Particle Swarm Optimization Discreto para o problema do Caixeiro Viajante

Bruno Mamede de Araújo Moura[1], **Gustavo Henriques da Cunha**[1], **Vitor Hugo Palmié Peixoto**[1]

[1]Ciência da Computação
Universidade Federal de São João del-Rei (UFSJ)

Resumo. *Este trabalho tem como intuito resolver um dos problemas clássicos da computação, o problema do caixeiro-viajante (PCV), utilizando a metaheurística Particle Swarm Optimization (PSO), que é um método de otimização que busca melhorar a solução de um problema simulando aspectos comportamentais e sociais de seres vivos, como em uma revoada de pássaros.*

1. Introdução

O problema do caixeiro-viajante é um dos exemplos de problemas de otimização combinatória mais famosos que encontramos na computação. Por ser, em sua natureza, um problema difícil de resolver, pertencente à classe NP-Difícil, mas fácil de explicar e ilustrar, ele se tornou um problema comum para testarmos diferentes algoritmos de aproximação, nos quais buscamos encontrar uma solução boa em tempo polinomial.

Neste trabalho, usamos uma dessas abordagens, o algoritmo PSO, que, simulando o comportamento social de seres vivos, trabalha com um conjunto de soluções que funcionam como partículas ou indivíduos, onde, compartilhando informações, elas buscam melhorar suas posições.

Com o objetivo de observar e entender o funcionamento do PSO no problema, conduzimos testes para observar seu comportamento com diferentes parâmetros e entradas, traçando gráficos e tabelas, buscando encontrar o conjunto de parâmetros que funciona melhor para nossas entradas.

2. Referencial Teórico

O PSO já foi utilizado para resolver o caixeiro-viajante, como em Clerc (2000) e Wang et al. (2003), além de diversas otimizações serem propostas, como em Goldbarg et al. (2005). Nos baseamos em alguns aspectos dessas soluções, mas buscamos um código simples, com o intuito de entender como o algoritmo busca uma melhor solução, assim não nos preocupando tanto com a otimização e o desempenho.

Além do PSO, vários outros métodos são famosos para resolver o PCV, como a busca local ou algoritmos genéticos, que são abordados em vários livros e artigos, como, por exemplo, em Talbi, Metaheuristics: From Design to Implementation (2009), onde podemos ver como o PSO se assemelha a algumas dessas soluções.

Para implementarmos o PSO Discreto nos baseamos em partes nos materias do Prof. Marcone J. F. Souza da Universidade Federal de Ouro Preto (UFOP), que por sua vez se baseou nos materiais da Profa. Estéfane G. M. de Lacerda da Universidade Federal do Rio Grande do Norte (UFRN).

3. Metodologia

Para este trabalho, utilizaremos a linguagem Python, onde, pelo terminal, indicamos o endereço do arquivo de instância, o tamanho da população, o número de iterações, além de duas constantes usadas no algoritmo que influenciam o resultado.

Em nosso problema, armazenaremos o grafo em uma matriz, lendo os dados de um arquivo contendo as coordenadas cartesianas de cada cidade, onde, no código, calculamos a distância entre elas. Note que, para esse problema, só consideramos grafos completos.

4. O Problema do Caxeiro Viajante

Nesta seção, passaremos brevemente pelos conceitos do problema do caixeiro-viajante.

Dado um grafo $G = (V, A)$ onde temos n vértices V que representam as cidades e arestas A que ligam essas cidades com certo peso que representa a distância entre elas, precisamos encontrar um circuito de menor distância possível que comece em uma cidade, passe por todas as outras uma vez e retorne para a cidade de origem.

Como já dito anteriormente, o PCV é NP-difícil; sendo assim, não conhecemos um algoritmo em tempo polinomial que encontre uma resposta ótima para o problema, e é por isso que abordagens aproximadas são populares alternativas para resolvê-lo.

5. Particle Swarm Optimization

Concebido por James Kennedy e Russell Eberhart em 1995, o algoritmo se originou simplesmente como uma tentativa de simulação simplificada de uma revoada de pássaros (Kennedy e Eberhart, 1995), onde mais tarde foi observado o potencial do modelo para otimizar problemas, inicialmente de variáveis contínuas.

Para simular o aspecto cooperativo que podemos encontrar nos pássaros, o PSO clássico guarda a experiência individual de cada indivíduo mais uma experiência global que guia os outros indivíduos para um estado melhor, como se fossem pássaros se comunicando entre si.

O PSO começa ao criar uma população de m indivíduos de forma aleatória. Chamamos esses indivíduos de partículas, e devemos movimentá-las pelo espaço de busca, usando as experiências pessoal e global, tentando convergir para uma posição mais vantajosa. A forma como a partícula se movimentará pelo espaço será influenciada por sua velocidade, que será calculada pela fórmula abaixo:

$$v_{ij} = wv_{ij}^k + c_1 r_1 (p_{best_{ij}}^k - x_{ij}^k) + c_2 r_2 (g_{best_j}^k - x_{ij}^k)$$

para $i = [1, ..., m]$ e $j = [1, ..., n]$, onde m é o número de partículas na população e n o número da dimensão de cada solução.

Note que o fator $p_{best_{ij}}^k - x_{ij}^k)$ da equação representa a experiência pessoal e o fator $(g_{best_j}^k - x_{ij}^k)$ representa a experiência global. O fator w atua com o intuito de diversificar a solução.

Com a velocidade calculada, utilizamos a seguinte equação para atualizar a partícula:

$$x_i^{k+1} = x_i^k + v_i^{k+1}$$

Assim, em cada iteração teremos uma atualização em cada partícula, que irão caminhar em direção a melhor solução até convergirem em algum ponto.

Podemos montar assim o pseudocódigo do PSO:

```
inicialize a nuvem de partículas
repita
   para i = 1 até m
      se f(x_i) < f(p_i) então
         p_i = x_i
         se f(x_i) < f(g) então
            g = x_i
         fim se
      fim se
      para j = 1 até n
         r_1 = rand() , r_2 = rand()
         v_ij = w v_ij + c_1 r_1 (p_i - x_ij) + c_2 r_2 (g_j - x_ij)
      fim para
      x_i = x_i + v_i
   fim para
até satisfazer o critério de parada
```

Figura 1. Pseudocódigo

6. PSO para resolver o Problema do Caxeiro Viajante

O PCV possui natureza discreta, por isso é necessário adaptarmos o PSO para usá-lo nesta situação.

O algoritmo discreto funcionará de forma semelhante ao que vimos no alogoritmo usual, onde começa com uma população inicial gerada de maneira aleatória, composta por vetores que são permutações das n cidades do grafo, de forma que temos um circuito que passe por cada uma.

A função de avaliação que dará o *fitness* ou qualidade de cada solução será simplesmente a soma da distância das cidades, onde buscamos minimizar esse valor.

A primeira mudança necessária para o PSO que usaremos no PCV é a mudança no vetor de velocidade. Temos que o vetor de velocidades será um vetor de transposições que resultará, quando aplicado em uma sequência durante um paço do algoritmo, em uma nova sequência (Clerc, 2000). Seja o vetor de transposições v a velocidade, seu comprimento dado por $||v||$, podemos definir a velocidade como $v = (i_1, j_1), (i_2, j_2), ..., (i_n, j_n)$ que significa substituir os números das posições (i_1, j_1), depois os números em (i_1, j_1), até finalmente os números em (i_n, j_n).

Usamos a velocidade para atualizar a partícula, praticando as transposições que ela guarda, como no exemplo:

Dado uma partícula $P = (1, 2, 3, 4, 5)$ e um vetor de velocidade $v = (1, 3), (2, 5)$, aplicando a operação $P' = P \oplus v$ temos, sucessivamente, $(3, 2, 1, 4, 5)$ e $(3, 5, 1, 4, 2)$.

Para obetermos a velocidade, precisamos da posição de duas partículas, P_1, P_2, onde calcularemos a velocidade v com base na operação $v = P_1 - P_2$, que de maneira direta mostra as transposições necessárias para sair da posição da partícula P_2 e chegar na partícula P_1. Dessa forma conseguimos reproduzir as partes $(p^k_{best_{ij}} - x^k_{ij})$ e $(g^k_{best_j} - x^k_{ij})$ da equacão de velocidade do PSO.

O operador de multiplicação da velocidade por um coeficiente pode ser definido como o quanto da velocidade será mantida, onde de maneira direta, temos que a multiplicação de uma constante c por uma velocidade v significa que escolheremos c transposições aleatórias de v. Como na equação temos que c pode ser um número real, em nosso algoritmo fizemos a adaptação de que arrendondaremos esse valor para o inteiro mais próximo. Além disso, se tivermos que $||v|| < c$, simplismente pegaremos todas as transposições em v. Se $c = 0$, $cv = \oslash$.

No caso da soma de duas velocidades, $v_1 + v_2$, devemos apenas concatená-los, como por exemplo:

Se $v_1 = ((1, 3), (2, 4))$ e $v_2 = ((3, 5), (2, 6))$, temos $v_1 + v_2 = ((1, 3), (2, 4), (3, 5), (2, 6))$.

Com isso, adptamos a equação original para o problema do PCV, com a exceção do fator w de diversificação, que na literatura, como em (Wang et al., 2003), foi observado bons resultados com $w = 0$, além de que pela maneira que implementamos o algoritmo, não vimos uso nele.

Para rodar o algoritmo, devemos ajustar os parâmetros c_1 e c_2, que nos dizem o quanto o indivíduo confia em si mesmo e o quanto ele confia no conjunto. Além disso, devemos ajustar o tamanho da população e o número de iterações. Como esses fatores podem afetar muito a qualidade do algoritmo, realizamos testes para observar uma combinação boa de parâmetros.

Por fim, decidimos o critério de parada sendo a forma mais simples, até que a última iteração se realize. Para otimizar o algoritmo podemos pensar em formas mais inteligentes para fazer a parada, como o número de vezes que a melhor solução não sofreu alteração, mas com queríamos apenas observar o funcionamento do algoritmo, optamos por algo simples.

Além do critério de parada, exitem diversas formas de melhorar nosso algoritmo para conseguir melhores soluções, como podemos ver em diversis artigos na literatura.

7. Ajuste de Parâmetros

Para melhorar a qualidade do algoritmo, fizemos diversos testes onde testamos diferentes combinações de parâmetros, na instância dada na especificação do trabalho com 12 cidades, além de uma instância extra com mais cidades, tendo 48.

Segue as tabelas com os testes realizados, com os parâmetros usados a média

e o desvio padrão calculados a partir dos melhores *fitness* de cada execução depois de rodarmos 25 vezes cada combinação:

Iterações	População	C1	C2	Media	Desvio
200	150	0.9	1.5	31.5645	1.7798
150	100	0.9	1.5	31.7009	1.53619
150	100	1	2	31.9558	1.72534
150	100	2	2	32.3281	2.17394
150	100	0.9	1	32.3434	1.80911
150	100	1	1	32.4457	1.54495
150	100	2	1	32.6743	1.84227
150	100	1.5	2	32.7394	1.71882
100	100	2	2	32.8791	1.68719
50	100	2	2	33.1016	1.75613
150	100	0.9	2	33.2288	2.31515
100	50	1	1.5	33.2646	1.66612
100	50	0.9	1	33.5406	2.46196
100	50	1	1	33.5585	2.16894
150	50	2	2	33.7602	1.53137
100	50	2	2	33.8859	2.24588
50	50	2	2	33.9221	2.1917
100	50	1	2	34.2128	2.04271

Figura 2. Tabela Instância 1

Iterações	População	C1	C2	Media	Desvio
400	300	0.9	2	789.692	40.6394
500	400	0.9	2	790.288	52.3326
400	300	1	1.5	799.982	51.0787
400	300	0.9	1	800.373	49.7569
400	300	1	1	802.323	49.1013
400	300	2	2	813.372	44.8242
400	300	0.9	1.8	815.609	37.4261
400	300	0.9	1.5	816.977	48.3357
300	300	2	2	817.361	35.6321
400	300	1	2	817.739	39.2394
400	300	0.5	2	829.033	40.0859
300	200	2	2	857.528	57.5316

Figura 3. Tabela Instância 2

Podemos observar que os parâmetros sejam diferentes entre as duas instâncias, principalmente as constantes, já que o número da população e de iterações deve ser ajustado com base no tamanho do grafo, ainda temos semelhanças marcantes, como c_1 ser menor que c_2. Podemos ver também que colocar um número muito grande de iterações e de população depois de certo ponto não melhora tão significativamente a qualidade do algoritmo, apenas deixando ele mais lento.

Com base na tabela, pegamos a combinação de instâncias que resultaram em melhor resultados e fizemos alguns testes em cima delas, com o objetivo de analizar o comportamento do algoritmo. Primeiro, rodamos o algoritmo para cada instância uma vez e analizamos a população em cada iteração, guardando o melhor e o pior *fitness*, além da média e do desvio da população, e traçamos os respectivos gráficos das figuras 4 e 5.

Podemos perceber que toda a população converge para o melhor *fitness* encontrado depois de algum tempo. Para otimizarmos o algoritmo poderíamos pensar em alguma forma de disturbio da população, para que não caiam tão rapidamente em ótimos locais.

Vemos também que a partir do momento que toda a população converge, não temos mais melhorias na solução, o que nos diz que na verdade não precisaríamos de um número de iterações tão grande como o utilizado.

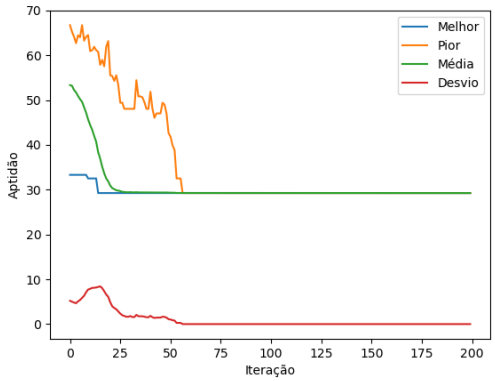

Figura 4. Gráfico Instância 1

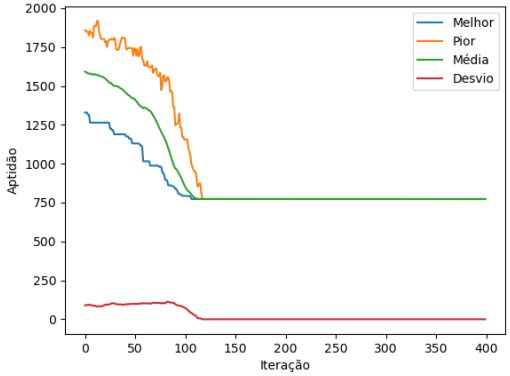

Figura 5. Gráfico Instância 2

Depois disso, com os mesmos parâmetros, executamos o algoritmo 10 vezes para cada instância para traçarmos um gráfico sobre o comportamento de cada execução durante cada iteração, com os gráficos das figuras 6 e 7.

Em ambos os gráficos podemos perceber como, no geral, os parâmetros utilizados nos trazem boas melhoras sobre as soluções aleatórias originais. Devido a natureza não determinística do algoritmo devemos executá-lo várias vezes se quisermos achar uma solução mais próxima possível da ótima.

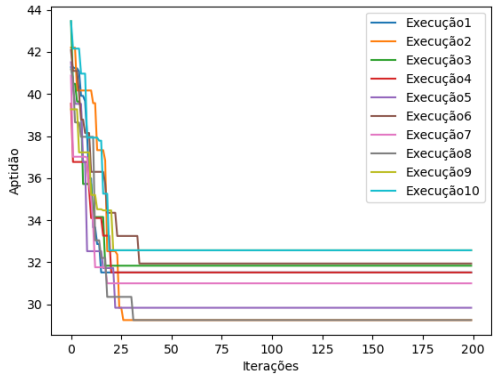

Figura 6. Gráfico Instância 1

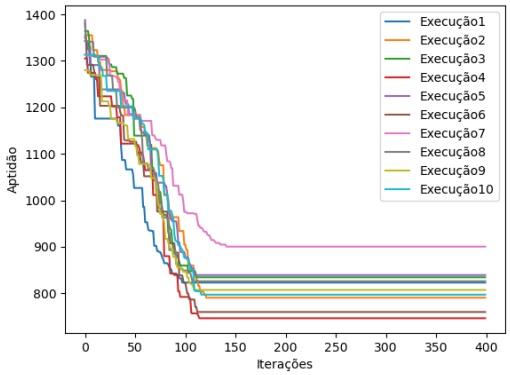

Figura 7. Gráfico Instância 2

8. Conclusão

Com a utilização do PSO, pudemos ver uma forma diferente de resolver o PCV, utilizando um conhecimento prévio para tomar decisões.

Vimos também como o método resolve o problema e como podemos ajustar os parâmetros para otimizar a busca, sendo importante a realização de diversos testes.

Devemos notar que existe ainda a possibilidade de otimização do algoritmo, onde podemos até mesmo combinar elementos de outros métodos, como algoritmos gulosos ou algoritmos genéticos, que possam levar a soluções mais refinadas.

Além disso, o PSO se mostra uma ferramenta poderosa para a resolução de problemas combinatórios, principalmente com a abordagem discreta que utilizamos, onde o número de situações em que o algoritmo pode ser útil aumenta.

Referências

CLERC, M. Discrete Particle Swarm optimization: Illustrated by the Traveling Salesman Problem. 2000 Artigo disponível em http://clerc.maurice.free.fr/pso/pso_tsp/Discrete_PSO_TSP.htm. Acessado em 16/11/2004.

GOLDBARG, E. F. G. ; GOLDBARG, M. C. ; SOUZA, G. R. Otimização por Nuvens de Partículas para o Problema do Caixeiro Viajante, XXXVII SIMPÓSIO BRASILEIRO de PESQUISA OPERACIONAL , Gramado, RS, 2005.

SOUZA, M. J. F. Particle Swarm Optimization (PSO). 2021 Disponível em http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/PSO.pptx. Acessado em 16/11/2004.

KENNEDY, J.; EBERHART, R. Particle Swarm Optimization, Proceedings of the IEEE International Conference on Neural Networks, v. 4, Perth, Australia, 1942-1948, 1995.

WANG, K.-P.; HUANG, L.; ZHOU, C.-G.; PANG, W. Particle swarm optimization for Traveling Salesman Problem. Proceedings of the Second International Conference on Machine Learning and Cybernetics, Xi'an, China, 1583-1585, 2003.