APLICAÇÃO DE METAHEURÍSTICA HÍBRIDA UTILIZANDO GRASP E BUSCA TABU NA RESOLUÇÃO DO PROBLEMA DE ROTEAMENTO DE VEÍCULO

MICHEL DIAS DE ARRUDA
micheldarruda@gmail.com

RESUMO

Este trabalho aborda o Problema do Roteamento de Veículos e tem como objetivo a minimização dos custos da distância percorrida por um veículo atendendo a restrição de capacidade desse. Para resolução desse problema serão usada uma abordagem híbrida através da utilização das metaheurísticas GRASP e Busca Tabu em conjunto. Busca-se Contribuir com um algoritmo de baixo tempo computacional e que possa obter soluções de boa qualidade. O algoritmo tem como algoritmo central para criação de soluções a Busca Tabu, que em sua construção inicial utiliza-se da fase de construção do GRASP. Nos experimentos, no que tange a construção do GRASP, foi utilizado um valor adaptativo para o tamanho da lista de candidatos que serve como base para a seleção aleatória do elemento a ser inserido na solução e no que tange a Busca Tabu, foram utilizados valores fixos do tamanho da Lista Tabu e número máximo de iterações. Para visualização dos resultados que podem ser obtidos pelo algoritmo foram utilizadas de instâncias conhecidas da literatura e também de comparações com suas soluções.

PALAVARAS CHAVE. metaheurísticas, busca tabu, GRASP, problema do roteamento de veículos, algoritmo híbrido.

1. Introdução

Nas últimas décadas tem se notado um crescente interesse de pesquisadores pelo Problema de Roteamento de Veículos (PRV). Esse interesse pode ser entendi através das motivações que acercam a resolução desse problema.

A principal motivação para estudos é a crescente necessidade da redução de custos com transporte em aplicações logísticas, uma vez que o transporte pode representar 15% do valor final do produto repassado ao consumidor.

O Problema de Roteamento de Veículos (PRV) proposto por Dantzig & Ramser (1959) consiste em um conjunto de consumidores com suas necessidades de demanda, um determinado número de veículos, com suas capacidades, para atendê-los e um depósito onde se iniciam e terminam as rotas cada rota da solução.

Este trabalho tem como objetivo desenvolver um modelo para abordar o Problema de Roteamento de Veículos utilizando como algoritmo a Busca Tabu e em sua construção de solução inicial a etapa de construção do GRASP.

O principal objetivo é contribuir com um algoritmo de baixo tempo computacional e que possa obter soluções de boa qualidade.

O trabalho apresenta no capítulo 2 a contextualização do Problema de Roteamento de Veículos, trazendo conceitos, nomenclaturas, definições, variações e sua formulação matemática para o problema.

Apresenta no capítulo 3 a metodologia desenvolvida para a resolução do problema, através da definição dos métodos utilizados e do planejamento dos experimentos.

O capítulo 4 apresenta os resultados.

O capítulo 5 apresenta as conclusões e os possíveis trabalhos futuros.

2. Revisão da Literatura

Esta seção apresenta conceitos, nomenclaturas, definições e variações do Problema do Roteamento de veículos. Além disso, apresenta formulação matemática para um problema genérico considerando restrições como: capacidade dos veículos e distância percorrida.

2.1 Visão Geral

O Problema de Roteamento de Veículos (PRV) é um problema da área da otimização combinatória. Consiste no atendimento das demandas de um conjunto de clientes por intermédio de veículos, que partem de um ou mais depósitos. A principal restrição do Problema de

Roteamento de Veículos é que cada veículo *v* possui uma capacidade C_v e o somatório de todas as demandas dos clientes atendidos por um veículo *v* não pode ultrapassar C_v.

Dantzig e Ramser foram os primeiros autores a formular o Problema de Roteamento de Veículos, em 1959, quando estudaram a aplicação real na distribuição de gasolina para estações de venda de combustíveis.

A função objetivo depende das características do problema. Os mais comuns são minimizar o custo total da operação, minimizar o tempo total de transporte, minimizar a distância total percorrida, minimizar a utilização de veículos, etc.

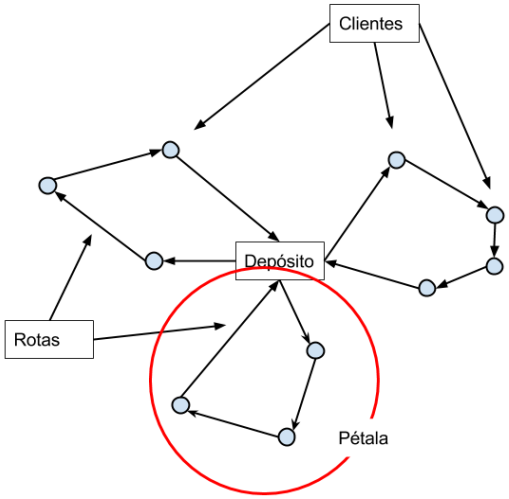

Figura 1 - Demonstração do roteamento de veículos

2.2 Tipos de Problema do Roteamento de Veículos

Além do Problema de Roteamento de Veículo clássico, que é abordado nesse trabalho, existem diversas variações do problema com diversas características como abaixo.

Tabela 1 - Fonte: Bodin, Golden, Assad, (1983)

Características	**Possíveis Opções**
Tamanho da frota	Um veículo ou múltiplos veículos
Tipo de frota	Homogênea e heterogênea
Domicílio dos veículos	Único depósito e múltiplos depósitos
Natureza das demandas	Estocástica e determinística
Localização das demandas	Nos nós e nos arcos
Tipo de rede	Não-direcionada, direcionada, euclidiana
Restrições quanto a capacidade do veículo	Capacidades iguais, diferentes e sem capacidade
Tempo máximo de duração da rota	O mesmo tempo para todas as rotas, diferentes para rotas diferentes e tempo não imposto
Operação	Entrega, recolhimento e ambos

Custos	Variáveis por rota e fixo
Objetivos	Minimizar custo total das rotas, minimizar a soma dos custos fixos e variáveis e minimizar números de veículos requeridos

A partir das características listadas acima pode-se obter diversas variações do problema de roteamento de veículos como na figura abaixo.

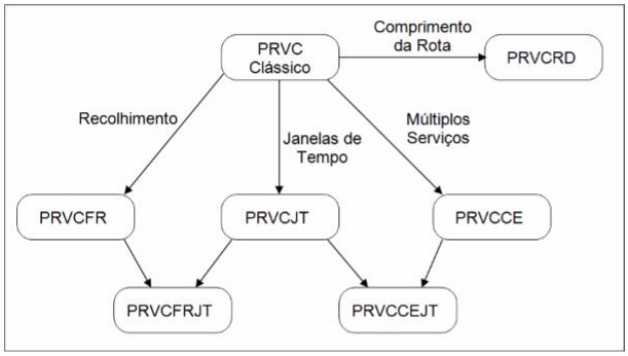

Figura 2 - Variações básicas do PRV (Galafassi, 2011)

As variações são:

Problema do Roteamento de Veículos Capacitados com Restrinções de Distância (PRVCDR) – É considerado o comprimento da rota.

Problema do Roteamento de Veículos Capacitados com Fretes de Regresso (PRVCFR) – É considerado o recolhimento de materiais na volta para o depósito

Problema do Roteamento de Veículos Capacitados com Janelas de Tempo (PRVCJT) – É considerada uma janela de tempo para atendimento ao cliente.

Problema do Roteamento de Veículos Capacitados com Coletas e Entregas (PVRCCE) – É considerado múltiplos serviços, coletas e entregas na mesma rota.

A partir dessas variações acima poder ser criadas outras variações:

Problema do Roteamento de Veículos Capacitados com Fretes de Retorno e Janelas de Tempo (PRVCFRJT)

Problema do Roteamento de Veículos Capacitados com Coletas e Entregas e Janelas de Tempo (PRVCCEJT)

2.3 Formulação Matemática

O problema pode ser definido sobre um grafo completo $G=(V,A)$ sendo $V = \{v_0, v_1, \ldots, v_n\}$ um conjunto de vertices e $A=\{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ um conjunto de arestas. O vértice v_0 representa o depósito e o restante os clientes.

Os clientes possuem uma demanda c_i a ser atendida.

Por último, têm-se a matriz $D=(d_{ij})$ que representa a distância de deslocamento associado à aresta (i,j).

$$\text{Minimizar} \sum_i \sum_j \sum_v c_{ij} X_{ij}^v; \tag{6.1}$$

sujeito a:

$$\sum_i \sum_v X_{ij}^v = 1, \text{ para todo } j; \tag{6.2}$$

$$\sum_j \sum_v X_{ij}^v = 1, \text{ para todo } i; \tag{6.3}$$

$$\sum_i X_{ip}^v - \sum_j X_{pj}^v = 0, \text{ para todo } p, v; \tag{6.4}$$

$$\sum_i d_i \left(\sum_j X_{ij}^v \right) \leq Q_v, \text{ para todo } v; \tag{6.5}$$

$$\sum_{j=1}^{n} X_{0j}^v \leq 1, \text{ para todo } v; \tag{6.6}$$

$$\sum_{i=1}^{n} X_{i0}^v \leq 1, \text{ para todo } v; \tag{6.7}$$

Onde X^v_{ij} são variáveis binárias que indicam se o arco (v_i, v_j) é utilizado pelo veículo v. A função objetivo de minimização da distância/custo/tempo aparece na equação (6.1).

As restrições representadas nas equações (6.2) e (6.3) juntas garantem que cada vértice é atendido por um único veículo.

A equação (6.4) garante que o veículo deixa o vértice aonde foi atender a demanda tão

logo alcance este vértice.

A capacidade do veículo Q_v é expressa na equação (6.5) aonde d_i representa a demanda a cada vértice i.

As restrições (6.6) e (6.7) expressam que a disponibilidade do veículo não pode ser excedida.

3. Metodologia

O algoritmo foi desenvolvido através da utilização das metaheurísticas Busca Tabu e GRASP e executado através de algumas tecnologias. Essa seção tem como objetivo a apresentação das metodologias desenvolvidas.

3.2 Modelagem das Classes

A codificação do algoritmo foi realizada através da linguagem JAVA e esse foi modelado previamente com base no paradigma orientado a objetos a fim de manter o padrão adotado pelo JAVA e se utilizar dos benefícios do paradigma, como a organização e fácil legibilidade.

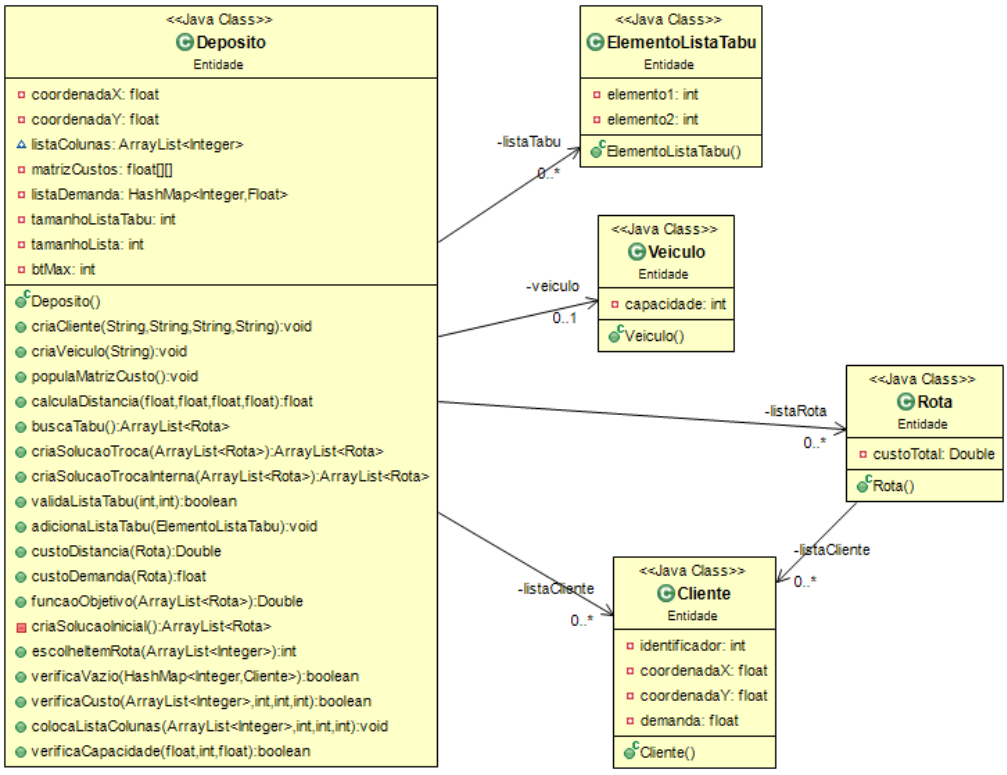

Figura 3 - Diagrama de Classes Simplificado

3.3 Matriz de Custos

Com base na formulação matemática, que indica a geração de um grafo completo entre todos os vértices, foi criada a matriz de distâncias entre todos os vértices.

A matriz de distâncias entre todos os vértices tem como objetivo ser a base para a execução das metaheurísticas Busca Tabu e GRASP, pois fornece o valor de comparação entre as distâncias dos elementos candidatos a entrar na solução.

A matriz de distância é criada antes da execução das metaheurísticas e alocada em memória para que possa ser consultada. A criação dela desfaz a necessidade de se calcular as distancias em tempo de execução.

Essas distâncias são obtidas através do cálculo da distância euclidiana ($\sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2)}$) e utilizando as coordenadas de cada elemento.

Tabela 2 – Matriz de Custos

Matriz de Distância Entre Vértices				
	0	1	2	3
0		D_{01}	D_{02}	D_{03}
1	D_{10}		D_{12}	D_{13}
2	D_{20}	D_{21}		D_{23}
3	D_{30}	D_{31}	D_{32}	
...				

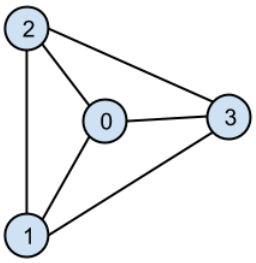

Figura 4 - Grafo Completo

3.4 Construção Inicial

A construção da solução inicial é realizada utilizando a fase de construção do GRASP que tem como característica a construção gulosa, aleatória e adaptativa.

É realizada de maneira iterativa onde uma elemento viável é incluído na solução por iteração. A cada iteração é criada uma lista de elementos melhor avaliados e nessa é escolhida de forma aleatória um elemento. Na próxima iteração a lista de elementos é adaptada considerando a inclusão do último elemento na solução.

Para seleção de elemento na lista de elementos melhor avaliados foi utilizada a forma totalmente aleatória com o parâmetro $\alpha = 1$.

Abaixo pode ser visualizado o algoritmo criado para a criação da solução inicial utilizando a fase de construção GRASP que será utilizada na Busca Tabu.

```
rotas[] criaSolucaoInicial(){
        rotas[];

        Enquanto(listaCandidatos != vazio){
                        if(elementoPivo == deposito){
                                rota = novaRota();
                                rota.adicionaElemento(deposito);
                                rota.adicionaCusto(0);
                        }

                        Para(elemento : listaCandidatos){
                                Se(verificaMelhorElem(listaElementosSelecionados)
                                        && verificaCapacidade(custoDemanda(rota),
                                                              veiculo.capacidade,
                                                              elemento.demanda)){

                                        colocaElementoLista(listaElementosSelecionados, elemento);
                                }
                        }

                        if(listaElementosSelecionados.tamanho() != 0){
                                elemento = escolheItemRota(listaElementosSelecionados);

                                rota.adicionaElemento(elemento);
                                rota.adicionaCusto(rota.custo() + custo(elementoPivo, elemento));

                                elementoPivo = elemento;

                                listaCandidatos.remove(elemento);
                                listaElementosSelecionados.limpar();
                        } else {
                                elementoPivo = deposito;

                                rota.adicionaElemento(deposito);
                                rota.adicionaCusto(rota.custo() + custo(elementoPivo, elemento));
                                rotas.adicionaRota(rota);
                        }
        }
        returna rotas;
}
```

3.5 Busca Tabu

A Busca Tabu é o algoritmo central que foi adotado para a resolução do Problema de Roteamento de Veículo, pois é considerada por muitos a melhor metaheurística para esse fim.

Utiliza-se da Lista Tabu, como memória, a fim de restringir movimentos repetidos e escapar de ótimos locais.

Abaixo pode ser visualizado o algoritmo utilizado. Primeiro passo desse algoritmo é a criação da solução inicial através da fase de construção do GRASP, essa definida na seção anterior. Após isso, é ajustado o número de máximo de iterações que o loop principal vai executar.

No loop, são chamadas as funções que se utilizam de troca entre elementos a fim de criar soluções através dos vizinhos da solução atual. O valor obtido nessas trocas é comparado com o valor da solução atual e se há melhora, a solução passa a ser a solução obtida através das trocas.

```
rotas[] buscaTabu(){

        solucao = criaSolucaoInicial();

        iter = 0;
        melhorIter = 0;
        btMax = 1000;

        Enquanto((iter - melhorIter) <= btMax){
                iter++;

                solucaoTemp = criaSolucaoTrocaInterna(solucao)

                Se(funcaoObjetivo(solucaoTemp) < funcaoObjetivo(solucao)){
                        solucao = solucaoTemp;
                        melhorIter = iter;
                }

                solucaoTemp = criaSolucaoTroca(solucao)

                Se(funcaoObjetivo(solucaoTemp) < funcaoObjetivo(solucao)){
                        solucao = solucaoTemp;
                        melhorIter = iter;
                }
        }
        retorna solucao;
}
```

3.5.1 Construção de Vizinhanças

3.5.1.1 Construção por movimento inter-pétalas

Este movimento faz todas as combinações possíveis entre os clientes de duas pétalas e todas as combinações possíveis entre pétalas. Em cada troca é calculada a função objetivo permanecendo a melhor solução ao final de todas possíveis combinações. O melhor movimento de troca é registrado na Lista Tabu.

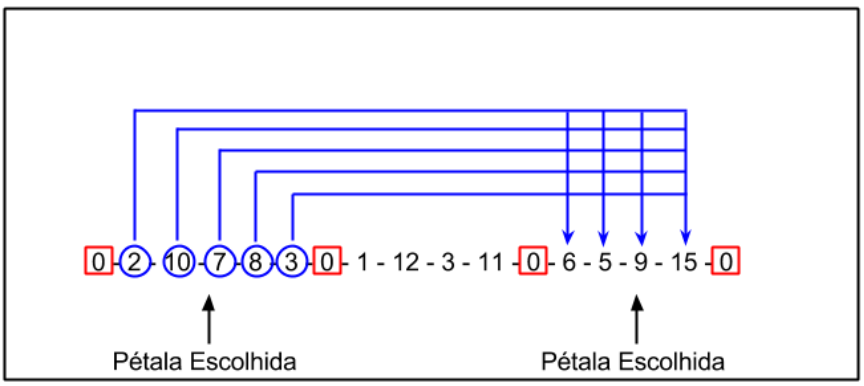

Figura 5 - Troca de elementos inter-pétalas

Abaixo está representado o algoritmo que demonstra a troca de elementos inter-pétalas.

```
rotas[] criaSolucaoTrocaInterna(solucao){
        rotas[];
        solucaoCriada = solucao;

        Para(rotaBase : solucao.rotas){

                Para(i=0; rotaBase.tamanho(); i++){
                        elementoBase = rotaBase[i];

                        Para(rota : solucao.rotas){

                                Para(i=0; rota.tamanho(); i++){
                                        elemento = rota[i];

                                        Se(rotaBase.demanda() - elementoBase.demanda() + elemento.demanda < veiculo.capacidade

                                           && rota.demanda() + elementoBase.demanda() - elemento.demanda < veiculo.capacidade){

                                                Se((i,j) nao existe em listaTabu){

                                                        RotaTempBase.removeElemento(elementoBase);
                                                        RotaTempBase.adicionaElemento(elemento);

                                                        RotaTemp.removeElemento(elemento);
                                                        RotaTemp.adicionaElemento(elementoBase);

                                                        solucaoTemp.adicionaRota(RotaTempBase);
                                                        solucaoTemp.adicionaRota(RotaTemp);

                                                        valorTemp = funcaoObjetivo(solucaoTemp);
                                                        valorCriada = funcaoObjetivo(solucaoTemp);

                                                        if(valorTemp < valorCriada){
                                                                solucaoCriada = solucaoTemp;

                                                                indiceElemento1 = i;
                                                                indiceElemento2 = j;
                                                        }
                                                }
                                        }
                                }

                        }
                }

        }

        listaTabu.adicionaElemento(indiceElemento1, indiceElemento2);
}
```

3.5.1.2 Construção por movimento intra-pétala

Este movimento faz todas as combinações possíveis entre os clientes de uma pétala e é realizada em todas as pétalas. Em cada troca é calculada a função objetivo permanecendo a melhor solução ao final de todas possíveis combinações. O melhor movimento de troca é registrado na Lista Tabu.

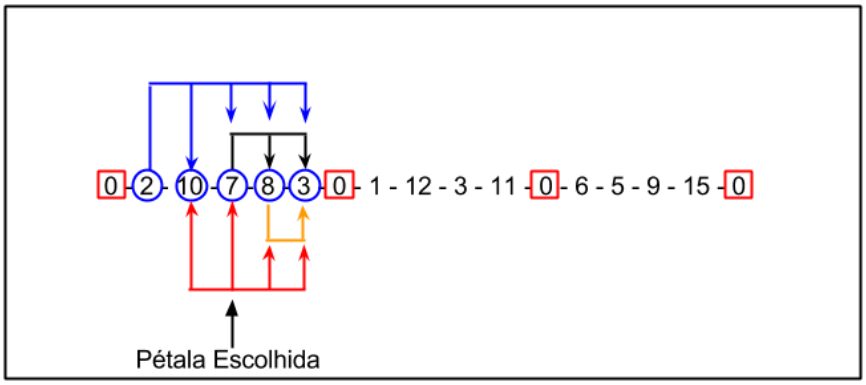

Figura 6 - Troca de elementos intra-pétala

Abaixo está representado o algoritmo que demonstra a troca de elementos intra-pétala.

```
rotas[] criaSolucaoTrocaInterna(solucao){
        rotas[];
        solucaoCriada = solucao;

        Para(rota : solucao.rotas){
                rotaBase = rota;

                Para(i=0; rota.tamanho(); i++){
                        elemento = rota[i];

                        Para(j=0; rota.tamanho(); j++){
                                elemento2 = rota[j];

                                Se((i,j) nao existe em listaTabu){

                                        RotaTemp.removeElemento(i);
                                        RotaTemp.removeElemento(j);

                                        RotaTemp.adicionaElemento(i, elemento);
                                        RotaTemp.adicionaElemento(j, elemento2);

                                        solucaoTemp.adicionaRota(RotaTemp);

                                        valorTemp = funcaoObjetivo(solucaoTemp);
                                        valorCriada = funcaoObjetivo(solucaoCriada);
                                        Se(valorTemp < valorCriada){
                                                solucaoCriada = solucaoTemp;

                                                indiceElemento1 = i;
                                                indiceElemento2 = j;
                                        }

                                }
                        }
                }

        }

        listaTabu.adicionaElemento(indiceElemento1, indiceElemento2);
}
```

3.6 Execução

Para a obtenção de resultados e visualização da eficiência do algoritmo foram utilizadas instancias propostas por Augerat et al. Nas seções seguintes são listadas instâncias utilizadas, assim como suas estruturas. Além disso, são exibidas as tecnologias utilizadas na execução do projeto e como os resultados foram obtidos.

3.6.1 Instâncias

As instâncias tem uma estrutura a qual podem-se obter o tamanho do veículo e a demanda e posicionamento de clientes e depósito. Abaixo podemos ver um exemplo de instância, onde o cliente de número 1(um) é o depósito, com demanda 0(zero) e coordenadas (40,50), e os demais os clientes tem suas próprias demandas e posicionamentos.

Tabela 3 - Exemplo de instância

Tamanho do Veículo	100		
Número Cliente	Coordenada X	Coordenada Y	Demanda
1	40.00	50.00	0.00
2	45.00	68.00	10.00
3	45.00	70.00	30.00
4	42.00	66.00	10.00
5	42.00	68.00	10.00
6	42.00	65.00	10.00
7	40.00	69.00	20.00
8	40.00	66.00	20.00
9	38.00	68.00	20.00
10	38.00	70.00	10.00
11	35.00	66.00	10.00
12	35.00	69.00	10.00

Na tabela abaixo são exibidos os dados referentes as instâncias testadas, como nome, melhor solução obtida, número de elementos, número de veículos na melhor solução e arquivo de entrada do programa.

A escolhe dessas instâncias entre as inúmeras disponíveis para utilização teve como ponto principal a variação da quantidade de elementos entre elas, para que possa ser visualizado o comportamento do algoritmo quando há um aumento de elementos.

Tabela 4 - Dados sobre as instâncias

Nome arquivo original	Melhor Solução	Número de elementos	Número de Veículos	Arquivo de entrada
A-N32-k5	784	32	5	Dado2.txt
A-N33-k5	661	33	5	Dado3.txt
A-N65-k9	1177	65	9	Dado4.txt
A-N48-k7	1073	48	7	Dado5.txt

3.6.2 Tecnologias Utilizadas

Para codificação do projeto foi utilizado a linguagem JAVA baseando-se no paradigma orientado a objetos. A IDE utilizada para codificação foi o Eclipse na versão Luna.

O computador utilizado para codificação e execução dos algoritmos tem como processador um Intel Core i7-4500U 1.80GHz, 8gb de memória RAM, 1tb de HD e utilizando como sistema operacional o Windows 8.1.

3.6.3 Realização da Execução

A realização da execução ocorreu em dois passos: ajustes do parâmetros e coleta dos resultados.

O ajuste dos parâmetros foi realizado através de diversas execuções visando a melhoria dos resultados. Parâmetros obtidos:

Fase de Construção GRASP

- Lista de elementos melhor avaliados = 20% do tamanho da lista de candidatos

Fase da Busca TABU

- Número de iterações sem melhora = 10000
- Tamanho Lista Tabu = 100

Após os ajustes foram coletados resultados através do registro de 5 execuções seguidas. Os resultados são exibidos da seção seguinte.

4. Resultados

Abaixo pode ser visualizada as tabelas com os resultados obtidos na execução do algoritmo utilizando instâncias propostas por Augerat et al.

Resultados Instância A-N32-k5

Tabela 5 - Resultado instância A-N32-k5

Ideal	Inicial	Final	Tempo (ms)
784	1003.285	959.608	7310
784	1144.062	1087.884	6034
784	1240.038	1078.806	5607
784	1315.449	1068.517	6192
784	1225.636	1064.657	6704

Resultados Instância A-N33-k5

Tabela 6 - Resultado instância A-N33-k5

Ideal	Inicial	Final	Tempo (ms)
661	1236.771	959.385	9586
661	1008.710	970.544	6637
661	1061.165	915.399	7313
661	892.835	892.835	8863
661	853.774	801.396	7208

Resultados Instância A-N65-k9

Tabela 7 - Resultado instância A-N65-k9

Ideal	Inicial	Final	Tempo (ms)
1177	2253.539	1873.430	23500
1177	2037.372	1797.374	21456
1177	1940.846	1719.049	30598
1177	2091.685	1793.750	18863
1177	2207.386	1911.951	19448

Resultados Instância A-N48-k7

Tabela 8 - Resultado instância A-N48-k7

Ideal	Inicial	Final	Tempo (ms)
1073	1415.268	1388.326	14547
1073	1497.395	1477.289	15509
1073	1331.357	1331.357	17012
1073	1770.026	1488.210	14313
1073	1562.929	1417.795	13773

5. Conclusões e trabalhos futuros

Esse trabalho foi desenvolvido com o objetivo da criação de um algoritmo para a resolução do Problema de Roteamento de Veículos. Foi desenvolvido uma Busca Tabu incorporando a criação da solução inicial a fase de construção do GRASP. Na Busca Tabu foi utilizada a Lista Tabu auxiliar ao algoritmo a buscar a solução em uma vizinhança maior que conseguiria em uma Busca Local simples. Nessa busca por novas vizinhanças foi utilizada as trocas entre elementos intra-pétala e inter-pétalas.

Na execução foi utilizada instâncias da conhecidas na literatura e proposta por Augerat et al. Durante a execução foram ajustados os parâmetros e visualizado os resultados.

Quanto ao resultado, foi satisfatório, porém acredita-se que pode melhorar com um melhor estudo do ajuste dos parâmetros.

O estudo do ajuste dos parâmetros seria um trabalho futuro, como a implementação do GRASP no lugar da Busca Tabu ou a mudança da fase de construção do GRASP para outra heurística de construção e realizar um estudo comparativo em cima do comportamento das múltiplas implementações.

Referências

ARAKAKI, Reinaldo GI. O problema de roteamento de veículos e algumas metaheurísticas. Monografia apresentada para o Exame de Qualificação do Curso de Computação Aplicada-Instituto Nacional de Pesquisas Espaciais, 1998.

NEVES, Tiago A.; SOUZA, M. J. F.; MARTINS, A. X. Resolução do problema de Roteamento de Veículos com Frota Heterogênea e Janelas de Tempo. Monografia-Apresentada no curso de Ciência da Computação, da Universidade Federal de Ouro Preto, MG, 2004.

SIMAS, Ettiene Pozzobom Lazzeris. Utilizando a Busca Tabu na Resolucao do Problema de Roteamento de Veiculos. 2007. Tese de Doutorado. MSc Thesis, Universidade do Vale do Rio dos Sinos.

GALAFASSI, Cristiano. Aplicação de metaheuristicas na abordagem do problema de roteamento de veículos capacitados com janelas de tempo. 2011. Tese de Mestrado. Universidade do Vale do Rio dos Sinos.