Israel Pereira de Souza

Um algoritmo baseado em Evolução Diferencial com Busca Local para o Problema de Roteamento de Veículos Capacitados

Vitória, ES

2019

Israel Pereira de Souza

Um algoritmo baseado em Evolução Diferencial com Busca Local para o Problema de Roteamento de Veículos Capacitados

Monografia apresentada ao Curso de Ciência da Computação do Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Departamento de Informática

Orientador: Profª Maria Claudia Silva Boeres

Vitória, ES

2019

Israel Pereira de Souza

Um algoritmo baseado em Evolução Diferencial com Busca Local para o Problema de Roteamento de Veículos Capacitados/ Israel Pereira de Souza. – Vitória, ES, 2019-

42 p. : il. (algumas color.) ; 30 cm.

Orientador: Profª Maria Claudia Silva Boeres

Monografia (PG) – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Departamento de Informática, 2019.

1. Problema de Roteamento de Veículos Capacitados 2. Meta-heurísticas 3. Evolução Diferencial 4. CVRP I. Boeres, Maria Claudia Silva. II. Universidade Federal do Espírito Santo. III. Um algoritmo baseado em Evolução Diferencial com Busca Local para o Problema de Roteamento de Veículos Capacitados

Israel Pereira de Souza

Um algoritmo baseado em Evolução Diferencial com Busca Local para o Problema de Roteamento de Veículos Capacitados

Monografia apresentada ao Curso de Ciência da Computação do Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Vitória, ES, 12 de dezembro de 2019:

Profª Maria Claudia Silva Boeres
Orientadora

Eduardo Uchoa Barboza
Departamento de Engenharia de Produção - UFF

Maria Cristina Rangel
Departamento de Informática - UFES

André Manhães Machado
Petrobras

Vitória, ES
2019

"Termina isso logo pra gente jogar Knights and Bikes."
(Violeta de Freitas)

Resumo

Este trabalho tem como propósito o desenvolvimento de um algoritmo baseado na meta-heurística Evolução Diferencial (ED), do inglês *Differential Evolution*, combinada com algoritmos de busca local para resolver o Problema de Roteamento de Veículos Capacitados (PRVC), em inglês *Capacitated Vehicle Routing Problem (CVRP)*, formulado e amplamente conhecido na área de Otimização Combinatória. Como a ED foi proposta para problemas contínuos, foi utilizado um mecanismo para utilização desta meta-heurística em problemas combinatórios. O comportamento deste algoritmo foi investigado utilizando um conjunto de 6 pacotes de instâncias de diferentes autores na literatura, obtidos no CVRPLIB, uma biblioteca conhecida do problema. Por fim os resultados foram comparados com trabalhos de outros autores na literatura.

Palavras-chaves: Problema de Roteamento de Veículos Capacitados, Meta-heurística, Metaheurística, Evolução Diferencial, Busca Local, Benchmark, CVRPLIB.

Lista de ilustrações

Lista de tabelas

Lista de Algoritmos

Lista de abreviaturas e siglas

PRV	Problema de Roteamento de Veículos
PRVC	Problema de Roteamento de Veículos Capacitados
VRP	*Vehicle Routing Problem*
CVRP	*Capacitated Vehicle Routing Problem*
ED	Evolução Diferencial
DE	*Differential Evolution*
ACO	*Ant Colony Optimization*
DELS	*Differential Evolution with Local Search*
CDELS	*Combinatorial Differential Evolution with Local Search*
SA	*Simulated Annealing*

Sumário

1 Introdução

O Problema de Roteamento de Veículos (PRV), em inglês *Vehicle Routing Problem (VRP)*, é um problema conhecido da otimização combinatória por sua alta complexidade ($\mathcal{NP}$-hard) (LENSTRA; KAN, 1981). De forma geral, seu objetivo é minimizar a distância total percorrida por um conjunto de veículos que devem atender à demanda de um conjunto de cidades.

O *VRP* em sua versão clássica, também denominado (*Capacitated VRP*), possui veículos idênticos, um único depósito central e apenas restrições de capacidades dos veículos são impostas (TOTH; VIGO, 2002). As cidades devem ser visitadas apenas uma vez e cada veículo inicia e finaliza seu trajeto no posto (geralmente tratado como a cidade ou cliente 0) (BRAEKERS; RAMAEKERS; NIEUWENHUYSE, 2016). Assim, uma solução para esse problema é representada por uma permutação de rotas formadas por um conjunto de clientes.

Neste trabalho desenvolvemos um algoritmo baseado na meta-heurística Evolução Diferencial (ED) (STORN; PRICE, 1997; PRICE; STORN; LAMPINEN, 2006), do inglês *Differential Evolution (DE)* e analisamos o seu comportamento na resolução do *CVRP*.

A ED é um algoritmo evolutivo proposto para problemas contínuos. Soluções de um problema resolvido pela ED são tratadas como indivíduos ou cromossomos que são organizados em populações e o método simula a evolução dessas populações. Esses indivíduos tendem a herdar as melhores características dos pais ao longo das gerações, seja por Mutação, Cruzamento (*Crossover*) ou Seleção. Cada indivíduo da população é designado como alvo, para aplicação dos operadores.

A diversificação dos indivíduos pela ED é causada por Mutação e *Crossover* e após o uso destes operadores o operador de Seleção é aplicado. Trocas de cidades aplicadas por Mutação e *Crossover* são caracterizadas perturbações. O operador de Mutação possui uma constante F que controla o impacto de perturbação dos indivíduos, de forma que para maiores valores de F são gerados indivíduos mutantes mais diversificados que os indivíduos originais na população. O *Crossover* é utilizado para aumentar a diversidade dos indivíduos gerados, sendo executado após a mutação. Por fim o operador de Seleção escolhe o melhor entre o indivíduo perturbado (modificado pelas ações de Mutação e *Crossover*) e aquele antes de sofrer a ação desses operadores (o indivíduo alvo).

Propomos uma adaptação para o mecanismo de Mutação da ED para utilização em problemas de origem combinatória, em questão o *CVRP*. Nesta adaptação efetuamos trocas entre pares de posições das componentes, ao invés de aplicar o operador no conteúdo. Essas posições representam visitações de clientes no *CVRP*. Desta forma as perturbações

são trocas nas posições das cidades pertencentes a uma solução, de forma que quando aplicadas, não duplicam ou removem um cliente do indivíduo.

Testes computacionais foram realizados com a adaptação proposta da ED para o *CVRP* considerando 6 pacotes de instâncias de diferentes autores obtidos no CVRPLIB, uma conhecida biblioteca de dados para o *CVRP*, disponibilizada em *CVRPLIB*, *http://vrp.atd-lab.inf.puc-rio.br, acessado em:19/09/2019*. Comparações com outros trabalhos da literatura são realizadas e discutidas na seção de resultados computacionais (Seção 6).

A seção 2 apresenta um referencial teórico com versões adaptadas do problema e um estudo sobre a frequência de publicações destas versões no período de 2009 a 2015. A seção 3 apresenta a descrição do problema estudado neste trabalho, com sua definição matemática. A seção 4 apresenta a meta-heurística aqui investigada, a Evolução Diferencial, em sua versão original apresentada por Rainer Storn e Kenneth Price. A seção 5 apresenta a adaptação para o *CVRP* realizada neste trabalho para uso da ED em problemas de origem combinatória. A seção 6 discute o desempenho do algoritmo em 6 pacotes de instâncias da literatura, em conjunto com as análises e estudos que foram utilizados para calibração dos parâmetros do algoritmo. A seção 7 apresenta as conclusões e uma discussão sobre os resultados obtidos neste trabalho.

2 Referencial Teórico

O Problema de Roteamento de Veículos, do inglês *Vehicle Routing Problem (VRP)*, consiste em obter uma sequência ótima de visitação de um conjunto de veículos considerando um conjunto de clientes, que também são referenciados como cidades na literatura, e é um dos mais importantes e estudados problema de otimização combinatória (TOTH; VIGO, 2002).

O problema foi introduzido em 1959 (DANTZIG; RAMSER, 1959), para modelar a entrega de gasolina aos postos com a proposta de uma formulação matemática e abordagem algoritmica (TOTH; VIGO, 2002). Em 1964, Clark e Wright propuseram uma heurística aprimorando o método de Dantzig (CLARKE; WRIGHT, 1964). Com base nestes trabalhos, diversos modelos matemáticos e técnicas para obtenção das soluções ótimas e aproximadas foram propostas para variações do *VRP*. Os estudos do *VRP* são motivados por sua relevância e dificuldade.

Existem na literatura diversas variações do *VRP*, das quais podemos destacar: *VRPTW (Time Windows)*, na qual cada cidade possui um período estipulado no qual deve ser atendida (EL-SHERBENY, 2010), *VRPDB (Delivery and back-haul)* onde cada cidade possui uma demanda de recebimento que é encaminhada do posto à cidade, e de entrega, que é encaminhada da cidade ao posto (SHOSHANA, 1996), *Heterogeneous Fleet VRP (HFVRP)* também conhecido como *Mixed Fleet VRP* na qual a capacidade dos veículos é variada (BRAEKERS; RAMAEKERS; NIEUWENHUYSE, 2016).

O *VRP* em sua versão clássica, chamado *Capacitated VRP*, consiste em obter um conjunto de rotas ótimas em que todos os clientes correspondem as entregas e as demandas são determinísticas, conhecidas previamente e não podem ser repartidas entre os veículos. Os veículos são idênticos, há um único depósito central e apenas restrições de capacidades dos veículos são impostas (TOTH; VIGO, 2002). As cidades devem ser visitadas apenas uma vez e cada veículo inicia e finaliza seu trajeto no posto (geralmente tratado como a cidade ou cliente 0) (BRAEKERS; RAMAEKERS; NIEUWENHUYSE, 2016). Em problemas reais, podemos considerar que um veículo faz mais de uma rota no momento que ele finaliza a rota anterior e recomeça uma nova rota, sempre começando e finalizando seu trajeto no posto para cada rota, porém este veículo é considerado algoritmicamente como múltiplos veículos.

Cerca de 71.25% de um total de 277 trabalhos correlatos publicados neste tema no período de 2009 a 2015, fazem uso de meta-heurísticas para solucionar o *VRP* e suas derivações, seguido pelos métodos exatos que foram utilizados em certa de 17.13% dos artigos e por fim os trabalhos fazem uso de heurísticas clássicas (9.79%), métodos de

solução em tempo real (3.36%) e simulações (2.14%) segundo (BRAEKERS; RAMAEKERS; NIEUWENHUYSE, 2016).

Em (BRAEKERS; RAMAEKERS; NIEUWENHUYSE, 2016) é proposta a distinção de meta-heurísticas e heurísticas clássicas. Exemplos de heurísticas clássicas incluem heurísticas construtivas e algoritmos de busca local como λ-opt. Laporte define heurísticas clássicas como heurísticas que não permitem a deterioração da solução durante o processo de busca do ótimo (LAPORTE, 2009).

A Tabela 1 apresenta a taxa observada da presença das características do *VRP* abordada nos trabalhos analisados por Braekers et al. Como as características podem ser combinadas a soma das porcentagens é superior a 100%.

Característica	**Presença (%)**
Capacitated vehicles	90.52
Time Windows	37.92
Backhauls	18.65
Heterogeneous vehicles	16.51
Multiple depots	11.01
Recourse allowed	9.48
Multi-period time horizon	8.87
Precedence and coupling constraints	8.56
Subset covering constraints	8.56
Split deliveries allowed	6.12
Stochastic demands	6.12
Time-dependent travel times	3.06
Stochastic travel times	2.75
Dynamic requests	2.45
Unknown demands	1.83
Unknown travel times	1.53

Tabela 1 – Cenário das especifícações abordadas no conjunto de 277 artigos observados entre 2009 e 2015 (BRAEKERS; RAMAEKERS; NIEUWENHUYSE, 2016).

3 Descrição do Problema

O *CVRP* pode ser representado por um grafo $G = (V, E)$ onde o conjunto de vértices $V = \{c_0, c_1, ..., c_N\}$ representa os clientes, também apresentados como cidades na literatura, em que cada cliente $c_i \in V$ possui coordenadas cartesianas (x_i, y_i) que serão utilizadas para o cálculo da distância. O conjunto de arestas E expressa a visitação dos clientes.

Uma rota consiste em uma sequência de clientes a serem atendidos por um veículo. A ordem desta sequência determina a ordem de visitação dos clientes. O vértice c_0 representa o posto, também apresentado como garagem ou depósito na literatura, que deve ser o ponto de partida e chegada de cada rota. O número de rotas para uma instância é equivalente ao número de veículos determinados como entrada.

Uma instância do problema corresponde a uma aplicação, com os clientes que serão atendidos definidos e as devidas restrições para resolução. Os parâmetros N, K, Q e q_i são parâmetros de entrada do problema, fornecidos pela instância, em que N representa o número de clientes que devem ser atendidos, K representa o número de veículos disponíveis, Q representa a capacidade máxima de cada veículo e q_i representa a demanda do cliente i. Além desses valores, a instância também deve fornecer as posições cartesianas de cada cliente, inclusive do posto. O parâmetro k é utilizado no modelo matemático e no algoritmo implementado para representar o índice ou número da rota em questão. Além disso, o parâmetro G é utilizado como índice para descrever a geração atual do algoritmo. A variável de decisão denotada como x_{ij}^k, tem valor 1 quando há visitação do cliente i à j na rota k e 0 em outro caso.

Notações:

N número de clientes a serem atendidos.

K número de veículos.

Q capacidade máxima de cada veículo.

d_{ij} distância do cliente i ao cliente j, com $d_{ij} = d_{ji}\ \forall i, j \in \{0, 1, ..., N\}$.

q_i demanda do cliente i, $\forall i \in \{0, 1, ..., N\}$ e $q_0 = 0$.

Variáveis de decisão:

$$x_{ij}^k = \begin{cases} 1, & \text{se o arco}(i,j) \text{ pertence a rota do veículo } k. \\ 0, & \text{em outro caso.} \end{cases}$$

A formulação matemática do *CVRP* é apresentada em (DANTZIG; RAMSER, 1959), (CLARKE; WRIGHT, 1964).

4 A meta-heurística Evolução Diferencial

A Evolução Diferencial (ED), do inglês *Differential Evolution*, foi proposta por (STORN; PRICE, 1997; PRICE; STORN; LAMPINEN, 2006) e possui três mecanismos principais: Mutação, Cruzamento (*Crossover*) e Seleção, que são aplicados a uma geração ou população de indivíduos, que também são chamados de cromossomos ou genes. Cada um destes indivíduos é designado como alvo para aplicação dos operadores.

A ED possui 3 parâmetros principais, que são: a taxa de Mutação $F \in (0, 2]$, taxa de *Crossover* $CR \in [0, 1]$ e o número de indivíduos da população np que tem por valor mínimo 4. Além destes, um parâmetro adicional pode ser utilizado para definir o número máximo de populações que serão geradas.

A constante F controla o raio de perturbação dos indivíduos na Mutação, de forma que para maiores valores de F a mutação causa um maior impacto sobre a população, resultando em uma maior diversificação das características dos indivíduos na geração. O *Crossover*, aplicado após a Mutação, é utilizado para aumentar a diversidade dos indivíduos gerados e é controlado pelo parâmetro CR de forma que quanto maior o CR, uma maior parte do indivíduo tende a ser cruzada. Por fim, o operador de Seleção escolhe o melhor entre o indivíduo perturbado (modificado pela ação da Mutação e *Crossover*) e o indivíduo alvo.

4.1 Mutação

Para cada indivíduo alvo $X_i \in \text{População}_G$, ou seja $X_{i,G}$, com $i \in 1, 2, \ldots, np$, outros três indivíduos $X_{r_1,G}, X_{r_2,G}, X_{r_3,G}$ são selecionados aleatoriamente com $i \neq r_1 \neq r_2 \neq r_3$. A diferença ponderada entre os indivíduos $X_{r_2,G}$ e $X_{r_3,G}$ é adicionada em um indivíduo base $X_{r_1,G}$ gerando então o indivíduo mutante $X_{mut,G+1}$. A equação (4.1) apresenta a fórmula para o cálculo de um indivíduo mutante.

$$X_{mut,G+1} = X_{r_1,G} + F(X_{r_2,G} - X_{r_3,G}) \tag{4.1}$$

4.2 *Crossover*

Com intuito de aumentar a diversidade da população o *Crossover* é aplicado. Para cada componente j do indivíduo um número aleatório $rand(j)$ é gerado e se este for menor ou igual a taxa de *Crossover* CR, a componente selecionada para o novo indivíduo

$X_{cross,G+1}$ será obtida do indivíduo mutante, caso contrário é selecionada a componente do indivíduo alvo $X_{ri,G}$. A equação (4.2) descreve como o indivíduo $X_{cross,G+1}$ é gerado.

$$X_{cross,G+1} = (X_{1cross,G}, X_{2cross,G}, ..., X_{Dcross,G})$$

$$X_{jcross,G+1} = \begin{cases} X_{jmut,G+1}, & \text{if rand(j)} \leq \text{CR.} \\ X_{ji,G}, & \text{caso contrário.} \end{cases} \tag{4.2}$$

Para garantir que o indivíduo cruzado será diferente do indivíduo alvo, o indivíduo cruzado $X_{cross,G+1}$ recebe uma componente de posição aleatória do indivíduo mutante no início do procedimento, sem avaliação pelo CR.

4.3 Seleção

Um critério guloso é utilizado para seleção do indivíduo $X_{i,G+1}$ que fará parte da próxima geração. Se o indivíduo perturbado $X_{cross,G+1}$ possuir custo inferior que o indivíduo alvo $X_{i,G}$, $X_{cross,G+1}$ será selecionado para a próxima geração como $X_{i,G+1}$. Em outro caso $X_{i,G}$ será passado para a próxima geração como $X_{i,G+1}$.

4.4 Pseudocódigo

O pseudocódigo da Evolução Diferencial é apresentado no Algoritmo 1.

Algoritmo 1: Evolução Diferencial

```
1  Geração ← Gera População Inicial
2  do
3      NovaGeração ← ∅
4      for i = 1 to np do
5          X_{mut,G+1} ← Mutação(X_{r1,G}, X_{r2,G}, X_{r3,G}, F)
6          X_{cross,G+1} ← Crossover(X_{i,G}, X_{mut,G+1}, CR)
7          V_{i,G+1} ← Seleção(X_{i,G}, X_{cross,G+1})
8          NovaGeração ← NovaGeração∪V_{i,G+1}
9      end for
10     Geração ← NovaGeração
11     G ← G + 1
12 while critério de convergência não atingido;
```

4.5 Estratégias

Existem variações da Evolução Diferencial, também chamadas de técnicas ou estratégias. As estratégias clássicas apresentadas em (STORN; PRICE, 1997) tratam a Mutação e *Crossover*. Na Mutação o indivíduo base selecionado na população pode ser um indivíduo aleatório, correspondendo ao tipo *rand* de Mutação, ou o melhor indivíduo da população, correspondendo a estratégia denominada *best* de Mutação. Além disso, também é apresentada a utilização de duas diferenças ponderadas para geração do indivíduo mutante, ao invés de apenas uma como apresentado na equação 4.1. Para o *Crossover* duas técnicas são apresentadas, sendo elas o tipo binário (*bin*) que é executado em todas as componentes todo o indivíduo, conforme a subseção 4.2 e o tipo exponencial (*exp*), em que o primeiro valor aleatório que for maior que a taxa de *Crossover* (*CR*) finalizará o processo e as demais componentes do indivíduo são copiadas do alvo.

O algoritmo pode ser esquematizado como *DE/x/y/z*, em que:

x *especifica o tipo do indivíduo base que será utilizado na mutação*

y *representa o número de vetores diferença usados na mutação*

z *especifica o crossover utilizado*

As subseções 4.1 e 4.2 descrevem a estratégia: *DE/rand/1/bin*.

Em (ONWUBOLU; DAVENDRA, 2009) é apresentado uma nova estratégia para a Mutação, denominada *rand-to-best* com objetivo de aprimorar a estratégia aleatória (*rand*) inserindo um fator mais guloso na técnica ao incorporar o melhor indivíduo $X_{best,G}$. Para isso um novo fator λ de controle é apresentado, que controla o impacto de $X_{best,G}$ na geração do indivíduo mutante.

A Tabela 2 apresenta formulações de diversas estratégias do algoritmo ED.

Estratégias	Formulações
DE/rand/1/bin	$X_{mut,G+1} = X_{r_1,G} + F(X_{r_2,G} - X_{r_3,G})$
DE/best/1/bin	$X_{mut,G+1} = X_{best,G} + F(X_{r_2,G} - X_{r_3,G})$
DE/rand/2/bin	$X_{mut,G+1} = X_{r_1,G} + F(X_{r_2,G} + X_{r_3,G} - X_{r_4,G} - X_{r_5,G})$
DE/best/2/bin	$X_{mut,G+1} = X_{best} + F(X_{r_2,G} + X_{r_3,G} - X_{r_4,G} - X_{r_5,G})$
DE/rand-to-best/1/bin	$X_{mut,G+1} = X_{i,G} + \lambda(X_{best,G} - X_{i,G}) + F(X_{r_2,G} - X_{r_3,G})$
DE/rand/1/exp	$X_{mut,G+1} = X_{r_1,G} + F(X_{r_2,G} - X_{r_3,G})$
DE/best/1/exp	$X_{best} = X_{r_1,G} + F(X_{r_2,G} - X_{r_3,G})$
DE/rand/2/exp	$X_{mut,G+1} = X_{r_1,G} + F(X_{r_2,G} + X_{r_3,G} - X_{r_4,G} - X_{r_5,G})$
DE/best/2/exp	$X_{mut,G+1} = X_{best,G} + F(X_{r_2,G} + X_{r_3,G} - X_{r_4,G} - X_{r_5,G})$
DE/rand-to-best/1/exp	$X_{mut,G+1} = X_{i,G} + \lambda(X_{best,G} - X_{i,G}) + F(X_{r_2,G} - X_{r_3,G})$

Tabela 2 – Estratégias da Evolução Diferencial (ONWUBOLU; DAVENDRA, 2009)

As estratégias que utilizam o melhor indivíduo (*best*) como o indivíduo base da Mutação tendem a convergir mais rápido, porém são mais propensas a estagnar em mínimos locais. De forma similar, as estratégias que usam indivíduos aleatórios (*rand*) tendem a gerar resultados melhores, sendo menos propensas à mínimos locais, porém são mais lentas. As estratégias que utilizam o tipo *rand-to-best* de Mutação tem por objetivo manter a qualidade de convergência do tipo *rand* de Mutação, porém com melhor eficiência na convergência fornecida pelo tipo *best* de Mutação.

5 Um algoritmo baseado em Evolução Diferencial com Busca Local para o Problema de Roteamento de Veículos Capacitados

Nesta seção apresentamos o algoritmo completo, proposto e implementado neste trabalho, denominado CDELS. Seu pseudocódigo é apresentado no Algoritmo 2. O algoritmo inicia com a construção da população inicial que é descrita na subseção 5.4. Após isso, cada indivíduo pertencente a população é designado como alvo e para cada indivíduo alvo, um indivíduo mutante é gerado seguindo a subseção 5.1 que descreve a Mutação. O *Crossover* é aplicado de acordo com a subseção 5.2 e então é aplicada a Busca Local no indivíduo cruzado, de acordo com a subseções 5.3, 5.3.1, 5.3.2 e por fim o indivíduo alvo e o indivíduo intensificado por busca local são avaliados e o indivíduo que possuir menor custo é selecionado para a próxima geração. Estes procedimentos são realizados enquanto um número máximo de gerações não for atingido e o algoritmo não tiver encontrado a melhor solução. Para obtenção das distâncias entre os clientes de forma mais eficiente uma matriz de distâncias é construída na inicialização do algoritmo.

Optamos por não descartar indivíduos inviáveis (que infrigiram as restrições) e não foram viabilidades pelo algoritmo *Drop One Point Infeasible*, aplicando uma penalidade em seu custo e então esses indivíduos são passados por seleção para decidir se farão parte da próxima geração. Observamos que com bons valores de penalidade, esses indivíduos quando selecionados, possuíam poucas rotas que infrigiam as restrições e ao manter na geração, houve convergência mais rápida dos indivíduos viáveis e no longo das gerações esses indivíduos inviáveis também eram viabilizados.

5.0.1 O algoritmo CDELS

Algoritmo 2: CDELS

```
1  Geração ← Gera População Inicial
2  do
3      NovaGeração ← ∅
4      for i = 1 to np do
5          X_{mut,G+1} ← Mutacao(X_{r1,G}, X_{r2,G}, X_{r3,G}, F)
6          X_{cross,G+1} ← Crossover(X_{i,G}, X_{mut,G+1}, CR)
7          U_{i,G+1} ← BuscaLocal(X_{cross,G+1})
8          V_{i,G+1} ← Seleção(X_{i,G}, U_{i,G+1})
9          NovaGeração ← NovaGeração∪V_{i,G+1}
10     end for
11     Geração ← NovaGeração
12     G ← G+1
13 while não atingiu o máximo de gerações e não encontrou a melhor solução;
```

5.1 Mutação

A Mutação da ED em sua forma clássica gera um vetor de números reais e desta forma, para utilizar o algoritmo ED em problemas combinatórios é necessário discretizar estes números reais. Existem diversos algoritmos na literatura que convertem os números reais para inteiros nos indivíduos da população manipulados pela ED.

Os métodos *Smallest Order Value (SOV)* (QIAN et al., 2008) e *Largest Order Value (LOV)* (QIAN et al., 2009), que tem como base o método *random keys* (BEAN, 1994), ranqueiam os números reais e os converte para inteiros considerando sua ordem na sequência. Por exemplo, ao aplicar o *SOV* em uma sequência $S = (0.31, 0.12, 0.4)$ o resultado seria (2,1,3) e ao aplicar o *LOV*, (2,3,1). O método *SOV* foi utilizado, por exemplo, em (TEOH; PONNAMBALAM; KANAGARAJ, 2015) e (MINGYONG; ERBAO, 2010), que utilizaram o método para solução de variações do *VRP*.

Neste trabalho é proposta a troca das posições das componentes ao invés de modificar diretamento o seu valor, que são os clientes no *CVRP*. Desta forma as perturbações são trocas entre clientes no indivíduo, que ao serem aplicadas não duplicam ou removem clientes no indivíduo.

Para efetuar a operação de Mutação são necessários o fator de mutação F e quatro indivíduos distintos da população, sendo estes: $X_{r_1,G}$, $X_{r_2,G}$, $X_{r_3,G}$ e o indivíduo alvo $X_{i,G}$. O parâmetro F controla o raio de perturbação, de forma que quanto maior o valor de F, indivíduos mutantes mais diversificados tendem a ser gerados. Para reprodução desta característica foi definido o número de perturbações $p = \lceil \frac{N}{2} \times F \rceil$ que serão efetuadas

na Mutação, com $F \in (0, 1]$. E assim, quanto maior o F, uma maior quantidade de perturbações será efetuada. A limitação de F imposta com valor máximo de 1 ao invés de 2, tem como propósito evitar trocas redundantes no indivíduo.

Para cada movimento é selecionado um cliente aleatório c_j de $X_{r_2,G}$. A posição $p_{x3,G}$ de c_j então é obtida no indivíduo $X_{r_3,G}$. Por fim os clientes c_j e c_k são trocadas no novo indivíduo $X_{mut,G+1}$, onde c_k é o cliente na posição p_{x3} em $X_{mut,G+1}$. A Figura 1 apresenta um exemplo deste movimento.

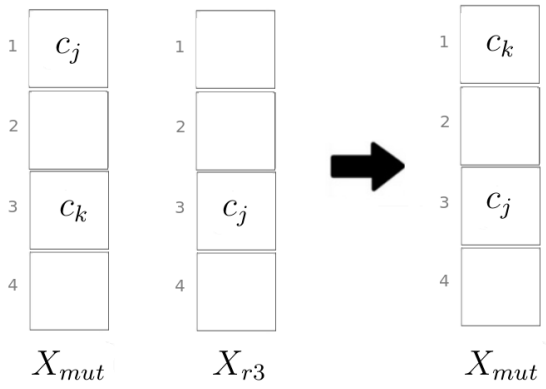

Figura 1 – Exemplo de perturbação na Mutação.

Com o passar das gerações, os indivíduos da população tendem a possuir boas características e os clientes tendem a estar em boas posições. O objetivo deste algoritmo é fazer a diversificação dos indivíduos com foco em preservar as informações que foram obtidas ao longo das gerações. O Algoritmo 3 apresenta o pseudocódigo da operação de Mutação implementada.

Algoritmo 3: Pseudocódigo da Mutação Combinatória

Input: $X_{r_1,G}$, $X_{r_2,G}$, $X_{r_3,G}$, F, N.
Output: $X_{mut,G+1}$.

```
1  inicialize X_{mut,G+1} como um clone de X_{r1,G}
2  for i ← 1 to ⌈(N/2) × F⌉ do
3      c_j ← cliente aleatório de X_{r2,G}
4      p_x3 ← posição de c_j on X_{r3,G}
5      c_k ← cliente na posição p_x3 in X_{mut,G+1}
6      if existe um cliente c_k na posição p_x3 em X_{mut,G+1} then
7          troque c_j e c_k em X_{mut,G+1}
8      else
9          remova a cidade c_j de X_{mut,G+1} e reinsira na última posição da rota definida
             por p_x3
10     end if
11 end for
```

O procedimento de Mutação inicia criando o indivíduo $X_{mut,G+1}$ com as características do indivíduo base $X_{r_1,G}$ e uma quantidade de trocas das componentes são feitas em $X_{mut,G+1}$, considerando as posições (localização do cliente no indivíduo, informando rota e posição na rota) de outros indivíduos que são considerados no procedimento.

Como as rotas podem possuir tamanhos diferentes, é possível que seja requisitada uma componente em uma posição além do final da rota de um indivíduo, e assim é possível que a posição p_{x3} não exista em $X_{mut,G+1}$. Se isto acontecer, o cliente do fim desta rota é selecionado para o movimento. Desta forma, o cliente c_j é removido de $X_{mut,G+1}$ e reinserido na última posição da mesma rota que c_j reside em $X_{r_3,G}$. Para não inserir rotas vazias, esta troca só é feita se a rota possuir mais de um cliente além do posto.

5.2 Crossover

A aplicação da operação de Crossover tem por finalidade aumentar a diversidade dos indivíduos perturbados. Neste trabalho utilizamos a operação padrão. O pseudocódigo é descrito no Algoritmo 4.

Algoritmo 4: Pseudocódigo do Crossover

Input: $X_{i,G}$, $X_{mut,G+1}$, CR, K.
Output: $X_{cross,G+1}$.

```
1  inicialize X_{cross,G+1} como um clone de X_{i,G}
2  zzz cruze uma posição aleatória em X_{cross,G+1} com X_{mut,G+1}
3  for i ← 1 to K do
4      foreach posição p_j na rota k_i em X_{mut,G+1} do
5          if rand(j) ≤ CR then
6              c_j ← cliente na posição p_j em X_{mut,G+1}
7              if existe cliente c_k na posição p_j em X_{cross,G+1} then
8                  c_k ← cliente na posição p_j em X_{cross,G+1}
9              else
10                 c_k ← cliente na última posição da rota definida por p_j em X_{cross,G+1}
11             end if
12             troque c_j e c_k em X_{cross,G+1}
13         end if
14     end foreach
15 end for
```

O procedimento de *Crossover* se inicia criando o indivíduo $X_{cross,G+1}$ com as características do indivíduo alvo $X_{i,G}$. Uma permutação é executada para garantir que $X_{cross,G+1}$ seja diferente do indivíduo alvo, então uma componente de posição aleatória é perturbada em $X_{cross,G+1}$, considerando $X_{mut,G+1}$. Após isso, o ciclo principal se inicia e para cada rota, todas as componentes são avaliadas e se o número aleatório gerado (*rand(j)*) for menor ou igual a taxa de *Crossover* a permutação será efetuada.

De forma similar à Mutação, se for requisitada uma posição que esteja além do fim da rota, o cliente do fim desta rota é selecionado e este movimento só é efetuado se o cliente a ser removido estiver em uma rota que possua outro cliente além dele e do posto.

5.3 Busca Local

Na Busca Local foram utilizados os algoritmos 2-opt com Descida Profunda, do inglês *Deepest Descent*, e uma adaptação do *Drop One Point* proposto em (TEOH; PONNAMBALAM; KANAGARAJ, 2015), que chamamos de *Strong Drop One Point*. Além destes, para tratar soluções inviáveis, propomos um algoritmo em que chamamos de *Drop One Point Infeasible*. O pseudocódigo da Busca Local é apresentado pelo Algoritmo 5 e cada um dos demais algoritmos nas subseções seguintes.

Algoritmo 5: Pseudocódigo da Busca Local

Input: $X_{cross,G+1}$.
Output: $X_{cross,G+1}$.
do
  $X_{cross,G+1} \leftarrow$ 2-opt$(X_{cross,G+1})$
  $X_{cross,G+1} \leftarrow$ *StrongDropOnePoint*$(X_{cross,G+1})$
  if $X_{cross,G+1}$ *é inviável* **then**
    $X_{cross,G+1} \leftarrow$ *DropOnePointInfeasible*$(X_{cross,G+1})$
  end if
while *houve melhora*;

5.3.1 *Strong Drop One Point*

Cada cliente do indivíduo é reinserido na melhor posição de outra rota, isto é, a posição que retornar o menor custo e em uma rota que possua capacidade disponível para atender o cliente. Para respeitar o número de veículos exigido, caso a rota possua apenas um cliente, este não é removido. Como as inserções só são feitas em rotas que possuem capacidade para o cliente, este algoritmo é capaz de viabilizar indivíduos inviáveis ao longo de sua execução, porém nos problemas mais difíceis isto nem sempre ocorre e então o algoritmo *Drop One Point Infeasible* é aplicado.

O pseudocódigo do procedimento *Strong Drop One Point* é apresentado no Algoritmo 6.

Algoritmo 6: Pseudocódigo do *Strong Drop One Point*

for $i \leftarrow$ *1 to* K **do**
  foreach *cliente* c_j *na rota* k_i **do**
    if *há uma rota diferente de* k_i *que possua capacidade disponível para atender* c_j **then**
      reinsira c_j na posição e rota que forneça o menor custo e possua capacidade disponível para atender c_j
    end if
  end foreach
end for

5.3.2 *Drop One Point Infeasible*

Se um indivíduo da população possui pelo menos uma rota inviável, então é submetido ao procedimento *Drop One Point Infeasible*. Este procedimento seleciona

aleatoriamente uma rota k_i que seja inviável. Seja c_l um cliente pertencente a k_i tal que exista uma rota k_j que suporte c_l, então c_l é reinserido na melhor posição de k_j. Este algoritmo é executado enquanto o indivíduo ainda for inviável e existirem clientes para serem realocados. Caso o algoritmo finalize e o indivíduo ainda seja inviável, é adicionado um valor ao seu custo definido pelo parâmetro *Penalidade* do algoritmo.

O pseudocódigo do procedimento *Drop One Point Infeasible* é apresentado no Algoritmo 7.

Algoritmo 7: Pseudocódigo do Drop One Point Infeasible

```
1 do
2     selecione uma rota inviável aleatória k_i
3     if há um cliente c_i em k_i que pode ser realocado then
4         reinsira c_i na melhor posição de uma rota aleatória que tenha capacidade
            disponível para c_i
5     end if
6 while o indivíduo continua inviável e há clientes para serem realocados;
```

5.4 População Inicial

O algoritmo proposto neste trabalho efetua principalmente trocas e reinserções das componentes (clientes) nos indivíduos, então uma população inicial composta por indivíduos com diversidade de posições das componentes é necessária para boa convergência.

Para geração da população inicial são construídos dois tipos de indivíduos. Sendo um com maior densidade de componentes no início, em que as cidades são inseridas começando na primeira rota até a última (denso superior). O segundo cromossomo é construído com maior densidade das componentes no fim e assim as cidades começam a ser inseridas da última rota para a primeira (denso inferior). Um número aleatório $z \in \{0,1\}$ é gerado e se $z = 0$ então é construído um indivíduo denso superior, em outro caso é construído um indivíduo denso inferior.

Na construção do indivíduo, um cliente aleatório $c_j \in V = \{c_1,...,c_N\}$ é selecionado e inserido no indivíduo seguindo os critérios de inserção superior ou inferior. Caso a rota de inserção possua capacidade disponível para c_j este será inserido nesta rota, em outro caso c_j será inserido em uma nova rota e removido de V após a inserção. Porém caso o indivíduo já esteja usando todas as rotas disponíveis, os clientes remanescentes são inseridos na rota de última inserção do indivíduo. Desta forma são geradas ao menos $K - 1$ rotas viáveis no indivíduo.

6 Experimentos Computacionais e Resultados Obtidos

O algoritmo CDELS foi implementado sequencialmente e testado em uma máquina com processador Intel i5-3570, 4GB RAM, sistema operacional Ubuntu 16.04 e compilado com GCC 5.4.0 com a flag de otimização -O3.

Para cada instância dos pacotes A, B, E, F, M, P, obtidos no CVRPLIB, foram executados 10 testes usando os 10 primeiros números inteiros como semente. A representação de ponto flutuante adotada foi precisão dupla (*double*) e em cada operação estes valores foram arredondados para o inteiro mais próximo como especificado no TSPLIB95 (REINELT, 1995). Entretanto, mesmo utilizando apenas inteiros para representar o fitness das soluções, o resultado apresentado pode ser real, que foi gerado pela média das soluções.

6.1 Parâmetros de entrada do algoritmo

Um conjunto de parâmetros de entrada são necessários. São eles:

np	número de indivíduos da população.
F	taxa de Mutação.
CR	taxa de *Crossover*.
Penalidade	penalidade aplicada às soluções inviáveis.
MaxGen	limite máximo de gerações do algoritmo.

Em seu trabalho, (DAS; SUGANTHAN, 2010) mostra que bons valores de np para a ED estão entre $3N$ e $8N$, onde N é o tamanho do indivíduo, que no problema de roteamento é representado como o número de clientes do indivíduo.

Com base no trabalho de (DAS; SUGANTHAN, 2010), os autores (TEOH; PONNAMBALAM; KANAGARAJ, 2015) investigaram os valores de np entre $3N$ e $8N$ e selecionaram a relação $np = 3N$ para seu algoritmo por boa qualidade de solução e tempo. Por este motivo adotamos a mesma relação nos nossos testes.

6.1.1 Calibração dos parâmetros $F \times CR$

Com respeito aos parâmetros F e CR, foi feito um estudo para sua calibração. Foram testadas todas as combinações $F \times CR$, $F, CR \in [0.1, 0.9]$ com intervalos de 0.1. O

Pacote A (Augerat,1995) com 27 instâncias foi utilizado por completo para calibração destes parâmetros e em cada combinação o algoritmo foi executado 10 vezes com *MaxGen*=10,000 e a média das soluções foi armazenada. A partir de testes preliminares o parâmetro *Penalidade*=100 apresentou bons resultados, sendo adotado nesta calibração.

Para a calibração dos parâmetros, os valores de tempo foram computados desde a inicialização do algoritmo até a finalização e desta forma caso a execução não tenha obtido o ótimo da instância, o tempo é calculado considerando todas as gerações executadas pelo algoritmo (*MaxGen*). Porém ao obter o ótimo da instância, o algoritmo finaliza computando assim um número menor de Gerações que *MaxGen*. Desta forma, estratégias ou parâmetros que obtêm o ótimo com mais frequência terão destaque nas comparações.

As Figuras 2, 3, 4, 5 apresentam o gráfico performance das estratégias *rand/1/bin*, *best/1/bin*, *rand/1/exp* e *best/1/exp* respectivamente, em que o eixo horizontal apresenta os valores de CR e o eixo vertical os valores de F. Cada ponto do gráfico apresenta um acumulador sobre a quantidade de vezes que a solução ótima foi obtida nas execuções que fizeram uso desses parâmetros. Considerando as 27 instâncias do pacote A, o valor máximo que pode ser atingido é 270.

As Figuras 6, 7 apresentam a comparação de tempo das estratégias consideradas, que utilizam *Crossover* binário e exponencial, respectivamente. O eixo horizontal denota os valores de CR testados e o eixo horizontal denota o tempo em segundos da média aritmética de tempo das 270 execuções do algoritmo. As curvas representam os valores de F testados e cada ponto das curvas denotam a média de tempo da combinação $F \times CR$ testada na estratégia em questão.

A Tabela 3 apresenta um resumo de performance das duas melhores combinações dos parâmetros $F \times CR$ observados para cada estratégia no processo de calibração. Esta tabela apresenta o *Score* (número de vezes que o ótimo foi obtido) *Fitness* e CPU que descrevem a média aritmética de custo e tempo computacional em segundos, respectivamente, das 270 execuções.

A Tabela 4 apresenta um resumo geral de performance das estratégias no processo de calibração. O *Score* representa a quantidade de soluções ótimas obtidas naquela estratégia, *Fitness* representa a média aritmética do custo das soluções e CPU apresenta a média aritmética do tempo computacional em segundos das execuções.

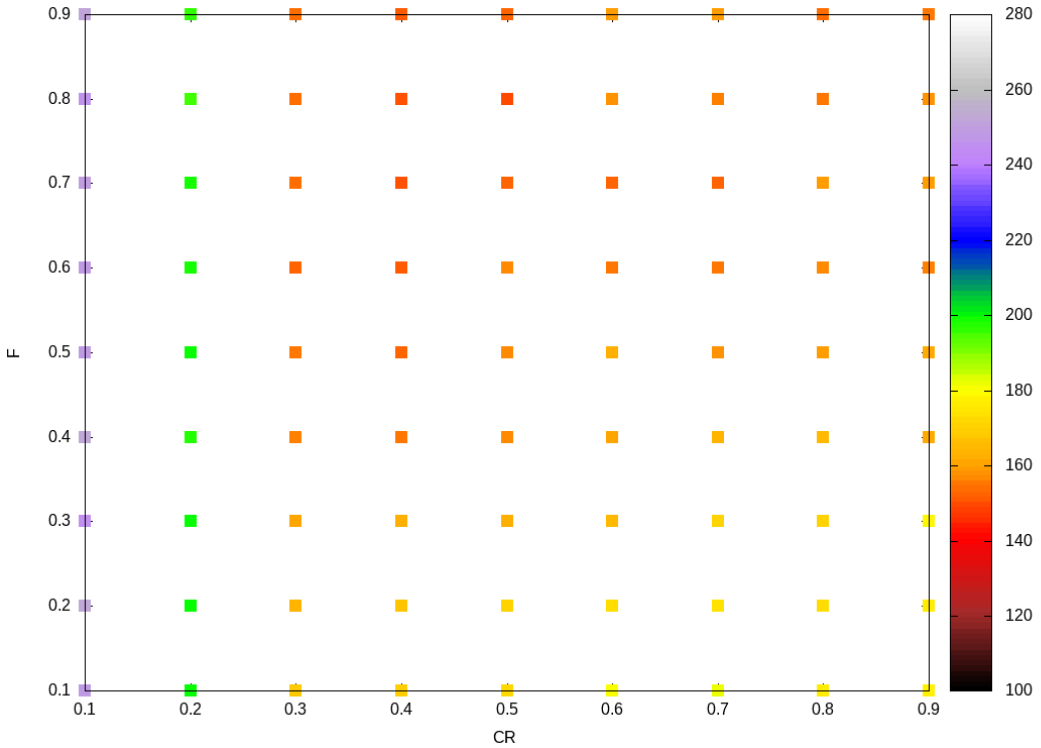

Figura 2 – Comparação de performance de $F \times CR$ com a estratégia *rand/1/bin* usando 27 instâncias do pacote A (Augerat, 1995) com *MaxGen*=10,000.

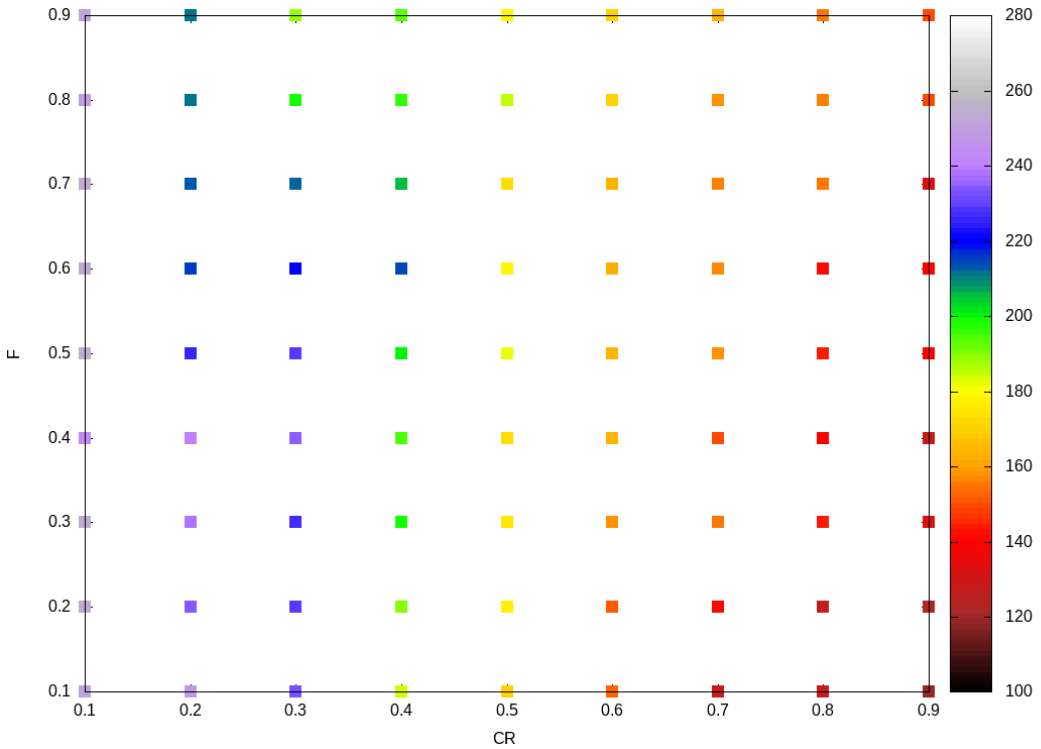

Figura 3 – Comparação de performance de $F \times CR$ com a estratégia *best/1/bin* usando 27 instâncias do pacote A (Augerat, 1995) com *MaxGen*=10,000.

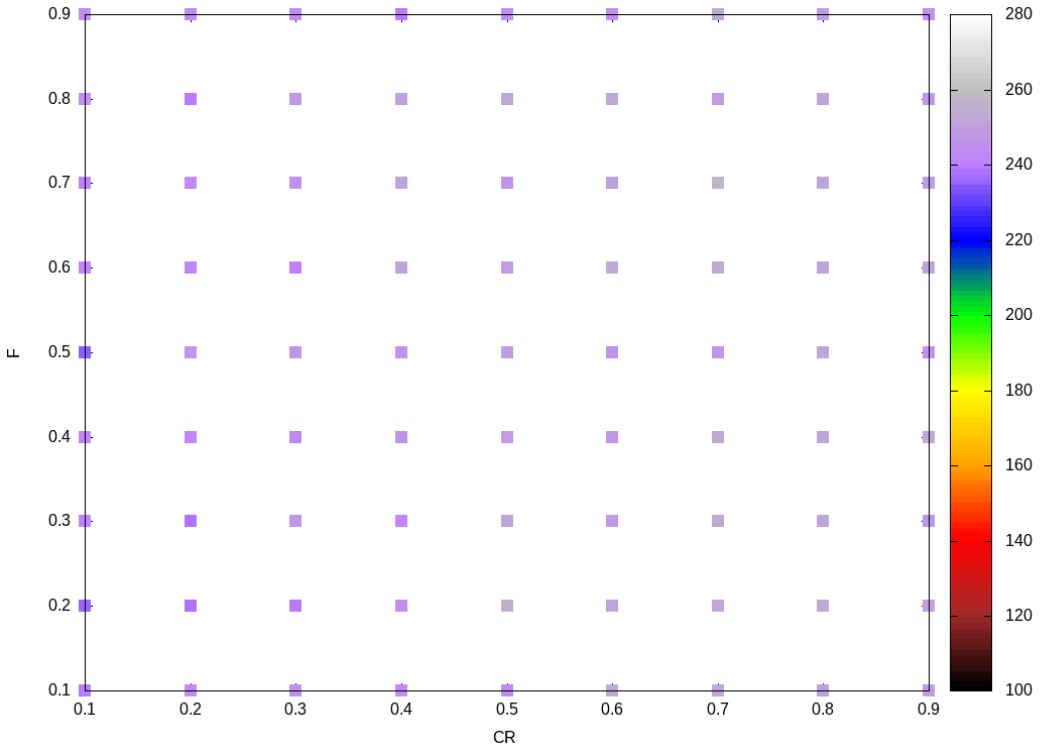

Figura 4 – Comparação de performance de $F \times CR$ com a estratégia *rand/1/exp* usando 27 instâncias do pacote A (Augerat, 1995) com *MaxGen*=10,000.

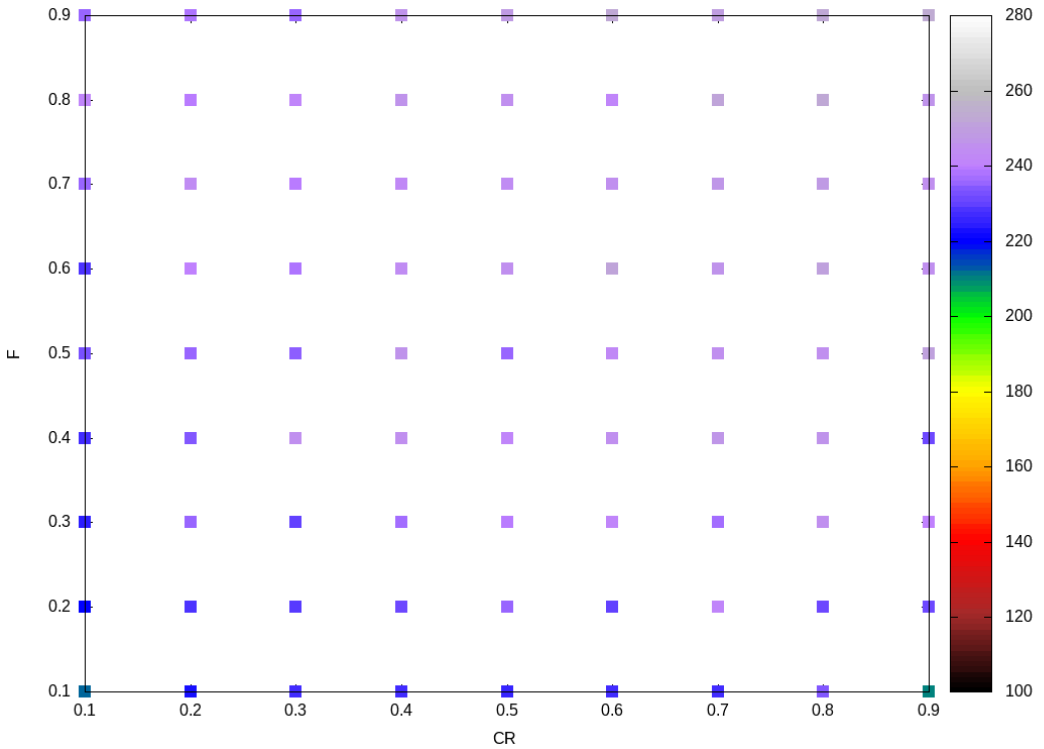

Figura 5 – Comparação de performance de $F \times CR$ com a estratégia *best/1/exp* usando 27 instâncias do pacote A (Augerat, 1995) com *MaxGen*=10,000.

Estratégia	F	CR	Score	Fitness	CPU (s)
rand/1/bin	0.2	0.1	252	1042.14	28.05
rand/1/bin	0.4	0.1	252	1042.16	27.57
best/1/bin	0.5	0.1	255	1042.10	25.37
best/1/bin	0.3	0.1	254	1042.13	26.59
rand/1/exp	0.7	0.7	257	1042.01	15.72
rand/1/exp	0.2	0.5	256	1042.14	12.20
best/1/exp	0.9	0.9	253	1042.11	20.12
best/1/exp	0.8	0.8	252	1042.07	17.06

Tabela 3 – Dupla de melhores parâmetros $F \times CR$ de cada estratégia no pacote A (Augerat, 1995) testado com *MaxGen*=10,000.

Estratégia	Score	Fitness	CPU (s)
rand/1/bin	14145	1047.24	73.26
best/1/bin	14947	1044.19	29.57
rand/1/exp	**19993**	**1042.20**	**16.93**
best/1/exp	19320	1042.36	19.05

Tabela 4 – Performance das quatro estratégias testadas no pacote A (Augerat, 1995) com *MaxGen*=10,000.

A estratégia *rand/1/exp* apresentou maior robustez, atingindo a solução ótima em 91.41% das execuções no pacote A. A estratégia *best/1/exp* foi a segunda mais robusta, atingindo a solução ótima em 88.34% das execuções no pacote A. A estratégia *best/1/exp* obteve alta performance em tempo sob as estratégias, porém a estratégia *rand/1/exp* encontrou o ótimo em mais execuções, apresentando média geral em tempo também menor, considerando que a execução finaliza ao encontrar o ótimo. A estratégia *rand/1/exp* com F=0.7, CR=0.7 se apresentou como a melhor combinação, obtendo a solução ótima em 95.18% das execuções no pacote A. Como neste trabalho buscamos qualidade de solução a estratégia *rand/1/exp* com F=0.7, CR=0.7 foi definida para este algoritmo.

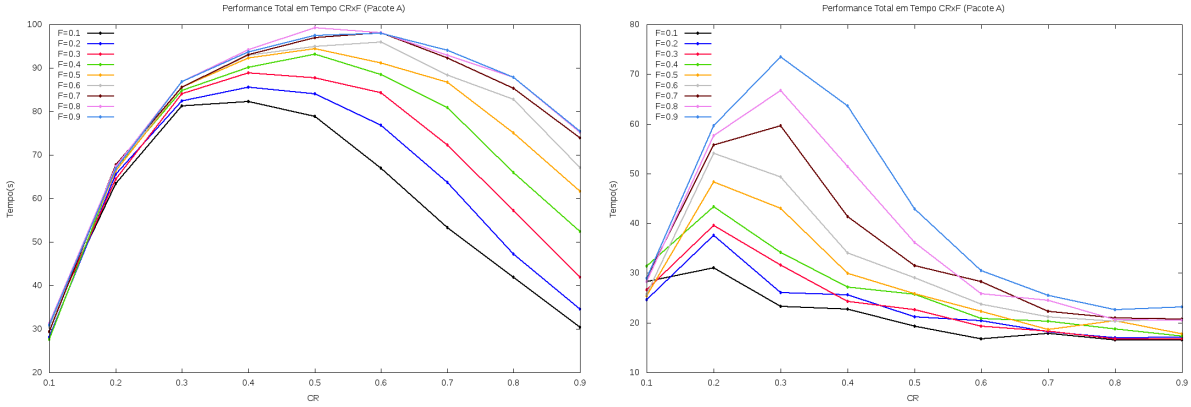

Figura 6 – Comparação de tempo das estratégias a esquerda *rand/1/bin* e a direita *best/1/bin* usando 27 instâncias do pacote A (Augerat, 1995) com *MaxGen*=10,000.

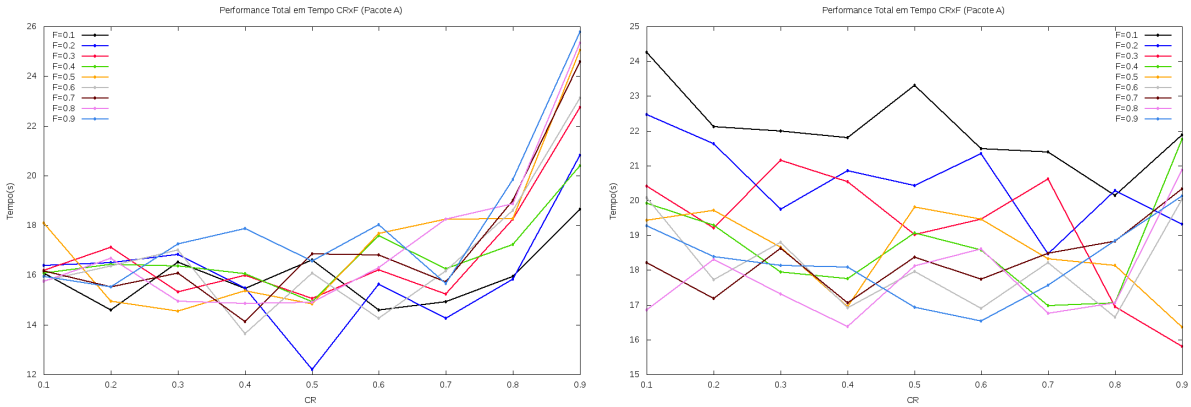

Figura 7 – Comparação de tempo das estratégias a esquerda *rand/1/exp* e a direita *best/1/exp* usando 27 instâncias do pacote A (Augerat, 1995) com *MaxGen*=10,000.

O tipo de *Crossover* possuiu o maior impacto na calibração dos parâmetros de $F \times CR$ e a utilização do melhor indivíduo levou a convergência mais rápida do algoritmo, mas a utilização de indivíduos aleatórios levou, no geral, a melhor qualidade de solução, apresentando maior robustez ao algoritmo na utilização de indivíduos aleatórios.

O tipo binário de *Crossover* apresentou bom comportamento com taxas baixas de CR, na qual a menor taxa de CR testada (0.1) apresentou bom comportamento para todos os valores de F testados, em que a pior combinação obteve o ótimo em 90% das execuções e a melhor combinação obteve o ótimo em 94.44% das execuções, utilizando *MaxGen*=10,000.

6.2 Testes Computacionais

O algoritmo proposto foi testado em 6 pacotes de instâncias de diferentes autores, obtidos no CVRPLIB, *disponível em: http://vrp.atd-lab.inf.puc-rio.br, acessado em:19/09/2019*. O CVRPLIB é uma biblioteca reconhecida do *CVRP*. Os pacotes selecionados foram: A, B, P do autor (Augerat, 1995), E dos autores (Christofides and Eilon, 1969), F do autor (Fisher, 1994) e M dos autores (Christofides, Mingozzi and Toth, 1979).

Os parâmetros utilizados para este algoritmo foram: F=0.7, CR=0.7, estratégia: *rand/1/exp*, *Penalidade*=100, *MaxGen*=200k para os pacotes A,B,E,F,P e *MaxGen*=500k para o pacote M.

A porcentagem do desvio foi calculada na forma: $\%\text{Dev} = \frac{\{SOLUTION - BKS\}}{BKS} \times 100\%$, em que BKS é a melhor solução conhecida da instância e $SOLUTION$ é o valor médio obtido com o algoritmo. A coluna $T_M(s)$ apresenta o tempo em segundos do algoritmo, referente à média das execuções até o critério de parada.

Para comparação dos resultados 4 algoritmos da literatura foram selecionados. Foram estes, DELS (TEOH; PONNAMBALAM; KANAGARAJ, 2015), um algoritmo que também utiliza a metaheurística ED para o *CVRP*, porém faz o uso da Mutação contínua na diversificação, PSO (CHANDRAMOULI; SRINIVASAN; NARENDRAN, 2012), CVRP-FA (ALTABEEB; MOHSEN; GHALLAB, 2019) e LNS-ACO (AKPINAR, 2016). Instâncias que não foram encontradas no *CVRPLIB* foram desconsideradas na comparação.

As Tabelas 5, 6, 7 apresentam a comparação de resultados dos algoritmos CDELS, DELS, PSO e CVRP-FA, nos pacotes A, B, P respectivamente. A Tabela 11 apresenta a performance do algoritmo LNS-ACO no pacote A. A Tabela 8 apresenta a performance dos algoritmos CDELS e DELS no pacote E. As Tabelas 9, 10 apresentam a performance do algoritmo CDELS nos pacotes F, M.

Os trabalhos usados para comparação não apresentaram resultados para todos os pacotes aqui testados. O algoritmo LNS-ACO foi o único que apresentou tempo de resolução dos pacotes em seu trabalho. Por isso, seus resultados no pacote A foram apresentados na Tabela 5.

Diferente das demais instâncias, a população inicial gerada para as instâncias M-n200-k16 e P-n55-k15 não possuiu qualquer solução viável. Para a instância M-n200-k16 o algoritmo foi capaz de viabilizar as soluções, finalizando com todas as soluções viáveis. Porém, para a instância P-n55-k15 o algoritmo não foi capaz de viabilizar qualquer solução e então não foi possível apresentar algum resultado nesta instância.

		CDELS			DELS		PSO		CVRP-FA		
Instância	*BKS*	*Avg.*	*%Dev*	$T_M(s)$	*Solution*	*%Dev*	*Solution*	*%Dev*	*Best*	*Avg.*	*%Dev*
A-n32-k5	784	784	0.00	0.05	-	-	784	0.00	796	798.1	1.80
A-n33-k5	661	661	0.00	0.02	661	0.00	661	0.00	661	661	0.00
A-n33-k6	742	742	0.00	0.02	742	0.00	742	0.00	742	742.7	0.09
A-n34-k5	778	778	0.00	0.04	778	0.00	778	0.00	778	778	0.00
A-n36-k5	799	799	0.00	0.13	799	0.00	799	0.00	799	804.2	0.65
A-n37-k5	669	669	0.00	0.04	669	0.00	669	0.00	669	669	0.00
A-n37-k6	949	949	0.00	0.11	949	0.00	949	0.00	949	955.5	0.68
A-n38-k5	730	730	0.00	0.25	730	0.00	730	0.00	730	730.7	0.10
A-n39-k5	822	822	0.00	0.23	822	0.00	822	0.00	822	822	0.00
A-n39-k6	831	831	0.00	0.43	831	0.00	831	0.00	831	834.6	0.43
A-n44-k6	937	937	0.00	0.43	937	0.00	937	0.00	937	937	0.00
A-n45-k6	944	944	0.00	2.65	944	0.00	944	0.00	953	959.4	1.63
A-n45-k7	1146	1146	0.00	0.48	1146	0.00	1146	0.00	1147	1153.3	0.64
A-n46-k7	914	914	0.00	0.88	914	0.00	914	0.00	914	914	0.00
A-n48-k7	1073	1073	0.00	0.62	1073	0.00	1073	0.00	1073	1073	0.00
A-n53-k7	1010	1010	0.00	4.27	1010	0.00	1014	0.40	1011	1014.8	0.48
A-n54-k7	1167	1167	0.00	3.10	1167	0.00	1170	0.26	1172	1172	0.43
A-n55-k9	1073	1073	0.00	0.91	1073	0.00	1073	0.00	1074	1078.6	0.52
A-n60-k9	1354	1354	0.00	5.09	1354	0.00	1356	0.15	1355	1364.7	0.79
A-n61-k9	1034	1034	0.00	12.20	1035	0.10	1038	0.39	1039	1048.4	1.39
A-n62-k8	1288	1288	0.00	56.49	1288	0.00	1288	0.00	1298	1312.2	1.88
A-n63-k9	1616	1616	0.00	24.89	1624	0.50	1626	0.62	1630	1644.8	1.78
A-n63-k10	1314	1314	0.00	203.67	1316	0.15	1320	0.46	1314	1333.4	1.48
A-n64-k9	1401	1401	0.00	234.09	1416	1.07	1409	0.57	1420	1424.9	1.71
A-n65-k9	1174	1174	0.00	29.96	1181	0.60	1177	0.26	1178	1180.7	0.57
A-n69-k9	1159	1159	0.00	12.06	1165	0.52	1162	0.26	1162	1174	1.29
A-n80-k10	1763	1763	0.00	135.01	1779	0.91	1778	0.85	1773	1787.2	1.37

Tabela 5 – Comparação de performance dos algoritmos no Pacote A (Augerat, 1995).

		CDELS			DELS		PSO		CVRP-FA		
Instância	*BKS*	*Avg.*	*%Dev*	$T_M(s)$	*Solution*	*%Dev*	*Solution*	*%Dev*	*Best*	*Avg.*	*%Dev*
B-n31-k5	672	672	0.00	0.03	672	0.00	672	0.00	672	672	0.00
B-n34-k5	788	788	0.00	0.03	788	0.00	788	0.00	788	789.7	0.22
B-n35-k5	955	955	0.00	0.04	955	0.00	955	0.00	955	955.1	0.01
B-n38-k6	805	805	0.00	0.20	805	0.00	805	0.00	806	806.2	0.15
B-n39-k5	549	549	0.00	0.31	549	0.00	549	0.00	550	553.9	0.89
B-n41-k6	829	829	0.00	0.36	829	0.00	829	0.00	829	829.8	0.10
B-n43-k6	742	742	0.00	0.19	742	0.00	742	0.00	742	742	0.00
B-n44-k7	909	909	0.00	0.20	909	0.00	909	0.00	909	913.3	0.47
B-n45-k5	751	751	0.00	0.66	751	0.00	751	0.00	751	754.2	0.43
B-n45-k6	678	678	0.00	2.25	678	0.00	678	0.00	686	692.8	2.18
B-n50-k7	741	741	0.00	0.07	741	0.00	741	0.00	741	744.8	0.51
B-n50-k8	1312	1312	0.00	33.77	1313	0.08	1315	0.23	1318	1329.6	1.34
B-n51-k7	1032	1032	0.00	1.34	1033	0.10	1038	0.58	-	-	-
B-n52-k7	747	747	0.00	0.90	747	0.00	747	0.00	747	747.6	0.08
B-n56-k7	707	707	0.00	0.63	707	0.00	707	0.00	709	713.9	0.98
B-n57-k7	1153	1153	0.00	15.53	1166	1.13	1162	0.78	1153	1162.5	0.82
B-n57-k9	1598	1598	0.00	8.72	1599	0.06	1598	0.00	1610	1615.2	1.08
B-n63-k10	1496	1496	0.00	151.06	1504	0.53	1496	0.00	1503	1540.6	2.98
B-n64-k9	861	861	0.00	6.75	861	0.00	864	0.35	862	887.8	3.11
B-n66-k9	1316	1316	0.00	28.14	1322	0.46	-	-	1319	1324.8	0.67
B-n67-k10	1032	1032	0.00	178.52	1032	0.00	1034	0.19	1042	1067.6	3.45
B-n68-k9	1272	1272.1	0.01	1118.87	1281	0.71	1273	0.08	1278	1288.1	1.27
B-n78-k10	1221	1221	0.00	73.57	1230	0.74	1249	2.29	1224	1248	2.21

Tabela 6 – Comparação de performance dos algoritmos no Pacote B (Augerat, 1995).

		CDELS			DELS		PSO		CVRP-FA		
Instância	*BKS*	*Avg.*	*%Dev*	$T_M(s)$	*Solution*	*%Dev*	*Solution*	*%Dev*	*Best*	*Avg.*	*%Dev*
P-n16-k8	450	450	0.00	0.01	450	0.00	450	0.00	450	450	0.00
P-n19-k2	212	212	0.00	0.01	212	0.00	212	0.00	212	212	0.00
P-n20-k2	216	216	0.00	0.01	216	0.00	216	0.00	216	216	0.00
P-n21-k2	211	211	0.00	0.01	211	0.00	211	0.00	211	211	0.00
P-n22-k2	216	216	0.00	0.01	216	0.00	216	0.00	216	216	0.00
P-n22-k8	603	603	0.00	0.01	603	0.00	603	0.00	603	602.5	0.00
P-n23-k8	529	529	0.00	0.01	533	0.76	529	0.00	529	529	0.00
P-n40-k5	458	458	0.00	0.04	458	0.00	458	0.00	458	459.9	0.00
P-n45-k5	510	510	0.00	0.15	510	0.00	510	0.00	510	510	0.00
P-n50-k7	554	554	0.00	0.78	554	0.00	554	0.00	554	557.3	0.00
P-n50-k8	631	631	0.00	14.94	641	1.58	635	0.63	631	633.4	0.00
P-n50-k10	696	696	0.00	115.39	696	0.00	696	0.00	697	702.9	0.14
P-n51-k10	741	741	0.00	0.99	742	0.13	741	0.00	742	750.3	0.13
P-n55-k7	568	568	0.00	6.09	568	0.00	568	0.00	568	573.5	0.00
P-n55-k10	694	694.1	0.01	502.00	694	0.00	695	0.14	698	699.4	0.58
P-n55-k15	989	-	-	-	989	0.00	989	0.00	-	-	-
P-n60-k10	744	744	0.00	1.98	744	0.00	744	0.00	749	752.6	0.67
P-n60-k15	968	968	0.00	2.36	968	0.00	968	0.00	968	983.2	0.00
P-n65-k10	792	792	0.00	3.23	792	0.00	795	0.38	792	799.3	0.00
P-n70-k10	827	827	0.00	51.44	827	0.00	827	0.00	827	827.9	0.00
P-n76-k4	593	593	0.00	140.94	593	0.00	593	0.00	593	598.8	0.00
P-n76-k5	627	627	0.00	245.66	629	0.32	627	0.00	628	630.7	0.16
P-n101-k4	681	681	0.00	249.55	685	0.59	681	0.00	681	684.7	0.00

Tabela 7 – Comparação de performance dos algoritmos no Pacote P (Augerat, 1995).

		CDELS			DELS		CVRP-FA		
Instância	*BKS*	*Avg.*	*%Dev*	$T_M(s)$	*Solution*	*%Dev*	*Best*	*Avg.*	*%Dev*
E-n22-k4	375	375	0.00	0.01	375	0.00	375	375	0.00
E-n23-k3	569	569	0.00	0.01	569	0.00	569	569	0.00
E-n30-k3	534	534	0.00	0.05	534	0.00	534	534.6	0.11
E-n33-k4	835	835	0.00	0.05	835	0.00	835	845.2	1.22
E-n51-k5	521	521	0.00	2.26	521	0.00	521	521	0.00
E-n76-k7	682	682	0.00	60.04	689	1.03	683	688.9	1.01
E-n76-k8	735	735	0.00	136.44	738	0.41	-	-	-
E-n76-k10	830	830	0.00	318.13	843	1.57	835	844.1	1.70
E-n76-k14	1021	1021	0.00	360.67	1042	2.06	1029	1039.3	1.79
E-n101-k14	1067	1067	0.00	2188.39	1086	1.78	-	-	-

Tabela 8 – Comparação de performance dos algoritmos no Pacote E (Christofides and Eilon, 1969).

		CDELS		
Instância	*BKS*	*Avg.*	*%Dev*	$T_M(s)$
F-n45-k4	724	724	0.00	0.21
F-n72-k4	237	237	0.00	9.01
F-n135-k7	1162	1162	0.00	9514.87

Tabela 9 – Performance no Pacote F (Fisher, 1994).

		CDELS		
Instância	*BKS*	*Avg.*	*%Dev*	$T_M(s)$
M-n101-k10	820	820	0.00	19.47
M-n121-k7	1034	1034	0.00	6277.50
M-n151-k12	1015	1016.7*	0.17	66294.17
M-n200-k16	1274	1329.3**	4.34	88168.63
M-n200-k17	1275	1288.6**	1.07	157286.67

** apenas 2 execuções obtiveram o ótimo*
*** nenhuma execução obteve o ótimo*

Tabela 10 – Performance no Pacote M (Christofides, Mingozzi and Toth, 1979).

		LNS-ACO		
Instância	*BKS*	*Worst*	*Best*	*CPU(s)*
A-n32-k5	784	784	784	856.21
A-n33-k5	661	661	661	900.06
A-n33-k6	742	742	742	947.86
A-n34-k5	778	778	778	909.04
A-n36-k5	799	799	799	1055.60
A-n37-k5	669	669	669	1103.70
A-n37-k6	949	949	949	1113.40
A-n38-k5	730	730	730	1139.20
A-n39-k5	822	822	822	1202.80
A-n39-k6	831	832.99	831	1266.00
A-n44-k6	937	941.97	937	1567.80
A-n45-k6	944	962.97	957.97	1728.10
A-n45-k7	1146	1151.96	1146	1741.70
A-n46-k7	914	917.02	914	1804.50
A-n48-k7	1073	1084.05	1084.05	1978.60
A-n53-k7	1010	1016.97	1010	2323.70
A-n54-k7	1167	1174.00	1167	2497.40
A-n55-k9	1073	1073.97	1073	2771.00
A-n60-k9	1354	1359.01	1354	3345.40
A-n61-k9	1034	1075.98	1066.98	3355.70
A-n62-k8	1288	1312.99	1307.96	3363.30
A-n63-k10	1314	1337.00	1328.98	3800.10
A-n63-k9	1616	1653.98	1648.97	3651.20
A-n64-k9	1401	1422.02	1415.01	3831.80
A-n65-k9	1174	1191.02	1185.04	3854.20
A-n69-k9	1159	1178.01	1170.01	4460.90
A-n80-k10	1763	1822.06	1815.01	6493.60

Tabela 11 – Performance no Pacote A (Augerat, 1995) do algoritmo LNS-ACO.

O algoritmo LNS-ACO (um híbrido entre dos algoritmos *Large Neighbourhood Search* e *Ant Colony Optimization*), apresentado em (AKPINAR, 2016), foi codificado em MATLAB 7.9 e executado em um Intel i7-2640 CPU 2.80 GHz. Os resultados do LNS-ACO no pacote A são apresentados na Tabela 11.

7 Conclusão e Trabalhos Futuros

Os resultados obtidos neste trabalho utilizando a adaptação do algoritmo para utilização dos mecanismos da Evolução Diferencial e as estratégias de Busca Local propostas demonstraram que o algoritmo gerado é bastante competitivo no cenário do problema.

Após a calibração dos parâmetros e com as estratégias definidas, o teste de performance do algoritmo obteve o ótimo em 99,81% das 530 execuções no pacote A, B, F, sendo 270, 230 e 30 execuções respectivamente. No pacote E foi obtido o ótimo em 100% das 110 execuções. Em uma instância do pacote P e M a geração inicial aleatória utilizada não foi capaz de gerar qualquer indivíduo. No pacote M o algoritmo foi capaz de viabilizar estas soluções ao longo de sua execução, porém no pacote P o algoritmo não foi capaz de viabilizar qualquer indivíduo e desta forma não foi possível apresentar a solução para esta instância P. No pacote M apenas 44% das execuções obtiveram o ótimo e no pacote P, com exceção de 1 instância em que nenhuma solução foi obtida, apenas uma execução não atingiu o ótimo e o ótimo foi obtido em todas as demais execuções.

Apesar da meta-heurística Evolução Diferencial ter sido proposta para problemas contínuos, percebemos que a utilização dela em problemas combinatórios também pode ser eficaz. Com a adaptação utilizada em (TEOH; PONNAMBALAM; KANAGARAJ, 2015), discutida brevemente na seção 5, o algoritmo também foi capaz de obter o ótimo, mas a partir da instância A-n61-k9 e com exceção da instância A-n62-k8 o algoritmo não conseguiu o ótimo nas execuções no pacote A. O algoritmo CDELS foi capaz de obter o ótimo em 100% das execuções neste pacote, demonstrando que a utilização das posições ao invés do cálculo com o conteúdo pode ser bastante eficaz.

Como trabalhos futuros, pretendemos utilizar técnicas mais inteligentes para a geração da população inicial, de forma a garantir mais riqueza de posições na geração inicial, que ao contrário leva à estagnação do algoritmo e convergência precoce, buscando também soluções viáveis mesmo nas instâncias que esta geração é dificultada. Pretendemos também paralelizar o algoritmo, que neste trabalho foi implementado de forma sequencial. Além disso, adaptar o algoritmo para que funcione com outras variações do VRP e estender os testes em novos pacotes com instâncias maiores como o pacote X (Uchoa et al.), em que as instâncias possuem 101 a 1001 clientes.

Referências

AKPINAR, S. Hybrid large neighbourhood search algorithm for capacitated vehicle routing problem. *Expert Systems with Applications*, Elsevier, v. 61, p. 28–38, 2016. Citado 2 vezes nas páginas 34 e 39.

ALTABEEB, A. M.; MOHSEN, A. M.; GHALLAB, A. An improved hybrid firefly algorithm for capacitated vehicle routing problem. *Applied Soft Computing*, Elsevier, v. 84, p. 105728, 2019. Citado na página 34.

BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, INFORMS, v. 6, n. 2, p. 154–160, 1994. Citado na página 22.

BRAEKERS, K.; RAMAEKERS, K.; NIEUWENHUYSE, I. V. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, Elsevier, v. 99, p. 300–313, 2016. Citado 4 vezes nas páginas 7, 11, 13 e 14.

CHANDRAMOULI, A.; SRINIVASAN, L. V.; NARENDRAN, T. Efficient heuristics for large-scale vehicle routing problems using particle swarm optimization. *International Journal of Green Computing (IJGC)*, IGI Global, v. 3, n. 2, p. 34–50, 2012. Citado na página 34.

CLARKE, G.; WRIGHT, J. W. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, Informs, v. 12, n. 4, p. 568–581, 1964. Citado 2 vezes nas páginas 13 e 16.

DANTZIG, G. B.; RAMSER, J. H. The truck dispatching problem. *Management science*, Informs, v. 6, n. 1, p. 80–91, 1959. Citado 2 vezes nas páginas 13 e 16.

DAS, S.; SUGANTHAN, P. N. Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, IEEE, v. 15, n. 1, p. 4–31, 2010. Citado na página 28.

EL-SHERBENY, N. A. *Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods*. [S.l.]: Journal of King Saud University-Science, 2010. v. 22.3. 123-131 p. Citado na página 13.

LAPORTE, G. Fifty years of vehicle routing. *Transportation Science*, INFORMS, v. 43, n. 4, p. 408–416, 2009. Citado na página 14.

LENSTRA, J. K.; KAN, A. R. Complexity of vehicle routing and scheduling problems. *Networks*, Wiley Online Library, v. 11, n. 2, p. 221–227, 1981. Citado na página 11.

MINGYONG, L.; ERBAO, C. An improved differential evolution algorithm for vehicle routing problem with simultaneous pickups and deliveries and time windows. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 23, n. 2, p. 188–195, 2010. Citado na página 22.

ONWUBOLU, G. C.; DAVENDRA, D. *Differential evolution: A handbook for global permutation-based combinatorial optimization*. [S.l.]: Springer Science & Business Media, 2009. v. 175. Citado 3 vezes nas páginas 7, 19 e 20.

PRICE, K.; STORN, R. M.; LAMPINEN, J. A. *Differential evolution: a practical approach to global optimization*. [S.l.]: Springer Science & Business Media, 2006. Citado 2 vezes nas páginas 11 e 17.

QIAN, B. et al. Scheduling multi-objective job shops using a memetic algorithm based on differential evolution. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 35, n. 9-10, p. 1014–1027, 2008. Citado na página 22.

QIAN, B. et al. An effective hybrid de-based algorithm for multi-objective flow shop scheduling with limited buffers. *Computers & Operations Research*, Elsevier, v. 36, n. 1, p. 209–233, 2009. Citado na página 22.

REINELT, G. Tsplib95. *Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Heidelberg*, v. 338, 1995. Citado na página 28.

SHOSHANA, A. *The vehicle-routing problem with delivery and back-haul options*. [S.l.]: Research Logistics, 1996. v. 43.3. 415-434 p. Citado na página 13.

STORN, R.; PRICE, K. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, Springer, v. 11, n. 4, p. 341–359, 1997. Citado 3 vezes nas páginas 11, 17 e 19.

TEOH, B. E.; PONNAMBALAM, S. G.; KANAGARAJ, G. Differential evolution algorithm with local search for capacitated vehicle routing problem. *International Journal of Bio-Inspired Computation*, Inderscience Publishers (IEL), v. 7, n. 5, p. 321–342, 2015. Citado 5 vezes nas páginas 22, 25, 28, 34 e 40.

TOTH, P.; VIGO, D. *The vehicle routing problem*. [S.l.]: SIAM, 2002. Citado 2 vezes nas páginas 11 e 13.