transcends limitations, your consciousness expands in every direction and you find yourself in a new, great and wonderful world. Dormant forces, faculties and talents become alive, and you discover yourself to be a greater person by far than you ever dreamed yourself to be.

Patañjali, criador das práticas do Ioga.

Resumo

Insira algum texto interessante aqui.

Abstract

Insert some interesting text here.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Listagens

Lista de Abreviaturas

AE Algoritmo Evolutivo

AG Algoritmo Genético

1 *Introdução*

A resolução de diversos problemas se dá na forma de algoritmos, de instruções bem definidas. No entanto, alguns algoritmos podem pedir inúmeras instruções até concluírem, o que pode até mesmo inviabilizar a solução encontrada. A Inteligência Artificial pode atuar sobre tais problemas de modo a interagir com o problema e aprender com ele a encontrar uma solução. Ótimos candidatos para esta tarefa são os chamados *Algoritmos Evolutivos (AE)*.

Algoritmos evolutivos são aqueles que se baseiam nos princípios de evolução natural da Biologia, e são aplicados em um modo particular de solução de problemas: o da tentativa-e-erro. Tais algoritmos seguem um framework mais ou menos comum, atuante sobre diferente *gerações* de um problema, por meio de mudanças e combinações de *indivíduos* existentes numa *população* (4). Tal framework, conjuntamente com as ações evolutivas, pode ser visto na figura 1.

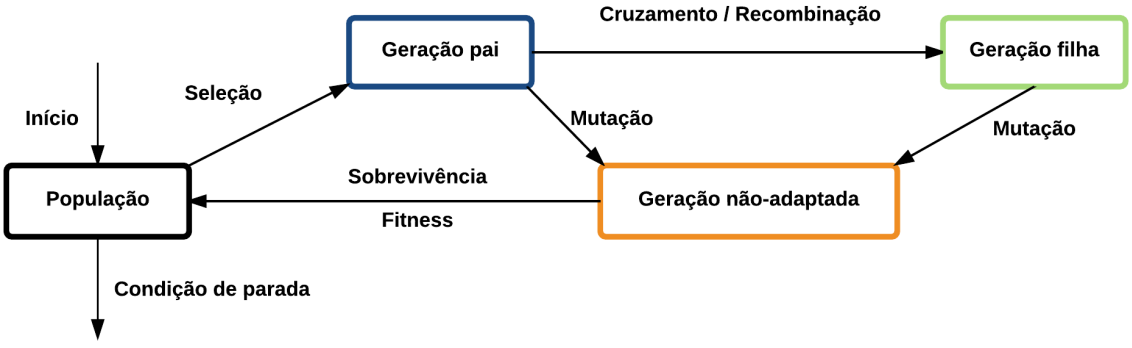


Figura 1: Framework de um algoritmo evolutivo.

Cada indivíduo está tentando resolver o problema durante a execução do algoritmo evolutivo. A população contém estes indivíduos e é o alvo de interesse do processo evolutivo, e uma geração é a população que sobrevive após um ciclo de processos evolutivos do Algoritmo Evolutivo (AE) sobre o problema. Similar ao processo evolutivo, seus com-

ponentes principais são as operações de variação (mutação e recombinação) e de seleção (seleção da geração pai e sobrevivência), chamadas aqui simplesmente de *operações de evolução* (3).

Este trabalho se utilizou de um grupo específico de Algoritmo Evolutivo, chamado de **Algoritmo Genético (AG)**. Um AG possui, como a menor estrutura de seus indivíduos, o *gene*. Um gene costuma ter apenas duas propriedades: uma expressividade, normalmente associada a um valor numérico, e uma forma de mudá-la aleatoriamente.

Para um AG, a *mutação* é uma mudança não controlada de um indivíduo feita a partir da mudança na expressividade de um ou mais genes. A *recombinação* envolve a mistura de genes vindos de dois indivíduos que são cruzados. A *seleção* envolve uma escolha a dedo dos melhores exemplares para cruzamento (a geração pai). A *sobrevivência* envolve a rejeição de indivíduos que não estejam aptos o suficiente para resolver o problema. Tal aptidão é normalmente associada a uma função definida conjuntamente com o problema, chamada de *função de fitness*.

1.1 Motivação

É difícil por si só garantir que um algoritmo evolutivo encontre boas soluções mesmo para os problemas mais simples, dada a natureza de sua implementação. Garantir uma execução eficiente do mesmo é fundamental.

1.2 Objetivo

Este trabalho propôs implementar um algoritmo genético capaz de resolver problemas determinados de diferentes complexidades e encontrar boas soluções após uma quantidade razoável de gerações.

1.3 Abordagem

Em termos de implementação, o AG deve ser tal que, uma vez aplicado sobre um problema e uma população, as gerações se desenvolvam automaticamente. O trabalho foi então dividido em 4 etapas:

[1] Escolha e implementação de problemas compatíveis com a aplicação do AE;

[2] Implementação do AG e de suas operações de evolução;

[3] Otimização das funções de mutação e recombinação;

[4] Análise de performance e coleta de dados.

As primeiras duas etapas são interdependentes, e precisaram ser completadas primeiro. A terceira etapa funciona como a etapa de otimização do AG, e precisou ser tratada separadamente. A última etapa é uma etapa de análise, e foi encima dela que as conclusões foram feitas.

De modo a permitir a reutilização deste código em outros problemas, o AG foi feito de modo a ser compatível com mais de um problema. As ferramentas de análise e coleta de dados consideraram tanto métricas comuns da literatura quanto parâmetros específicos dos problemas abordados.

Optou-se por utilizar Python como linguagem principal do código feito para este trabalho.

1.4 Plano de Trabalho

Este trabalho foi planejado de forma que as etapas de otimização e análise do AE demorassem mais tempo. A validação dos resultados foi feita com base na literatura encontrada e nos resultados obtidos por algoritmos de código aberto (*open source*) aplicados aos mesmos problemas.

1.5 Organização do Trabalho

- **Capítulo 1:** Introdução
- **Capítulo 2:** Problemas Escolhidos
- **Capítulo 3:** Algoritmo Genético
- **Capítulo 4:** Otimização de Mutação e Recombinação
- **Capítulo 5:** Validação com Outros Trabalhos
- **Capítulo 6:** Conclusões e Trabalhos Futuros

2 *Problemas Escolhidos*

Por mais que a estrutura básica de um algoritmo evolutivo seja capaz de resolver múltiplos problemas, é importante que ele seja validado por problemas de diferentes naturezas. Para isso, este trabalho focou sua atenção na resolução de três problemas com implementações diferentes para a construção do algoritmo genético.

Como o AG atua diretamente com populações, um problema deve defini-las de antemão, tanto em termos de indivíduo quanto em termos de gene. Conjuntamente, é necessário definir quão apto um indivíduo está frente à solução que ele propõe. Isso é feito por meio da *função de fitness*. Para o número de indivíduos na população (ao menos inicialmente), utilizou-se um valor padrão de 100 indivíduos.

Três problemas foram escolhidos: OneMax Booleano (5), OneMax Real e o problema do Caixeiro Viajante (1). Os dois primeiros foram escolhidos em termos não só de sua facilidade de implementação, mas também porque são problemas "fáceis" em termos de encontro de solução ótima (como será detalhado a seguir), o que permite testar hipóteses e heurísticas de modo bem mais simples. O terceiro problema foi escolhido por não só ter uma literatura rica, mas também por ser um problema complexo, cujos resultados podem ser analisados e comparados de modo mais rico.

2.1 OneMax Booleano

Dado um conjunto de 100 bits iniciados em 0, o AG deve ser capaz de tornar todos os bits iguais a 1.

Gene

Utilizou-se aqui o *BooleanGene*, um gene com expressividade booleana (0 ou 1). Sua operação de mutação consiste numa operação semelhante a jogar cara-e-coroa, trocando o valor expresso para 0 ou 1 aleatoriamente, com igual probabilidade.

Indivíduo

Cada indivíduo tentará resolver o problema, o que faz com que cada indivíduo tenha 100 genes do tipo BooleanGene.

Função de fitness

Conta-se o número de genes de um indivíduo que sejam iguais a 1. Quanto maior a contagem, melhor.

2.2 OneMax Real

Dado um conjunto de 100 variáveis reais iniciadas em 0, o AG deve ser capaz de tornar todas elas o mais próximo de 1. Este OneMax possui uma caminhada bem mais lenta que o anterior, pois um gene booleano possui apenas dois estados, o que faz com que as ações de mutação permitam uma evolução muito mais rápida, enquanto um gene real muda sua expressividade num espectro bem maior.

Gene

Utilizou-se aqui o *RealGene*, um gene com expressividade real entre 0.0 e 1.0. Sua operação de mutação consiste numa escolha aleatória de um número real no intervalo [0.0, 1).

Indivíduo

Cada indivíduo tentará resolver o problema, o que faz com que cada indivíduo tenha 100 genes do tipo RealGene.

Função de fitness

Soma-se a expressividade de todos os genes de um indivíduo. Quanto maior a contagem, melhor. Feita de maneira apropriada, esta função pode ser a mesma utilizada para o OneMax Booleano.

2.3 Caixeiro Viajante

Dado um conjunto de cidades e as distâncias entre elas, o AG deve ser capaz de descobrir qual o menor caminho que possibilita a um caixeiro visitar todas as cidades uma única vez e retornar à cidade original. Tal problema é NP-Hard, e avaliar se uma solução candidata é algo tão complexo quanto a resolução do problema em si.

Gene

Utilizou-se o *IntegerGene*, um gene com expressividade inteira entre 0 e K-1, com uma operação de mutação capaz de escolher aleatoriamente um valor inteiro neste intervalo. A inicialização deste gene possui K como parâmetro.

Indivíduo

No caso de um indivíduo do problema do Caixeiro Viajante, foi pensado que o mesmo deveria ser capaz de gerar, a partir da expressividade de seus genes, um percurso que passasse uma única vez por todas as cidades. Para isso, os genes aqui foram organizados de modo um pouco diferente dos problemas OneMax.

Digamos, por exemplo, que um caixeiro na cidade A precise passar pelas cidades [B, C, D, E, F] e voltar à cidade A. O indivíduo de tal problema teria então quatro genes (o número total de cidades, menos dois) criados da seguinte forma:

- O primeiro gene possui expressividade de 0 a 4;
- O segundo gene possui expressividade de 0 a 3;
- O terceiro gene possui expressividade de 0 a 2;
- O quarto gene possui expressividade de 0 a 1.

Digamos que um dos indivíduos do AG tenha, pela expressividade de seus genes, os valores [3, 0, 1, 0]. Para se calcular o percurso feito por tal indivíduo, escolhe-se a cidade da lista naquela posição, a qual é removida antes de se escolher a próxima cidade. Ou seja:

- Gene 1: [3] mapeia a cidade E na lista [B, C, D, E, F]. Sem ela, a lista se torna [B, C, D, F];

- Gene 2: [0] mapeia a cidade B na lista [B, C, D, F]. Sem ela, a lista se torna [C, D, F];
- Gene 3: [1] mapeia a cidade D na lista [C, D, F]. Sem ela, a lista se torna [C, F];
- Gene 4: [0] mapeia a cidade C na lista [C, F]. Sem ela, a lista se torna [F].

Como [F] foi a única cidade que tais genes não escolheram, ela será visitada por último. Com isso, o indivíduo com genes [3, 0, 1, 0] traz o percurso A -> E -> B -> D -> C -> F -> A.

O percurso inicial terá sempre genes com expressividade zero. No exemplo fornecido, o caminho inicial (trazido por [0, 0, 0, 0]) de todos os indivíduos seria A -> B -> C -> D -> E -> F -> A.

Função de fitness

A função de fitness aqui calcula a distância percorrida pelo caixeiro no trajeto completo trazido pelo indivíduo, considerando as distâncias percorridas após se utilizar o algoritmo de Dijkstra. Quanto menor a distância percorrida, melhor. Note que não se tenta verificar se tal distância é a menor - minimizá-la se torna um traço evolutivo do AG.

Para isso, os genes foram organizados em cada indivíduo de acordo com as seguintes considerações:

(1) O grafo a ser analisado pelo AG é conexo e não-direcionado;

(2) Escolhendo-se ir da cidade A à cidade B, a distância percorrida a partir desta decisão será sempre a menor possível (mesmo que seja necessário passar por outras cidades);

(3) Por decisão de um indivíduo, uma cidade não pode ser visitada mais de uma vez (não se impede, no entanto, que uma cidade no caminho de duas outras seja atravessada em nome de um atalho, por exemplo).

Para atender ao segundo ponto, uma etapa anterior à da execução do AG foi a de trazer as menores distâncias entre quaisquer dois nós. Para isso, executou-se o algoritmo de Dijkstra (2) em cada um dos nós, de modo a se obter tais distâncias.

Para garantir que cada gene deste indivíduo seja independente, os genes funcionarão de um jeito diferente dos demais problemas. Digamos, por exemplo, que um caixeiro em

A precise passar pelas cidades [B, C, D, E, F] e voltar à cidade A. O indivíduo de tal problema teria então quatro genes (o número de cidades que ele precisa visitar além da sua cidade de origem, menos um) que funcionam da seguinte forma:

- O primeiro gene possui expressividade de 0 a 4;
- O segundo gene possui expressividade de 0 a 3;
- O terceiro gene possui expressividade de 0 a 2;
- O quarto gene possui expressividade de 0 a 1.

Digamos que um dos indivíduos do AG tenha, pela expressividade de seus genes, os valores [3, 0, 1, 0]. Para se calcular o percurso feito por tal indivíduo, escolhe-se a cidade da lista naquela posição, a qual é removida antes de se escolher a próxima cidade. Ou seja:

- Gene 1: [3] mapeia a cidade E na lista [B, C, D, E, F]. Sem ela, a lista se torna [B, C, D, F];
- Gene 2: [0] mapeia a cidade B na lista [B, C, D, F]. Sem ela, a lista se torna [C, D, F];
- Gene 3: [1] mapeia a cidade D na lista [C, D, F]. Sem ela, a lista se torna [C, F];
- Gene 4: [0] mapeia a cidade C na lista [C, F]. Sem ela, a lista se torna [F].

Como [F] foi a única cidade que tais genes não escolheram, ela será visitada por último. Com isso, o indivíduo com genes [3, 0, 1, 0] traz o percurso A -> E -> B -> D -> C -> F -> A.

Para este trabalho, se, digamos, B for um atalho entre A e E, B ainda seria percorrida duas vezes (ou seja, o percurso com tal atalho seria A -> B -> E -> B -> D -> C -> F -> A). Isso não chega a desrespeitar a ideia do terceiro ponto, pois B não é visitada uma segunda vez (por decisão do indivíduo) no percurso A -> E.

Com isso, é garantido que todas as cidades serão visitados pelo caixeiro ao menos uma vez, é considerado retornar à cidade original, e todos os percursos são feitos do modo mais rápido.

Algoritmo de Dijkstra

A função de fitness aqui calcula a distância percorrida pelo caixeiro no trajeto completo trazido pelo indivíduo, considerando as distâncias percorridas após se utilizar o algoritmo de Dijkstra. Quanto menor a distância percorrida, melhor. Note que não se tenta verificar se tal distância é a menor - minimizá-la se torna um traço evolutivo do AG.

3 Algoritmo Genético

O pseudocódigo tradicional de um algoritmo evolutivo pode ser dado por (6):

Algoritmo 1: Pseudocódigo do Algoritmo Genético.

```
begin
    P ← InicializarPopulacao(fitness);
    P.fitness();
    t = 0;
    while t < número de gerações do
        Pais ← Selecao(P);
        Filhos ← Crossover(Pais);
        P ← P ∪ Filhos;
        Mutacao(P);
        P.fitness();
        P ← Sobrevivem(P);
    end
end
```

Cada uma destas funções principais (todas exceto a função de fitness, que é uma das entradas do problemas) pode ser feita de diferentes maneiras. No entanto, tais implementações independem do problema a ser resolvido, o que torna sua confecção e manutenção bem mais simples.

3.1 Seleção

A escolha dos pais é feita por.

3.2 Crossover

Escolhidos os pais, optou-se neste trabalho por embaralhá-los aleatoriamente numa lista e, escolhendo-os dois a dois, verificar se os dois pais serão cruzados, de acordo com uma probabilidade p_c. Se sim, os pais terão seus materiais genéticos misturados por uma ação chamada *crossover*.

Crossover envolve a quebra da sequência de genes de dois indivíduos em um mesma secção, com a subsequente troca de material na região delimitada por esta secção. Este trabalho se utilizou do crossover em dois pontos, o qual divide as sequências em duas partes distintas, ocorrendo troca do material genético entre estas partes. Tal ação é mostrada na figura 1.

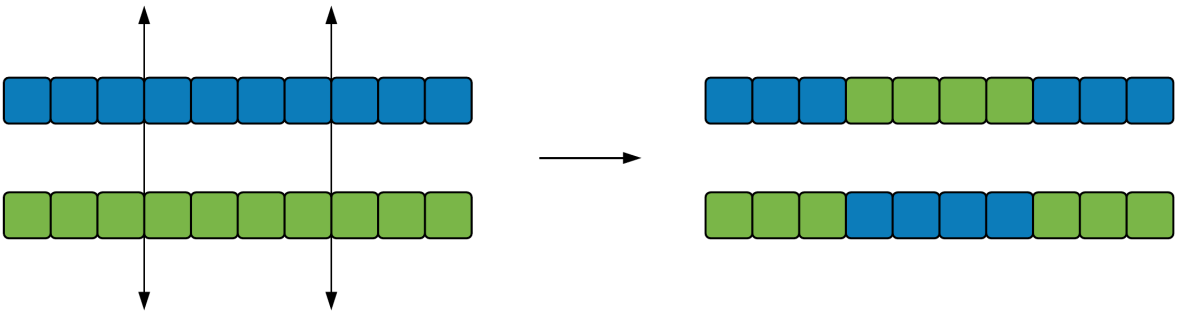


Figura 2: Ação de crossover em dois pontos.

Toda ação de crossover gera duas sequências de genes (filhas) novas a partir das sequências originais (pais), adicionando dois indivíduos novos a cada crossover bem sucedido.

3.3 Mutação

A ação de mutação funciona, em sua estrutura mais básica, da seguinte forma: itera-se sobre todos os genes da população um a um. Com uma probabilidade p_m, é avaliado se o gene deveria ou não alterar sua expressividade. Se sim, o gene tem seu valor alterado aleatoriamente.

3.4 Sobrevivência

Texto.

4 *Otimização de Mutação e Recombinação*

Texto aqui.

5 Validação com Outros Trabalhos

Texto aqui.

6 Conclusões e Trabalhos Futuros

Texto aqui.

Referências

1 D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: a computational study*. Princeton university press, 2011.

2 E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

3 A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

4 A. E. Eiben and J. E. Smith. *Introduction To Evolutionary Computing*, volume 53. Springer-Verlag, 2003.

5 P. Giguere and D. E. Goldberg. Population sizing for optimum sampling with genetic algorithms: A case study of the onemax problem. *Genetic Programming*, 98:496–503, 1998.

6 ICMC-USP. Algoritmos genéticos. `http://conteudo.icmc.usp.br/pessoas/andre/research/genetic/`. [Online; acessado em 06 de novembro de 2016].

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. REGISTRO N°	4. N° DE PÁGINAS
TC			15

5. TÍTULO E SUBTÍTULO:

Resolução Otimizada de Problemas com uso de Algoritmos Evolutivos

6. AUTOR(ES):

Cássio dos Santos Sousa

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica - ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Algoritmo Evolutivo, Inteligência Artificial.

9.PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Solicite preenchimento dos campos 2, 3 e 9 – envie este formulário para doc.pt@ita.br

10. APRESENTAÇÃO: **X Nacional** **Internacional**

ITA, São José dos Campos. Curso de Graduação em Engenharia de Computação. Orientador: Carlos Henrique Quartucci Forster. Publicado em 2016.

11. RESUMO:

Insira algum texto interessante aqui.

12. GRAU DE SIGILO:

(X) OSTENSIVO **( ) RESERVADO** **( ) SECRETO**