Metaheurística para otimização de parâmetros de robô Roomba

1° Antônio Espínola Navarro Neto
Divisão de Engenharia Mecânica
Instituto Tecnológico de Aeronáutica
São José dos Campos, Brasil
antonio.neto@ga.ita.br

2° Henrique Silva Simplício
Divisão de Engenharia Aeronáutica e Aeroespacial
Instituto Tecnológico de Aeronáutica
São José dos Campos, Brasil
henrique.simplicio@ga.ita.br

***Resumo*—O presente trabalho se baseia em utilizar algoritmos baseados em metaheurística para otimizar parâmetros de um robô Roomba. Tal robô pode realizar uma série de movimentos baseados em uma Behavior Tree, que incluem andar para frente, andar em espiral, andar para trás e rotacionar. O tempo que leva para cada ação é de grande importância para o tempo total que leva para o Roomba passar por toda, ou maior parte, da área que ele atua. Desse modo, utiliza-se 4 algoritmos de metaheurística para otimizar tais parâmetros e achar uma solução ótima.**

***Palavras-chave*—otimização, metaheurísticas, algoritmos genéticos, simulated annealing, particle swarm optimization, CMA-ES**

I. Introdução Teórica

Metaheurística é um procedimento ou heurística cujo objetivo é achar, gerar ou selecionar uma heurística que pode vim a produzir uma solução suficientemente ótima para um problema de otimização.

Existe uma grande quantidade de algoritmos metaheurísticos, baseados em inúmeras estratégias de buscas. No presente trabalho, usa-se 4 algoritmos em específico, apresentados a seguir:

A. Algoritmos Genéticos

Em um algoritmo genético, uma população de soluções candidatas para um problema de otimização evolui para soluções melhores, baseados na teoria da evolução de Darwin. Cada solução candidata é chamada de cromossomo, que são constituídos por genes (e.g. um bit ou uma dimensão), que podem ser mutados e alterados;

A evolução geralmente começa a partir de uma população de indivíduos gerados aleatoriamente e é um processo iterativo, com a população em cada iteração chamada de geração. Em cada geração, a aptidão de cada indivíduo da população é avaliada; a aptidão é geralmente o valor da função objetivo no problema de otimização que está sendo resolvido. Os indivíduos mais aptos são estocasticamente selecionados da população atual, e o gene de cada indivíduo é modificado (recombinado e possivelmente mutado aleatoriamente) para formar uma nova geração. A nova geração de soluções candidatas é então usada na próxima iteração do algoritmo.

Comumente, o algoritmo termina quando um número máximo de gerações foi produzido ou um nível de aptidão satisfatório foi alcançado para a população [1].

B. Simulated Annealing

O algoritmo de Arrefecimento Simulado substitui a solução atual por uma solução próxima (i.e., na sua vizinhança no espaço de soluções), escolhida de acordo com uma função objetivo e com uma variável T (dita Temperatura, por analogia). Quanto maior for T, maior a componente aleatória que será incluída na próxima solução escolhida. À medida que o algoritmo progride, o valor de T diminui, começando a convergir para uma solução ótima local.

Esta técnica começa sua busca a partir de uma solução inicial qualquer. O procedimento principal consiste em um loop que a cada iteração gera aleatoriamente um único vizinho θ' da solução corrente θ.

A cada geração de um novo vizinho, é testada a variação do valor da função objetivo. Para uma variação negativa, o algoritmo aceita a solução θ' e ela passa a ser a solução corrente. Para variação positiva, é gerado um número aleatório entre 0 e 1, caso for maior do que fator de Boltzmann ($e^{(-\Delta/T)}$, onde Δ é a variação da função custo), rejeita-se a solução e aceita-se caso contrário. Para uma variação igual a zero, a aceitação é indiferente.

A temperatura T assume inicialmente um valor elevado Após um número fixo de iterações, a temperatura é gradativamente diminuída por uma razão de resfriamento. Como esse procedimento se dá no início, há uma chance maior de se escapar de mínimos locais.

O procedimento é finalizado quando a temperatura chega a um valor próximo de zero e nenhuma solução que piore o valor da melhor solução seja mais aceita. A solução obtida quando o sistema encontra-se nesta situação evidencia o encontro de um mínimo local [2].

C. Particle Swarm Optimization – PSO

O PSO é um algoritmo de otimização baseado em população, no qual partículas (candidatos a solução) são geradas aleatoriamente dentro de um espaço de busca e atualizadas, iterativamente, a partir de suas velocidades, como mostrado na Equação 1.

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{v}_i \tag{1}$$

onde $\mathbf{v}_i$ é a velocidade da partícula e $\mathbf{x}_i$ é a posição de cada partícula.

Além disso, as velocidades de cada partícula também são atualizadas a cada iteração, como apresentado na Equação 2.

$$\mathbf{v}_i = \omega \mathbf{v}_i + \varphi_p r_p \left(\mathbf{b}_i - \mathbf{x}_i\right) + \varphi_g r_g \left(\mathbf{b}_g - \mathbf{x}_i\right) \qquad (2)$$

onde $\mathbf{b}_i$ é a melhor posição de cada partícula, $\mathbf{b}_g$ é a melhor posição considerando todas as partículas, ω, φ_p, $e\,\varphi_g$ são hiperparâmetros (chamados de *inertia weight*, *cognitive parameter* e *social parameter*, respectivamente) e $r_p\,e\,r_g$ são números aleatórios entre 0 e 1.

É interessante ressaltar que, para a inicialização das partículas, escolhem-se a quantidade delas e os limites das posições e das velocidades. Além disso, para evitar que as partículas saiam das fronteiras delimitadas, é comum o uso de heurísticas. A heurística mais simples, a qual foi usada na implementação aqui adotada, é limitar ambas – posição e velocidade – pelos seus mínimos e máximos.

D. CMA-ES

O CMA-ES (*Covariance Matrix Adaptation Evolution Strategy*) é um algoritmo evolucionário utilizado para problemas de otimização difíceis, não-lineares, não-convexos, em domínios contínuos [3]. Ele é considerado o algoritmo estado da arte na computação evolucionária.

Tal algoritmo, conforme o nome indica, é uma estratégia de evolução – na qual, a cada geração, amostram-se (distribuição gaussiana multivariada) novos candidatos a solução a partir da média e covariância das melhores amostras – na qual tanto a média, quanto o passo e a matriz de covariância são adaptados, a cada geração, utilizando-se pesos para cada uma das melhores amostras. Além disso, é importante ressaltar a preocupação em prover bons valores para os hiperparâmetros – de modo que o algoritmo funcione em diferentes tipos de problemas – como grande vantagem do CMA-ES.

Para a aplicação do CMA-ES, é necessário fornecer uma solução inicial, um desvio padrão inicial (passo), assim como um critério de parada.

II. Implementação

O problema se baseia em otimizar uma função custo que retorna o tempo de simulação do robô Roomba. O robô tem três tipos de movimento: andar para frente, andar em espiral e, ao entrar em contato com uma parede, andar para trás. Em suma, a função possui como argumentos o tempo que ele anda para frente, o tempo que ele anda em espiral, o raio inicial da espiral, o fator de incremento do raio da espiral, e o tempo que ele anda para trás ao entrar em choque com a parede. A função realiza três simulações e retorna a média do tempo que demora para o robô passear por 60% dos pixeis da região. Nasce a necessidade de realizar mais de uma simulação para dados parâmetros pelo fato de que, ao terminar de andar para trás, o Roomba gira um ângulo aleatório, que cria um caráter estocástico para a função.

Desse modo, o problema de otimização nasce ao se procurar parâmetros que minimizem o tempo de simulação.

III. Resultados e Discussões

Os melhores parâmetros obtidos pelos 4 algoritmos são apresentados nas Tabelas I, II e III. Além disso, na Tabela IV, apresentam-se os tempos médios resultantes do uso dos melhores parâmetros obtidos por cada algoritmo.

TABELA I
MELHORES PARÂMETROS `MOVE_FOWARD_TIME` E `MOVE_IN_SPIRAL_TIME` OBTIDOS PARA CADA UM DOS 4 ALGORITMOS.

Algoritmo	`move_foward_time`	`move_in_spiral_time`
AG	3,6	24
SA	3,326	19,021
PSO	2,756	27,926
CMA-ES	-4,305	24,184

TABELA II
MELHORES PARÂMETROS `GO_BACK_TIME` E `SPIRAL_FACTOR` OBTIDOS PARA CADA UM DOS 4 ALGORITMOS.

Algoritmo	`go_back_time`	`spiral_factor`
AG	0,6	0,04
SA	0,5	0,05
PSO	0,358	0,044
CMA-ES	0,100	2,699

TABELA III
MELHOR PARÂMETRO `INITIAL_RADIUS_SPIRAL` OBTIDO PARA CADA UM DOS 4 ALGORITMOS.

Algoritmo	`initial_radius_spiral`
AG	0,24
SA	0,2
PSO	0,216
CMA-ES	10,147

TABELA IV
TEMPOS OBTIDOS COM OS MELHORES PARÂMETROS PARA CADA UM DOS 4 ALGORITMOS.

Algoritmo	**Melhor tempo (s)**
AG	164
SA	147,44
PSO	118,19
CMA-ES	209,94

Dos resultados, nota-se que o resultado obtido pelo PSO foi significativamente superior aos obtidos pelos demais algoritmos. Tal fato pode estar relacionado tanto à quantidade de partículas (40) como ao número de iterações (500) utilizados. Além disso, para tal algoritmo, utilizou-se a média de 5 tempos (obtidos para cada conjunto de parâmetros, uma vez que o comportamento da simulação é estocástico) em vez de 3 (utilizado nos demais algoritmos).

Além disso, o resultado obtido pelo CMA-ES foi o pior entre eles – o que não era esperado. Tal resultado pode estar relacionado ao baixo tamanho da população utilizada (5) assim como ao número de iterações (50). Também, analisando os valores obtidos para `move_foward_time`, `spiral_factor` e `initial_radius_spiral`, percebe-se que os parâmetros não permaneceram dentro

dos limites adotados para os outros algoritmos, de modo que os valores obtidos foram muito diferentes. Inclusive, vale ressaltar que o `move_foward_time` negativo não poderia ser possível, ressaltando que na implementação do CMA-ES utilizando a biblioteca `cma` [3] não verifica-se se as soluções estão dentro do intervalo de interesse, de forma que tal mecanismo deveria ter sido implementado manualmente.

IV. Conclusão

Do que foi exposto e discutido, conclui-se que, considerando tempos de simulação aproximadamente iguais, o algoritmo PSO obteve melhor desempenho dentre os 4 utilizados. Além disso, faz-se necessário ressaltar que, para obter resultados concretos e que possam ser utilizados de fato para comparar a efetividade dos algoritmos nesse problema, seria necessário aumentar significativamente a quantidade de simulações rodadas.

Ainda na linha desse raciocínio, é importante destacar que o tempo médio obtido para um conjunto de parâmetros está longe de representar o valor real, uma vez que, dado que trata-se de um processo estocástico, seriam necessárias muitas repetições para encontrá-lo e foram usadas apenas de 3 a 5. Também, o próprio número de iterações usado em cada otimização foi pequeno (de 100 a 500), o que pode ter ocasionado a não convergência dos algoritmos.

Referências

[1] D. E. Goldberg, "Genetic algorithms in search, optimization, and machine learning," EUA: Addison-Wesley, 1989.

[2] M. Pincus, "A Monte-Carlo method for the approximate solution of certain types of constrained optimization problems," Journal of the Operations Research Society of America, Nov–Dec 1970. doi:10.1287/opre.18.6.1225

[3] N. Hansen, "The CMA evolution strategy," Dec. 2016. Accessed: July 17, 2022. [Online]. Available: https://cma-es.github.io/

[4] J. Kennedy and R. Eberhart, "Particle swarm optimization," Proceedings of ICNN'95 - International Conference on Neural Networks, 1995, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.

[5] M. R. Bonyadi and Z. Michalewicz, "Particle Swarm Optimization for Single Objective Continuous Space Problems: A Review," Evol Comput, 2017, pp. 1–54 vol. 25 (1). doi: https://doi.org/10.1162/EVCO_r_00180

Apêndice

a) Manual para execução do código: Cada algoritmo foi implementado no seu respectivo arquivo: `algoritmo_genetico.py`, para o Algoritmo Genético; `simulated_annealing.py`, para o *Simulated Annealing*; `pso.py`, para o *Particle Swarm Optimization*; e `cmaes.py`, para o CMA-ES. Para rodar um algoritmo, basta executar o arquivo correspondente.

Para alterar o número de simulações usadas para o cálculo do tempo médio para determinado conjunto de parâmetros, basta alterar o valor da variável `n`, na linha 21 do arquivo `behavior_tree_test.py`.