Otimização Discreta por Nuvem de Partículas aplicada ao Problema do Caixeiro Viajante

Dario José Aloise (UFRN) dario@dimap.ufrn.br

Marilyn Cristine Serafim de Oliveira (UFRN) marilyn@ppgsc.ufrn.br

Thales Lima Silva (UFRN) thales@engcomp.ufrn.br

Resumo
Particle Swarm Optimization (PSO) ou Otimização por Nuvem de Partículas é uma metaheurística que surgiu da intenção de simular o comportamento de um conjunto de pássaros em vôo com seu movimento localmente aleatório, mas globalmente determinado. Esta técnica tem sido muito utilizada na resolução de problemas contínuos não-lineares e pouco explorada em problemas discretos. Este artigo apresenta o funcionamento desta metaheurística com novas adaptações para sua aplicação em problemas de otimização discreta. Ao final, são apresentados resultados de experimentos computacionais para algumas instâncias do problema do caixeiro viajante, a fim de demonstrar a eficiência do método na resolução de problemas desta categoria.
Palavras-chave*: Nuvem de Partículas, Otimização Global, Otimização Combinatória.*

1. Introdução

As técnicas de computação evolucionária (programação evolucionária [3], algoritmos genéticos [4], estratégias evolucionárias [7] e programação genética [10]) são motivadas pela evolução da natureza. Uma população de indivíduos, os quais representam as possíveis soluções para o problema, são manipulados de acordo com as regras de sobrevivência que melhor se adaptam aos operadores genéticos, tais como mutação, cruzamento e reprodução. A melhor solução evolui através das gerações. Essas técnicas têm sido aplicadas com sucesso em muitas áreas do conhecimento, e mais especificamente, em problemas de Otimização. Baseando-se nas técnicas evolucionárias, Eberhart e Kennedy desenvolveram um algoritmo diferente, denominado de Particle Swarm Optimization (PSO) ou Otimização por Nuvem de Partículas, inspirado na simulação do comportamento social de um bando de pássaros em revoada com movimento localmente aleatório, mas globalmente determinado [1, 2, 5, 6]. No PSO, assim como em outros algoritmos, existe uma população de indivíduos, chamados de nuvem (ou enxame) de partículas, que ao invés de utilizar operadores genéticos, evoluem através da cooperação e competição entre si por diversas gerações. As partículas se beneficiam de sua própria experiência e da experiência de outros membros do enxame durante a busca de uma melhor fitness (alvo: comida, local para pouso, proteção de predadores, etc).

2. Estrutura do Algoritmo

Em sua versão original o *PSO* é muito similar a algumas técnicas da computação evolucionária, como os algoritmos genéticos (*GA*), onde o sistema é inicializado com uma população de soluções randômicas. Entretanto, difere dessas outras técnicas populacionais, pois nenhum operador inspirado pelos procedimentos de DNA é aplicado na população para obter uma nova geração de indivíduos. Em vez de mutação, são utilizadas *partículas*, movendo-se em um espaço de busca n-dimensional, sendo cada uma delas uma solução potencial para o problema. Cada partícula possui também uma velocidade randômica para que possa percorrer o espaço de soluções do problema. A i-ésima partícula é representada por:

$$X_i = (X_{i1}, X_{i2},\ldots, X_{iD}).$$

A melhor posição prévia, ou seja, a posição que dá o melhor valor de aptidão da i-ésima partícula, é registrada e representada por:

$$P_i = (P_{i1}, P_{i2},\ldots, P_{iD}).$$

O índice da melhor partícula entre todas as partículas na população é representado pelo símbolo *g*. A taxa da mudança de posição, que é chamada de velocidade, para partícula i é representada definida por

$$V_i = (V_{i1}, V_{i2},\ldots, V_{iD}).$$

A movimentação de cada partícula é baseada em três parâmetros: *fator de sociabilidade* que determina a atração das partículas para a melhor posição descoberta por qualquer elemento do enxame; *fator de individualidade* que determina a atração da partícula com sua melhor posição e a *velocidade máxima* que delimita o movimento, uma vez que esse é direcional e determinado.

As partículas são então manipuladas de acordo com as seguintes equações:

$$(1)\ V_{id} = W^* V_{id} + c_1 * rand(.) * (P_{id} - X_{id}) + c_2 * Rand(.) * (P_{gd} - X_{id})$$
$$(2)\ X_{id} = X_{id} + V_{id}$$

Onde:

c_1 e c_2 - são duas constantes positivas que correspondem as componentes cognitivas e sociais;

rand(.) e *Rand (.)* - são duas funções aleatórias no intervalo [0,1];

W - é o fator de inércia que determina a *diversificação* ou *intensificação* das partículas.

A Equação (1) é usada para calcular a nova velocidade da partícula de acordo com sua velocidade anterior e as distâncias entre sua posição atual, sua melhor posição e a melhor posição do grupo. A partir daí, a partícula "voa" para uma nova posição de acordo com a equação (2). O desempenho de cada partícula é medido de acordo com uma função de aptidão pré-definida que é relacionada ao problema a ser resolvido. O fator de inércia W é empregado para controlar o impacto da velocidade anterior na velocidade atual, influenciando assim as habilidades de exploração global e local das partículas. Um fator de inércia maior facilita exploração global, ou seja, a procura por novas áreas dentro do espaço, enquanto um fator de inércia menor tende a facilitar exploração local para refinar a área de procura atual. A seleção satisfatória do fator de inércia *W* pode prover um equilíbrio entre habilidades de exploração global e local, podendo dessa forma requerer menos repetições, em média, para encontrar o valor ótimo.

Cada partícula mantém o rastro de suas coordenadas no espaço problema, que estão associadas à melhor solução (*fitness*) que ela tenha encontrado até então. O valor do *fitness* também é armazenado. Esse valor é chamado *pbest*. Outro valor que é rastreado pela versão global do *Particle Swarm Optimization* é o melhor sobre todos os valores, e sua posição, obtido por qualquer partícula na população. Esse valor é chamado de *gbest*.

O conceito do *PSO* consiste de, a cada passo, mudança de velocidade (aceleração) as partículas alcancem suas posições de *pbest* e *gbest.* O processo original para implementação da versão global do PSO é realizado de acordo com a Figura 1:

Passo 0 (Definição das variáveis):
P o tamanho da população do PSO.
PSO [i] a posição da i-ésima partícula da população do PSO, que representa uma solução candidata para o problema.
fitness[i] o custo da função da i-ésima partícula.
V[i] a velocidade da partícula.
G_{best} um índice para a melhor posição global.
$P_{best}[i]$ a posição de melhor localização da i-ésima partícula.
$P_{best\_fitness}[i]$ o melhor fitness local visitado pela i-ésima partícula.

Passo 1 (Inicialização): Para cada partícula i na população:
Passo 1.1: Inicialize o *PSO[i]* randomicamente.
Passo 1.2: Inicialize *V[i]* randomicamente.
Passo 1.3: Avalie o *fitness[i]*.
Passo 1.4: Inicialize $G_{best.}$
Passo 1.5: Inicialize *Pbest[i]* com uma cópia de $PSO[i] \ \forall \ i \leq P$.

Passo 2: Repita até que um critério de parada seja satisfeito
Passo 2.1: Encontre o G_{best} tal que $fitness[G_{best}]$ *fitness[i]* $i \leq P$.
Passo 2.2: Para cada partícula *i*:
$P_{best}[i] = PSO[i]$ *se* $fitness[i] > P_{best\_fitness}[i] \ \forall \ i \leq P$.
Passo 2.3: Para cada particular *i*:
Atulaize *V[i]* e *PSO[i]* de acordo com as equações 1 e 2.
Passo 2.4: Avalie o $fitness[i] \ \forall \ i \leq P$

Figura 1: Algoritmo PSO Clássico

O funcionamento do algoritmo resume-se em:

1. Inicialização de uma população de partículas com posições e velocidades aleatórias em um espaço de busca *d*-dimensional.
2. Para cada partícula, avalia-se a função de fitness encontrado e a otimização desejada em d variáveis.
3. Compara-se o fitness e o *pbest* da partícula. Se o **valor atua**l for melhor que o *pbest* então, o valor do *pbest* é substituído pelo valor do fitness encontrado, e a **posição** do *pbest* pela **posição atual** no espaço *d*-dimensional.
4. O valor do fitness é comparado com os melhores valores encontrados em toda a população. Se o **valor atual** for melhor que o *gbest* então, o *gbest* é modificado para o índice e valor da partícula atual.
5. Modifica-se então a velocidade e posição da partícula de acordo com as equações (1) e (2) respectivamente.
6. Executa-se um loop para o passo 2 até que um critério seja encontrado: normalmente, um fitness bom o suficiente ou um número máximo de iterações pré-estabelecido.

3. PSO Discreto

As modificações propostas para o *PSO discreto* podem ser ditas de certa forma, elegantes, pois elas preservaram toda a estrutura do algoritmo *PSO original* e, além disso, inseriu o *PSO discreto* em uma nova classe de problemas. As mudanças sugeridas foram: Os vetores de posição atual e melhor posição foram substituídos por valores discretos, e o vetor velocidade,

por probabilidades onde um valor discreto em particular, modificará uma determinada iteração.

3.1. População inicial

As partículas no *PSO discreto* são representadas através de arestas/nós, sendo cada um destes elementos uma dimensão, representadas em um vetor posição.

3.2. Operadores Discretos

3.2.1. Velocidade

No espaço discreto, a velocidade não é mais somente um número. Ela deve ser um operador que, quando aplicado a uma posição, dê outra posição. Sendo assim, a velocidade é definida como uma lista de N transposições: velocidade = $\{(i_1,j_1),(i_2,j_2), \ldots (i_N,j_N)\}$, que significa a troca dos vértices de índices i_1 e j_1, i_2 e j_2, etc. até a troca dos vértices i_N e j_N.
A velocidade pode ser dita equivalente, ou seja, velocidade $v_1 = v_2$, se nós aplicarmos v_1 ou v_2 em uma posição e obtivermos o mesmo resultado. Uma velocidade nula é uma lista vazia, ∅. $|v|$ denota o tamanho da velocidade v, que é igual ao número de transposições em v.

3.2.2. Movimentação (Posição + Velocidade)

Seja P uma posição e v uma velocidade, logo, P' será uma nova posição onde $P' = P + v$, onde devem ser aplicadas todas as transposições pertencentes a v.
Por exemplo, seja $P=[2,3,4,5,1]$, $v = \{(1,2), (2,4)\}$, a operação é realizada da seguinte forma: Aplique a transposição $(1,2)$, ou seja, troque o conteúdo da 1ª célula e da 2ª célula em P, onde teremos $P_1=[3,2,4,5,1]$. Em seguida, aplique a transposição $(2,4)$, ficando $P_2=[3,5,4,2,1]$. Como a última transposição foi alcançada, uma nova posição P' é obtida: $P'=P_2=[3,5,4,2,1]$.

3.2.3. Obtenção da Velocidade (Posição – Posição)

Sejam $P1$ e $P2$ duas posições. A diferença $P1-P2$ é definida como sendo a velocidade v, onde o sinal de " – " possui um novo significado. Através dessa diferença obteremos a lista de transposições, resultando na seguinte equação: $P_2 = P_1 + v$.

Como podemos verificar no exemplo logo abaixo:

Exemplo: Dadas duas posições $P_1= [1\ 2\ 3\ 4\ 5]$ e $P_2= [2\ 3\ 1\ 5\ 4]$:
$P_1[1] = P_2[3] = 1$, logo a primeira transposição será $(1,3)$, então teremos um $P_2'=P_2 + S(1,3)$. Portanto, $P_2' = [1\ 3\ 2\ 5\ 4]$
$P_1[2] = P_2'[3] = 2$, logo a segunda transposição será $(2,3)$, então teremos $P_2''= P_2' + S(2,3)$. Portanto, $P_2'' = [1\ 2\ 3\ 5\ 4]$
Seguindo na seqüência, o terceiro operador será $(4,5)$, logo teremos a lista de transposição (v), que será igual a $P_1 – P_2 = \{(S(1,3), S(2,3), S(4,5))\}$.

3.3. Adição entre Velocidades (Velocidade + Velocidade)

A adição entre duas velocidades consiste na concatenação de listas de transposições. Resultando uma nova lista.

3.4. Multiplicação (Coeficiente * Velocidade)

Seja c um coeficiente e v uma velocidade. Então são possíveis quatro casos, dependendo do valor de c:

(i) $c = 0$, teremos $c*v = \emptyset$;
(ii) $c \in [0,1]$, truncamos v por $(1-c) *|v|$;
(iii) $c > 1$, aplicamos v, c vezes para a parte inteira e aplicamos (ii) para a parte fracionária;
(iv) $c < 0$, não é definido pra este tipo de implementação.

4. Aplicação ao Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) é descrito por um conjunto de *n* cidades e uma matriz de distância entre elas, tendo o seguinte objetivo: o caixeiro viajante deve sair de uma cidade chamada origem, visitar cada uma das *n - 1* cidades restantes apenas uma única vez e retornar à cidade origem percorrendo a menor distância possível. Em outras palavras, deve ser encontrada uma rota fechada (ciclo hamiltoniano) de comprimento (ou custo) mínimo que passe exatamente uma única vez por cada cidade [8, 9]. A ordem da visita das cidades pode ser escolhida pelo caixeiro. Entretanto, caso o caixeiro deseje economizar tempo e energia, ele deverá procurar o menor caminho de forma que todas as cidades sejam visitadas uma única vez e então retornar a sua cidade de origem.

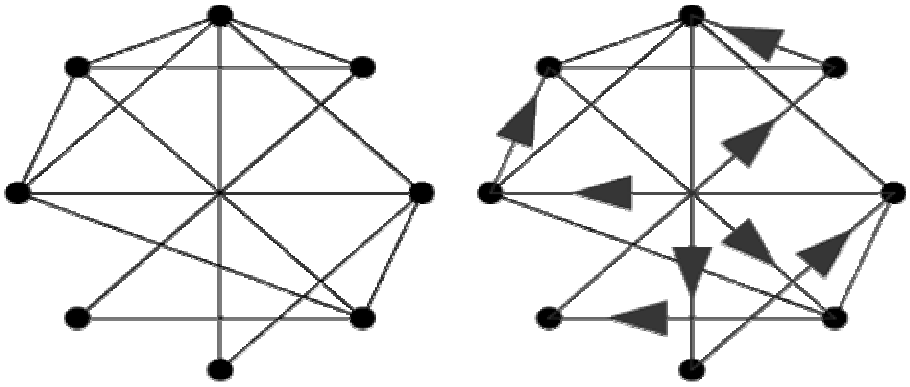

Figura 2: Possível representação para o Problema do Caixeiro Viajante.

O algoritmo PSO Discreto em sua versão clássica, não apresentou resultados satisfatórios para o Problema do Caixeiro viajante, então, neste trabalho foram desenvolvidas e implementadas outras estratégias para melhorar os resultados obtidos.

A **1ª estratégia** experimentada foi incrementar o PSO clássico com a inserção de uma busca local, evitando assim que as partículas caíssem em ótimos locais e ali permanecessem até o final da execução. O procedimento para realização da busca local foi o *First Improvement*. Além da busca local, foi inserido também na 1ª estratégia o conceito de vizinhança por bairros. As partículas foram agrupadas de 10 em 10 em elementos, chamados de bairros. Esses bairros se comunicam através de pontos de interseção, onde uma determinada partícula pode pertencer a vários bairros ao mesmo tempo. Em cada bairro é eleita uma partícula de melhor posição e essas melhores partículas, devido aos pontos de interseção, se comunicam, passando suas posições para as outras partículas dos outros bairros.

A **2ª estratégia** desenvolvida foi a divisão das partículas em bairros separados sem pontos de intersecção. Neste procedimento, é selecionada a melhor partícula de cada bairro (população) e a estas partículas são permitidas 100 execuções do algoritmo, separadamente de seus bairros, para que elas possam melhorar os resultados encontrados. Após isso, cada partícula volta para seu bairro de origem passando as outras o melhor resultado encontrado que passa a ser o *gbest*. O procedimento de busca local foi mantido na 2ª e na 3ª estratégia.

Na **3ª estratégia** foi mantida a estrutura de bairros separados da estratégia 2 sendo adicionada a ela algumas partículas soltas. Estas partículas não terão sua orientação baseada nos vetores de melhor posição da partícula e melhor posição global, ao invés disso o vetor com a melhor posição da partícula é substituído por um vetor de posições aleatórias que levam a partícula em direção ao *gbest* de forma indireta. O principal objetivo aqui é fazer com que as partículas não caiam em ótimos locais, além de permitir uma maior variabilidade às posições das partículas.

5. Resultados Computacionais

Os testes foram realizados com 10 instâncias da TSPLIB para problemas simétricos e assimétricos. O tempo de execução variou de 0.5 a 30 minutos no pior caso.
Os piores e os melhores resultados obtidos, assim como a média das execuções, podem ser visualizados nas tabelas 1,2 e 3.

Tabela 1: Comparação, entre diferentes abordagens para o problema, das piores soluções obtidas.

Estratégia \ Instância	**PSO Clássico**	**Estratégia 1** (Bairros Ligados e Busca Local)	**Estratégia 2** (Bairros Separados e Busca Local)	**Estratégia 3** (Bairros Separados e Partículas Soltas)	**Ótimo**
br17	53	49	39	39	39
ftv35	3554	1835	1844	1826	1473
swiss42	3640	1604	1548	1581	1273
p43	11710	6597	5670	5681	5620
brasil58	109109	33983	32615	34290	23395
ftv70	8232	3227	3153	3199	1950
kro124	119486	57366	55601	57904	36230
ftv170	19587	7113	6960	7117	2755
rbg323	5863	1644	1624	1631	1326
rbg403	7378	3010	2876	2691	2465

Tabela 2: Comparação, entre diferentes abordagens para o problema, das melhores soluções obtidas.

Estratégia \ Instância	**PSO Clássico**	**Estratégia 1** (Bairros Ligados e Busca Local)	**Estratégia 2** (Bairros Separados e Busca Local)	**Estratégia 3** (Bairros Separados e Partículas Soltas)	**Ótimo**
br17	44.0	44.3	39.0	39.0	39
ftv35	3253.5	1710.3	1657.4	1685.8	1473
swiss42	3307.6	1455.4	1417.0	1422.7	1273
p43	6429.5	5667.3	5641.6	5647.5	5620
brasil58	90636.2	30291.3	29352.0	29764.5	23395
ftv70	7813.0	2822.1	2874.7	2873.4	1950
kro124	112087	52940.5	50716.4	51029.6	36230
ftv170	17403.8	6623.0	6355.4	6408.6	2755
rbg323	5715.5	1581.1	1564.2	1579.9	1326
rbg403	7146.6	2945.7	2762.0	2636.2	2465

Tabela 3: Comparação entre as médias das soluções obtidas das diferentes abordagens.

Estratégia \ Instância	**PSO Clássico**	**Estratégia 1** (Bairros Ligados e Busca Local)	**Estratégia 2** (Bairros Separados e Busca Local)	**Estratégia 3** (Bairros Separados e Partículas Soltas)	**Ótimo**
br17	41	39	39	39	39
ftv35	3099	1644	1542	1606	1473
swiss42	3106	1374	1284	1372	1273
p43	6118	5655	5624	5637	5620
brasil58	84872	30270	27551	27387	23395

ftv70	7431	2797	2730	2665	1950
kro124	105832	50034	48385	49089	36230
ftv170	16580	6267	5995	5960	2755
rbg323	5641	1549	1517	1549	1326
rbg403	7031	2733	2692	2617	2465

6. Conclusões

O PSO pertence a uma classe de algoritmos baseado na natureza que tem apresentado bons resultados em Problemas Contínuos de Otimização Combinatória, mas ainda pouco utilizado em problemas discretos. Neste artigo, foram apresentadas algumas estratégias na Otimização Discreta por Nuvem de Partículas (PSO) aplicada ao Problema do Caixeiro Viajante, onde demonstrou ser bastante eficiente na busca de bons resultados.

Ainda há muito a ser estudado sobre o PSO Discreto. As técnicas de busca local utilizadas neste trabalho apresentaram resultados satisfatórios, porém, em alguns testes, o tempo de execução ficou um pouco elevado (30 min), o que pode inviabilizar a sua aplicação a instâncias muito grandes do problema.

A implementação de técnicas que possibilitem resultados satisfatórios, em um tempo de execução viável para instâncias maiores do Problema do Caixeiro Viajante são as prioridades para continuidade da pesquisa. Esta idéia, dentre outras, estão além deste artigo, mas são áreas que estão sendo exploradas.

7. Referências

[1] Eberhart, R.C., Dobbins, R.C., and Simpson, P. (1996), Computational Intelligence PC Tools, Boston: Academic Press.

[2] Eberhart, R.C., and Kennedy, J. (1995). A New Optimmizer Using Particles Swarm Theory, Proc. Sixth International Symposium on Micro Machine and Human Science (Nagoya, Japan), IEEE Service Center, Piscataway, NJ, 39-43.

[3] Fogel, L.J. (1994), Evolutionary Programming in Perspective: the Top-down View. In Computational Intelligence: Imitating Life, J.M. ZURADA, R.J. Marks II, and C.J. Robinson, Eds., IEEE Press, Piscataway, NJ.

[4] Goldberg, D.E. (1989), Genetic Algorithms in Search, Optimization, and Machine Learning, Reading MA: Addison-Welsey.

[5] Kennedy, J., and Eberhart, R.C. (1995). Particle Swarm Optimization, Proc. IEEE International Conference on Neural Networks (Perth, Australia), IEEE Service Center, Piscataway, NJ, IV: 1942-1948.

[6] Kennedy, J. (1997), The Particle Swarm: Social Adaptation of Knowledge, Proc. IEEE International Conference on Evolutionary Computation (Indianapolis, Indiana), IEEE Service Center, Piscataway, NJ, 303-308.

[7] Koza, J.R. (1992), Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA.

[8] Laporte, G. & Martello, S. *The selective traveling salesman problem. Discrete Applied Mathematics 26, 193-207.* ,1990

[9] Lawer, E.L., Lenstra, J.K, Rinooy Kan, A.H.G., Shmoys, D.B. The Traveling Salesman Problem, John Wiley and Sons, Chichester, 1985.

[10] Rechemberg, I. (1994), Evolution Strategy, In Computational Intelligence: Imitating Life, J.M. Zurada, R.J. Marks II, and C. Robinson, Eds., IEEE Press, Piscataway, NJ.

[11] *Traveling Salesman Problem* em http://mathworld.wolfram.com/ Traveling Salesman Problem.html acessado em 14/06/2005