Aplicação de Algoritmo de Nuvem de Partículas no Problema do Caixeiro Viajante

CCO-727 Otimização Inteligente de Sistemas Produtivos – Profº Edilson Kato

Diego Luiz Cavalca – diego.cavalca@dc.ufscar.br

RESUMO

O presente trabalho visa analisar e comparar, em termos de desempenho computacional, a qualidade das soluções obtidas usando Algoritmos de Nuvem de Partícula (do inglês, *Particle Swarm Optimization* - PSO) para solução do Problema do Caixeiro Viajante (PCV). O PCV foi o primeiro problema apresentado na literatura com o intuito de buscar um melhoramento nas rotas de veículos, no entanto trata-se de um problema NP-hard, ou seja, não é possível encontrar uma solução ótima em tempo computacional válido. Visto que soluções ótimas não são ideais para esse tipo de problema, métodos heurísticos são estudados para buscar resultados satisfatórios. PSO é uma meta-heurística que surgiu da intenção de simular o comportamento de um conjunto de pássaros em voo, 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 meta-heurística, 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 (PCV), disponibilizadas na TSPLIB, a fim de demonstrar a eficiência do método na resolução de problemas desta categoria. O melhor resultado conhecido para a base de dados TSPLib EIL51.tsp foi 426. O algoritmo proposto nesse trabalho atingiu como melhor resultado o valor de 429.48 para o problema em questão.

1. INTRODUÇÃO

O Problema do Caixeiro Viajante (PCV) consiste em estabelecer uma rota que passe por cada nó de um grafo, uma única vez, retornando ao nó inicial no final do percurso.

Este roteiro Hamiltoniano deve ser feito de modo que a distância total percorrida seja mínima. O conjunto de rotas possíveis para o PCV Simétrico, isto é, se a distância do ponto a

ao ponto *b* é igual ao do ponto *b* ao ponto *a*, é *a*, é o resultado de todas as combinações possíveis e pode ser calculado por *(n − 1)!*, sendo *n* o número de nós.

Este problema pertence a classe de problemas conhecida por NP-Hard, isto é, não existem algoritmos com limitação polinomial capazes de resolvê-lo. Assim a quantidade de passos de um algoritmo que possa solucioná-lo otimamente não pode ser dada por uma função polinomial do tamanho de sua entrada. Logo, apenas os problemas de pequeno porte podem ser solucionados de forma ótima.

Problemas maiores tornam-se tornam-se inviáveis através dos métodos exatos, haja vista o esforço computacional que seria exigido para resolvê-los. Muitas abordagens de algoritmos heurísticos, que fornecem soluções factíveis próximas da ótima, têm sido desenvolvidas para resolver os problemas NP-Hard, apresentando soluções aproximadas e as algumas vezes ótimas para o problema.

Diante disto, *Particle Swarm Optimization* (PSO) ou Otimização por Nuvem de Partículas é uma meta-heurística que surgiu da intenção de simular o comportamento de um conjunto de pássaros em voo, com seu movimento localmente aleatório, mas globalmente determinado.

Portanto, neste trabalho é apresentado desenvolvimento de um algoritmo de nuvem de partículas proposto no trabalho de (ROSENDO, 2010) para a resolução do Problema do Caixeiro Viajante, no qual é demonstrado a aplicação de meta-heurísticas combinadas para tratar do contexto discreto no qual este problema está inserido, pois a abordagem clássica do PSO foi desenvolvida para tratar a otimização de funções contínuas, o qual não apresenta resultados viáveis e/ou satisfatórios para o PCV.

2. APLICAÇÃO DO ALGORITMO PSO PARA O PROBLEMA DO CAIXEIRO VIAJANTE

2.1. ALGORITMO *PARTICLE SWARM OPTIMIZATION* DISCRETO

Conforme (ROSENDO, 2010), a seguir são descritos os principais termos utilizados no PSO:

- ***Swarm* ou Enxame:** População/Nuvem do algoritmo;
- **Partícula:** Indivíduo da população ou enxame. Cada partícula possui uma de- terminada posição no espaço de busca do problema e esta posição representa uma solução em potencial para o problema tratado;

- **Velocidade (Vetor):** Responsável por comandar o processo de otimização, a velocidade de uma partícula determina a direção na qual ela se movimentará, com objetivo de melhorar sua posição atual. Ela é atualizada, durante as iterações do algoritmo, de acordo com as melhores posições, sendo que esta atualização varia de acordo com a estratégia utilizada;
- **Pbest:** Melhor posição já alcançada pela partícula;
- **Lbest:** Melhor posição já alcançada por uma partícula pertencente à vizinhança de uma determinada partícula;
- **Gbest:** Melhor posição já alcançada por uma partícula em toda população. • Líderes: Partículas da população que possuem os melhores valores da função objetivo para o problema;
- **Coeficiente de inércia (w):** Usado para controlar a influência dos valores anteriores da velocidade no cálculo da velocidade atual;
- **Fator de individualidade (c1):** Influencia na atração que a partícula tem em direção à melhor posição já encontrada por ela mesma (Pbest);
- **Fator de sociabilidade (c2):** Influencia na atração que a partícula tem em direção à melhor posição já encontrada por qualquer partícula vizinha a ela (Lbest);
- **Topologia de Vizinhança:** Determina o conjunto de partículas usado como vizinhança de uma determinada partícula.

Como citado, no algoritmo PSO clássico, apresentado em (GOLDBARG; GOLDBARG; LUNA, 2016), a velocidade é o operador responsável pela movimentação da partícula. Ao analisar a equação de atualização da velocidade

$$V(t+1) = w * V(t) + c1 * \varphi 1 * (Pbest(t) - X(t)) + c2 * \varphi 2 * (Lbest(t) - X(t))$$

percebe-se que este operador engloba, em uma única operação, os três movimentos possíveis, e quando aplicado à partícula, faz com que a mesma seja influenciada pelos três caminhos de uma só vez proporcionalmente aos coeficientes w, c1 e c2. Entretanto quando se lida com problemas discretos, como o estudado neste trabalho, pode não haver um único operador capaz de englobar os três movimentos.

Além disso, na PSO tradicional a partícula é codificada como um conjunto de variáveis reais que representam a sua localização num espaço multidimensional. E todas as dimensões normalmente são independentes umas das outras, de forma que a

aplicação da velocidade à posição da partícula é realizada independentemente em cada dimensão. Esta é uma das principais características do algoritmo PSO.

Como cita (SHI et al., 2007), e defende (ROSENDO, 2010), entretanto, desta forma o algoritmo PSO tradicional não é aplicável a problemas combinatórios, pois assim, o algoritmo violaria regras de restrição gerando soluções inválidas para o problema em questão.

Portanto, é possível afirmar que o algoritmo PSO foi inicialmente projetado para a resolução de problemas de otimização contínua. Desse modo, tal algoritmo deve passar por uma série de adaptações para se tornar capaz de resolver problemas discretos.

Logo, neste trabalho o movimento M1 é implementado como uma busca local. Outro operador de velocidade é considerado quando uma partícula tem que se mover de sua posição atual para outra (*Pbest* ou *Lbest*, M2 ou M3). Uma maneira natural de realizar essa tarefa é executar o *Path relinking* entre as duas soluções. A principal diferença entre o presente trabalho e as demais abordagens que descrevem PSO discretos utilizando busca local e *Path reliking* diz respeito aos operadores de velocidade, pois no algoritmo proposto todas as operações, ou movimentos, são aplicados à partícula em cada iteração.

$$\vec{V}(t+1) = \overbrace{w * \underbrace{\vec{V}(t)}_{\textbf{M1}}}^{\textbf{Local Search}} + \overbrace{c1 * \Phi 1 * \underbrace{\left(\vec{P}_{best}(t) - \vec{x}(t)\right)}_{\textbf{M2}} + c2 * \Phi 2 * \underbrace{\left(\vec{L}_{best}(t) - \vec{x}(t)\right)}_{\textbf{M3}}}^{\textbf{Path relinking}}$$

Figura 1 – *Equação de atualização da velocidade*
Fonte: (ROSENDO, 2010)

Conforme supracitado, no PSO clássico, a velocidade é um operador que cobre os três movimentos em uma única operação. Assim, no algoritmo proposto a velocidade também foi definida como o conjunto de três operações que são aplicadas a cada partícula.

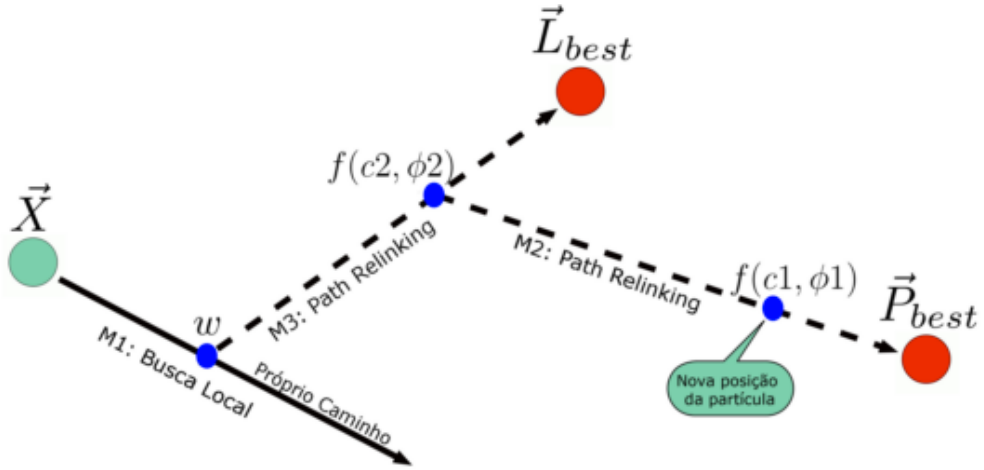

Figura 2 – *Representação gráfica do algoritmo proposto por* (ROSENDO, 2010)

De maneira geral, como pode ser observado na figura acima, primeiro a busca local é realizada até o limite estipulado por w, então o *Path relinking* para o movimento M3 é aplicado até o limite estabelecido por c2 e φ2 através da função *f* que pode ser implementada de diferentes formas dependendo do problema tratado. Finalmente, o mesmo é feito para o movimento M2 cujo limite é estabelecido por c1 e φ1.

Assim, o pseudocódigo do algoritmo PSO desenvolvido neste trabalho, com base na pesquisa realizada por (ROSENDO, 2010), é apresentado a seguir:

1. *Define os parâmetros do PSO*
2. **for** i = 0 até *tamanhoEnxame* do
    a. Inicia X^i com valores aleatórios
    b. Avalia X^i
    c. Define Pbesti e Gbest com base em Xi
3. **end for**
4. **while** *Não atingir critério de parada* **do**
    a. **for** i = 0 até *tamanhoEnxame* **do**
        i. X^i = *buscaLocal*(X^i, *w)*
        ii. X^i = *PathRelinking*(X^i, *Gbest, c2)*
        iii. X^i = *PathRelinking*(X^i, *Pbest*^{i}, *c1)*
        iv. Avalia X^i
        v. Define Pbest^i e Gbest com base em X^i
    b. **end for**
5. **end while**
6. **return** Gbest

Algoritmo 1 – *Pseudocódigo do PSO proposto*

Fonte: Autor

2.2. COEFICIENTES E FUNÇÕES AUXILIARES

Um dos principais objetivos deste trabalho consiste em desenvolver um algoritmo de meta-heurística mantendo as principais características do PSO original. Para manter este equilíbrio, é necessário definir os valores dos coeficientes de maneira adequada e, fundamentalmente, manipular o cálculo das funções de busca envolvidas.

2.2.1. Fator de inércia *w*

(ROSENDO, 2010) recomenda nivelar o custo de processamento da busca local no movimento M1 em relação ao *Path relinking*, de tal maneira esta busca não utilize um baixo números dos nós do problema como base para tais execuções. Além disso, o número de execuções deveria ser variável, portanto, *w* = (0,1] foi implementado como um limite para o algoritmo *2-opt*.

Sendo assim, toda vez que este algoritmo é chamado, com base no fator de inercia *w*, a seguinte equação é usada para definir o número de movimentos da busca:

$$m = (int)\, n * w$$

Onde *n* é o número de cidades do problema, consequentemente, e *m* se refere ao número de possíveis vértices para início das execuções do *2-opt*.

2.2.2. Coeficientes c1, c2 e c_{pr}

Segundo (GOLDBARG; GOLDBARG; LUNA, 2016), todo algoritmo PSO deve conter o coeficiente c1 (fator de individualidade) implementado como uma variável real que influencia a partícula a seguir a sua melhor posição já encontrada, e c2 (fator de sociabilidade) que influencia a partícula em direção a melhor posição já encontrada pela vizinhança.

(ROSENDO, 2010) define que os coeficientes c1 e c2 são valores reais predefinidos no início do algoritmo, sendo c1 = c2 = c = (0, 1), já os coeficientes φ1 e φ2 são gerados aleatoriamente a cada execução do *Path relinking* (φ = [0, 1)) de maneira distinta para cada movimento (M2 e M3). Assim, os coeficientes

atuam dentro do algoritmo *Path relinking* através da função *calculaPassos* que é representada pela seguinte equação:

$$return\ distExata(Pini, Pobj) * f(c, \varphi)$$

A função f utiliza os coeficiente *c* e φ para determinar o número de passos que serão percorridos entre a posição de origem (*Pini*) e destino (*Pobj*). Esta função pode ser implementada de diversas maneiras distintas, de acordo com o problema, mas (ROSENDO, 2010) sugere, neste caso, extrair a média aritmética dos dois valores

$$f = \frac{c + \varphi}{2}$$

Por fim, Assim, de maneira empírica, (ROSENDO, 2010) define os valores do coeficiente c_{pr} sendo 0.2 e 1, pois estes apresentaram maior impacto positivo no desempenho do algoritmo.

2.3. BUSCA LOCAL 2-OPT

A busca local é uma técnica de resolução de problemas de otimização através de sucessivas pequenas mudanças em uma possível solução.

Algoritmos de busca local são utilizados em problemas de otimização computacionalmente complexos. Este tipo de algoritmo, geralmente, atua sobre uma solução pré-existente (s) fazendo consecutivas alterações na tentativa de obter uma solução melhor para o problema em questão.

Para este trabalho, o algoritmo utilizado para a busca local foi o *2-opt*. Este é o método de busca mais simples para o problema do caixeiro viajante, sendo a vizinhança N(S) da solução S definida pelo conjunto de soluções que podem ser alcançadas através da permutação de duas arestas não-adjacentes em S. Este movimento é chamado de 2-interchange e é descrito na figura abaixo:

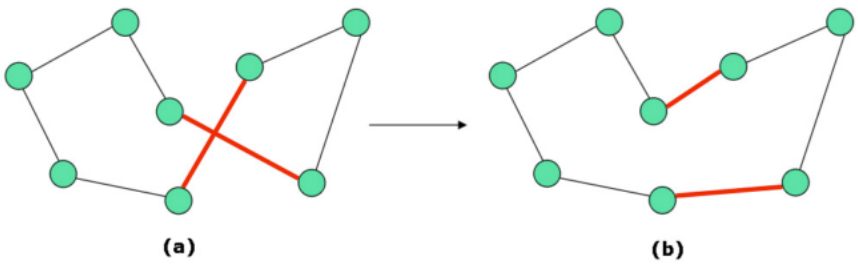

Figura 3: Um caminho (a) antes e (b) depois do movimento *2-interchange*.

Vale ressaltar que dependendo do tipo de implementação de busca local todas as trocas possíveis são efetuadas antes de ser tomada a decisão, ou pelo menos até que se encontre a primeira troca que melhore a solução, se essa troca existir.

Na abordagem desenvolvida, esta busca é limitada pelo fator w, o qual garante a restrição das trocas em busca de um indivíduo de menor custo.

Em resumo, o algoritmo busca a sequência de arestas {x1, x2, ..., xk} que, quando trocadas pelas arestas {y1, y2, ...,, yk}, retornam um caminho factível e de menor custo.

Para este trabalho, foi utilizado o algoritmo 2-opt proposto por (WANTCHAPONG KONGKAEW; 2012), o qual foi levemente modificado a fim de atender a restrição imposta pelo coeficiente *w*, de acordo com a função m anteriormente detalhada.

2.4. PATH RELINKING

O Path relinking foi sugerido originalmente como uma abordagem para integrar estratégias de intensificação e diversificação.

O algoritmo consiste na geração de um caminho de soluções intermediárias entre uma solução inicial x′ e uma solução objetivo x″. Tal solução (x″) deve pertencer à vizinhança de x′ e é chamada na literatura solução guia (*guiding solution*). Para gerar tal caminho, é necessário apenas que sejam aplicados progressivamente movimentos a x′ de forma a reduzir a distância entre esta e a solução objetivo x″. Assim, o caminho é formado a partir da sequência de soluções x′ = x(l), x(2), ...,x(r) = x″, tal que a solução x(i + 1) é criada a partir de x(i) a cada passo, escolhendo um movimento que deixe um número reduzido de movimentos restantes para atingir x".

Neste trabalho, o *Path relinking* foi utilizado na implementação dos movimentos M2 e M3. A versão utilizada se baseia na proposta a qual F(x″) < F(x′), ou seja, busca a melhor solução a partir de avanços em uma solução base (GOLDBARG; GOLDBARG; LUNA, 2016). Quanto à adaptação ao PSO, naturalmente x′ foi implementado como a posição atual da partícula (*Pini*) e x″ como a posição da partícula objetivo (*Pobj*). Assim, *Pobj* representa uma das duas posições:

$$Pobj = \begin{cases} Pbest, & se\ movimento = M2 \\ Lbest, & se\ movimento = M3 \end{cases}$$

Antes de executar qualquer movimento, ou passo, na *Pini* em direção a *Pobj*, a distância entre as duas soluções deve ser calculada. Existem várias funções que

podem ser utilizadas para o cálculo de distância entre soluções de problemas discretos. A função utilizada nesse trabalho foi a função de distância exata (*distExata*), *exact match distance*, descrita em Ronald (1998). Este é um método considerado computacionalmente barato no qual a distância é definida, basicamente, como o total de dimensões cujos valores são diferentes entre si. Por exemplo:

$$Pini = \{1, 7, \mathbf{5}, 6, \mathbf{2}, 8, \mathbf{4}, \mathbf{3}, 1\}$$
$$Pobj = \{1, 7, \mathbf{2}, 6, \mathbf{4}, 8, \mathbf{3}, \mathbf{5}, 1\}$$
$$distExata(Pini, Pobj) = 4$$

O movimento ocorre a partir de uma alteração em uma dada dimensão de *Pini* e esta alteração é aplicada de acordo com o problema em questão. Apesar disso, o método de cálculo da quantidade de movimentos que serão aplicados não difere, este método é chamado *calculaPassos* e sua descrição, juntamente com os parâmetros c e φ, é apresentada na próxima seção (5.3).

Durante a execução do *Path relinking*, a cada alteração na solução atual, uma nova solução é criada. Mas isso não significa que toda solução intermediária precisa ser avaliada. Assim, o parâmetro c_{pr} = (0, 1] foi criado. Este coeficiente limita o número de soluções intermediárias que são avaliadas durante o Path relinking. Portanto, quanto maior o valor de c_{pr}, mais soluções avaliadas haverá entre *Pini* e *Pobj*. A ideia é tornar o algoritmo proposto mais eficiente em termos computacionais, já que a reavaliação completa da solução a cada alteração pode representar um custo computacional significativo dependendo do problema que está sendo abordado. Além disso, o estabelecimento deste parâmetro faz com que as ações tomadas no *Path relinking* seja limitada a um fator constante, conforme proposto no PSO clássico (c1 e c2) para os movimentos da partícula.

O pseudocódigo do algoritmo abaixo mostra como o *Path relinking* foi definido neste trabalho.

1. distancia = *distExata(Pini, Pobj)*
2. totalPassos = *calculaPassos(distancia, c,φ)*
3. avalicoes = 0
4. intervalo = $1/c_{pr}$
5. **for** passo = 1 até totalPassos **do**
    a. escolhe dimensão d
    b. aplica alteração em $Pini_d$
    c. **if** passo >= intervalo ∗ avaliacoes **then**
        i. avalia Pini
        ii. **if** Pini melhor que Pbest then
            1. Pbest = Pini
        iii. **end if**
        iv. avaliacoes++
    d. **end if**
6. **end for**
7. **return** Pini

Algoritmo 3 – *Pseudocódigo Path Relinking*

Fonte: (ROSENDO, 2010)

Primeiramente, a distância, o número de passos que serão executados e o intervalo entre as avaliações (de acordo com c_{pr}) são calculados. Logo após começam as iterações, a cada passo, uma alteração é aplicada a *Pini* (linha 5.b). Na sequência, o algoritmo verifica se o intervalo de passos sem avaliar a solução foi atingido (linha 5.c), em caso positivo *Pini* é avaliada, então é verificado se esta nova posição é melhor que *Pbest* (linha 5.c.ii). Em caso positivo a posição atual de *Pini* é guardada como *Pbest* e, consequentemente, utilizada posteriormente como *Pobj* em um novo processamento do método *Path relinking*, depois disso o contador de avaliações é incrementado na (linha 5.c.iv). Após todas as iterações, a posição alcançada ao final da última troca é retornada como a nova posição da partícula.

Conforme (ROSENDO, 2010), como o TSP é um problema modelado em forma de grafo, é impossível modificar apenas uma dimensão da partícula por alteração mantendo a viabilidade da solução. Assim, para executar cada alteração a fim de movimentar *Pini* em direção a *Pobj*, foi utilizado o operador de troca (swap).

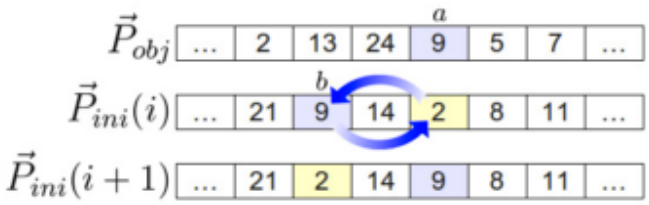

***Figura 4** – Operação de Swap dentro do Path relinking*

Como ilustrado na *figura 4*, o procedimento de aplicação deste operador é relativamente simples, uma vez que num determinado passo (iteração) *i* do *Path relinking*, uma dimensão aleatória *a* é escolhida tal que $Pobj_a \neq Pini_a$, depois é encontrada a dimensão *b* tal que $Pini_b = Pobj_b$ e então são trocados os valores contidos em $Pini_a$ e $Pini_b$.

Vale ressaltar que, no caso do TSP, a função de distância exata não é capaz de fornecer o número de passos de distância entre a solução inicial e final, visto que, em uma única troca duas dimensões de *Pini* podem assumir o mesmo valor de *Pobj*, como evidencia a a figura 4, no caso de $Pobj_b = 2$. E a probabilidade de isto ocorrer aumenta de acordo com o número de operações já realizadas, pois quanto mais trocas aplicadas à *Pini*, mais esta será semelhante à *Pobj*.

Portanto, na implementação para o TSP a distância calculada pela função de distância exata é uma estimativa do máximo de trocas consecutivas necessário para que *Pini* se transforme em *Pobj*.

O fato do algoritmo não calcular previamente com exatidão o número de trocas que uma solução se encontra da outra não acarreta problemas para geração das soluções, visto que a própria equação que calcula o número de passos leva em conta o coeficiente aleatório φ, cuja função é justamente variar a quantidade de trocas aplicadas à partícula a cada iteração.

3. METODOLOGIA

Os algoritmos foram desenvolvidos utilizando a ferramenta MATLAB 2015a, testados sob o sistema operacional OSX El Captain 10.11.6, numa máquina com processador de 2.26GHz Intel Core 2 Duo, memória RAM de 8gb 1067MHz DDR3 e placa de vídeos NVIDIA GeForce 9400 256MB e 1Tb de HD.

3.1. DEFINIÇÕES E REGRAS DO PCV

O PCV não permite rotas com valores duplicados, ou seja, cada cidade deve ser visitada apenas uma vez e a rota deve ser encerrada na cidade de partida inicial.

3.2. CONFIGURAÇÕES DE TESTES DO ALGORITMO

A tabela 1 mostra os valores que foram testados nos experimentos do algoritmo PSO proposto neste trabalho e os valores em negrito representam a configuração padrão inicial do algoritmo para tais experimentos. Estes valores foram extraídos do trabalho de (ROSENDO, 2010), no qual o autor verificou que o algoritmo apresentou maior sensibilidade a este conjunto.

Parâmetro	Valores testados
Iterações	10
Tamanho do enxame	30
c_{pr}	0.2; 1
w	0.2; 0.5; 0.9
c1 e c2	0.2; 0.5; 0.8

Tabela 1: Parâmetros testados nos experimentos de sensibilidade

4. RESULTADOS

Para a avaliação do PCV os dados usados foram os do problema EIL51.tsp, disponibilizado pela TSPLIB (TSPLIB, 2010), que possui tamanho de 51 nós. Os resultados obtidos variavam de acordo com mudanças na quantidade de gerações e no tamanho da população de indivíduos.

Conforme o resumo geral sobre a performance deste PSO apresentado abaixo, (reportado de forma completa no ANEXO I), o melhor resultado obtido para o PCV aplicado ao conjunto de dados EIL51.tsp foi **429.48**:

Min.	*1st Qu.*	*Median*	*Mean*	*3rd Qu.*	*Max*
429.48	436.1	438.5	452.7	472.4	524.9

Tabela 2: Resumo geral de resultados do algoritmo proposto.

É interessante notar que este resultado está a apenas **3.48** pontos do melhor resultado conhecido para este conjunto de dados.

Também vale ressaltar que a mediana dos resultados obtidos situou em **438.5**, atingindo um resultado abaixo de 10% em comparação à meta – benchmark estabelecido para

o conjunto de dados EIL51- conforme pode ser vista representada pela linha vertical vermelha no gráfico de distribuição dos resultados abaixo:

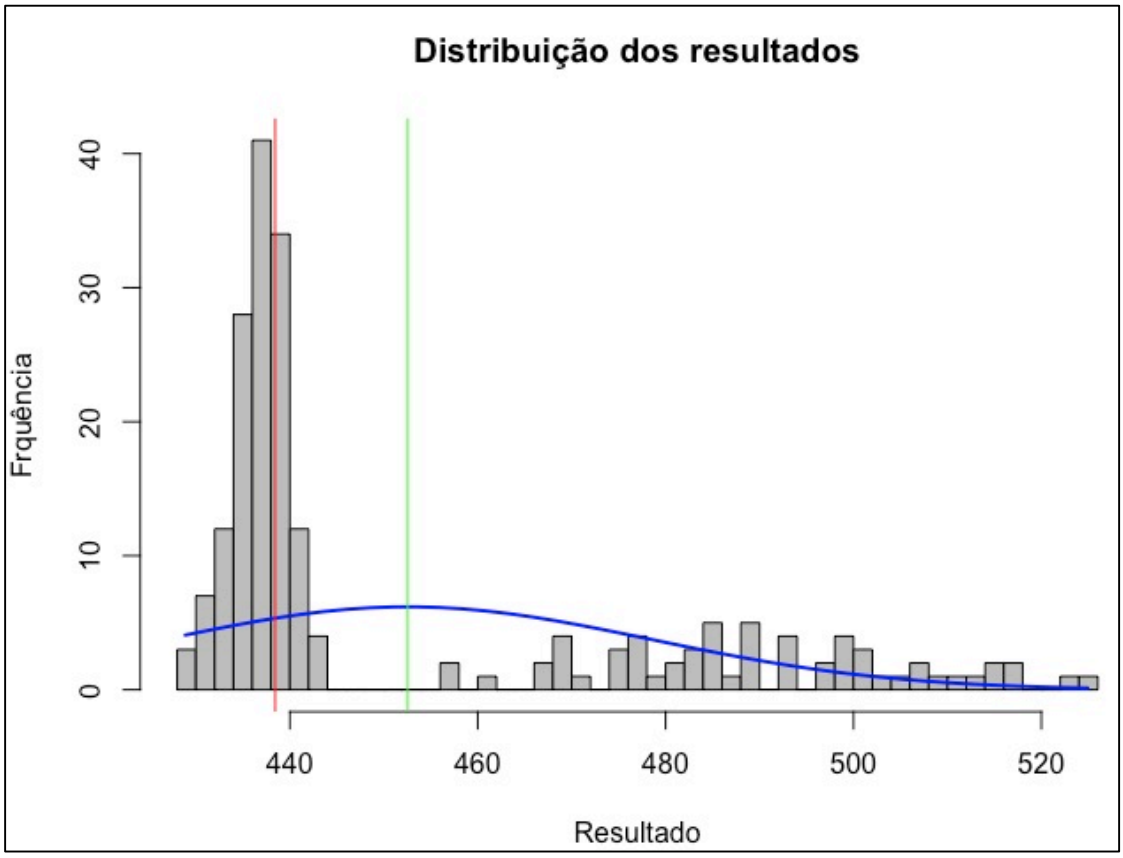

Figura 5 - Distribuição dos resultados de testes do algoritmo. Linha vermelha representa o valor da mediana no eixo x; a linha verde representa o valor médio.

Conforme o resumo exibido na tabela 2, é possível notar a integridade do algoritmo proposto. Tal característica fica evidente, quando se observa que 75% dos resultados situaram entre a mínima rota obtida (429.48) e o valor de 472.35, para todas as combinações de parâmetros envolvidos nos testes do algoritmo; um valor aceitável dentro do escopo de avaliação proposto para o trabalho, ou seja, estar numa margem de 10% do melhor resultado encontrado na literatura para o conjunto de dados EIL51. Abaixo, é possível observar com mais detalhes a distribuição dos resultados do conjunto de testes obtidos:

Prob.	**< 1%**	**25%**	**50%**	**75%**	**100%**
Resultado	429.48	436.09	438.49	472.35	524.91

Tabela 3 – Análise das probabilidades de resultados do algoritmo

Interessante notar que o melhor resultado encontrado ocorre, em geral, menos de 1%, o que podemos constatar uma casualidade, de fato comprovado ao analisarmos o conjunto de dados resultante, e vemos que tal rota ocorreu apenas duas vezes, como vemos abaixo:

ID	*NUMITER*	*SWARMSIZE*	*W*	*C1*	*C2*	*CPR*	*BESTROUTE*	*TIME (s)*
55	10	30	0.9	0.5	0.5	0.2	429.48	157.99
60	10	30	0.9	0.5	0.5	0.2	429.48	160.29

Tabela 4 – Duplicidade do melhor resultado obtido pelo algoritmo

Na figura 6, é possível observar o comportamento do algoritmo em busca da menor rota, para quando o resultado obtido foi o melhor dentre todos encontrados nos testes (rota mínima observada):

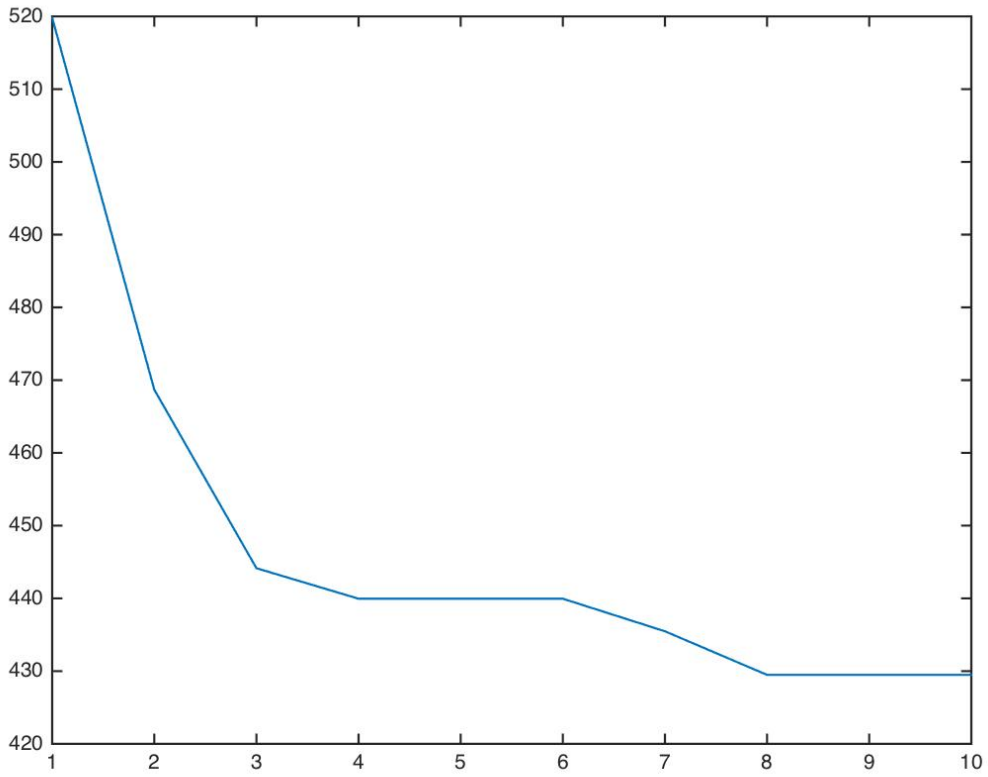

Figura 6: Relação entre fitness (caminho, eixo y) e iterações (eixo x) do algoritmo quando este obteve o melhor resultado no conjunto de dados.

Em tempo, a distribuição visual do caminho encontrado pode ser visualizada a seguir:

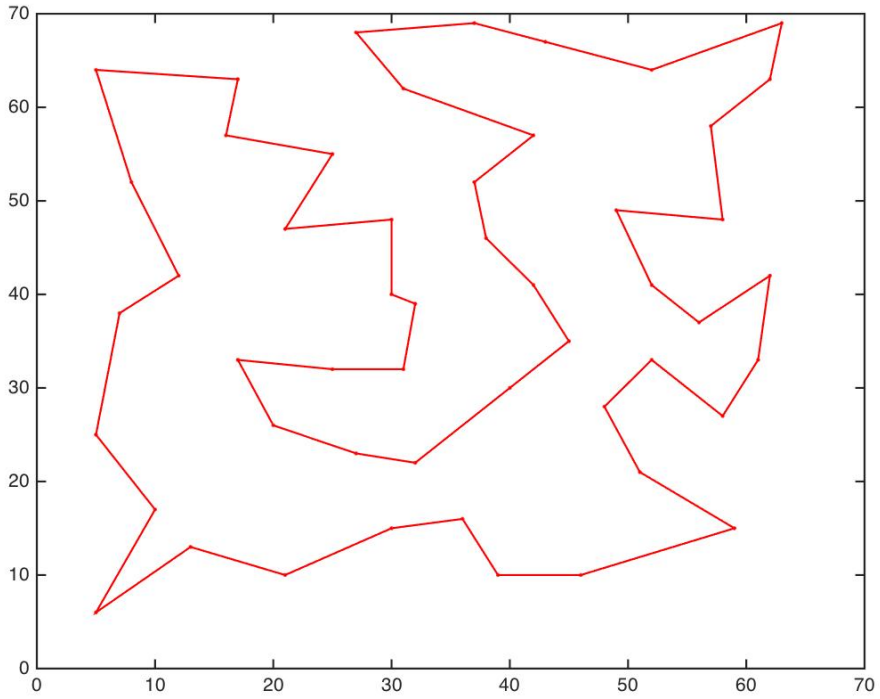

Figura 7: Representação do melhor caminho encontrado pelo PSO.

Como pressuposto fundamental para a eficiente execução do algoritmo PSO, inclusive o desenvolvido neste trabalho, a correta configuração de parâmetros garante maiores chances de êxito na resolução do problema de otimização. Partindo desta premissa, assumindo os dados de testes resultante, podemos extrair um gráfico de correlação de variáveis a fim de detectar a sensibilidade do algoritmo ao ajuste destes parâmetros, o qual podemos visualizar abaixo:

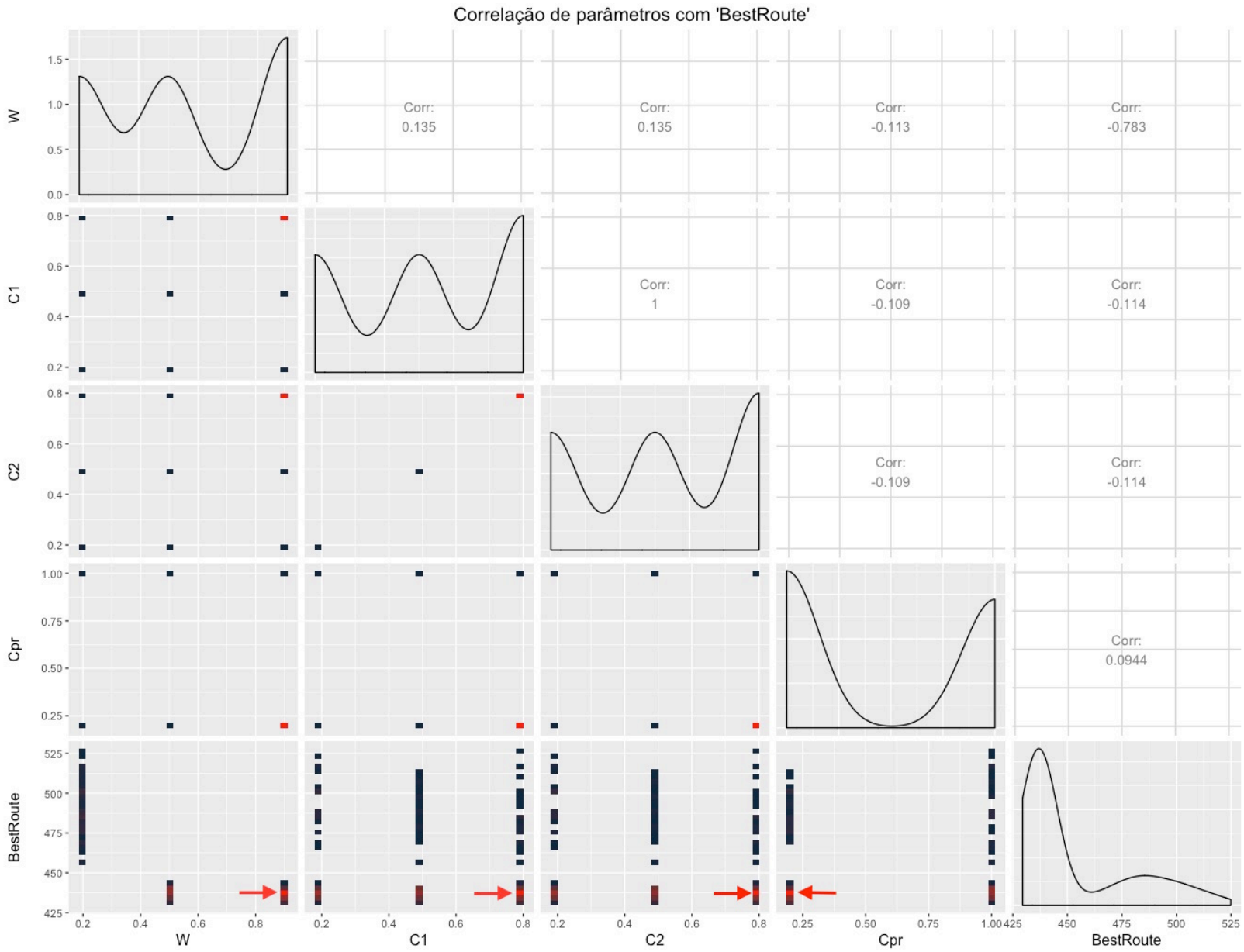

Figura 8: Gráfico de correlação de parâmetros, exibindo sensibilidade do atributo ‘BestRoute’ em relação aos demais analisados.

Assim, fica evidente que o resultado, representado pela variável 'BestRoute', possui maior sensibilidade a certos valores das variáveis de configuração do algoritmo, apresentando resultados geralmente próximos aos melhores (mínimo) de acordo com a combinação de valores dos parâmetros, indicados no gráfico pela seta vermelha em cada atributo em questão.

Portanto, de acordo com o algoritmo PSO desenvolvido, os testes realizados e os dados aqui apresentados, o melhor comportamento possível deste ocorre quando os parâmetros são ajustados conforme a tabela abaixo:

Parâmetro	**Valor ideal**
Iterações	10
Tamanho do enxame	30
c_{pr}	0.2
w	0.9
c1 e c2	0.8

Tabela 5: Parâmetros testados nos experimentos de sensibilidade

5. CONCLUSÃO

O algoritmo PSO clássico se mostra restrito a problemas de otimização contínua, o que inviabiliza sua aplicação sem modificações em contextos discretos, como o PSO. Todavia, há na literatura inúmeros trabalhos que pesquisam abordagens inspiradas em nuvens de partículas para a resolução do PCV. Tais abordagens levam em consideração o desenvolvimento híbrido do PSO se apoiando em meta-heurísticas de busca local e otimização combinatória para realizar os movimentos das partículas de maneira eficiente.

Conforme demonstrado neste trabalho, a partir destas adaptações, utilizando busca local *2-opt* para o movimento do indivíduo e *Path relinking* para o movimento da vizinhança, o PSO apresenta resultado satisfatório comparado a outros algoritmos disponíveis.

Como próximos trabalhos, no contexto do PSO proposto, outras estratégias poderiam ser implementadas, como estabelecer um valor dinâmico para o coeficiente *w,* conforme (ROSENDO, 2010) propõe, o que leva a decrescer uniformemente o valor desta a cada iteração. Outra evolução plausível pode ser utilizar um algoritmo mais consistente para realizar a busca local, como por exemplo o *LK-concorder*, no qual diversas literaturas apontam este como sendo mais eficiente para o PCV. Além disso, o *Path relinking* implementado neste foi o mais simples disponível na literatura – baseado em avanço – que pode ser facilmente substituído por versões mais complexas a fim de avaliar o impacto que teria no cenário do PSO aplicado a problemas discretos.

REFERÊNCIAS

GOLDBARG, M. C.; GOLDBARG, E. G.; LUNA, H. P. L. **Otimização combinatória e meta-heurísticas: algoritmos e aplicações**. 1. ed. ed. Rio de Janeiro: Elsevier, 2016.

ROSENDO, M. **Um Algortimo de otimização por nuvem de particulas para resolução de problemas combinatorios**. [s.l: s.n.].

SHI, X. H. et al. Particle swarm optimization-based algorithms for TSP and generalized TSP. **Information Processing Letters**, v. 103, n. 5, p. 169–176, 2007.

WANTCHAPONG KONGKAEW 2-OPT ALGORITHM (2012). Disponível em < https://www.mathworks.com/matlabcentral/fileexchange/38758-an-algorithm-based-on-gaussian-process-regression-model-for-the-traveling-salesman-problem/content/TSP_GPR/opt_2.m> Acesso em 27 set. 2016.

TSPLIB - Traveling Salesman Problem Library. Disponível em: <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/> Acesso em 30 ago. 2016.

ANEXO I - ALGORITMOS DESENVOLVIDOS E DEMAIS INSUMOS

O código-fonte, resultados, análise dos dados (linguagem R), e demais arquivos desenvolvidos neste trabalho estão disponíveis em https://github.com/diegocavalca/mestrado/tree/master/CCO-727/Trabalho2-PSO_TSP