Um estudo sobre Otimização por Partículas aplicado ao problema de roteamento de veículos com demandas estocásticas

Vinícius Renan de Carvalho
Departamento de Informática(Dinf)
Universidade Federal do Paraná
Curitiba, Brasil
vrcarvalho@inf.ufpr.br

Aurora Trinidad Ramirez Pozo
Departamento de Informática(Dinf)
Universidade Federal do Paraná
Curitiba, Brasil
aurora@inf.ufpr.br

Abstract— Este artigo estuda algumas propostas da literatura da aplicação do PSO) ao problema VRP. O VRP (Vehicle Routing Problem) é um problema considerado NP - Difícil, e devido a este fato diversas técnicas heurística e meta-heurísticas tem sido utilizadas a fim de encontrar soluções de alta qualidade em um razoável tempo computacional. Dentre estas heurísticas podemos destacar o PSO (Particle Swarm Optimization) é considerada uma técnica bem atraente devido a sua simplicidade de implementação e possuir um bom desempenho na busca por soluções.

O PSO possui uma grande aplicabilidade, e devido a isso tem sido aplicado em contextos discretos ampliando assim as possibilidades do algoritmo. Entretanto, em sua formulação original o algoritmo trabalharia somente com valores no contexto contínuo devido às suas equações. Sendo assim foi-se necessário adaptar as funções em questão para o espaço discreto. Clerc propôs a mudança nos operadores das funções de velocidade e movimento enquanto Marinakis manteve a função de velocidade original, mas substituindo a função de movimento pelo uso do Path Relinking, fazendo com que as soluções atuais sejam conectadas as melhores soluções, e o uso do VNS (Variable Neighbourhood Search) para exploração da partícula no espaço de soluções.

Keywords— Inteligência Artificial, PSO; CENTPSO; CNTPSO; VRP; SVRP; VRPSD

I. INTRODUÇÃO

O *VRP* (*Vehicle Routing Problem*) é descrito como um problema onde veículos partindo de um mesmo ponto de origem tem como objetivo atender a demanda de clientes dispersos geograficamente. Existem porém alguns modelos de VRP em que alguma variável pode ser estocástica como por exemplo o *VRPSD* (trabalhado neste artigo), onde a demanda do cliente é apenas conhecida no momento em que esta é atendida.

Problemas VRP são considerados problemas *NP-Difícil*, e desta maneira requerem alto custo computacional. Assim torna-se necessário o uso de heurísticas e meta- heurísticas para encontrar soluções de qualidade em tempo razoável.

Dentre elas podemos destacar o *PSO (Particle Swarm Optimization)* é um algoritmo inicialmente proposto por Kennedy e Eberhart, e simula o comportamento social de organismos sociais pelo uso de movimentos que tendem a seguir os melhores indivíduos, seguir sua melhor memória ou procurar recursos pela exploração do espaço existente.

Com o passar dos anos diversas modificações foram propostas para o *PSO*, que inicialmente foi idealizado para problemas contínuos, para atender a problemas discretos. Dentre eles podemos ressaltar o *TSP* (*Travel Salesman Problem* ou Problema do Caixeiro Viajante) e o *VRP*, algoritmo que este artigo busca trabalhar.

No contexto do *VRP* temos adaptações do PSO Discreto de Clerc [6] que inicialmente foi proposto para o *TSP* e aplicado posteriormente para o *VRP*, o *CNTPSO* que apresenta uma nova forma de trabalhar o movimento das partículas através do *Path Relinking* e *VNS(Variable Neighbourhood Search)*, e o *CENTPSO* que possui diversos atributos do *CNTPSO* e trabalha com vizinhança expansiva de forma semelhante ao *PSOENT* [3].

Este artigo tem como objetivo realizar um experimento comparando os algoritmos PSO, CNTPSO e CENTPSO utilizando 15 problemas VRP (com tamanho entre 16 e 60 nós e com diferentes quantidades de veículos utilizados) e executando cada experimento 40 vezes para obter a média para cada problema em cada algoritmo.

Este artigo é organizado da seguinte forma: Na seção 2 o VRPSD é descrito. Na seção 3 o PSO e suas variantes são mostradas. Na seção 4 é mostrada a metodologia de implementação empregada. Finalmente na seção 5 são apresentados os resultados obtidos

II. PROBLEMA DE ROTEAMENTO DE VEÍCULOS

O Problema de Roteamento de Veículos (PRV) também conhecido como *Vehicle Routing Problem(VRP)* é um problema considerado NP-Difícil, e devido a este fato diversas técnicas heurística e meta-heurísticas tem sido utilizadas a fim de encontrar soluções de alta qualidade em um razoável tempo computacional [1].

Neste problema veículos com capacidade finita tem como objetivo visitar apenas uma vez cada cliente partindo de um ponto de origem denominado depósito central. Os clientes estão geograficamente dispersos e cada um deles possui uma demanda a ser atendida, sendo que esta é conhecida previamente. O veículo parte de um depósito e visita cada cliente exatamente uma vez e retorna para o depósito. Isto é chamado de uma priori tour [8].

Uma variação deste problema é o Problema Estocástico de Roteamento de Veículos, também conhecido *como Stochastic Vehicle Routing Problems (SVRPs)*, onde algum parâmetro como cliente ou demanda dos clientes é uma variável estocástica que segue uma probabilidade que pode ou não ser conhecida. Neste grupo de problemas temos o *VRPSD (VRP with Stochastic Demand)* ou em português PRVDE (PRV com Demandas Estocásticas) onde a demanda do consumidor é apenas conhecida quando o veículo chega ao cliente, sendo que esta é a maior diferença entre o *VRP* e o *VRPSD*, onde todas as demandas são conhecidas previamente.

Um exemplo de solução para este problema é dado pelo caminho associado a um determinado veículo, onde esta solução possui um valor de distância total, no problema esta distância deve ser minimizada.

Segundo [8] O *VRPSD* é definido como um grafo completo $G = (V, A, D)$,

onde:

• $V = \{0, 1, ..., n\}$ é um conjunto de nós onde o nó 0 é o depósito,

• $A = \{(i, j): i, j \in V, i = j\}$ é o conjunto de arcos,

• $D = \{d_{ij} : i, j \in V, i = j\}$ é a distância entre o nó i e o nó j.

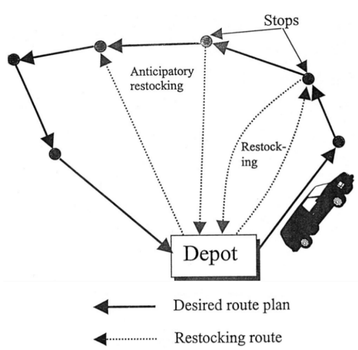

Fig1. Exemplo de VRP. [5]

Um veículo com capacidade *Q* tem que entregar produtos para os clientes baseado em suas demandas minimizando o tamanho total esperado baseado no seguinte [8]

- As demandas dos consumidores $(\xi_i, \; i = 1, ..., n)$ são variáveis estocásticas independentemente distribuídas com distribuições conhecidas.
- A demanda real de cada consumidor é conhecida apenas quando o veículo chega no cliente.
- A demanda $\xi_i, \; i = 1, ..., n$ não excede a capacidade do veículo Q, e segue uma probabilidade discreta de distribuição $p_{ik} = Prob(\xi_i = k), k = 0, 1, 2, ..., K \leq Q$.

Em uma priori tour, como pode ser visto em Fig.1, caso um veículo não possua capacidade suficiente para realizar a entrega a um determinado cliente, o mesmo deve retornar ao depósito para reabastecimento. Existe o caso porém de reabastecimento preventivo, onde antes de o veículo seguir para o próximo cliente ele volta para o depósito para reabastecer e em seguida segue para o cliente em questão.

Para calcular a distância esperada, são utilizadas as seguintes fórmulas:

$$f_j(q) = Minimum(f_j^p(q), f_j^r(q)) \quad (1)$$

$$f_j^p(q) = d_{j,j+1} + \sum_{k \leq q} f_{j+1}(q-k)p_{j+1,k} + \sum_{k>q} [2d_{j+1,0} + f_{j+1}(q+Q-k)]p_{j+1,k} \quad (2)$$

$$f_j^r(q) = d_{j,0} + d_{0,j+1} + \sum_{k=1}^{K} f_{j+1}(Q-k)p_{j+1,k} \quad (3)$$

Sendo:

- $f_j(q)$ o custo esperado partindo do consumidor j para diante (assim, se j =0 então $f_0(q)$ denota o custo esperado para a priori tour).
- $f_j^p(q)$ o custo esperado da rota quando o veículo não retorna para o deposto, mas vai para o próximo cliente
- $f_j^r(q)$ o custo esperado quando o veículo retorna para o depósito para reabastecimento preventivo.

III. PSO

A. *Particle Swarm Optimization*

O PSO(Particle Swarm Optimization) foi inicialmente proposto por Kennedy e Eberhart e é um algoritmo inspirado no comportamento e na dinâmica dos movimentos dos pássaros, insetos e peixes.

Este algoritmo é composto por partículas, sendo que cada partícula representa uma solução no espaço de soluções e se movimenta buscando otimizar uma função de aptidão, função esta que é predefinida e é relacionada ao problema a ser resolvido.

Cada partícula possui três parâmetros que definem qual movimento será adotado pela mesma, sendo estes o fator de sociabilidade, que determina o quanto a melhor partícula do enxame irá influenciar o movimento da partícula em questão; fator de individualidade que determina o quanto a partícula seguirá a sua melhor posição já atingida e a inércia que é a força com que a partícula seguirá para a descoberta de uma nova solução.

Estes fatores são aplicados na obtenção da velocidade da partícula, a qual determinará para qual posição a partícula irá

A velocidade de uma partícula no PSO é dada pela seguinte equação:

$$V_{ij}\,(t+1) = W * V_{ij}\,(t) + c1 * rand1 * (pbest_{ij} - x_{ij}(t)) + c2 * rand2 * (gbest_{ij} - x_{ij}(t)) \qquad (4)$$

Onde:

c1 é uma constante positiva aplicada ao fator cognitivo

c2 é uma constante positiva aplicada ao aprendizado social

W é o fator de é uma constante positiva aplicada a velocidade anterior.

rand1 e **rand2** são valores aleatórios no intervalo [0,1].

pbest é a melhor posição já visitada por esta partícula.

lbest é a melhor posição já visitada pelo enxame.

A partir da velocidade calculada é possível realizar o movimento com a seguinte equação:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \qquad (5)$$

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[7].

Com o passar do tempo melhoras foram acrescentadas ao PSO, sendo estas utilizadas nos algoritmos tratados. Dentre elas temos:

Redução linear da ponderação da Inércia

$$c_1 = c_{1,\min} + \frac{c_{1,\max} - c_{1,\min}}{Iter_{\max}} * t \qquad (6)$$

$$c_2 = c_{2,\min} + \frac{c_{2,\max} - c_{2,\min}}{Iter_{\max}} * t \qquad (7)$$

Fator de contrição modificando a função de velocidade para a equação (8).

$$V_{ij}\,(t+1) = X(V_{ij}\,(t) + c1 * rand1 * (pbest_{ij} - x_{ij}(t)) + c2 * rand2 * (gbest_{ij} - x_{ij}(t))) \;) \qquad (8)$$

onde:

$$X = \frac{2}{\left|2 - c - \sqrt{c^2 - 4c}\right|} \; e \; c = c_1 + c_2 > 4 \qquad (9)$$

B. *PSO Discreto*

Devido a sua grande aplicabilidade o PSO tem sido aplicado em contextos discretos ampliando assim as possibilidades do algoritmo. Entretanto, em sua formulação original, o algoritmo trabalharia somente com valores no contexto contínuo devido às suas equações. Sendo assim foi-se necessário adaptar as funções em questão para o espaço discreto.

Neste artigo trabalhamos com a implementação de *Clerc* [6], que utiliza a mesma função de velocidade do *PSO* modificando os operadores matemáticos nela empregada, passando a chamar funções que terão um comportamento diferenciado. Sendo subtração entre posições, multiplicação entre coeficiente e velocidade e adição entre velocidades (concatenação entre as listas). A partir disto a função de velocidade passa a retornar um operador que, quando aplicado na função de movimentação realiza a troca de posições no espaço da solução. Sendo assim, a velocidade passa a ter o seguinte formato:

$$v_{i=\{(i_1,j_1),(i_2,j_2),\ldots,(i_n,j_n)\}} \qquad (10)$$

Seja c um coeficiente e v uma velocidade. Então são possíveis quatro casos, dependendo do valor de c [7]:

i) c == 0, teremos c*v = ∅;

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.

A modificação dos operadores também altera o comportamento da função de movimento, mas a mesma ainda preserva sua descrição original. Um exemplo desta aplicação seria:

$$\begin{aligned} x_i &= (1, 2, 3, 4, 5, 1) \\ v_i &= \{(1, 2), (2, 3)\} \\ x_i' &= (3, 1, 2, 4, 5, 3) \end{aligned} \qquad (11)$$

C. *CNTPSO*

O CNTPSO *(Combinatorial Neighbourhood Topology PSO)* foi proposto por Marinakis e Marinaki [2] para solucionar o problema CVRP. Este algoritmo utiliza-se da equação (8) para cálculo de velocidade, mas trabalha sua movimentação utilizando-se do *Path Relinking* e *VNS (Variable Neighbourhood Search).*

Para determinar qual movimento a partícula deve seguir, o algoritmo primeiramente calcula a média das velocidades existentes pela seguinte equação:

$$average_v = \frac{\sum_{j=1}^{d} v_{ij}(t+1)}{d} \qquad (12)$$

Este valor é utilizado para comparar com os resultados das equações (13) e (14) e seu valor determina qual movimento será executado. Os movimentos são *NOPR* (*No Path Relinking*), quando o valor da média é inferior a *L1* (13), *PRPB (Path Relinking with Personal Best*), quando a média está entre os valores de L1 e L2 (14) e *PRGB*(*Path Relinking with Global Best*) quando a média for superior a *L2*.

$$L_1 = (u_{bound} - l_{bound)} * (w_1 - \frac{w_1}{Iter_{max}} * t) + l_{bound)} \qquad (13)$$

$$L_2 = (u_{bound} - l_{bound)} * (w_2 - \frac{w_2}{2*Iter_{max}} * t) + l_{bound)} \qquad (14)$$

Onde t é a iteração corrente, $iter_{max}$ é o número máximo de iterações, U_{bound} e L_{bound} são respectivamente maiores e menores limites para cada partícula.

Normalmente na literatura, os valores U_{bound} e L_{bound} são +4 e -4 respectivamente. Se em alguma iteração em um elemento da partícula o valor violar estes limites, então este elemento será inicializado com o novo valor que violou estes limites [1].

O *PR* gera novas soluções pela exploração de trajetórias que conectam soluções de alta qualidade partindo de uma destas soluções, chamada *starting solution*, e por gerar um caminho na vizinhança que leva para a outra solução, chamada *target solution*[1].

A seguir, temos um exemplo da execução do *Path Relinking* para uma instância com 10 nós onde a solução corrente, melhor solução pessoal e melhor solução global são apresentadas.

Current solution	1	2	3	4	5	6	7	8	9	10
Global Best	6	4	7	2	3	1	9	10	8	5
Personal Best	3	2	8	5	6	9	4	10	7	1
	1	2	**3**	**4**	**5**	*6*	*7*	**8**	**9**	**10**
1 iteration	1	2	3	4	5	6	7	8	9	10
2 iteration	1	2	3	4	5	6	7	8	9	10
3 iteration	1	2	**7**	4	5	6	**3**	8	9	10
4 iteration	1	2	7	4	5	6	3	8	9	10
5 iteration	1	2	7	4	**3**	6	**5**	8	9	10
6 iteration	1	2	7	4	3	**9**	5	8	**6**	10
7 iteration	1	2	7	**5**	3	9	**4**	8	6	10
8 iteration	1	2	7	5	3	9	4	**10**	6	**8**
9 iteration	1	2	7	5	3	9	4	10	**8**	**6**
10 iteration	1	2	7	**6**	3	9	4	10	8	**5**

Tab. 1. Exemplo de PR. [1]

Nesta tabela, na quarta linha, os números em **negrito** indicam que será executado o *PR* seguindo a melhor solução global (*PRGB*) para o item, caso esteja em *itálico* o PR deve seguir a melhor solução pessoal (*PRPB*), caso esteja em fonte normal, o *Path Relinking* não deve ser executado para o item.

A cada iteração, quando um item é trocado o mesmo é sinalizado em negrito indicando a troca.

Os nós 1 e 2 não passarão por *PR* e qualquer tentativa de troca com estes nós será impedida (como por exemplo na iteração 4). As outras trocas foram executadas como visto na *Tabela1*.

Nos itens não alterados pelo *PR* é executado o *VNS* utilizando os algoritmos *2-opt e 3-opt* a fim de procurar melhores soluções.

Sendo:

Iteração	Tipo	Troca
1	NOPR	-
2	NOPR	-
3	GBPR	3 -> 7
4	GBPR	Não executado
5	GBPR	5 -> 3
6	PBPR	6 -> 9
7	PBPR	5 -> 4
8	GBPR	8 -> 10
9	GBPR	6 -> 8
10	GBPR	6 -> 5

Tab. 2. Caminho do PR

D. CENTPSO

O CENTPSO (*Combinatorial Expanding Neighbourhood Topology PSO*) combina atributos de dois algoritmos sendo o *CNTPSO* e o PSOENT [3]. O *CENTPSO* herda do *CNTPSO* a forma com que este lida com o movimento das partículas e do PSOENT o modelo de vizinhança expansiva.

Este algoritmo combina a vantagem de uma topologia de busca local na vizinhança e a topologia de busca global na vizinhança. Esta topologia é denominada Topologia de Vizinhança Expansiva (*Expanding Neighbourhood Topology*) e foi inicialmente proposta em [3]. O algoritmo inicia com uma vizinhança de tamanho 2 e a cada iteração a vizinhança é incrementada até que seja igual ao número de partículas. Caso o número máximo seja atingido antes do fim do algoritmo o número retorna ao tamanho 2. Desta forma não haverá iterações consecutivas onde o tamanho da vizinhança seja o mesmo [1]. Com este procedimento cada partícula tem mais habilidades de exploração por se mover por um número de iterações em enxames pequenos e dentro de um grande enxame.

Tam	Enxames
3	{7,**1**,2}, {1,**2**,3}, {2,**3**,4},{3,**4**,5},{4,**5**,6}, {5,**6**,7}, {6,**7**,1}
5	{6,7,**1**,2,3}, {7,1,**2**,3,4}, {1,2,**3**,4,5}, {2,3,**4**,5,6}, {3,4,**5**,6,7}, {4,5,**6**,7,1}, {5,6,**7**,1,2}
7	{5,6,7,**1**,2,3,4}, {6,7,1,**2**,3,4,5}, {7,1,2,**3**,4,5,6}, {1,2,3,**4**,5,6,7}, {2,3,4,**5**,6,7,1}, {3,4,5,**6**,7,1,2}, {4,5,6,**7**,1,2,3}

Tab. 3. Exemplo de expansão

A tabela 3 mostra um exemplo de expansão onde Tam. é o tamanho do enxame e as partículas referência estão em negrito.

IV. METODOLOGIA

A. Implementação

Inicialmente os algoritmos recebem como parâmetro: um arquivo problema, a quantidade de iterações a serem executadas e a quantidade de partículas a serem utilizadas pelo problema. Para os algoritmos *CNTPSO* e *CENTPSO* a quantidade de iterações utilizada pelo *VNS* também é especificada.

Assim que o algoritmo inicia a sua execução, obtém-se lendo o arquivo qual a posição no eixo x e y para cada nó, a demanda para cada cliente, a quantidade de veículos existente para o problema e a capacidade de carga para cada veículo. Após isto são criadas partículas conforme a quantidade passada por parâmetro e iniciadas com valores aleatórios que não dupliquem um Nó na rota de algum veículo, ou aloquem um mesmo nó em mais de uma rota.

Cada partícula é formada por um vetor que contém rotas de veículos, sendo que cada rota possui um respectivo veículo e uma sequência de nós do grafo do problema, indicando por qual caminho o veículo em questão deve percorrer.

Deve se ressaltar que a função de avaliação utilizada é a função descrita na equação (1) e que a distância de uma cidade X até uma cidade Y é dada pela distância euclidiana entre as cidades em questão.

O PSO Discreto possui um ótimo global, um enxame de partículas com tamanho fixo e se utiliza da função (10) para cálculo de velocidade e da função (8), adaptada com os operadores já descritos, para movimentação e a cada iteração o ótimo global é atualizado até que o número de iterações máxima seja atingido.

O *CNTPSO* possui várias diferenças do *PSO* Discreto, primeiramente por usar a função (8) do *PSO* para calcular a velocidade, realizar a média (12) e se utilizar das equações (13) e (14) para definir qual ação será executada para cada item da solução (Tab.1 e Tab.2). Itens classificados como *NOPR* são adicionados a um vetor para posteriormente serem processados via *VNS*, itens *PRGB* ou *PRPB* são trocados pelas soluções correspondentes. Assim, caso a lista de itens *NOPR* não seja vazia, o VNS é executado um determinado número de vezes (passado por parâmetro). A lista *NOPR* existe para indicar quais itens podem ser mudados, impedindo assim que itens que já foram contemplados pelo *PRGB* ou *PRPB* não sejam modificados.

Caso a lista *NOPR* atenda o tamanho mínimo para executar a troca, os algoritmos 2-opt e 3-opt são executados e a cada iteração os novos resultados são comparados entre si baseados na equação (1) para verificar qual modificação obteve melhor fitness. Ao final do processo *VNS* obtém-se uma solução que será atribuída à partícula em questão. Após todas as partículas terem sido processadas, o algoritmo verifica as melhores soluções pessoais e globais, recalcula velocidades e inicia uma nova iteração caso a quantidade máxima de iterações não tenha sido atingida.

O *CENTPSO* é idêntico ao *CNTPSO* no que diz respeito a funções, *PR* e *VNS*, mas usa uma estratégia expansiva, onde a ao invés de um enxame utiliza *N* enxames, onde *N* é a quantidade de partículas [1,4], e devido a este fato necessita de um vetor de enxames. O tamanho de cada enxame é variável, iniciando com tamanho *2* até o tamanho que englobe todas as partículas, ou seja, *N*. Além disso cada enxame possui um ótimo local e dessa forma cada enxame é processado individualmente, onde a equação de velocidade (8) e o *Path Relinking* são executadas tomando como base o ótimo local.

B. Bases utilizadas

As bases utilizadas estão disponíveis em [9]. Cada base contém o posicionamento no plano cartesiano de cada cliente, a demanda de cada cliente, a quantidade de veículos empregada no problema e a distância total da solução ótima.

Estas bases foram também empregadas em [1, 2, 10] e devido a este fato é considerado interessante o seu uso.

	Problema	**Opt**	**Nós**	**Veículos**
1	A-n60-k9.vrp	1354	60	9
2	A-n32-k5.vrp	784	32	5
3	P-n50-k8.vrp	631	50	8
4	E-n51-k5.vrp	521	51	5
5	E-n33-k4.vrp	835	33	4
6	P-n60-k15.vrp	968	60	15
7	P-n19-k2.vrp	212	19	2
8	E-n22-k4.vrp	375	22	4
9	A-n48-k7.vrp	1073	48	7
10	P-n16-k8.vrp	450	16	8
11	A-n39-k6.vrp	831	39	6
12	P-n55-k7.vrp	568	55	7
13	A-n36-k5.vrp	799	36	5
14	P-n22-k2.vrp	216	22	2
15	A-n45-k7.vrp	1146	45	7

C. Métricas de avaliação

O problema visa a minimização do somatório da distância percorrida por todos os veículos, onde valores menores são considerados superiores. Este cálculo leva em consideração a quantidade de veículos e distâncias entre nós que são valores determinísticos e a demanda de cada cliente que por definição utiliza demandas estocásticas. Devido o uso de uma variável estocástica, para que comparações entre algoritmos possam ser realizadas deve se assumir valores determinísticos para demandas ao invés de estocásticos.

Para isto utilizamos as modificações propostas em [10].

- Todas as demandas dos clientes são assumidas como distribuídas por Poisson.
- Para cada cliente, coloca-se a demanda esperada igual a demanda determinística original do cliente
- Distâncias de rota, localizações de nós e capacidade de veículos são deixadas sem mudanças e determinísticas.

D. Recursos empregados

Neste artigo foram implementados os algoritmos *PSO* Discreto, *CNTPSO* e *CENTPSO* utilizando a linguagem Java com o *JDK* 7. Os experimentos foram executados utilizando 15 problemas com diferentes quantidades de nós e veículos utilizados. Cada problema foi executado 40 vezes em cada algoritmo em quatro computadores distintos com a configuração CPU 2.3 GHz, 1GB de RAM, utilizando *Ubuntu 14.04.*

A configuração dos algoritmos foi a seguinte:

	PSO	CNTPSO	CENTPSO
partículas	80	80	80
iterações	3500	3500	3500
c1,min = c2,min	2	2	2
c1,max = c2,max	5	5	5
w1	-	0.8	0.8
w2	-	0.9	0.9
VNS	-	5	5
ubound	4	4	4
lbound	-4	-4	-4

Tab. 4. Configuração dos algoritmos

V. Resultados e Discussão

Os resultados exibidos são as médias obtidas para cada problema conforme em (Tab. 5). Resultados mostraram superioridade do CENTPSO em 14 casos sendo que no problema 10 houve um empate entre os três algoritmos, CNTPSO em 11 casos foi superior ao PSO e em 3 casos o PSO foi superior ao CNTPSO.

Os melhores resultados de fitness apresentados foram obtidos para o VRP mais de 40 nós, sendo que o percentual médio de melhora em problemas com mais de 40 nós é em média de aproximadamente 7,64%, enquanto com menos de 40 nós é em média de aproximadamente 6,35% em relação ao segundo melhor resultado, seja ele obtido pelo PSO ou CNTPSO.

Com relação ao o tempo de execução é possível verificar que em todos os casos o CENPSO necessitou de maior tempo de execução seguido pelo CNTPSO e o PSO.

VI. Conclusão

Foi visualizado que o método CENTPSO proposto por Marinakis[1] mostrou superioridade em relação aos demais, que segundo o autor deve-se a capacidade de expansão do mesmo, permitindo assim explorar melhor o espaço de soluções, expansão não empregada no CNTPSO.

Entretanto o tempo necessário na execução do CNTPSO e seus resultados próximos do CENTPSO mostram que o algoritmo é uma excelente opção no que se diz respeito ao custo beneficio.

Uma possível melhora nos algoritmos CNTPSO e CENTPSO seria utilizar outros algoritmos diferentes do 2-opt e 3-opt devido ao fato de possibilitar uma melhor exploração do espaço da busca por estar diretamente relacionado com o fator de mutação do algoritmo.

Referencias

[1] Marinakis, Y. and M. Marinaki (2013). Combinatorial expanding neighborhood topology particle swarm optimization for the vehicle routing problem with stochastic demands. GECCO 2013: 49-56

[2] Marinakis, Y. and M. Marinaki (2013). Combinatorial Neighborhood Topology Particle Swarm Optimization Algorithm for the Vehicle Routing Problem, M. Middendorf and C. Blum (Eds.): EvoCOP 2013, LNCS 7832, pp. 133-144.

[3] Marinakis, Y. and M. Marinaki, (2013). Particle Swarm Optimization with Expanding Neighborhood Topology for the Permutation Flowshop Scheduling Problem, Soft Computing.

[4] Suganthan PN (1999) Particle swarm optimizer with neighborhood operator. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp 1958–1962

[5] Yang, W. H., Mathur, K., Ballou, R. H. (2000). Stochastic vehicle routing problem with restocking, Transportation Science, 34, 99-112.

[6] Clerc, M. (2004). Discrete Particle Swarm Optimization Illustrated by the Traveling Salesman Problem. New Optimization Techniques in Engineering 219-239

[7] Aloise J. A., Serafim M. C., Silva T. L. (2006). Otimização discreta por nuvem de partículas aplicada ao problema do caixeiro viajante, GEPROS, 2,87-95

[8] Bianchi, L., Birattari, M., Manfrin, M., Mastrolilli, M., Paquete, L., Rossi-Doria, O., Schiavinotto, T. (2006). Hybrid Metaheuristics for the Vehicle Routing Problem with Stochastic Demands. Journal of Mathematical Modelling and Algorithms, 5(1), 91-110.

[9] Vehicle Routing Data Set, Computational Infrastructure for Operations Research at http://www.coin-or.org/SYMPHONY/branchandcut/VRP/data/Vrp-All.tgz.

[10] Christiansen, C.H., Lysgaard, J. (2007). A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. Operations Research Letters, 35, 773-78

Prob	Opt	PSO		CENTPSO		CNTPSO	
	Bf	Bf	Tempo	Bf	Tempo	Bf	Tempo
1	1354	2082.508	38,9441	1409.077	267,6558	1486.989	111,9600
2	784	865.7817	11,8132	675.0448	85,0362	727.6297	29,6339
3	631	1084.348	30,2304	784.4155	193,1756	850.3444	87,6131
4	521	1134.727	16,3633	798.3925	116,4163	867.5327	55,8316
5	835	778.2106	10,8335	614.8419	68,6262	683.5821	28,9103
6	968	1359.139	72,1997	897.8753	513,8088	992.0475	227,3710
7	212	190.0015	3,3156	188.8167	20,8217	207.1706	8,87027
8	375	347.8175	6,9626	334.6671	47,5327	363.7313	19,2037
9	1073	1556.059	23,5607	1089.859	172,0748	1160.973	69,6552
10	450	196.2247	10,1363	196.2247	65,1405	196.2247	26,1055
11	831	1204.293	21,1720	859.0172	142,5002	964.99202	56,6176
12	568	1080.18	26,8328	744.5253	193,3879	800.39687	66,9846
13	799	986.6755	16,2955	741.8793	90,1287	827.03949	36,8121
14	216	222.5939	3,8862	204.991	27,1829	230.28956	8,7193
15	1146	1319.401	25,9009	951.2175	159,2446	1056.7583	68,9528

Tab. 5. Resultados Obtidos com a execução dos 3 algoritmos, onde Bf é o melhor fitness e o tempo é dado em minutos