Algoritimo genético para resoler o problema do caixeiro viajante com o problemas de mochila para cálculo de rotas ótimas para robôs

Killdary A. Santana[1]

[1]Departamento de Engengharia Elétrica e da Computação
Universidade Federal do Ceará (UFC) – Campus Sobral
Caixa Postal 62010-560 – Sobral – CE – Brazil

killdary.aguiar@gmail.com

Abstract. *Route planning is becoming a daily task for activities that require the help of robots, such as robotic manipulators on assembly lines and transport robots in industries. The great challenge of these robots is to run a better route within their energy limitations. This paper presents the study and development of an application for the planning of beasted routes in the Traveling Salasmen Problem (TSP) combined with the Knapsack Problem (KP) using genetic algorithm (GA).*

Resumo. *O planejamento de rotas está se tornando a cada dia uma tarefa corriqueira para atividades que necessitem do auxílio de robôs, como manipuladores robóticos em linhas de montagem e robôs de transporte em indústrias. O grande desafio desses robôs é executar uma melhor rota dentro das suas limitações energéticas. Este artigo apresenta o estudo e desenvolvimento de uma aplicação para o planejamento de rotas baseados no Problema do Caixeiro Viajante(PVC) combinado ao Problema da Mochila(PM) por meio de algorítmo genético(AG).*

1. Introdução

Com o avanço da tecnologia os robôs se tornaram mais eficazes na execução de diversas tarefas, podendo executar funções que antes eram consideradas impossíveis para uma máquina, deixando de ser um componente caro e acessível apenas para grandes indústrias e passando a atuar nas mais diversas aplicações que antes eram executadas por humanos, impulsionando assim o desenvolvimento de robôs autônomos.

A finalidade da pesquisa de robôs móveis com autonomia é construir máquinas para realizar tarefas com precisão e capazes de tomar decisões adequadas frente a uma situação inesperada.

Muitos projetos podem ser observados com as mais diversas finalidades, como exploração de ambientes não estruturados como florestas[Vicente Santos et al. 2017], exploração de planetas como marte[NASA 2017], futebol de robôs[ROBOCUP 2017], carros autônomos[Google 2019], aplicações agrícolas[Tabile et al. 2011], entre muitos outros.

Um dos maiores desafios encontrados em robôs móveis se dá no planejamento de sua trajetória e tem recebido grande atenção por parte dos pesquisadores, tanto na

indústria como no meio acadêmico, pois seu desenvolvimento está diretamente relacionado com a maior autonomia dos robôs. A complexidade do problema de planejamento do movimento tem motivado o desenvolvimento dos mais diversos algoritmos. Tal complexidade advém da necessidade de integrar a navegação do robô com o sensoriamento, a eficiência e o planejamento de rotas, como também poupar recurso importantes e preservação de suas partes mecânicas. A habilidade de planejamento de rotas deve estar presente na programação interna dos robôs para que os mesmos possam decidir quando será necessário alterar sua rota de modo a minimizar o impacto em compração ao planejamento inicial.

O problema de roteirização de robôs pode ser modelado através do Problema do Caixeiro Viajante(PCV), porém o PVC não é suficiente para modelar a roteirização de rotas dentro de limitações do robô, como limitação energética ou de peso máximo a ser transportado, sendo assim o mesmo deve ser combinado ao Problema da Mochila(PM) para uma modelagem mais próxima a realidade. Este trabalho tem como objetivo desenvolver um algorítmo genético que forneça a melhor rota para um robô maximizando os ganhos que o mesmo pode realizar dentro de uma limitação energética.

Na próxima seção será descrito o PCV e sua variação síncrona, o PM e a combinação do PCV e PM. Na seção 3 será abordado a fundamentação teórica do problema e da solução a ser implementada. A seção 4 abordará os métodos de implementação e suas metodologias. Na seção 5 será exposto os resultados obtidos oriundos da seção 4, enquanto a conclusão será apresentada na seção 6, e por fim na seção 7 os trabalhos futuros.

2. Problema do Caixeiro Viajante com o Problema da Mochila

Para uma melhor representação do problema de roteirização de robôs com restrições será combinado o PCV e o PM. A seguir será apresentado o PCV e sua variação síncrona, usada como base do algorítmo, o PM com sua variação do Problema da Mochila Binária, para adicionar a visitação de uma cidade apenas uma vez, e por fim a combinação dos dois problemas na formulação matemática final.

2.1. Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) tem sido muito utilizado no experimento de diversos métodos de otimização por ser, principalmente, um problema de fácil descrição e compreensão, mas de grande dificuldade de solução, uma vez que é NP-difícil(Non-Deterministic Polynomial time). O PCV determina que um vendedor tem N cidades no qual o mesmo deveria visitar todas as cidades, sem repetir nenhuma, e voltar para a cidade de partida de modo que o custo da viagem seja mínimo [Singh and Lodhi 2013]. O cálculo de rotas mínimas em robôs se assemelha ao PCV, sendo o robô o vendedor, as cidades os pontos objetivos que o robô deve visitar e as arestas a distância que o robô percorrerá. O PCV é um problema de otimização combinatória, no qual o número de soluções possíveis é representado na equação 1.

$$R(n) = (n-1)! \tag{1}$$

Onde:

- n: é o numero de cidades a serem visitadas;

2.1.1. Formulação Matemática Para o PCV

Seja o grafo $G(N, A)$ onde N representa o conjunto ($|N| = n$) e A o conjunto de arestas. Seja, uma matriz simétrica com custos ou distâncias mínimas entre os nós da rede considerando ainda que $c_{ij} = +\infty \forall i \in N$. A matriz $X[x_{ij}]$ é composta pelas variáveis de decisão do problema

$$x_{ij} = \begin{cases} 1 & \text{, se: o arco } a_{ij} \in \text{rota} \\ 0 & \text{, se: o arco } a_{ij} \notin \text{rota} \end{cases} \tag{2}$$

Desta forma, a formulação de Programação Linear Inteira para o problema[Golden et al. 1980] pode ser escrita como:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{3}$$

Sujeito a:

$$\sum_{i=1}^{n} x_{ij} \qquad j = 1, 2, \ldots, n \tag{4}$$

$$\sum_{j=1}^{n} x_{ij} \qquad i = 1, 2, \ldots, n \tag{5}$$

$$X = x_{ij} \in S \tag{6}$$

$$x_{ij} = 0 \qquad \text{ou} \qquad x_{ij} = 1 \tag{7}$$

Os dois primeiros grupos de restrições (4) e (5) garantem que exatamente um arco (i, j) tem origem cada nó i da rota e exatamente um arco(i, j) é direcionado para um nó j da rota. A penúltima restrição (6) contém um subconjunto S que pode ser qualquer conjunto de restrições que impeça a formação de sub-rotas. Estas restrições são chamadas restrições de quebra de sub-rotas e podem ser, entre outras:

$$S = \left\{ (x_{ij}) \geq 1, \text{para todo conjunto próprio não vazio } Q \text{ de } N \right\} \tag{8}$$

$$S = \left\{ (x_{ij}) \leq |R| - 1, \text{ para todo subconjunto não vazio } R \text{ de } \{ 2, ..., n \} \right\} \tag{9}$$

$$S = \left\{ (x_{ij}) : y_i - y_j + n x_{ij} \leq (n-1) \text{ , para } 2 \leq i \neq j \leq n \text{ para alguns números reais } y_i \right\} \tag{10}$$

O PVC pode ser classificado em simétrico, onde os custos de um caixeiro ir da cidade A para a cidade B e vice-versa são os mesmos, ou assimétrico, onde os custos de movimentação de uma cidade A para uma cidade B podem ser diferentes de ir da cidade B para a cidade A. Para este trabalho será focado no problema **simétrico**.

2.1.2. Problema do Caixeiro Viajante Simétrico

O PCVS possui a direção da arestas é irrelevante já que $c_{ij} = c_{ji}$. Uma vez que a direção não tem importância, é possível considerar que o grafo possui apenas um arco, sem direção, entre todos os pares de nós. Deste modo, $x_j \in 0, 1$ é a variável de decisão onde j percorre todos os arcos A do grafo e c_j é o custo de percorrer cada nó. Para encontrar um circuito no grafo, deve selecionar um subconjunto de arcos, no qual todos os nós estejam contidos. Assim, o problema pode ser formulado como:

$$R(n) = \frac{(n-1)!}{2} \tag{11}$$

de modo que o número de resposta é a metade da formulação (1). De modo semelhante ao assimétrico, os sub-circuitos devem ser eliminados. O problema pode então ser formulado como [de Almeida 2019]:

$$min \frac{1}{2} \sum_{j=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{12}$$

$$s.t. \sum_{k \in J(j)} x_k = 2 \ \text{ ,para todo } j = 1, ..., m \tag{13}$$

$$\sum_{j \in E(S)} x_j \leq |S| - 1 \ \text{ ,para todo } S \subset 1, ..., m \tag{14}$$

$$x_j = 0 \text{ ou } x_j = 1 \ \text{ ,para todo } j \varepsilon E \tag{15}$$

onde $J(j)$ é o conjunto de todos os arcos, não direcionados, ligados ao nó j e $E(S)$ é o subconjunto de todos os arcos que ligam as cidades em qualquer conjunto S não vazio que contenha todas as cidades[Wikipedia 2019].

2.2. Problema da Mochila (PM)

O PM é um problema de programação linear inteira, classificado como NP-hard. O PM consiste de uma mochila com capacidade limitada, dentro dessa mochila deve ser colocado uma lista de itens, cada um com seu peso e valor, de modo que seja agrupado um conjunto de itens na bolsa que sua soma de peso não ultrapassem a capacidade da mochila e a soma dos valores seja a maior possível[Martello and Toth 1990]. O cálculo da rota de um robô se assemelha ao PM, sendo o robô a mochila, a capacidade da mochila as restrições impostas ao robô, como peso máximo ou energia disponível, e os itens a serem colocados na mochilas são os pontos as serem visitados.

O PM possui diversas variações, porém um robô que tem sua rota com base no PCV só deverá passar por um ponto uma única vez, sendo assim a o PM binário é a representação dp PM que mais aderente ao problema.

2.2.1. PM binário e sua Formulação Matemática

A variação PM binária consiste de um conjunto de itens N, que é formado por uma lista de n itens j com valor p_j e peso w_j, cada item é de um tipo único, e o valor da capacidade da mochila sendo C. O objetivo é selecionar um subset de N no qual a soma dos pesos não exceda C e o valor seja o maior possível[Prof. Hans Kellerer 2004]. A lista de itens é composta por um vetor $Y[y_j]$, no qual

$$y_i = \begin{cases} 1 & \text{, se: o item } y_i \in \text{rota} \\ 0 & \text{, se: o item } y_i \notin \text{rota} \end{cases} \tag{16}$$

Desta forma, a formulação matemática para o problema[Prof. Hans Kellerer 2004][Martello and Toth 1990] pode ser descrita como: r eliminados. O problema pode então ser formulado como [de Almeida 2019]:

$$max \sum_{i=1}^{n} \sum_{i=1}^{n} p_i y_i \tag{17}$$

$$s.t. \sum_{i=1}^{n} w_i y_i \leq C \tag{18}$$

$$y_i \in \{0, 1\} \qquad i = 1, ..., m. \tag{19}$$

A restrição (18) garante que a capacidade da mochila não será ultrapassada. A restrição (19) só assume valor 1 ou 0. A equação (17) garante o maior valor possível na mochila.

2.3. Problema do Caxeiro Viajante Combinado ao Problema da Mocila (PCVPM)

Nesta seção, o objetivo deste artigo, a saber, o Problema do Caixeiro Viajante com Problema de Mochila (PCVPM). Pode-se dizer que esse problema é a junção do PCV Simétrico com o PM binário, de modo que, para formulá-lo, foi decidido que devem ser mantidas as principais características de cada um dos dois problemas componentes:

1. Cidades visitadas geram prêmios.
2. É necessário atender uma quantidade de cidades que proporcione a soma do custo entre cidades não ultrapasse o custo máximo estabelecido.

Obviamente, a primeira caracterísitca é relativa ao PM, enquanto a segunda é uma combinação entre PCV e PM, no qual o peso do item a ser posto na mochila foi substituído pelo custo de visitação entre as cidades. Observe que, dessa forma, nem todos as cidades precisam ser visitadas. O problema consiste em minimizar a soma dos custos de viagem, no qual sua rota é formada por clientes suficientes para percorrer uma distância menor ou igual a distância preestabelecida, maximizando os prêmios coletados a cada cliente visitado, de modo que as cidades inclusas na rota sejam visitadas uma única vez.

Mais precisamente, consideremos um grafo $G(N, A)$, onde N representa o conjunto ($|N| = n$) e A o conjunto de arestas. Seja, uma matriz simétrica com custos ou

distâncias mínimas entre os nós da rede considerando ainda que $c_{ij} = +\infty \forall i \in N$. As cidades x_i que compõe o conjunto N possuem um prêmio p_i. O custo máximo percorrido seja C .A matriz $X[x_{ij}]$ é composta pelas variáveis de decisão do problema:

$$x_{ij} = \begin{cases} 1 & \text{, se: o arco } x_{ij} \in \text{rota} \\ 0 & \text{, se: o arco } x_{ij} \notin \text{rota} \end{cases} \tag{20}$$

$$x_i = \begin{cases} 1 & \text{, se: a cidade } x_i \in \text{rota} \\ 0 & \text{, se: a cidade } x_i \notin \text{rota} \end{cases} \tag{21}$$

Desta forma, a formulação de Programação Linear Inteira para o problema pode ser escrita como:

$$max.f(x) = \alpha_P \sum_{i=0}^{n} p_i x_i - \beta_D \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{22}$$

sujeito as restrições:

$$\sum_{j=2}^{n} x_{1j} = 1 \tag{23}$$

$$\sum_{k=1}^{n-1} x_{in} = 1 \tag{24}$$

$$\sum_{i=1}^{n-1} x_{ni} = 0 \tag{25}$$

sobre a quantidade de visitações:

$$\sum_{i=1}^{n} x_{ij} \qquad j = 1, 2, \ldots, n \tag{26}$$

$$\sum_{j=1}^{n} x_{ij} \qquad i = 1, 2, \ldots, n \tag{27}$$

sobre a distância máxima a ser percorrida:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \leq C_T \tag{28}$$

sobre a aliminação de subrotas:

$$1 \leq u_i \leq n \qquad \forall i \in T \tag{29}$$

$$i_i - u_j + 1 \leq (1 + -x_{ij})n \qquad 2 \leq i \neq j \leq n \tag{30}$$

onde n é o tamanho do conjunto de cidades a serem visitadas, T é conjuntos de cidades que comõe uma rota viável, c_{ij} é o custode movimentação entre uma cidade e outra, p_i é o prêmio da i-ésima cidade, α_P e β_D são os pesos na fução objetivo relacionados a maximização da premiação e minimização do custo

Neste problema as equações (23) e (24) são restrições sobre o ponto inicial e final que devem ser inclusos na rota, a restição (25) impõe que nenhum ponto apos chegar ao ponto final seja visitado, (26) e (27) restringe que cada ponto seja visitado apenas uma única vez, a equação (28) limita a distância máxoma percorrida a C_T, subrotas são impedidas de ser geradas pelas equações (29) e (30).

O resultaddesta otimização será um conjunto de vertices ordenados a serem visitados pelo agente.

3. Algoritmo Genético

Os Algoritmos Genéticos (AGs) são técnicas de otimização inspirados no princípio da sobrevivência e reprodução dos indivíduos mais aptos, proposto por Charles Darwin[Coppin 2004][Goldberg 1989].

O Algoritmo Genético é baseado em um conjunto de soluções candidatas que representam uma solução para o problema de otimização que se deseja resolver. Uma solução é um candidato em potencial para otimizar o problema. A forma de representação desempenha um papel importante, uma vez que determina a escolha dos operadores genéticos.

As representações geralmente são listas de valores e por muitas vezes são mais baseadas em conjuntos de símbolos. Caso o conjunto de símbolos sejam contínuos, eles serão chamados de vetores, caso eles consistem em bits, eles serão chamadas de strings de bits. Em caso de problemas combinatórios, as soluções geralmente consistem em símbolos que aparecem em uma lista. A representação pode ser aplicado ao planejamento de rotas como um percurso realizado pelo caixeiro viajante. Os operadores genéticos produzem novas soluções a partir da representação escolhida e permite a aproximação da solução. A codificação da solução, no qual pode ser aplicado o processo evolutivo, é chamada de genótipo ou cromossomo.

O algoritmo 1 mostra o pseudocódigo do AG básico. No início, um conjunto de soluções, chamado de população, é inicializada. Esta inicialização é utilizada para gerar aleatoriamente o conjunto de soluções[Kramer 2017].

O principal ciclo geracional do Algoritmo Genético gera uma nova prole de soluções candidatas, através do cruzamento e da mutação, até a população estar completa.

3.1. Implementação

A ferramenta utilizada para realizar a implementação foi a linguagem $Python$. O Python ocupa o primeiro lugar no *ranking spectrun da IEEE*[CASS 2018] em linguagens de cálculos científicos, já possui funções nativas de cálculos semelhantes ao Matlab, porém as mesmas ainda se demonstraram insuficientes para adaptar com precisão de tempo o

Algoritmo 1: Algorítimo Genético

```
1  iniciar população; repeat
2      repeat
3          cruzamento;
4          mutação;
5          avaliação;
6          seleção;
7      until;
8      população completa;
9      atualização teste de Parada
10 until;
11 finalização
```

algoritmo escolhido, para resolver este problema foi utilizado o pacote Numpy, que é pacote cientifico da linguagem [Jones et al. 01].

Para atender às necessidades de um sistema baseado em AG's, tem-se como principais requisitos do sistema, a passagem de parâmetros: número de gerações, tamanho da população, tamanho de geração sem alteração no melhor indivíduo, peso máximoa ser percorrido, arquivo com pontos a serm visitados, arquivo com premiacoes de cada cidade e o ponto inicial. O programa foi criado projetado para ser executado através da linha de comando, para uma flexidade maior na mudanças dos parâmetros, aceitando a sintaxe:

Algoritmo 2: Execuçaõ em linha de comando do algorítmo

```
1  PCVPM_GA.py [size_generation] [size_population] [num_generation_limit]
     [max_coust] [towns_list] [weight_list] [begin_deposit]
```

A base do desenvolvimento deste artigo foi o algoritmo proposto por [Singh and Lodhi 2013],neste estudo é realizado a implementação de um algoritmo genético, no qual os resultados se demonstraram bastantes promissores. No estudo é utilizado um algoritmo em Matlab. Para o desenvolvimento embarcado a memória e o processamento são recursos preciosos, porém o Matlab exige, em seus requisitos mínimos, 2 Gb de RAM e 2,2 Gb de armazenamento, sendo que em plataformas embarcadas a memória RAM e a memória de armazenamento possuem capacidades inferiores ao exigido, desta forma não é possível utilizar o algoritmo original para os testes nas plataformas embarcadas. Para contornar este problema foi realizado uma adaptação do algorítimo para uma linguagem presentes nos sistemas citados, a linguagem escolhida foi o Python na sua versão 3.6 [Python 2019].

3.2. ARM SBCs

Um computador de placa única ARM (*single-board computer* - SBC) é um computador completo construído em uma única placa de circuito , com microprocessador ARM, memória, entrada/saída (E/S) e outros recursos necessários para um computador funcional. Os computadores de placa única foram feitos como sistemas de demonstração ou desenvolvimento, para sistemas educacionais ou para uso como controladores de

computador incorporados. Muitos tipos de computadores domésticos ou computadores portáteis integram todas as suas funções em uma única placa de circuito impresso. Estes dispositivos ganharam grande visibilidade em 2012 quando foi lançado pela *Raspberry Pi Foundation* o SBC *Raspberry Pi*, o intuito da fundação era disseminar o ensino de Ciência da Computação nas escolas, porém a placa demonstrou ser uma ótima alternativa para desenvolvimento dos mais diversos projetos na industria e na área acadêmica. Desde então diversos SBC's foram lançados para as mais diversas funcionalidades[Mitchell 2017].

A tabela 1 faz uma comparação das especificações das placas utilizadas neste estudo:

Tabela 1. Características BeagleBone Black

	BeagleBone Black	**Raspberry Pi 3**	**Raspberry Pi A+**	**Orange Pi One**
Processador	ARM Cortex A8	ARM Cortex-A53 64-bit	ARM 1176JZF-S	ARM H3 Quad-core
Clock	1 GHz	1 GHz	700 MHz	1 GHz
Ram	512 MB	1 GB	256 Mb	512 MB
Storage	2 GB on-board eMMC, MicroSD	MicroSD	MicroSD	MicroSD
Alimentação	210-460 mA @ 5V	800 mA @ 5V	200 mA @ 5V	300 mA @ 5V
Quantidade de GPIO	92	40	40	40
Preço	US$45.00	US$35.00	US$20.00	US$10.00

[Raspberry 2019][Orangepi 2017][Brown 2016][DiCola 2014]

Os dados acima foram levantados na data 2017-12-22.

4. Método de Implementação

A metodologia de desenvolvimento segue as seguintes etapas:

- Definição do ambiente de testes;
- Adaptação Algorítimo Genético para Python 3.6:
  Multiplataforma;
  Similaridade com Matlab;
  Vasta utilização em *data science* segundo a IEEE[CASS 2018];
- Portabilização para as plataformas;
- Critérios de execução:
  tempo limite;
  consumo de RAM;
  consumo de CPU;
- Ambiente físico de execução de testes;
- Interpretação dos testes.

Para realizar a comparação de desempenho proposto por esta pesquisa é fundamental que todas as placas sejam submetidas ao mesmo ambiente de teste, respeitando as especificações técnicas de cada plataforma. O ambiente de testes foi aplicado sobre um sistema operacional Linux, no qual para cada plataforma foi utilizado o sistema disponibilizado pelo seu respectivo fabricante, a lista abaixo exibe cada sistema utilizado:

- BeagleBobe Black: Debian 9.2 IoT[BeagleBoard 2017];
- Raspberry Pi 3: Raspbian Stretch with Desktop[Raspberry 2019];
- Raspberry Pi A+: Raspbian Stretch with Desktop[Raspberry 2019];
- Orange Pi One: Lubuntu [Orangepi 2017].

A base do desenvolvimento deste artigo foi o algoritmo proposto por [Singh and Lodhi 2013],neste estudo é realizado a implementação de um algoritmo genético, no qual os resultados se demonstraram bastantes promissores. No estudo é utilizado um algoritmo em Matlab. Para o desenvolvimento embarcado a memória e o processamento são recursos preciosos, porém o Matlab exige, em seus requisitos mínimos, 2 Gb de RAM e 2,2 Gb de armazenamento, sendo que em plataformas embarcadas a memória RAM e a memória de armazenamento possuem capacidades inferiores ao exigido, desta forma não é possível utilizar o algoritmo original para os testes nas plataformas embarcadas. Para contornar este problema foi realizado uma adaptação do algorítimo para uma linguagem presentes nos sistemas citados, a linguagem escolhida foi o Python na sua versão 3.6 [Python 2019].

O Python ocupa o primeiro lugar no *ranking spectrun da IEEE*[CASS 2018] em linguagens de cálculos científicos, já possui funções nativas de cálculos semelhantes ao Matlab, porém as mesmas ainda se demonstraram insuficientes para adaptar com precisão de tempo o algoritmo escolhido, para resolver este problema foi utilizado o pacote Numpy, que é pacote cientifico da linguagem [Jones et al. 01].

A análise das plataformas foi feita considerando os seguintes aspectos:

- tempo de execução: tempo utilizado pela placa para a execução do algoritimo;
- consumo de RAM: consumo de memória RAM realizado pelo GA;
- consumo de processamento: processamento consumido pelo GA.

A medida do tempo de execução foi realizada utilizando o módulo nativo do Python *timeit*,este módulo possui uma função chamada *repeat* no qual a definição encontradoa no *help()* da linguagem define:

> Esta é uma função conveniente que chama o *timeit()* repetidamente, retornando uma lista de resultados. O primeiro argumento *repeat* especifica quantas vezes deverá ser executado chamado o *timeit()*, o padrão é igual à 3; o segundo argumento especifica o *number* que define quantos testes são feitos em cada repetição,o valor padrão é igual 1000000.

Para a medida de uso de memória e de CPU foi utilizando o comando nativo do Linux *top* , este comando oferece um conjunto de estatísticas sobre o estado geral, assim como dos processos com maior atividade no sistema. Essa lista é apresentada e atualizada de 2 em 2 segundos. Depois de captar os dados de cada plataforma é realizada a comparação entre elas.

O ambiente de testes para cada plataforma foi realizada de maneira que simulasse os mesmos aspectos para todas as plataformas. O ambiente de testes possui as características:

- temperatura de 23 graus Celsius controlada por um ar-condicionado;
- versão do Python 3.6;
- versao do Numpy versão 1.13.3;
- parâmetros do algoritmo genético:
  tempo de execução: 60 minutos;
  população: 50;
  gerações: 100;
  a matriz de distâncias entre os pontos sempre o mesmo;
- parâmetros de execução:
  número de repetições: 10;
  número de execuções por repetição: 10;

Por fim a interpretação dos testes.

5. Resultados

Os experimentos são conduzidos para avaliar o desempenho de cada plataforma. Os testes em todas as plataformas foram executados sempre no mesmo ambiente sobe as mesmas condições físicas e de software. Para cada teste foi definido um tempo máximo de execução de 60 minutos, este tempo foi baseado no tempo médio de execução do algoritmo em uma plataforma não embarcada, notebook com processador core i7 de 1.8 GHz e 8 Gigabytes de memória RAM, definindo o mesmo número de repetições e execuções, 10 repetições e 10 execuções, o tempo de execução no notebook foi de 30 minutos, já nos testes das plataformas embarcadas este tempo foi dobrado para compensar sua baixa memória e seu baixo poder de processamento e executando o mesmo número de repetições e de execuções.

Ao iniciar os testes todas as plataformas concluiram a análise dentro do tem máximo definifo, com excexão do placa Raspberry Pi A+, a mesma não conseguiu executar o teste em tempo hábil, desta forma os testes foram abostrados e seus dados nos gráficos seguintes irão ser exibidos com valores zerados.

O foco do experimento não baseia-se apenas no tempo de execução do algoritmo mas também no uso de CPU e memória RAM.

5.1. Tempo de Execução

O tempo de execução de cada plataforma é definido pela média resultados retornados pela função *timeit.repeat()*.

5.2. Memória RAM

O consumo de RAM de cada plataforma é definido pela média resultados captados apartir da função do Linux *top*.

5.3. CPU

O uso de CPU de cada plataforma é definido pela média de resultados captados apartir da função do Linux *top*.

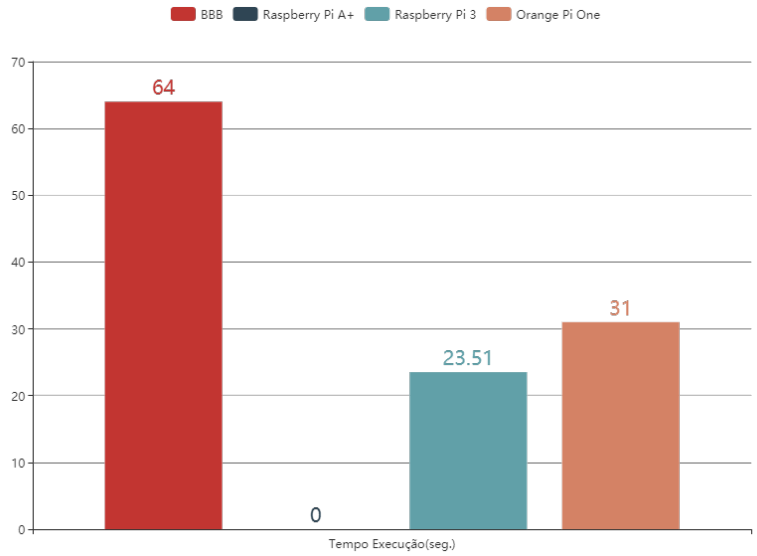

Fig. 1. Tempo médio de execução

5.4. Tabela de Comparação de desempenho

Tabela 2. Características BeagleBone Black

	BeagleBone Black	**Raspberry Pi 3**	**Raspberry Pi A+**	**Orange Pi One**
Tempo Execução (seg.)	64	23.51	0	31
RAM (Mb)	17,48	16	0	17,48
CPU (MHz)	980	1000	0	1000

6. Conclusão

Os testes demonstraram que o Raspberry Pi 3 possui uma melhor performance na execução do AG proposto, seguido do Orange Pi One e BeagleBone Black. O Raspberry Pi A+ demonstrou não ser uma boa opção para a execução do algorítimo.

Apesar do Raspberry Pi 3 possuir um melhor performance o Orange Pi One demonstrou ser uma ótima escolha considerando que seu desempenho aproximou-se do Raspberry Pi 3 e seu custo ser menor, ver tabela 1, sendo assim a melhor plataforma pode ser definida tendo em consideração seu uso final.

7. Trabalhos Futuros

Temdo em comparação dos resultados e de performance, os seguintes resultados devem persuadir trabalhos futuros. A comparação de performance e de custo não são as únicas métricas que devem ser levadas em consideração na escolha da platforma, a eficiência enérgética deve ser levada em conta, no qual poderá ser realizada em pesquisas futuras.

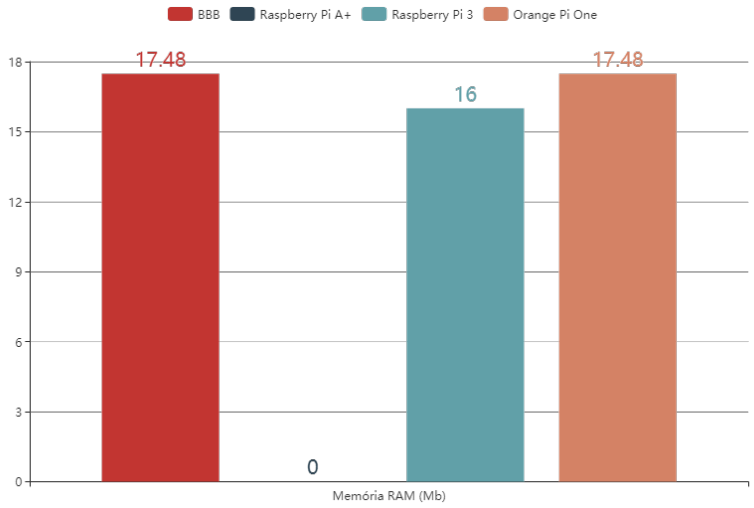

Fig. 2. Consumo médio de memória RAM

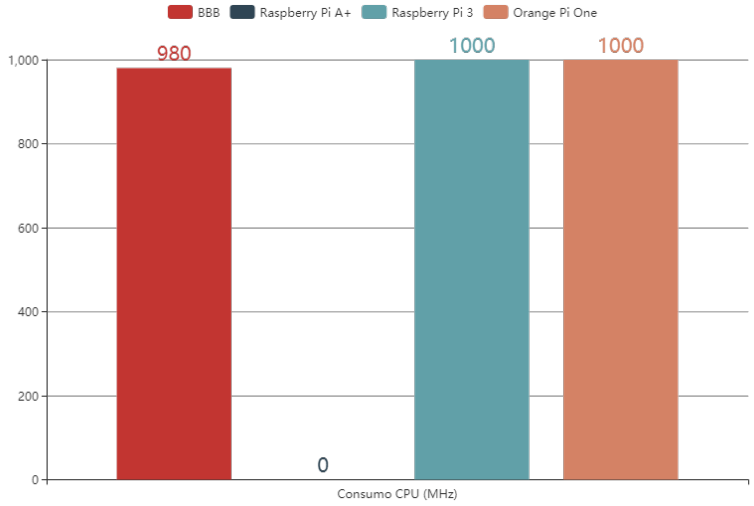

Fig. 3. Uso médio de memória CPU

References

BeagleBoard (2017). Beagleboneblack.

Brown, E. (2016). $10 orange pi one pits quad-core cortex-a7 against pi zero.

CASS, S. (2018). The 2018 top programming languages.

Coppin, B. (2004). *Artificial Intelligence Illuminated*. Jones and Bartlett Publishers, 1st ed edition.

de Almeida, T. A. (2019). Método do subgradiente para a resolução da relaxação lagrangeana do problema do caixeiro viajante simétrico.

DiCola, T. (2014). Power usage.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1 edition.

Golden, B., Bodin, L., Doyle, T., and W. Stewart, J. (1980). Approximate traveling salesman algorithms. *Operations Research*, 28(3-part-ii):694–711.

Google (2019). Just press go: designing a self-driving vehicle.

Jones, E., Oliphant, T., Peterson, P., et al. (2001–). SciPy: Open source scientific tools for Python. [Online; accessed 2017-12-21].

Kramer, O. (2017). *Genetic Algorithm Essentials*. Springer.

Martello, S. and Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA.

Mitchell, R. (2017). Battle of the sbcs: Beaglebone black, raspberry pi 3, and asus tinker board.

NASA (2017). Curiosity overview.

Orangepi (2017). orange pi one-orangepi.

Prof. Hans Kellerer, Prof. Ulrich Pferschy, P. D. P. a. (2004). *Knapsack Problems*. Springer-Verlag Berlin Heidelberg, 1 edition.

Python (2019). Python.

Raspberry (2019). Raspberry pi - teach, learn, and make with raspberry pi.

ROBOCUP (2017). Robocup about.

Singh, S. and Lodhi, E. A. (2013). Study of variation in tsp using genetic algorithm and its operator comparison.

Tabile, R. A., Godoy, E. P., Pereira, R. R. D., Tangerino, G. T., Porto, A. J. V., and Inamasu, R. Y. (2011). Design and development of the architecture of an agricultural mobile robot. *Engenharia Agrícola*, 31:130 – 142.

Vicente Santos, A., De Góes, E., Miranda, F., Freitas, G., Almir De Sena, J., Robinson, N., Dos Reis, S., Eduardo, P., Panta, G., Carvalho Ferreira, R., and Curty Cerqueira, R. (2017). Robô ambiental híbrido: um novo conceito em locomoção e monitoramento de áreas inóspitas na floresta amazônica.

Wikipedia (2019). Wikipedia.