Instituto de Ciências Matemáticas e de Computação - USP
Departamento de Sistemas de Computação
SSC-0713 - Sistemas Evolutivos Aplicados à Robótica
Prof. Alexandre Cláudio Botazzo Delbem

Relatório do Trabalho Prático de Algoritmo Genético

Gabriel Arantes de Lima e Silva 9266372
Guilherme Malacrida Alves 9311639

Dezembro - 2016

1.Introdução

A computação evolutiva (CE) se trata de um conjunto de técnicas que permite analisar um problema e encontrar soluções a partir do princípio de sobrevivência do mais adaptado da biologia. Desta forma, buscamos criar indivíduos e selecioná-los a partir de condições preestabelecidas, maximizar uma função por exemplo, e a partir disso podemos avaliar as influências dos diversos parâmetros que influenciam a “adaptabilidade” ou como chamamos o fitness de um indivíduo, além de encontrar os parâmetros que fornecem um ponto de melhor fitness local.
Um dos tipos mais conhecidos de algoritmos evolutivos são os chamados algoritmos genéticos (AG). A principal ideia por trás dos AGs é que considerar cada parâmetro como um gene de um DNA, então durante a criação de uma nova geração de indivíduos esses genes passaram por processos de permutação (crossover) e de variações randômicas (mutação). Desta forma ao selecionar indivíduos de uma geração para criar uma nova surge uma certa variedade no espaço de busca que permite explorar regiões ainda inexploradas enquanto converge para pontos de máximo local (ou mínimo dependendo do problema).

1.1 Descrição Geral do Problema Proposto

O problema que propomos resolver é uma aplicação direta do problema do caixeiro viajante com um veículo de tração diferencial em duas rodas que navega pelo campo utilizando a técnica conhecida como carrot chaser. Desta forma pretendemos otimizar a maneira com que o veículo percorre todos os pontos predeterminados para que o faça no menor tempo possível.

1.1.1 O problema do caixeiro viajante

O problema original do caixeiro viajante consiste em escolher a melhor rota que um caixeiro viajante pode percorrer para visitar um conjunto de cidades dado, sendo que o caixeiro não deve passar pela mesma cidade duas vezes, salvo a cidade inicial, que é também o ponto final do percurso. Normalmente as cidades estão conectadas em um grafo que limita os possíveis trajetos, mas neste trabalho assume-se que é possível ir de uma cidade para qualquer outra.

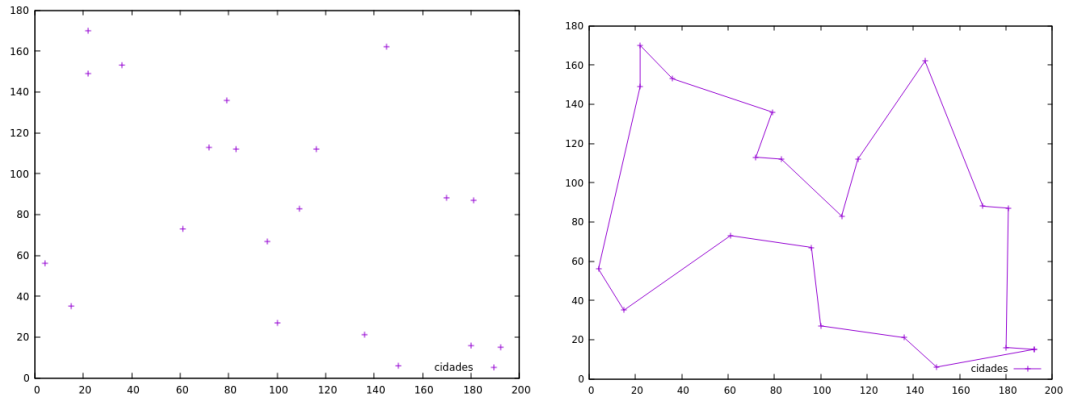

Figura 1.1: cidades e rota escolhida em um exemplo de problema do caixeiro viajante

1.1.2 Carrot Chaser

O carrot chaser (carrot following ou follow-the-carrot) é um método de navegação que permite que um veículo percorra uma linha entre dois waypoints de maneira suave com qualquer ponto de partida. A ideia básica por trás desse método está contida na figura abaixo. Encontrar a posição relativa e a orientação do veículo e então projetar essa posição na direção da linha entre os dois waypoints e deslocá-la de uma distância delta na direção do próximo waypoint, essa é a coordenada do carrot. A partir da posição do carrot define-se a direção que o veículo deve seguir. Comparando a direção que deve seguir com a orientação temos um fator que define a quantia de rotação necessária para o veículo.

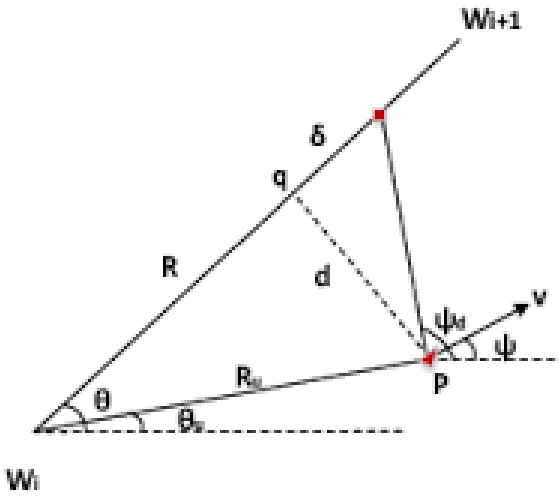

Figura 1.2: Ideia básica por trás do metodo carrot chaser

Desta forma é possível percorrer um caminho suave entre dois waypoints de uma maneira que evita acúmulo de erros.

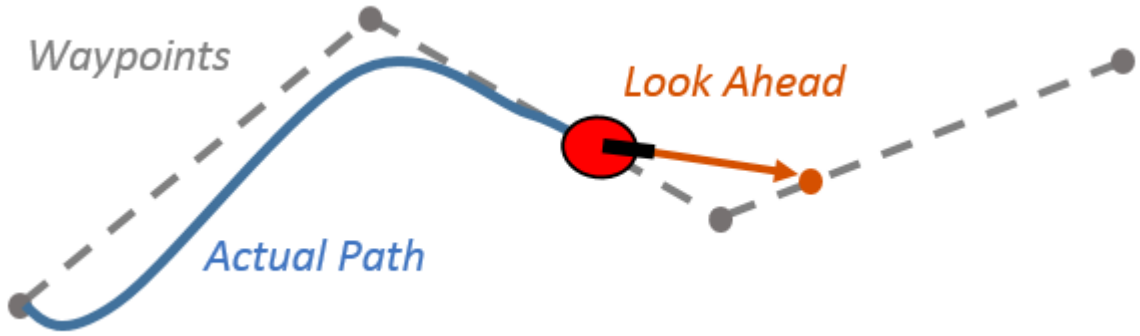

Figura 1.3: caminho percorrido com o carrot chaser ao seguir um conjunto de waypoints

O caminho resultante deste método varia principalmente com os parâmetros da velocidade do veículo e a distância delta usada no cálculo do carrot, podendo gerar caminhos completamente distintos.

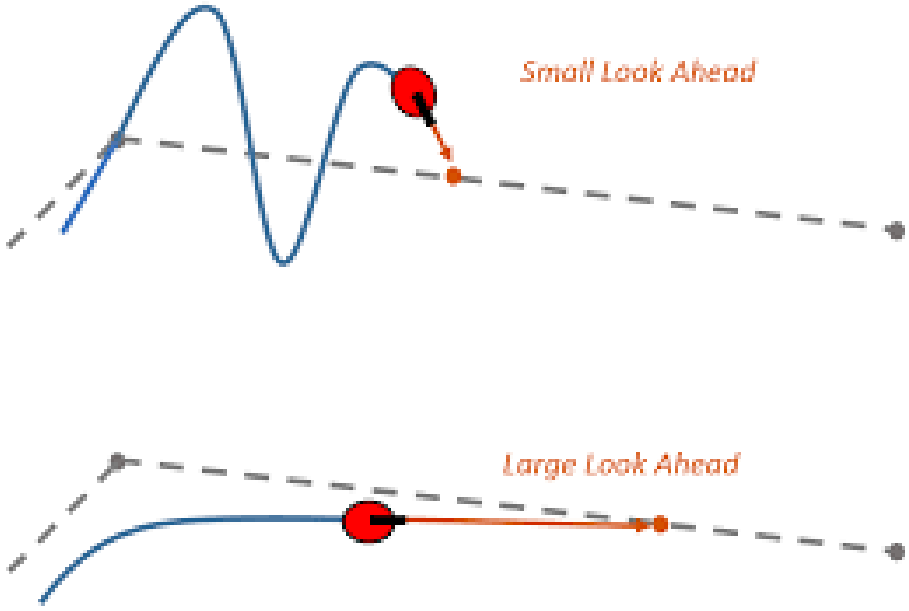

Figura 1.4: Efeito do parâmetro delta no caminho definido pelo carrot chaser.

1.2 O simulador

O simulador utilizado neste trabalho foi o “Gazebo 7” (link do site: http://gazebosim.org/). Ele é um simulador tridimensional que permite testar o comportamento de robôs através de sua “physics engine” com o auxílio de plugins (escritos em C++) que permitem controlar os elementos constituintes do robô, como suas juntas. Além de apresentar uma interface gráfica através da qual podemos visualizar e interagir com a simulação.

Para este projeto foram construídos através de arquivos SDF um ambiente de simulação (world) e um veículo (model), com os quais interagem o AG e os plugins de controle. O modelo é um veículo com duas rodas na parte frontal e um “caster wheel” na parte traseira, as curvas são feitas ao impor velocidades diferentes a cada roda. O ambiente por sua vez é constituído de um plano base sobre o qual se encontra o veículo e uma fonte de luz que ilumina o cenário.

2. Implementação

A implementação deste problema foi feita na linguagem C++ em três partes, um executável que contém o AG, um plugin que está ligado ao ambiente de simulação e um plugin que está ligado ao modelo. Cada um desses elementos atua de maneira distinta para que possamos avaliar e selecionar os melhores indivíduos.

2.1 AG

O AG atua na tentativa de otimizar essencialmente três parâmetros: a ordem dos waypoints (cidades), fixadas a velocidade e a distância do carrot (delta). Essa otimização busca fazer com que o robô percorra seu trajeto o mais rápido possível. Para gerar uma variabilidade de indivíduos que permitam explorar o espaço de busca foram utilizados os métodos de crossover e mutação que são descritos a seguir.

2.1.1 Crossover dos waypoints (cidades)

Para o crossover dos waypoints é utilizado um algoritmo de permutação chamado de crossover de ordem no qual são selecionados dois pais e sorteadas duas posições no vetor das cidades e caso a primeira posição seja anterior a segunda copiamos para o filho os genes do pai 1 entre as duas posições na ordem e local em que aparecem, caso contrário copiamos os genes de fora desse intervalo. Então completa-se os espaços remanescentes com os genes do pai 2 que não foram utilizados, na ordem em que aparecem no pai 2. O procedimento fica mais visível com um exemplo.

- Pai 1 = C E D B F A ;
- Pai 2 = E B A C D F;

Sorteamos então a 2° e a 5° posições, logo copiamos os genes do pai 1 no intervalo selecionado.

- Filho = X X D B X X

Por fim completamos com os genes não usados na ordem em que aparecem no pai 2.

- Filho = E A D B C F

A operação de crossover é aplicada durante a geração de todos os indivíduos (exceto os da geração original).

2.1.2 Mutação na ordem dos waypoints

O procedimento de mutação na ordem dos waypoints consiste em primeiramente sortear para cada posição do vetor um número aleatório, normalizá-lo e comparar seu valor com a taxa de mutação. Caso o valor sorteado seja menor que a taxa de mutação então sorteia-se uma segunda posição do vetor de waypoints e inverte-se os valores dessas duas posições.

2.1.3 Cálculo do fitness

O cálculo do fitness é feito ao se tomar o inverso do tempo decorrido durante o percurso realizado pelo veículo, ou seja, para calcular o fitness realizamos uma simulação em que um veículo com as características do indivíduo e medimos o tempo que este leva para percorrer todo o trajeto. Desta forma o melhor indivíduo será aquele com o maior fitness.

2.1.4 Geração de uma nova população

A formação de novos indivíduos para a geração seguinte é feita um a um. Para gerar cada novo indivíduo são realizados dois torneios, os dois vencedores então se tornam os pais, na ordem em que são selecionados. Aplica-se em seguida o algoritmo de crossover para gerar um filho que então passa pela mutação para gerar uma maior diversidade. Esse processo é repetido até que tenhamos o número desejado de indivíduos.

2.2 Interação entre o AG e o Simulador

O AG e o simulador comunicam-se através de troca de mensagens por socket TCP/IP, utilizando Google protobufs. Esse método baseia-se no estabelecimento de *topics*, canais de comunicação nos quais mensagens podem ser publicadas (por publishers) e lidas (por subscribers). O algoritmo assume o papel de ambos em diferentes *topics*, assim como o plugin do model.

Cada robô recebe, do AG, os seus genes, que representam a ordem dos waypoint a serem percorridos. Com a simulação iniciada, o robô realiza o percurso, enquanto escuta a um topic nativo do Gazebo, *World_Statistics*. Desse modo, é possível para o plugin adquirir o tempo total de percurso, e então publicar essa informação no *topic* Evolve. O AG, que permanece em repouso até que algo seja publicado em Evolve, utiliza a informação de fitness para o robô em questão.

O processo é repetido até que cada indivìduo da população possua seu fitness. O algoritmo genético aplica às relações de crossover e mutação, para gerar uma nova população. Assim, se dá a iteração da primeira geração. Desse modo, realizando o mesmo processo por um número definido de gerações, espera-se que ocorra uma convergência para algum gene que forneça uma solução ótima local para o problema.

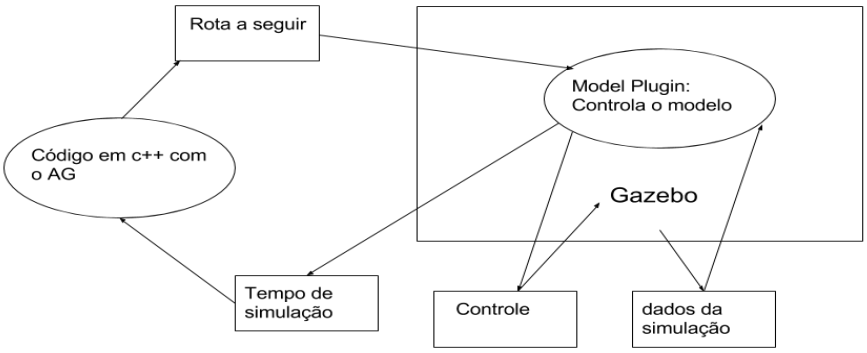

Figura 2.2.1: Diagrama de comunicação AG/Simulador.

3.Resultados

Até o momento os resultados foram apenas no teste isolado do AG sem o uso do simulador para o cálculo do fitness. Neste caso o fitness foi calculado com base na distância entre os pontos, media-se a distância em linha reta entre os pontos e ao somar as distâncias em linha reta de todos os pontos e tomar seu inverso era possível obter o fitness. Desta forma foi possível obter os resultados abaixo para um conjunto de pontos sorteados randomicamente.

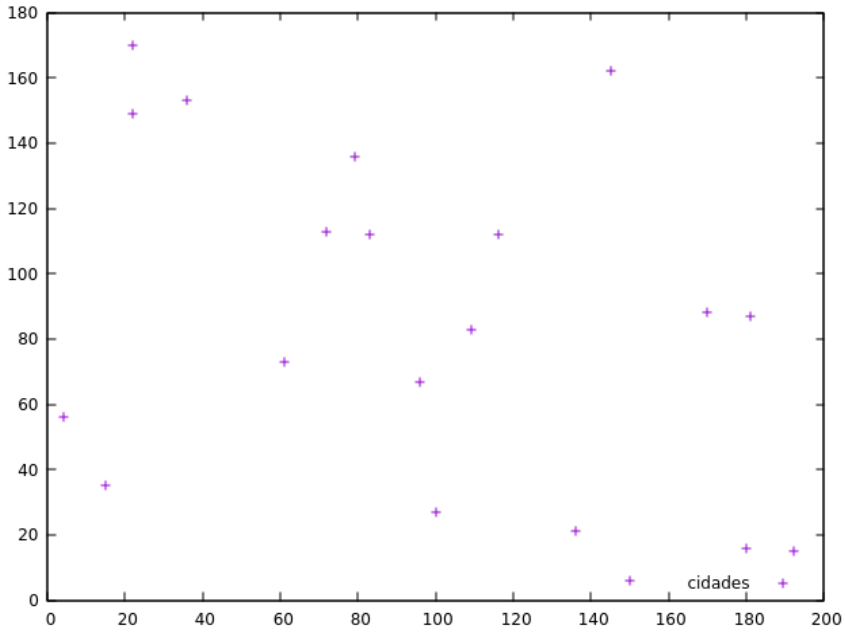

Figura 3.1: Conjunto de waypoints sorteados

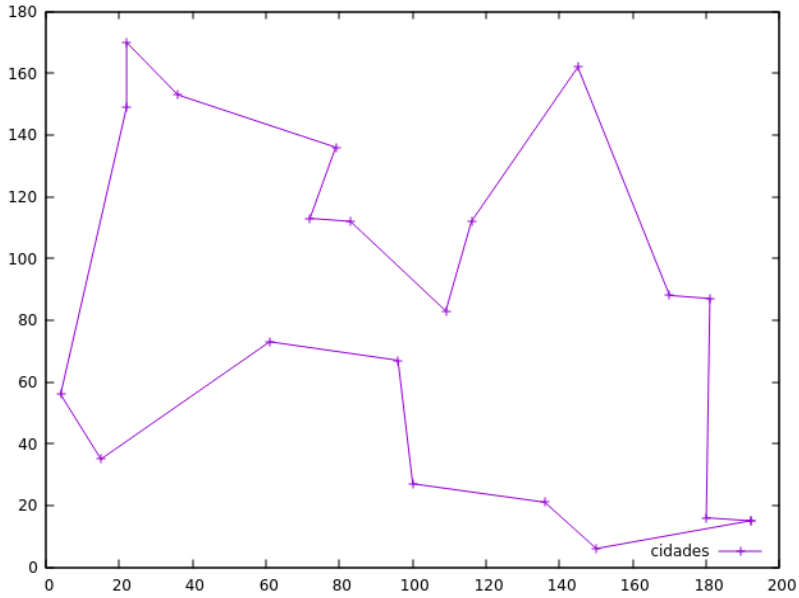

Figura 3.2: Caminho escolhido pelo AG por apresentar a menor distância

Um ponto importante que podemos observar é que, com o acréscimos de mais waypoints, a complexidade do problema se eleva muito. Não só ocorre um aumento na quantidade de operações que devem ser realizadas sobre os vetores de genes, que são custosas, mas também o universo a ser estudado cresce exponencialmente. Isso aumenta a dificuldade em explorar adequadamente o espaço de busca para encontrar a melhor solução fazendo que o algoritmo estagne em um ponto de máximo local, uma maneira de solucionar isso seria aumentar a população, mas isso torna o procedimento ainda mais custoso e demorado, principalmente quando levamos em conta o elevado custo computacional do algoritmo de crossover que é aplicado a todos os indivíduos. Uma população muito grande também é um problema, pois após um determinado tamanho a influência na exploração do espaço de busca passa a ser muito pequena se comparada ao aumento do processamento, mesmo para um número reduzido de cidades onde o custo das operações é reduzido.

4.Conclusão

O uso de um algoritmo evolutivo provou-se eficiente na resolução do problema do caixeiro viajante. Até onde o projeto pôde avançar, foi possível aplicar a teoria aprendida em sala para elaborar um programa que fornecesse, ao final de sua execução, o melhor caminho para percorrer uma série de waypoints.

Isso possui diversas aplicações práticas. Tópicos como o estudo de grafos e otimização de processos comumente utilizam algoritmos evolutivos, em função de sua implementação relativamente simples e capacidade de gerar respostas satisfatórias.

A ideia original por trás do projeto FieldWalker era de um robô que deve percorrer um determinado caminho, indo a locais especìficos para realizar uma tarefa. Com o auxílio do simulador, seria possível não só determinar a melhor ordem para realizar esses caminhos, como também a melhor velocidade e parâmetro de carrot para cada trecho. Infelizmente, não foi possível realizar essa parte, uma vez que o aprendizado para utilizar o Gazebo tomou um tempo muito longo.

O simulador é uma ferramenta robusta, e que possui excelente integração com controladores como o ROS. Entretanto, possui uma interface pouco intuitiva, o que atrapalha no desenvolvimento de projetos por aqueles que não estejam devidamente acostumados com o ambiente de desenvolvimento. Dessa forma, recomendamos que o Gazebo não seja utilizado por alunos sem experiência prévia com simuladores.