

**UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC  
CENTRO DE EDUCAÇÃO SUPERIOR DO ALTO VALE DO ITAJAÍ – CEAVI  
ENGENHARIA DE SOFTWARE**

**TIAGO FUNK**

**DESENVOLVIMENTO E AVALIAÇÃO DE UM NOVO COMPONENTE BASEADO  
EM PATH RELINKING PARA O ALGORITMO MEMPLAS NO PROBLEMA QCARS**

**IBIRAMA**

**2019**

**TIAGO FUNK**

**DESENVOLVIMENTO E AVALIAÇÃO DE UM NOVO COMPONENTE BASEADO  
EM PATH RELINKING PARA O ALGORITMO MEMPLAS NO PROBLEMA QCARS**

Trabalho de conclusão apresentado ao curso de Engenharia de Software do Centro de Educação Superior do Alto Vale do Itajaí (CEAVI), da Universidade do Estado de Santa Catarina (UDESC), como requisito parcial para a obtenção do grau de bacharel em Engenharia de Software.

**Orientador:** Prof. Dr. Fernando dos Santos

**IBIRAMA**

**2019**

**TIAGO FUNK**

**DESENVOLVIMENTO E AVALIAÇÃO DE UM NOVO COMPONENTE  
BASEADO EM PATHRELINKING PARA O ALGORITMO MEMPLAS NO  
PROBLEMA QCARS**

Trabalho de Conclusão apresentado ao Curso de Engenharia de Software, da Universidade do Estado de Santa Catarina, como requisito parcial para obtenção do grau de Bacharel em Engenharia de Software

**Banca Examinadora**

Orientador

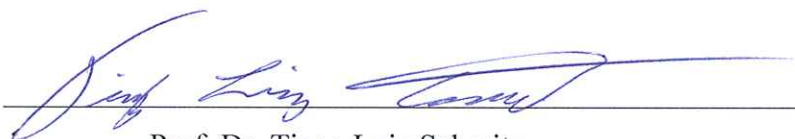


Prof. Dr. Fernando dos Santos  
UDESC/IBIRAMA

Membros:



Prof. Dr. Paolo Moser  
UDESC/IBIRAMA



Prof. Dr. Tiago Luiz Schmitz  
UDESC/IBIRAMA

**Ibirama – SC, 19 de novembro de 2019**

## RESUMO

Este trabalho é um estudo sobre o problema Quota Car Traveling Salesman (QCaRS). Este problema consiste em minimizar o custo de uma viagem entre um grupo de cidades, visitando apenas um subgrupo, e garantindo a visita de cidades que cumpram com uma satisfação mínima para o viajante. Cada cidade possui uma quota, que indica a satisfação do viajante em visitá-la. O viajante precisa também minimizar o custo do percurso. Este trabalho propôs desenvolver e avaliar um componente baseado em *Path Relinking* para o algoritmo genético MemPlas proposto por Goldberg et al. (2016). A hipótese considerada neste trabalho é de que a substituição do componente *Plasmídeo* pelo *Path Relinking* poderia trazer resultados melhores considerando qualidade de soluções. Dois experimentos foram realizados para avaliar esta hipótese. O primeiro experimento evidenciou que os resultados apresentados pelas três variações do algoritmo não apresentavam diferenças quando todas as instâncias foram analisadas juntas. Ao realizar uma análise por instância, notou-se que o algoritmo MemPlas mais o novo componente *Path Relinking* apresentou melhoras em algumas instâncias. No segundo experimento, ao remover o componente de busca local, o algoritmo MemPlas original apresentou os melhores resultados no geral e na análise instância por instância, rejeitando a hipótese.

**Palavras-chave:** QCaRS. Metaheurística. Algoritmos genéticos. MemPlas. Path Relinking.

## ABSTRACT

This paper is a study of the Quota Car Traveling Salesman problem (QCaRS). This problem consists in minimizing the cost of a trip between a group of cities by visiting only one subgroup, and ensuring the visit of cities that meet with minimal traveler satisfaction. Each city has a quota, which indicates the satisfaction of the traveler to visit it. The traveler also needs to minimize the cost of the route. This work proposes to develop and evaluate a *path relinking* component for the MemPlas genetic algorithm proposed by Goldberg et al. (2016). The hypothesis considered in this paper is that replacing the *Plasmid* component with *Path Relinking* could bring better results considering the quality of solutions. Two experiments were performed to evaluate this hypothesis. The first experiment showed evidence that there is no difference among overall results of the three algorithm variations when all instances were considered together. When performing an analysis per instance, it was noted that the MemPlas algorithm plus the new *Path Relinking* component showed improvements in some instances. In the second experiment, with the local search component disabled, the original MemPlas algorithm showed the best overall results, as well as in the instance by instance analysis, rejecting the hypothesis.

**Keywords:** QCaRS. Metaheuristic. Genetic Algorithms. MemPlas. Path Relinking.

## LISTA DE ILUSTRAÇÕES

Figura 1	– Template de metaheurística baseada em população . . . . .	13
Figura 2	– Template do algoritmo evolucionário . . . . .	14
Figura 3	– Exemplo de representação do indivíduo do MemPlas . . . . .	17
Figura 4	– Algoritmo MemPlas . . . . .	18
Figura 5	– Ordem de execução do método <i>realizarBuscaLocais</i> . . . . .	19
Figura 6	– Exemplo de inversão do indivíduo . . . . .	19
Figura 7	– Funcionamento do Crossover . . . . .	21
Figura 8	– Funcionamento do Plasmídeo . . . . .	22
Figura 9	– <i>Template</i> de busca dispersa . . . . .	23
Figura 10	– <i>Template</i> do Path Relinking . . . . .	24
Figura 11	– Algoritmo com componente baseado em <i>Path Relinking</i> . . . . .	26
Figura 12	– Algoritmo com plasmídeo e <i>Path Relinking</i> . . . . .	27
Figura 13	– Procedimento do Path Relinking - Do início para o final . . . . .	28
Figura 14	– Procedimento do Path Relinking - Do final para o início . . . . .	29
Figura 15	– Procedimento do Path Relinking - Aleatório . . . . .	30
Figura 16	– Boxplot com os resultados em instâncias euclidianas no experimento 1 . . . .	37
Figura 17	– Boxplot com os resultados em instâncias não euclidianas no experimento 1 . .	38
Figura 18	– Boxplot com os resultados em instâncias euclidianas no experimento 2 . . .	45
Figura 19	– Boxplot com os resultados em instâncias não euclidianas no experimento 2 .	45
Figura 20	– Boxplot com os resultados em instâncias euclidianas para ambos experimentos	49
Figura 21	– Boxplot com os resultados em instâncias não euclidianas para ambos experimentos . . . . .	49

## LISTA DE TABELAS

Tabela 1	– Variações do Path Relinking . . . . .	28
Tabela 2	– Parâmetros utilizados para a calibração do irace . . . . .	33
Tabela 3	– Parâmetros escolhidos pelo irace . . . . .	34
Tabela 4	– Resultados nas instâncias euclidianas no experimento 1 . . . . .	36
Tabela 5	– Resultados nas instâncias não euclidianas no experimento 1 . . . . .	37
Tabela 6	– Resultados do teste do ANOVA por instância no experimento 1 . . . . .	39
Tabela 7	– Resultados do teste de Tukey no experimento 1 . . . . .	40
Tabela 8	– Resultados nas instâncias euclidianas no experimento 2 . . . . .	43
Tabela 9	– Resultados nas instâncias não euclidianas no experimento 2 . . . . .	44
Tabela 10	– Resultados do teste de Tukey agrupados por algoritmos no experimento 2 . .	46
Tabela 11	– Resultados do teste do ANOVA por instância para o experimento 2 . . . . .	46
Tabela 12	– Resultados do teste de Tukey no experimento 2 para instâncias euclidianas .	47
Tabela 13	– Resultados do teste de Tukey no experimento 2 para instâncias não euclidianas	48

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	8
1.1	Problema	9
1.2	Objetivos	9
1.2.1	Objetivo Geral	9
1.2.2	Objetivos específicos	9
1.3	Justificativa	10
1.4	Hipótese	10
1.5	Metodologia	10
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	12
2.1	Metaheurísticas	12
2.2	Algoritmos Evolucionários	13
2.3	O Problema QCaRS	15
2.4	Algoritmo MemPlas	17
2.5	Path Relinking	22
2.6	Trabalhos Relacionados	24
<b>3</b>	<b>COMPONENTE <i>PATH RELINKING</i> PROPOSTO</b>	26
<b>4</b>	<b>AValiação EXPERIMENTAL</b>	31
4.1	Implementação dos Algoritmos	31
4.2	Materiais e Métodos dos Experimentos	32
4.2.1	Instâncias QCaRS	32
4.2.2	Calibração dos Algoritmos	32
4.2.3	Execução dos Algoritmos e Coleta de Dados	35
4.3	Resultados e Discussão	35
4.3.1	Experimento 1	35
4.3.2	Experimento 2	43
<b>5</b>	<b>CONCLUSÃO</b>	50
	<b>REFERÊNCIAS</b>	52



## 1 INTRODUÇÃO

Este trabalho é um estudo sobre o problema do Caixeiro Alugador com Quota — do inglês, *Quota Car Traveling Salesman (QCaRS)*. Este problema consiste em minimizar o custo de uma viagem entre um grupo de cidades, visitando apenas um subgrupo, e garantindo a visita de cidades que cumpram com uma satisfação mínima para o viajante. Cada cidade possui uma quota, que indica a satisfação do viajante em visitá-la. O viajante precisa também minimizar o custo do aluguel de veículos, custo de combustível, e outras despesas para fazer o percurso. Este problema tem aplicações para construção de roteiros de turismo para viajantes e construção de rotas para representantes de empresas que querem visitar várias filiais.

A formulação do problema QCaRS deriva do CaRS, que por sua vez, é baseado no Problema do Caixeiro Viajante (PCV). O PCV é um problema já clássico na literatura, que consiste em minimizar o custo de percorrer  $n$  vértices, partindo e finalizando a rota no mesmo vértice. O CaRS é uma generalização do PCV, onde o viajante deve também se preocupar com o custo de alugar veículos, que geram despesas de acordo com o tipo de veículo. O CaRS por derivar do PCV, também é um problema NP-difícil (MENEZES et al., 2017).

O QCaRS é muito semelhante ao CaRS. A diferença é que há uma quota em cada um dos vértices. O valor da quota é importante pelo motivo que o viajante em alguns casos não pode visitar todas as cidades, assim é necessário que visite apenas as de maior interesse. A rota que deve ser criada precisa atender uma satisfação mínima que o viajante possui (MENEZES et al., 2017).

Pelo fato do problema QCaRS ser NP-difícil, é comum a utilização de metaheurísticas para resolvê-los. Uma metaheurística especifica um *template* de solução algorítmica para problemas de otimização. Tal estratégia frequentemente usa iterações de refinamento das soluções. Metaheurísticas fornecem soluções satisfatórias para problemas grandes e complexos em um tempo razoável. Porém, ao contrário de métodos exatos, não há garantia que encontrem soluções ótimas (TALBI, 2009). Um exemplo de metaheurística é a *busca local*, que ao ser aplicada sobre uma solução candidata, executa pequenas alterações visando melhorá-la.

Este trabalho se propõe a realizar uma melhoria no algoritmo MemPlas proposto por Goldberg et al. (2016) para o problema QCaRS. O MemPlas é um algoritmo evolucionário com um operador plasmídeo. De acordo com Goldberg et al. (2016), um algoritmo evolucionário utiliza princípios da evolução para construir operadores e realizar uma busca no espaço de solução. O algoritmo se baseia na troca de genes. Seja da forma vertical, ou seja, herança de pais para filhos, ou ainda a troca de genes horizontal, que é aquisição de genes externos por meios que não são a herança. Plasmídeos são pequenas moléculas capazes de se replicar

independentemente, sendo um dos mecanismos que proporciona a troca de genes de forma horizontal. Os plasmídeos são importantes para a troca de genes horizontal, pois podem carregar uma quantidade de genes e pode reorganizá-los.

O funcionamento básico do MemPlas está em pegar uma população inicial de genes, aplicar o plasmídeo, aplicar uma série de buscas locais, em um número  $n$  de repetições. A fase do plasmídeo e das busca locais são chamadas nesse trabalho de componentes. A proposta deste trabalho é de implementar um *Path Relinking* no lugar do plasmídeo. A ideia principal do *Path Relinking* é gerar e explorar a trajetória no espaço de busca que conecta uma solução inicial e uma solução alvo. O caminho entre estas duas soluções no espaço de busca (espaço de vizinhança) geralmente gerará soluções que compartilham atributos comuns com as soluções de entrada (TALBI, 2009).

## 1.1 PROBLEMA

No trabalho de Menezes et al. (2017), é possível observar que as metaheurísticas adotadas na solução estado da arte, como por exemplo Goldberg et al. (2016) e no trabalho de Menezes et al. (2017), embora obtenham soluções em menos tempo, não são capazes de obter os mesmos resultados que *solvers* lineares em todas as instâncias. Nem mesmo os *solvers* apresentam confiabilidade da respostas. Quando se determina um limite de tempo para resolução de 80.000 segundos, algumas instâncias grandes não são resolvidas pelos *solvers*.

## 1.2 OBJETIVOS

### 1.2.1 Objetivo Geral

O objetivo é melhorar o algoritmo metaheurístico MemPlas, proposto por Goldberg et al. (2016), implementando novos componentes na tentativa de melhorar o estado da arte da solução do problema do QCaRS.

### 1.2.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

1. Propor um novo componente baseado em *Path Relinking* para o algoritmo MemPlas.
2. Realizar experimentos para coletar resultados sobre o tempo e qualidade da solução.

3. Realizar testes estatísticos para averiguar se o novo componente obteve resultado melhores que os anteriores propostos na literatura.

### 1.3 JUSTIFICATIVA

O problema QCaRS tem uma aplicação direta na construção de roteiros para visita de várias cidades por um viajante. Assim, implementar um algoritmo que resolva esse problema de forma rápida e com uma solução ótima pode impactar positivamente a vida de pessoas. Atualmente os algoritmos resolvem bem instâncias com poucas cidades e poucos carros, mas ao aumentar o número de ambos, nem mesmo *solvers* lineares conseguem dar resposta ótimas em tempo aceitável.

A utilização de um algoritmo que resolva o problema QCaRS em um tempo muito pequeno, na ordem de segundos, pode impactar a vida cotidiana das pessoas. Por exemplo, considere uma empresa que vende passeios turísticos. Esta empresa pode utilizar o algoritmo para construir passeios que atinjam um alto grau de satisfação para os clientes em um tempo pequeno. Essa empresa vai ter uma competitividade maior em comparação a outra que faça esse passeio de forma manual.

O algoritmo ainda pode ser aplicado em um contexto onde um funcionário de uma empresa precisa visitar filiais. O roteiro precisa ficar pronto pela manhã para que possa começar a visitar as filiais mais importantes o mais rápido possível. Outro exemplo é um técnico em segurança visitando instalações dentro da empresa para inspeção de segurança. A inspeção precisa começar rapidamente e deve ser completada de forma rápida para que a empresa volte a funcionar. Com o algoritmo é possível montar um roteiro em poucos segundos para visitar as instalações mais importantes e vitais para a segurança.

### 1.4 HIPÓTESE

A hipótese deste trabalho é de que o uso de uma estratégia de melhoria de soluções, baseada em elite e *Path Relinking*, produz soluções de melhor qualidade que o algoritmo MemPlas.

### 1.5 METODOLOGIA

O novo componente para o algoritmo MemPlas será implementado utilizando uma seleção do tipo *Path Relinking*. O componente do algoritmo MemPlas que será substituído será o plasmídeo/*crossover*, todos os outros componentes serão mantidos iguais.

Para realizar os experimentos, o algoritmo implementado será executado com as instâncias disponibilizadas por Goldbarg et al. (2016). Cada instância será executada 30 vezes, e será feita a coleta de tempo e de qualidade da solução apresentada pelo algoritmo proposto neste trabalho.

Os dados coletados nos experimentos serão submetidos a testes estatísticos. O objetivo será verificar se há diferença estatisticamente significativa entre o algoritmo proposto neste trabalho e o MemPlas em relação a qualidade da resposta.

Do ponto de vista da abordagem, este trabalho será quantitativo, pois este trabalho tem a finalidade de coletar dados de qualidade da solução. Estes dados serão utilizados para comparação com o algoritmo MemPlas para validar a hipótese que o componente baseado em *Path Relinking* traz melhores resultados. A comparação será feita a partir de testes estatísticos.

Em relação ao ponto de vista dos objetivos, será uma pesquisa descritiva, pois tem a intenção de verificar se os dados obtidos a partir dos experimentos tem relação com a troca do componente no algoritmo.

## 2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção será apresentada a base teórica, contextualizando os assuntos para uma melhor entendimento do problema QCaRS , sobre metaheurísticas e as técnicas utilizadas no desenvolvimento do sistema.

### 2.1 METAHEURÍSTICAS

As metaheurísticas são uma família de técnicas de aproximação para encontrar soluções em problemas de otimização. Essa família de técnicas fornece soluções aceitáveis em um tempo razoável para problemas difíceis e complexos. Mas ao contrário de métodos de otimização exatos, não há garantia que soluções ótimas possam ser obtidas. Metaheurística pode ser definidas como o mais alto nível de metodologias genéricas que podem ser usadas para guiar estratégias de projeto de heurísticas para problemas específicos de otimização (TALBI, 2009).

Segundo Gendreau, Potvin et al. (2010), metaheurísticas são métodos de solução que realizam iterações entre procedimentos de melhoria local e estratégias de alto nível para escapar de soluções ótimas locais que podem induzir ao erro durante a busca da solução. Metaheurísticas utilizam-se de soluções que são construídas de forma arbitrária e sobre esta solução são feitas operações com a intenção de melhorar a qualidade desta, respeitando restrições do problema e parando quando não é mais possível obter uma melhora. Um problema comum de ocorrer, é que a partir de uma solução arbitrária apenas é possível melhorá-la até um ponto que não chega suficientemente próximo da solução ótima, esse acontecimento é chamado de problema da solução ótima local. Como o algoritmo para quando não consegue mais melhorar a solução atual, ele considera que esta solução é a ótima e que não possível mais melhorá-la. Uma característica das metaheurísticas é contornar o problema de soluções ótimas locais tentando se aproximar ou igualar das soluções ótimas.

As metaheurísticas cresceram em popularidade nos últimos 20 anos. Diversas áreas as utilizam, como por exemplo otimização estrutural em eletrônica, aerodinâmica, dinâmica de fluídos, telecomunicações, aprendizado de máquina, modelagem de sistemas, processamento de imagens, planejamento de rotas, logística, entre outros (TALBI, 2009). Basicamente onde existir um problema complexo que precisa ser otimizado, é muito provável que seja possível utilizar metaheurísticas para resolvê-lo em um tempo aceitável com uma resposta suficientemente boa.

No desenvolvimento de metaheurísticas, dois fatores contraditórios devem ser analisados: a diversificação e a intensificação. A diversificação é a exploração do espaço de busca, visitando

regiões ainda não visitadas. Na intensificação, regiões promissoras são exploradas com mais detalhamento tentando melhor uma solução o máximo possível (TALBI, 2009). Quando uma metaheurística apenas se preocupa com a intensificação, muito provável que vá cair no problema da solução ótima local. Se o foco for a diversificação, várias partes do espaço de busca podem ser explorados sem muito detalhe, o que pode levar a descarte de soluções locais promissoras. Um equilíbrio entre os dois fatores é o ideal.

Embora metaheurísticas consigam encontrar soluções muito próximas das soluções ótimas, utilizá-las para resolver problemas simples ou instâncias pequenas de problemas difíceis não é aconselhável. Metaheurísticas não garantem soluções ótimas. Utilizar algoritmos exatos pode ser melhor para garantir a qualidade da solução (TALBI, 2009). Um *solver* utiliza algoritmos exatos. O ponto forte de metaheurísticas é encontrar soluções suficientemente próximas da solução ótima em instâncias grandes de problemas difíceis em um tempo consideravelmente menor que algoritmos exatos.

## 2.2 ALGORITMOS EVOLUCIONÁRIOS

Algoritmos evolucionários fazem parte da categoria das metaheurísticas baseadas em população. O ponto principal é realizar melhorias em uma população de soluções. A fase inicial é criar uma população. A fase seguinte é a substituição, onde uma nova população é gerada, e é realizado um processo de seleção entre a nova população e a anterior, que vai gerar a população que vai interagir com a próxima geração. O algoritmo finaliza apenas quando um critério de parada é satisfeito (TALBI, 2009). O *template* para uma metaheurísticas baseadas em população pode ser visto na figura 1.

Figura 1 – Template de metaheurística baseada em população

- 1:  $P = P_0$  /\* Geração da população inicial \*/
- 2: **repeat**
- 3:    $P'_i = \text{gerarPopulacao}()$  /\* Gera uma nova população para esta iteração \*/
- 4:    $P_{i+1} = \text{selecionePopulacao}(P_i \cup P'_i)$  /\* Seleção da população para a próxima \*/
- 5: **until** Critério de parada for satisfeito

Fonte: Adaptado de Talbi (2009).

A geração da população tem grande importância, pois desempenha um papel crucial na eficiência do algoritmo. O principal ponto é lidar com a diversificação. Caso a população não seja muito diversificada, pode ocorrer uma convergência prematura, que ocorre quando indivíduos muito parecidos são gerados, assim esta população não possui características que podem levar a solução ótima global. A forma de criar esta população pode ser classificada em quatro categorias: Geração aleatória, diversificação sequencial, diversificação paralela e iniciação

heurística. Na geração aleatória é criada a população inicial de forma aleatória. Cada elemento da população recebe valores randômicos que respeitam o intervalo possível pelo problema. Na diversificação sequencial, as soluções são geradas em sequência de tal forma que a diversidade é otimizada, respeitando um intervalo igual entre todos os elementos da população. Em uma estratégia de diversificação paralela, as soluções de uma população são geradas de maneira independente e em paralelo. Na inicialização heurística, pode-se utilizar uma heurística qualquer para criar a população (TALBI, 2009).

O critério de parada se resume em duas categorias. A primeira categoria é a estática, onde se conhece quando a busca vai ser parada. Um exemplo é quando uma constante indica o número máximo de iterações ou de gerações. A segunda categoria é a adaptativa, ocorre quando não se conhece quando a busca vai terminar. Utiliza-se geralmente critérios estatísticos que levam em consideração a população atual (TALBI, 2009).

Os algoritmos evolucionários são os mais estudados das metaheurísticas baseadas em população. Eles representam uma classe de algoritmos que simulam a evolução das espécies, e se baseiam na noção de competição. A intenção do algoritmo é a evolução de uma população de indivíduos. No início do algoritmo, a população é gerada aleatoriamente. Cada indivíduo representa uma possível solução para o problema e por consequência, tem uma aptidão que representa o valor que a função objetivo retorna quando submetida aos valores que o indivíduos possui. Em cada iteração do algoritmo, indivíduos são selecionados a partir da geração atual. O critério de seleção é sortear indivíduos dentro da população, onde se dá maior probabilidade de seleção para indivíduos com a melhor aptidão. Os indivíduos selecionados irão se reproduzir para gerar a prole, utilizando operadores de variação, como o *crossover* ou a mutação. O próximo passo é aplicar um critério de substituição para escolher quais indivíduos formarão a próxima geração, ocorrendo uma escolha entre indivíduos da geração atual e a prole gerada. O algoritmo vai realizar iterações enquanto o critério de parada não for alcançada (TALBI, 2009). O template do algoritmo evolucionário pode ser visto na figura 2.

Figura 2 – Template do algoritmo evolucionário

```

1:  $P_0 = gerarPopulacao()$  /* Gera a população inicial */
2: while Critério de parada for satisfeito do
3:    $avaliacao(P_i)$  /* Calcula a aptidão dos indivíduos */
4:    $P'_i = selecao(P_i)$  /* Seleciona os melhores indivíduos */
5:    $P'_i = reproducao(P'_i)$  /* Realiza o cruzamento */
6:    $avaliacao(P'_i)$  /* Calcula a aptidão da prole */
7:    $P_{(i+1)} = substituicao(P_i, P'_i)$  /* Insere os indivíduos da prole na população */
8: end while

```

Fonte: Adaptado de Talbi (2009).

### 2.3 O PROBLEMA QCaRS

O problema QCaRS envolve criar uma rota entre um conjunto de cidades, onde se deve minimizar os custos da viagem sempre realizando o aluguel de veículos. Os custos são o aluguel de carros, gastos com gasolina, pedágios e multas caso seja realizada uma devolução de veículo fora da cidade do aluguel. Deve ser considerado também que uma satisfação mínima deve ser atendida. A restrição de satisfação mínima é importante, pois nem sempre é viável visitar todas as cidades. No próximo parágrafo será apresentado a formulação matemática.

Seja um grafo completo  $G = (V, A)$ , onde  $V$  é um conjunto com  $n$  nós simbolizando as cidades e  $A$  é um conjunto de arcos simbolizando as estradas entre as cidades. Um valor de quota é atribuído para cada cidade. Existe também um conjunto  $C$  de veículos que está disponível para aluguel. Cada veículo possui custos associados a ele, como consumo de combustível, custo de aluguel, pedágios, etc. Por último o QCaRS considera um valor de quota mínimo que o viajante deve recolher durante a sua viagem, esse valor é representado por  $\omega$ . O objetivo do QCaRS é completar um caminho hamiltoniano em  $G$  pagando o mínimo possível com relação aos custos com os veículos. O QCaRS é um problema NP-Difícil (MENEZES et al., 2017).

As características do problema QCaRS são: não existe restrição de viagem entre as cidades, é possível ir de uma cidade para qualquer outra cidade; qualquer carro pode ser alugado em qualquer cidade; o carro alugado pode ser devolvido em qualquer cidade; cada carro só pode ser alugado uma vez; os custos de utilizar um carro para ir da cidade  $a$  para cidade  $b$  é igual ao valor de ir da cidade  $b$  até a cidade  $a$ ; um carro devolvido em uma cidade diferente daquela que foi alugado implica uma multa a ser paga (MENEZES et al., 2017).

A definição formal e o modelo matemático do problema QCaRS foram apresentados em Menezes, Goldberg e Goldberg (2014) e atualizados em Goldberg et al. (2016). As equações 2.1 até 2.14 apresentam o modelo matemático do QCaRS.



$$\text{Minimize: } \sum_{c \in C} \sum_{i, j \in V} d_{ij}^c * f_{ij}^c + \sum_{c \in C} \sum_{i, j \in V} y_{ij}^c * w_{ij}^c \quad (2.1)$$

$$\text{Sujeito a: } \sum_{c \in C} \sum_{j \in V} f_{1j}^c = \sum_{c \in C} \sum_{i \in V} f_{i1}^c = 1 \quad (2.2)$$

$$\sum_{c \in C} \sum_{i \in V} f_{ih}^c = \sum_{c \in C} \sum_{j \in V} f_{hj}^c \leq 1 \quad \forall h \in V \quad (2.3)$$

$$a_i^c = \left( \sum_{j \in V} f_{ij}^c \right) \left( \sum_{c' \in C, c' \neq c} \sum_{h \in V} f_{hi}^{c'} \right) \quad \forall c \in C, i \in V, i > 1 \quad (2.4)$$

$$e_i^c = \left( \sum_{j \in V} f_{ji}^c \right) \left( \sum_{c' \in C, c' \neq c} \sum_{h \in V} f_{ih}^{c'} \right) \quad \forall c \in C, i \in V, i > 1 \quad (2.5)$$

$$w_{ij}^c = a_j^c * e_i^c \quad \forall c \in C, \forall i, j \in V \quad (2.6)$$

$$\sum_{c \in C} a_1^c = 1 \quad (2.7)$$

$$\sum_{i \in V} a_i^c \leq 1 \quad \forall c \in C \quad (2.8)$$

$$\sum_{i \in V} a_i^c = \sum_{i \in V} e_i^c \quad \forall c \in C \quad (2.9)$$

$$\sum_{i \in V} \left( \left( \sum_{c \in C} \sum_{j \in V} f_{ij}^c \right) q_i \right) \geq \omega \quad (2.10)$$

$$2 \leq u_i \leq n \quad \forall i = 2, \dots, n \quad (2.11)$$

$$u_i - u_j + 1 \leq (n - 1) \left( 1 - \sum_{c \in C} f_{ij}^c \right) \quad \forall i, j = 2, \dots, n \quad (2.12)$$

$$f_{ij}^c * w_{ij}^c * a_i^c * e_i^c \in [0, 1] \quad (2.13)$$

$$u_i \in \mathbb{N} \quad (2.14)$$

A variável  $d_{ij}^c$  representa o custo de usar o carro  $c \in C$  para ir da cidade  $i$  até a cidade  $j$ . Um custo  $y_{ij}^c$  deve ser pago caso um carro  $c$  seja alugado em uma cidade  $i$  e entregue em  $j$ , onde as cidades  $i$  e  $j$  são diferentes. A variável  $u_i$  indica a posição na rota. As seguintes variáveis são binárias:  $f_{ij}^c$  recebe valor 1 para o caso que o carro  $c$  viaja da cidade  $i$  para a cidade  $j$  e valor 0 caso contrário;  $w_{ij}^c$  representa se o carro  $c$  é alugado na cidade  $j$  e entregue na cidade  $i$ ; a variável  $a_i^c$  representa se o carro  $c$  é alugado na cidade  $i$ ; e a variável  $e_i^c$  representa se o carro  $c$  foi entregue na cidade  $i$  (GOLDBARG et al., 2016).

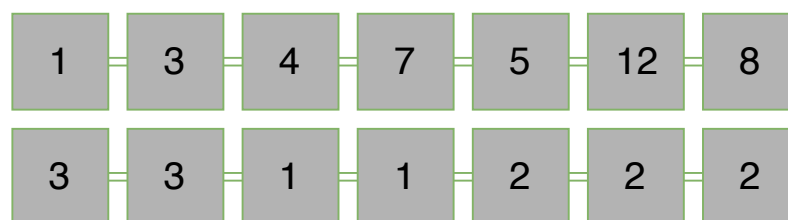
O primeiro termo da função objetivo (equação 2.1) expressa o custo das cidades visitadas e o segundo termo é o custo das multas pelos carros devolvidos em cidades diferentes daquelas em que foram pegos. A cláusula 2.2 obriga que a cidade 1 é a cidade que a rota deve começar e terminar. A cláusula 2.3 afirma que cada vértice deve ser visitado no máximo uma vez e se

um carro chega ao vértice  $i$ , então um carro deve deixar esse vértice. A cláusula 2.4 restringe se o carro  $c$  foi alugado na cidade  $i$ . A cláusula 2.5 limita se o carro foi entregue na cidade  $i$ . A cláusula 2.6 delimita se um carro foi alugado em uma cidade e entregue em outro. A cláusula 2.7 obriga que um carro seja alugado na cidade de início da rota. A cláusula 2.8 limita que um carro seja alugado apenas uma vez. A cláusula 2.9 obriga que um carro alugado seja entregue. A cláusula 2.10 obriga que o mínimo de quota seja coletado. As cláusulas 2.11 e 2.12 evitam sub-rotas. A cláusula 2.13 indica que as variáveis são binárias e a cláusula 2.14 indica que a variável  $u_i$  é um inteiro positivo.

## 2.4 ALGORITMO MEMPLAS

O algoritmo MemPlas, proposto por Goldberg et al. (2016), é um algoritmo evolucionário que utiliza os operadores de plasmídeo e *crossover*. A representação do indivíduo adotada para este algoritmo é de dois vetores, onde um vetor representa a sequência de cidades visitadas e a sequência de carros alugados é representado no outro. A aptidão é calculada com a equação 2.1. Na figura 3 é apresentado um exemplo de representação do indivíduo. Na parte de cima está o vetor que representa as cidades. Neste exemplo a rota começa na cidade 1, vai até a cidade 3, depois até a 4 e assim por diante. Na parte de baixo fica a ordem dos veículos alugados. O veículo 3 é pego na cidade 1, utilizado para ir para a cidade 3, depois para a cidade 4 e então devolvido. O veículo 1 é alugado na cidade 4 e devolvido na 5. O veículo 2 é alugado na cidade 5 e devolvido na cidade 1. A rota não mostra que a cidade 1 é a final, pois é uma restrição do problema e então fica implícito que todas as rotas terminam na cidade 1.

Figura 3 – Exemplo de representação do indivíduo do MemPlas



Fonte: Autor (2019).

O algoritmo MemPlas pode ser visto na figura 4. O algoritmo possui cinco parâmetros de entrada: o tamanho da população, o tamanho do plasmídeo, a taxa de recombinação  $pCross$ , o tamanho de conjunto de elite  $pElite$ , e o número máximo de iterações. O parâmetro  $pElite$  indica a proporção de melhores indivíduos da população que representam os que possuem melhor aptidão e devem ser selecionados para a próxima geração.

Figura 4 – Algoritmo MemPlas

```

1: Entradas : tamanhoPopulacao, tamanhoPlasmideo, pCross, pElite, numeroIteracoes
2: iteracoes = 0
3:  $P_0 = \text{gerarPopulacao}(\text{tamanhoPopulacao})$ 
4:  $\text{realizarBuscaLocais}(P_0)$ 
5:  $\text{elite} = \text{selecionarMelhores}(pElite)$ 
6: while iteracoes < numeroIteracoes do
7:    $\text{sel} = \text{selecione}(P_i)$ 
8:   if iteracoes / 10  $\neq$  0 then
9:      $\text{off} = \text{crossver}(\text{sel}, pCross)$ 
10:  else
11:     $\text{off} = \text{plasmideo}(\text{sel}, \text{tamanhoPlasmideo}, \text{elite})$ 
12:  end if
13:   $\text{realizarBuscaLocais}(P_i)$ 
14:   $\text{pop} = \text{torneioBinario}(\text{pop}, \text{off})$ 
15:  iteracoes = iteracoes + 1
16:   $\text{elite} = \text{atualiza}(pElite)$ 
17: end while

```

Fonte: adaptado de Goldberg et al. (2016).

O algoritmo inicia gerando uma população (linha 3). Cada indivíduo dessa população é criada com a ajuda da heurística de vizinhança próxima para criar o caminho entre duas cidades associando o custo do aluguel do carro para o caminho entre as duas cidades. A rota sempre vai começar na cidade 1. Enquanto a quota mínima não for alcançada, um carro  $c$  e uma cidade  $i$  são selecionados aleatoriamente para ir criando a rota, sempre levando em consideração que cidades já visitadas e carros já alugados não podem ser mais selecionados novamente para este indivíduo. Se durante a construção da rota todos os carros já foram alugados, o último carro deve ser utilizado para completar a rota até o fim. No momento que a quota é alcançada, então a rota deve ser finalizada realizando a volta para a cidade 1 (GOLDBARG et al., 2016).

O próximo passo é submeter o cromossomo a seis buscas locais, esses procedimentos visam melhorar a rota nos pontos onde os carros são trocados e a cota é coletada. São representadas na figura 4 (linha 4) pelo método *realizarBuscasLocais*. Estas seis buscas locais são representadas por seis métodos a saber: *removeSaving*, *invertSol*, *replaceSavingCar*, *replaceSavingCit*, *insertSavingCit* e *2-opt* e a ordem da execução pode ser vista na figura 5 (GOLDBARG et al., 2016).

O método *removeSaving* se concentra no custo da rota. Consiste em remover as cidades com as menores pontuações de satisfação da solução candidata, enquanto a exigência de satisfação ainda é atendida. O método *invertSol* inverte a ordem das cidades visitadas juntamente com os veículos associados. Por exemplo, considere a rota (1,2,3,4,5) onde o carro 1 é alugado na cidade 1 com a entrega na cidade 3 e o carro 2 é alugado na cidade 3 com a entrega na 1.

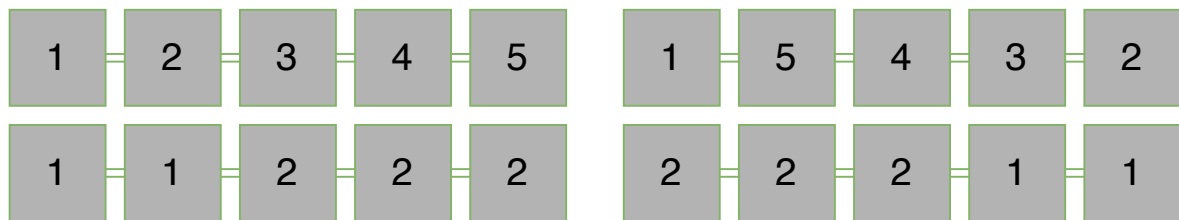
Figura 5 – Ordem de execução do método *realizarBuscaLocais*

- 1: *removeSaving(populacao)*
- 2: *invertSol(populacao)*
- 3: *insertSavingCit(populacao)*
- 4: *replaceSavingCit(populacao)*
- 5: *replaceSavingCar(populacao)*
- 6: *2opt(populacao)*

Fonte: adaptado de Goldberg et al. (2016).

Com a inversão, a rota fica (1,5,4,3,2) e o carro 2 é alugado na cidade 1 com a entrega na cidade 3 e o carro 1 é alugado na cidade 3 com a entrega sendo feita na cidade 1. Na figura 6 está a representação do exemplo do *invertSol*. No lado esquerdo está a rota original e no direita está a rota invertida.

Figura 6 – Exemplo de inversão do indivíduo



Fonte: Autor (2019).

Continuando com os métodos de busca locais, o *replaceSavingCar* tenta inserir carros que não foram utilizados na construção da rota. Cada carro que não está na rota é considerado. Cada carro fora da rota é inserido em cada posição iterativamente. Por exemplo, a rota (1,2,3,4,5) com os carros 2, 3 e 4, onde são alugados nas cidades 1, 3 e 5, respectivamente e devolvidos nas cidades 3, 5 e 1, respectivamente, criando o seguinte vetor para os carros (2,2,3,3,4). O carro 1 não está no vetor, então é iniciado a tentativa de inserir o veículo em todas as posições do vetor, criando as sequências (1,2,3,3,4), (1,1,3,3,4), (1,1,1,3,4) e assim por diante. O próximo passo começa a inserção a partir da segunda posição, como (2,1,3,3,4), (2,1,1,3,4), (2,1,1,1,4) e (2,1,1,1,1). É realizado esse processo em uma posição posterior até acabar as possibilidades. A sequência que apresentar o melhor resultado da função objetivo é escolhida.

O método *replaceSavingCit* tenta inserir cidades que não estão presentes na rota com intuito de aumentar a quota de satisfação. Por exemplo, a rota (1,3,4,5,6) não possui a cidade 2, assim o método tenta inserir a cidade em todas as posições: (1,2,4,5,6), (1,3,2,5,6), (1,3,4,2,6) e (1,3,4,5,2). O vetor que representa os carros não é alterado. No quinto método, chamado de *insertSavingCit* ocorre algo parecido com o *replaceSavingCit*, mas não há troca e sim apenas inserção de cidade cidade. No exemplo da rota (1,3,4,5,6), serão testados as rotas (1,2,3,4,5,6),

(1,3,2,4,5,6) e assim por diante até esgotar as possibilidades. O veículo que está associado a cidade anterior é associado para esta nova cidade inserida.

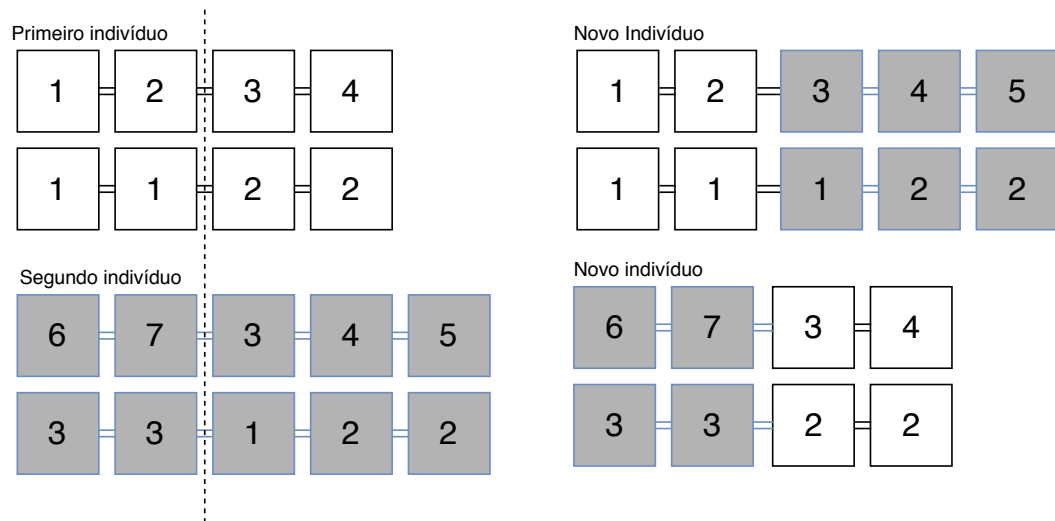
O último método de busca local é o *2-opt*. Ele consiste em eliminar duas arestas e reconectar os dois caminhos restantes de uma forma diferente para formar um novo caminho. Há apenas uma maneira de reconectar os caminhos que geram uma rota diferente. Entre todos os pares de arestas cuja troca *2-opt* diminui o comprimento da rota, o par que dá o menor rota é escolhido. A sequência dos veículos não é alterada (CROES, 1958).

O próximo passo do algoritmo MemPlas (linha 7) é selecionar os *pElite* indivíduos da população com a melhor aptidão, criando uma população elite. A partir desse ponto é iniciada as iterações sobre a população que irão parar apenas quando o critério de parada for alcançado, esse critério é um número fixo que é passado ao programa como parâmetro. O primeiro passo de cada iteração é realizar uma seleção de indivíduos. Logo após é iniciado o processo de recombinação. O MemPlas segue a lógica que a cada dez iterações, as nove primeiras devem utilizar o operador *crossover* e a décima é o operador plasmídeo.

O operador *crossover* consiste em pegar dois indivíduos que são escolhidos aleatoriamente com probabilidade uniforme. O valor de *pCross* representa a taxa de recombinação. O *crossover* utilizado é do tipo *one-point*, que consiste em dividir um cromossomo em um ponto e trocar as partes com outro cromossomo também dividido ao meio no mesmo ponto. Como o número do comprimentos dos cromossomos pode variar, um ponto aleatório é escolhido na faixa de índices do cromossomo menor. Cada um dos dois cromossomos partidos vão trocar as suas partes para gerar dois descendentes. Na figura 7 é possível ver um exemplo. O primeiro cromossomos possui quatro posições e o segundo cinco. O ponto escolhido para a recombinação foi entre a posição 2 e 3. Logo após é feita uma recombinação entre os dois trocando metade dos vetores.

O operador plasmídeo usa fragmentos externos de DNA para modificar os indivíduos. O MemPlas utiliza como fragmentos partes da população de elite (GOLDBARG et al., 2016). O método plasmídeo, (linha 11), recebe como parâmetro a população selecionada *sel*, um valor numérico *tamanhoPlasmideo* que representa o tamanho que o plasmídeo deve ter, e a população elite *elite*. O seu funcionamento consiste em pegar um cromossomo da população *sel* e um cromossomo da população *elite*. Após é selecionado um fragmento do tamanho *tamanhoPlasmideo* de forma aleatória do cromossomo da população *elite*. Esse fragmento é chamado de plasmídeo. Com o cromossomo da população *sel* é retirado um fragmento, também de tamanho *tamanhoPlasmideo*, de forma aleatória. A inserção do plasmídeo no cromossomo da população *sel* ocorre com o teste de inserção de todas as posições possíveis. Na figura 8 é possível ver dois indivíduos. O primeiro tem retirado um pedaço que iria se tornar o plasmídeo. O segundo

Figura 7 – Funcionamento do Crossover



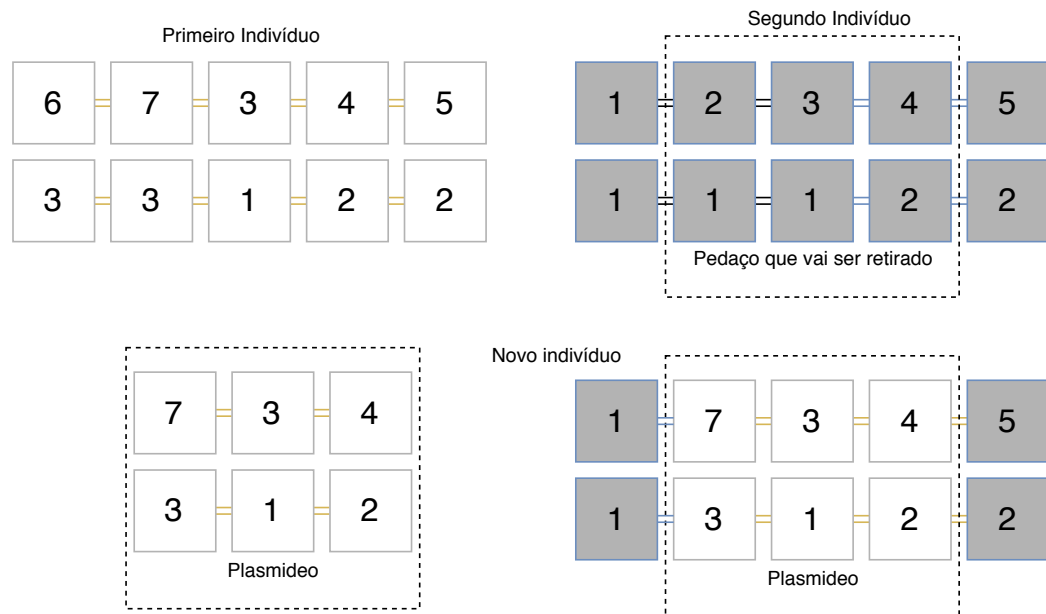
Fonte: Autor (2019).

tem selecionado um pedaço que vai ser retirado e substituído pelo plasmídeo gerando um novo indivíduo.

Os operadores *crossover* e plasmídeo não garantem que os indivíduos continuarão sendo válidos após a recombinação de genes. Pode ocorrer inconsistências com o cromossomo gerado. O MemPlas contorna esse problema executando um método que corrige as inconsistências. O seu funcionamento básico se divide em três fases: correção das cidades, correção dos carros e correção da restrição de quota mínima. Como não faz sentido visitar mais de uma vez uma cidade, cada repetição de cidades é substituída por um asterisco, que depois será substituído por uma cidade escolhida de forma aleatória das cidades ainda não presentes na solução. Se por acaso não existirem mais cidades, a posição do cromossomo que possuir um asterisco será deletada e cromossomo original será encurtado. A segunda fase consiste em detectar repetições de veículos, caso seja encontrado, é substituído por um asterisco. Em seguida, o vetor é percorrido da esquerda para a direita e cada asterisco é sobrescrita pelo veículo presente na posição imediatamente a esquerda do vetor. A terceira e última fase consiste em verificar se a rota coletou a quota mínima. O seu procedimento consiste em ordenar as cidades já visitadas em ordem crescente e iterativamente, começando pela primeira, trocar cidades visitadas por não visitadas com valor de quota maior até a quantidade mínima ser atendida. Se ao terminar a verificação pelo vetor ainda não for atendida a restrição mínima, então cidades serão adicionadas no final do vetor até que a restrição seja atendida. Neste caso, o carro atribuído a cada nova cidade adicionada é aquele atribuído à última cidade (GOLDBARG et al., 2016).

Para terminar a iteração, ainda são executadas as busca locais novamente para melhorar os indivíduos atuais (linha 13). É realizado um torneio binário para selecionar quais indivíduos

Figura 8 – Funcionamento do Plasmídeo



Fonte: Autor (2019).

da população atual e da população gerada a partir dos operadores de *crossover* e plasmídeo formarão a população que será utilizada na próxima iteração. O torneio binário consiste em selecionar indivíduos destas populações para que entrem em competição para terem o direito de fazer parte da nova população. O fato de ser um torneio binário é que serão selecionados sempre dois indivíduos por vez para competirem (LINDEN, 2008). Por último, é realizada a atualização de população de elite (linha 16).

## 2.5 PATH RELINKING

*Path Relinking* também faz parte das metaheurísticas baseadas em população, mais especificamente das buscas dispersas. O primeiro passo é gerar uma população inicial que satisfaça um critério de diversidade e qualidade. O grupo de referência é criado selecionando uma pequena parte dos melhores indivíduos. As soluções são combinadas para criar soluções iniciais para um procedimento de melhoria. A partir do resultado gerado, a população e o grupo de referência podem ser atualizados com indivíduos com melhor qualidade e diversidade. O processo apenas para quando o critério de parada é alcançado (TALBI, 2009). O template de uma busca dispersa pode ser visto na figura 9. O algoritmo se baseia em cinco métodos: *geracaoDiversificada*, *melhoria*, *atualizaçãoGrupoReferência*, *geracaoSubgrupo* e *recombinacao*.

O método *geracaoDiversificada* gera um grupo de soluções iniciais diversas. Para isto, algoritmos gulosos são utilizados para aplicar diversidade na busca enquanto seleciona as me-

Figura 9 – *Template* de busca dispersa

```

1: geracaoDiversificada()
2: melhoria()
3: atualizacaoGrupoReferencia()
4: while Critério de parada 1 não for satisfeito do
5:   geracaoSubgrupo()
6:   while Critério de parada 2 não for satisfeito do
7:     recombinacao()
8:     melhoria()
9:   end while
10:  atualizarGrupoReferencia()
11: end while

```

Fonte: adaptado de Talbi (2009).

lhores soluções. O método *melhoria* transforma uma solução inicial em uma ou mais soluções candidatas, geralmente é utilizando uma busca local. O terceiro método, o *atualizacaoGrupoReferencia* é para criar e manter o grupo de referência, garantindo a diversidade enquanto guarda soluções de melhor qualidade. O quarto método, chamado de *geracaoSubgrupo* opera sobre o grupo de referência, para produzir um subconjunto de soluções como base para criar soluções combinadas. O último método é a *recombinacao*, onde ocorre uma recombinação com o subgrupo gerado a partir do método *geracaoSubgrupo* (TALBI, 2009).

O *Path Relinking* permite explorar caminhos que conectam soluções elite encontradas pela pesquisa de dispersão. A ideia principal é gerar e explorar o caminho no espaço de busca conectando uma solução inicial e uma solução objetivo. A solução objetivo é uma solução com uma aptidão boa e que vai ser escolhida para a execução do *Path Relinking*. Segundo Talbi (2009), a ideia é realizar uma reinterpretação de combinações lineares dos pontos no espaço euclidiano como caminhos entre e além das soluções em um espaço de vizinhança. O caminho entre duas soluções no espaço de busca geralmente vai criar soluções que compartilham atributos com as soluções de entrada. A sequência de soluções vizinhas serão geradas a partir da solução inicial até alcançar a solução objetivo. A melhor solução gerada é retornada. O template do *Path Relinking* está na figura 10. Então, utilizar o *Path Relinking* serve para gerar soluções intermediárias entre a solução inicial e a solução objetivo, pois ao utilizar características da solução inicial, pode ocorrer de uma solução gerada ter melhor aptidão que a inicial e a objetivo.

O algoritmo leva em consideração dois pontos importantes, a seleção do caminho e as operações intermediárias. A seleção do caminho considera os fatores importantes para a geração do caminho. A qualidade do processo de escolha do caminho pode interferir no nível de intensificação e diversificação das soluções geradas. Escolher as melhores ou as piores solu-



Figura 10 – *Template* do Path Relinking

```

1: while distancia( s, t )  $\neq$  0 do
2:   /* Busca o melhor movimento que diminui a distância */
3:    $m = \text{encontrarMelhorMovimento}()$ 
4:    $\text{aplicarMovimento}(s, m)$ 
5: end while

```

Fonte: adaptado de Talbi (2009).

ções terá um impacto diferente na busca. A seleção de caminho também pode ser guiada pelo conjunto que representa o conjunto de componentes diferentes na solução inicial  $s$  e na solução alvo  $t$ . A seleção do caminho pode levar em consideração se parte da melhor solução e vai para uma pior, chamada de *Backward* ou se parte da pior e vai até a melhor, chamada *Forward*. As operações intermediárias são as operações que geram as soluções vizinhas que formam o caminho entre a solução inicial e a objetivo.

## 2.6 TRABALHOS RELACIONADOS

Além do trabalho que desenvolveu o algoritmo MemPlas, outros três trabalhos foram feitos tratando do algoritmo QCaRS. Todos os trabalhos apresentam uma metaheurística para resolver o problema, com a intenção de melhorar o estado da arte.

Em Goldberg, Goldberg e Menezes (2013) é proposto um algoritmo memético que utiliza uma procedimento de seis buscas locais. Um modelo matemático foi apresentado e submetido a resolução através de um *solver* com limitações de tempo e memória. Uma análise experimental foi realizada para investigar o potencial do algoritmo memético proposto. Os resultados coletados pelo algoritmo proposto foram comparados com os resultados do *solver*. Os autores do trabalho concluem que de uma forma geral, o desempenho em tempo computacional do algoritmo memético é significativamente superior ao método exato no conjunto de instâncias examinadas.

No trabalho de Menezes, Goldberg e Goldberg (2014), também é proposto um algoritmo memético, mas com uma pequena diferença. No algoritmo anterior, ao realizar o *crossover*, se os filhos tivessem melhor aptidão do que os pais, os filhos substituiriam os pais diretamente. Neste algoritmo, os filhos são inseridos na população e ao final dos cruzamentos, é feita uma seleção dos melhores indivíduos de toda população. Os resultados coletados pelo algoritmo proposto foram comparados com os resultados do *solver* que resolveu o modelo matemático. Os autores do trabalho concluem que o algoritmo memético também encontrou soluções de qualidade melhores que as produzidas pelo *solver*.

No artigo de Menezes et al. (2017) são apresentados três novos algoritmos para o problema QCaRS. Todos os algoritmos se baseiam em algoritmos GRASP – Greedy Randomized Adaptive Search Procedure. O GRASP consiste em criar uma solução inicial e depois efetuar uma busca local para melhorar a qualidade da solução. Os três algoritmos são: GRASP com *Variable Neighborhood Search*, GRASP com *Variable Neighborhood Search* e *Path Relinking* e GRASP 2-potencial com *Variable Neighborhood Search* e *Path Relinking*. Este último apresenta uma variação do processo de geração dos indivíduos iniciais, para evitar criar indivíduos inicialmente bons, mas que podem ser ruins mais tarde. Neste trabalho também houve uma coleta de resultados utilizando um *solver*. Os autores concluíram que utilizar abordagens GRASP demandaram pouco tempo de processamento comparado aos trabalhos com os algoritmos meméticos. O componente *Path Relinking* foi eficiente na melhoria da solução ótima encontrada pelo algoritmo, a um custo de tempo de processamento relativamente baixo. A nova abordagem proposta do 2-potencial não obteve resultados promissores. Os algoritmos analisados tiveram uma boa adaptação para as instâncias menores. Ao aumentar o tamanho da instância, os algoritmos perderam eficiência na busca pelo valor ótimo, e tendem a demandar maior tempo de processamento.

### 3 COMPONENTE *PATH RELINKING* PROPOSTO

O *Path Relinking* e algoritmos evolucionários se baseiam em metaheurísticas de população. Assim ambos podem ser utilizado em conjunto, ou um substituir o outro. Assim, este trabalho propõe um novo componente baseado em *Path Relinking*. Este novo componente fará parte do algoritmo MemPlas. Ele será utilizado de duas formas. A primeira é substituir os componentes de *crossover* e plasmídeo. A segunda é utilizar o novo componente após a execução das buscas locais, em conjunto com o *crossover* e plasmídeo. Na figura 11 está o algoritmo MemPlas onde o componente *Path Relinking* substitui o componente plasmídeo e o *crossover*. Na figura 12 está o algoritmo MemPlas que teve a adição do *Path Relinking*. Importante dizer apenas houve a troca entre os componentes já citados, outros componentes continuaram iguais.

Figura 11 – Algoritmo com componente baseado em *Path Relinking*

```

1: Entradas : tamanhoPopulacao, pElite, numeroIteracoes, selectionStr, intermediaryStr
2: interacoes = 0
3:  $P_0 = \text{gerarPopulacao}(\text{tamanhoPopulacao})$ 
4:  $\text{realizarBuscaLocais}(P_0)$ 
5:  $\text{elite} = \text{selecionarMelhores}(pElite)$ 
6: while interacoes < numeroIteracoes do
7:    $\text{sel} = \text{selecione}(P_i)$ 
8:    $\text{off} = \text{pathRelinking}(\text{elite}, \text{selectionStr}, \text{intermediaryStr})$ 
9:    $\text{realizarBuscaLocais}(P_i)$ 
10:   $\text{pop} = \text{torneioBinario}(\text{pop}, \text{off})$ 
11:   $\text{interacoes} = \text{interacoes} + 1$ 
12:   $\text{elite} = \text{atualiza}(pElite)$ 
13: end while

```

Fonte: Autor (2019).

O *Path Relinking* implementado neste trabalho se baseou no algoritmo proposto por Menezes et al. (2017). O *Path Relinking* utiliza um conjunto de soluções de elite. Assim, duas soluções são escolhidas para serem utilizadas pelo *Path Relinking*, a melhor e a segunda melhor. Durante o processo, a melhor solução gerada retorna e será adicionada no conjunto de elite, se esta for melhor que a pior solução do conjunto. O conjunto então deve ser reordenado do melhor para o pior para que o processo continue. A partir daqui o *Path Relinking* pega a melhor solução e a terceira melhor. Depois a melhor e a quarta melhor, até acabarem todas as soluções no conjunto de elite. Este *Path Relinking* utiliza uma estratégia de combinar a melhor solução com a n-ésima melhor solução.

O *Path Relinking* proposto por Menezes et al. (2017) considera que a melhor solução é a solução inicial e a n-ésima melhor solução é a solução final. Assim, pedaços da solução inicial

Figura 12 – Algoritmo com plasmídeo e *Path Relinking*

```

1: Entradas : tamanhoPopulacao, tamanhoPlasmideo, pCross, pElite
2:             numeroIteracoes, selectionStr, intermediaryStr
3: interacoes = 0
4:  $P_0 = \text{gerarPopulacao}(\text{tamanhoPopulacao})$ 
5: realizarBuscaLocais( $P_0$ )
6: elite = selecionarMelhores(pElite)
7: while interacoes < numeroIteracoes do
8:     sel = selecione( $P_i$ )
9:     if interacoes / 10  $\neq$  0 then
10:         off = crossver(sel, pCross)
11:     else
12:         off = plasmideo(sel, tamanhoPlasmideo, elite)
13:     end if
14:     off = pathRelinking(offspring, selectionStr, intermediaryStr)
15:     realizarBuscaLocais( $P_i$ )
16:     pop = torneioBinario(pop, off)
17:     interacoes = interacoes + 1
18:     elite = atualiza(pElite)
19: end while

```

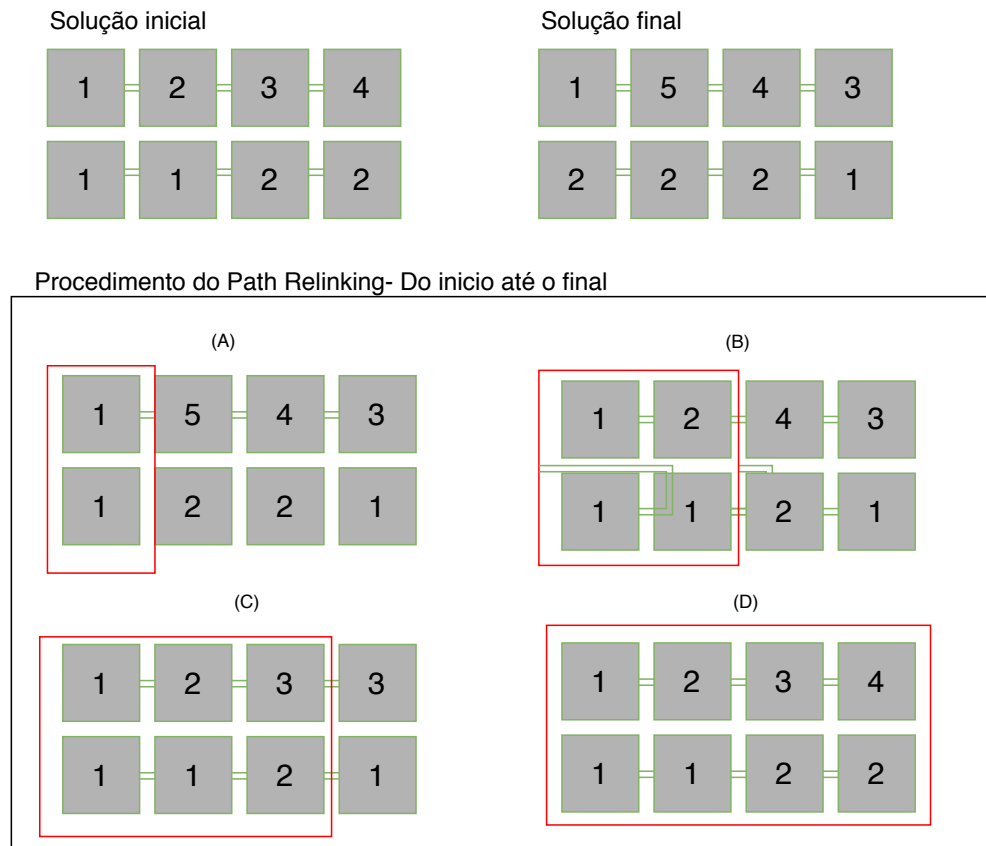
Fonte: Autor (2019).

serão interativamente inseridos na solução final, partindo do início da solução até chegar ao final dela. Um exemplo do processo pode ser visto na figura 13. Em (A), o par cidade/carro na posição 1 é copiado e inserido na posição 1 da solução final. Em (B), é feita a copia na segunda posição da solução inicial e é inserida na segunda posição da solução final. O processo continua até não haver mais índices na solução inicial ainda não copiados.

Este trabalho, além de implementar o *Path Relinking* já apresentado, implementará duas variantes. A primeira variante vai realizar a inserção de pedaços do final para o início dos indivíduos. Essa estratégia foi pensada para ver se inverter a ordem de inserção, seriam gerados indivíduos diferentes e possivelmente melhores que a estratégia anterior. Na figura 14 está a primeira variante. Em (A), o par cidade/carro foi copiado da última posição da solução inicial e é inserido na última posição da solução final. Em (B), é feita a cópia da penúltima posição da solução inicial e inserida na penúltima posição da solução final. O processo continua até não haver mais índices na solução inicial ainda não copiados.

A segunda variante vai inserir pedaços de forma aleatória da solução inicial na solução final, realizando um sorteio no intervalo de valores do tamanho do indivíduo. Essa estratégia foi escolhida para tentar remediar o problema de soluções inválidas que ambas as estratégias anteriores produzem (isto será detalhado a seguir). Na figura 15 está a segunda variante. Em (A), é sorteado uma posição, no caso a terceira. É feita a copia desta posição na solução inicial e

Figura 13 – Procedimento do Path Relinking - Do início para o final



Fonte: Autor (2019).

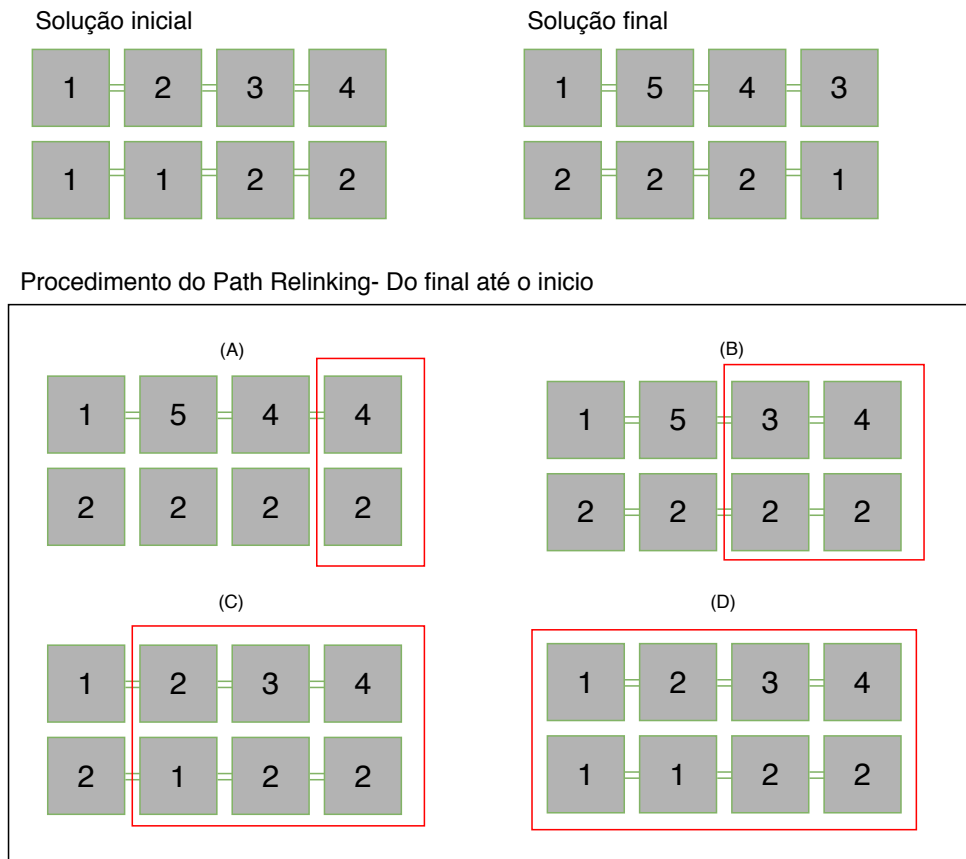
inserida na mesma posição na solução final. Em (B) é feito um novo sorteio, a posição 2 é escolhida. É feita a cópia desta posição na solução inicial e é inserida na solução final. O processo também continua até não houver mais índices na solução inicial ainda não copiados.

Será testado também se inverter a ordem das soluções escolhidas para o *Path Relinking* trará resultados melhores ao algoritmo. Ao realizar a inversão, a melhor solução será a solução final e a solução inicial será a n-ésima melhor. Na tabela 1 estão descritas todas as variantes do componente baseado em *Path Relinking* proposto neste trabalho.

Tabela 1 – Variações do Path Relinking

Nome	Ordem dos indivíduos	Estratégia Intermediária
steb	Melhor para Pior	Realiza a inserção do início para o final dos indivíduos.
stef	Pior para Melhor	Realiza a inserção do início para o final dos indivíduos.
etsb	Melhor para Pior	Realiza a inserção do final para o início dos indivíduos.
etsf	Pior para Melhor	Realiza a inserção do final para o início dos indivíduos.
rb	Melhor para Pior	Inserção aleatória.
rf	Pior para Melhor	Inserção aleatória

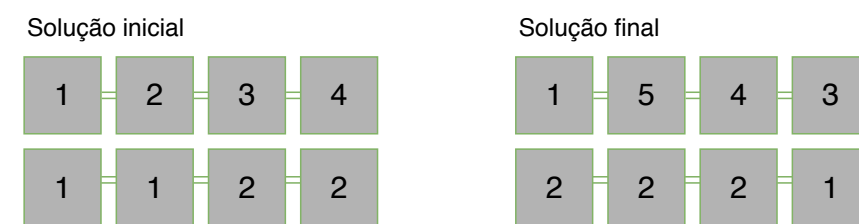
Figura 14 – Procedimento do Path Relinking - Do final para o início



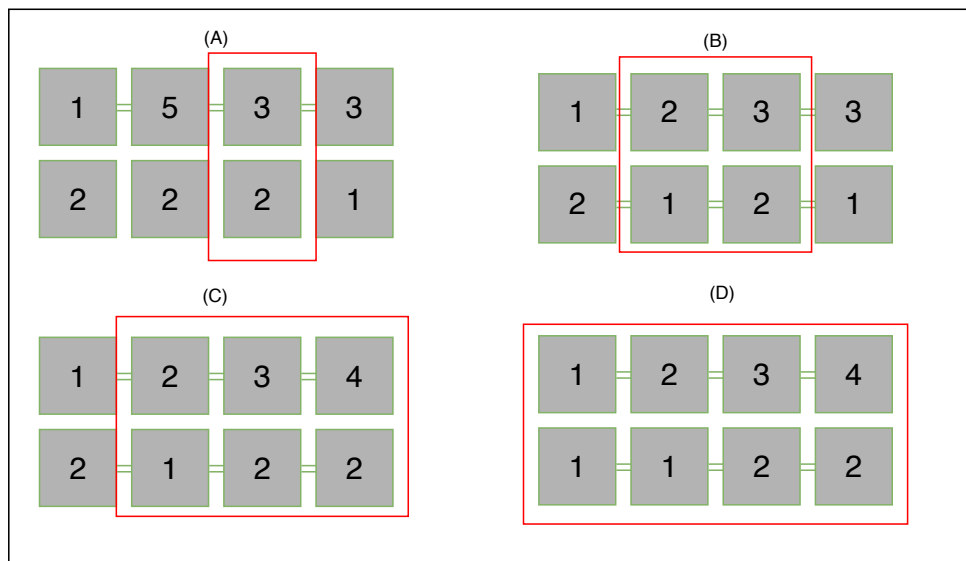
Fonte: Autor (2019).

Outro ponto que deve ser apresentado sobre o novo componente é que durante a inserção dos pedaços de gene da solução inicial na solução final, são gerados indivíduos inválidos. Os indivíduos inválidos são aqueles que, por exemplo, possuem cidades ou carros repetidos em seu trajeto. Este trabalho não criou uma estratégia para contornar este problema. Quando um indivíduo inválido é gerado, ele é ignorado. Assim sua aptidão não é calculado e não é considerado para o cálculo da melhor solução criada. Esta estratégia levou ao problema que muitos indivíduos gerados são inválidos e assim o novo componente acaba por retornar os indivíduos inicial ou final em muitos casos.

Figura 15 – Procedimento do Path Relinking - Aleatório



Procedimento do Path Relinking- Inserções aleatórias



Fonte: Autor (2019);

## 4 AVALIAÇÃO EXPERIMENTAL

Este capítulo descreve as etapas realizadas neste trabalho para avaliar o componente *Path Relinking* proposto. Na primeira seção, será apresentada uma descrição da estruturas e técnicas utilizadas para desenvolver os algoritmos deste trabalho. Na segunda seção serão explicadas algumas ferramentas e métodos utilizados para a realização dos experimentos e análise dos resultados. A terceira seção vai apresentar os resultados obtidos a partir dos experimentos.

### 4.1 IMPLEMENTAÇÃO DOS ALGORITMOS

Para avaliação da técnica proposta, os algoritmos foram desenvolvidos na linguagem C++. Os códigos fonte desenvolvidos neste trabalho estão disponíveis no GitHub.<sup>1</sup>

O algoritmo MemPlas foi implementado com base nos pseudocódigos apresentados por Goldberg et al. (2016). A implementação foi realizada “do zero”, por dois motivos. O primeiro motivo foi não ter acesso ao código original. O segundo é para uma comparação mais justa entre algoritmos. Todos os componentes originais do algoritmo MemPlas foram reescritos para poderem ser utilizados também no algoritmo com *Path Relinking* proposto neste trabalho. Assim, a comparação torna-se mais justa na questão de qualidade de solução e tempo de execução, pois o estilo e estruturas de programação usados para implementar os componentes afeta igualmente todas as variações do algoritmo.

O projeto foi estruturado em funções, onde cada função é um dos componentes do algoritmo. Cada componente pode utilizar outras funções. Assim, o código ficou organizado em componentes que podem ser chamados a qualquer momento durante o algoritmo. Esta separação facilitou ainda a remoção e/ou substituição para a realização dos experimentos.

Para garantir a correta implementação dos componentes, testes unitários foram implementados ao longo do processo de desenvolvimento dos algoritmos e componentes. Os testes unitários consistem em pedaços de código para verificar se um componente individual de software foi implementado corretamente. Esse componente pode ser um método ou procedimento, uma classe completa ou, ainda, um pacote de funções ou classes de tamanho pequeno a moderado (WAZLAWICK, 2013). Além disso, Pfleeger (2004) define como um teste que verifica se o componente funciona de forma adequada aos tipos de entradas esperadas. O teste de unidade é feito em um ambiente controlado, de modo que a equipe de teste possa fornecer ao componente a ser testado um conjunto de dados predeterminado, e observar quais as ações e dados são produzidos.

---

<sup>1</sup><<https://github.com/TiagoFunkUdescCeavi/MemPlas-And-Path-Relinking>>



Realizar os testes unitários garantiram os que componentes dos algoritmos implementados estavam funcionando conforme esperado. Isto evitou possíveis problemas e defeitos que poderiam passar despercebidos na implementação e que poderiam comprometer as etapas de calibração de parâmetros e experimentos. Testes unitários ajudam a garantir que os componentes estejam funcionando corretamente, para que possam ser reutilizados no futuro em outros algoritmos e trabalhos. A biblioteca que foi utilizada para a automação dos testes foi a Google Testing and Mocking Framework (GOOGLE INC., 2019).

## 4.2 MATERIAIS E MÉTODOS DOS EXPERIMENTOS

Esta seção descreve os materiais (instâncias do problema QCaRS) adotados nos experimentos, e também os métodos adotados para calibrar os algoritmos e coletar resultados.

### 4.2.1 Instâncias QCaRS

As instâncias utilizadas na avaliação experimental são as mesmas utilizadas por Goldberg et al. (2016) e estão disponíveis online.<sup>2</sup> Ao todo foram utilizadas 44 instâncias de problemas QCaRS. Nestas instâncias a quantidade de cidades varia entre 9 e 30, e a quantidade de veículos varia entre 2 a 5. As instâncias possuem as seguintes características: todos os carros podem ser alugados em todas as cidades; cada carro pode ser alugado apenas uma vez; e o custo de viajar para a cidade  $i$  até a cidade  $j$  é igual a viajar de  $j$  até  $i$ .

As instâncias utilizadas são divididas em dois grupos: instâncias euclidianas e instâncias não euclidianas. As instâncias representam coordenadas de cidades no mapa. Nas instâncias euclidianas, a distância entre cidades é calculada pela fórmula da distância euclidiana entre dois pontos. A distância entre cidades nas instâncias não euclidianas é calculada de forma diferente, pois nestas instâncias considera-se a curvatura do planeta e são utilizadas coordenadas gaussianas (WOLFE, 2012).

### 4.2.2 Calibração dos Algoritmos

O algoritmo MemPlas possui diversos parâmetros de entrada, tal como o *tamanhoPopulacao* e *numeroIteracoes*. Estes parâmetros precisam ser calibrados para se encontrar a combinação de valores que maximizar o desempenho do algoritmo nas instâncias consideradas. A ferramenta que foi utilizada para calibração é o *irace*. Esta ferramenta automatiza a configura-

<sup>2</sup><http://www.dimap.ufrn.br/lae/en/projects/CaRS.php>

ção dos parâmetros e define a melhor configuração possível baseado em execuções do algoritmo (LÓPEZ-IBÁÑEZ et al., 2011).

O *irace* recebe todos os parâmetros e uma especificação do tipo e intervalo de valores possíveis. Ao terminar o processo, fornece combinações de parâmetros que obtém os melhores resultados. Neste trabalho, foram realizados seis calibrações de parâmetros: algoritmo MemPlas original, que usa o componente plasmídeo; algoritmo MemPlas que utiliza o componente de *Path Relinking* proposto no lugar do componente plasmídeo; o MemPlas que utiliza os componentes plasmídeo e *Path Relinking* combinados; algoritmo MemPlas original sem buscas locais; algoritmo MemPlas utilizando apenas o componente *Path Relinking* sem buscas locais; e algoritmo MemPlas com ambos componentes sem buscas locais.

Na tabela 2 estão todos os oito parâmetros que precisaram ser calibrados. Na coluna *parâmetro* está o nome do parâmetro. A coluna *Tipo* traz o tipo do parâmetro, por exemplo, categoria é o tipo do dado que aceita apenas um conjunto finito de valores. A coluna *Intervalo* indica os possíveis valores que o *irace* deve levar em conta para realizar a calibração. Na coluna *Requisitos*, indica o requisito para aquele parâmetro ser utilizado, por exemplo, o parâmetro *SelectionStrategy* pode apenas ser utilizado quando o *Path Relinking* também está sendo utilizado.

Tabela 2 – Parâmetros utilizados para a calibração do *irace*

Parâmetro	Tipo	Intervalo	Requisitos
Strategy	Categoria	m, pr ou mpr	-
SizePopulation	Inteiro	100 a 300	-
Elite	Decimal	0.1 a 0.9	-
LimitIterations	Inteiro	100 a 2000	-
SizePlasmideo	Decimal	0.3 a 0.7	Strategy igual a m ou mpr
Cross	Decimal	0.1 a 0.9	Strategy igual a m ou mpr
SelectionStrategy	Categoria	a	Strategy igual a pr ou mpr
IntermediaryStrategy	Categoria	stef, steb, etsf, etsb, rf ou rb	Strategy igual a pr ou mpr

O parâmetro *Strategy* indica se o algoritmo vai utilizar o plasmídeo (m), *Path Relinking* (pr) ou ambos em sua execução (mpr). O *SizePopulation* indica o número de indivíduos que compõe a população. *Elite* indica a porcentagem de indivíduos da população que compõem a população de elite. *LimitIterations* indicam o número de vezes que a população deve receber a aplicação do operador MemPlas/*Path Relinking* e após gerar uma nova população. *SizePlasmideo* indica uma porcentagem do tamanho do indivíduo que é o tamanho do fragmento de gene utilizando no plasmídeo. O parâmetro *Cross* indica o tamanho da população que será utilizada para realização do *crossover*. *SelectionStrategy* indica qual estratégia está sendo utilizada na seleção da solução inicial e final no *Path Relinking*. Neste trabalho apenas foi desenvolvida

uma variação, onde o melhor indivíduo de uma população de elite é combinado com o n-ésimo melhor indivíduo. O último parâmetro indica qual estratégia intermediária do *Path Relinking* é utilizada. As estratégias são as mesmas da tabela 1. Para realizar esta calibração, foram utilizadas todas as instâncias disponíveis. Os resultados que o *irace* obteve estão na tabela 3.

Tabela 3 – Parâmetros escolhidos pelo *irace*

Parâmetro	<i>MemPlas</i>	<i>Path Relinking</i>	<i>MemPlas+ Path Rel.</i>	<i>Somente Buscas Locais</i>	<i>MemPlassem Buscas locais</i>	<i>Path Rel. sem Buscas locais</i>	<i>Mem. + Path R. sem Buscas locais</i>
SizePopulation	190	215	110	190	200	195	200
Elite	0.35	0.55	0.4	0.35	0.5	0.4	0.5
LimitIterations	1500	2000	3000	1500	1500	1900	2450
SizePlasmideo	0.50	-	0.5	-	0.5	-	0.5
Cross	0.50	-	0.85	-	0.55	-	0.5
SelectionStrategy	-	a	a	-	-	a	a
IntermediaryStrategy	-	stef	stef	-	-	etsf	rf

Para o algoritmo *MemPlas* original, o *irace* chegou aos parâmetros: população de 190 indivíduos, 35% dos indivíduos serão a população de elite, serão realizadas 1500 iterações sobre a população, o plasmídeo será metade do tamanho do indivíduo, metade da população será utilizada no cruzamento. Para o algoritmo baseado em *Path Relinking*: população de 215 indivíduos, 55% da população fará parte da elite, serão 2000 iterações na população e o componente intermediário escolhido foi o *stef* (inserção do início para o final com a ordem dos indivíduos do melhor para o pior). O algoritmo que une o componente *MemPlas* com *Path Relinking* teve os seguintes resultados: população de 110 indivíduos, 40% da população formará a população de elite, 3000 iterações, o tamanho do plasmídeo deverá ser do tamanho de 50% do tamanho do indivíduo, 85% da população será utilizada no cruzamento, a estratégia intermediária será o *stef* (inserção do início para o final com a ordem dos indivíduos do melhor para o pior). Os parâmetros para o algoritmo que apenas utiliza as buscas locais foi escolhido de forma arbitrária, apenas utilizando os parâmetros apontados como os melhores para o algoritmo *MemPlas*. Estes parâmetros foram escolhidos de forma arbitrária, pois apenas precisava-se de dados para comparação com os algoritmos.

Para o algoritmo *MemPlas* sem as buscas locais, o *irace* chegou ao resultado de 200 indivíduos, 50% da população serão a população de elite, 1500 iterações na população, o plasmídeo será do tamanho da metade do indivíduo, e 55% dos indivíduos será utilizada no cruzamento. O algoritmo *MemPlas* com *Path Relinking* sem as buscas locais chegou ao resultado de 195 indivíduos na população, 40% para população de elite, 1900 iterações na população e a estratégia intermediária será o *etsf* (inserção do final para o início com a ordem dos indivíduos do melhor para o pior). O algoritmo *MemPlas* com ambos indivíduos sem buscas locais teve resultado de 200 indivíduos, 50% da população para elite, 2450 iterações, metade do indivíduo será o tama-

nho do indivíduo, metade da população será utilizada no cruzamento e a estratégia intermediária será o rf (inserção do aleatória com a ordem dos indivíduos do melhor para o pior).

#### 4.2.3 Execução dos Algoritmos e Coleta de Dados

A partir dos valores de parâmetros encontrados pelo irace, foram coletadas a qualidade da solução e o tempo de duração da execução de cada algoritmo. Cada variação do algoritmo foi executada trinta vezes em todas as instâncias disponíveis. O computador onde os experimentos foram executados é um Intel Core 7-7700 CPU de 3.60 GHz, com memória RAM de 8 GB. Sistema operacional Linux Ubuntu 16.04 64 bits LTS. A versão do C++ utilizada foi a C++11 e o compilador foi g++ versão 5.4.0.

Para realizar a análise de dados, foi utilizada a biblioteca pandas (NUMFOCUS, 2019). O pandas é uma biblioteca de código aberto, que fornece estruturas de dados de alto desempenho e fáceis de usar e ferramentas de análise de dados para a linguagem de programação Python.

A biblioteca pandas fornece várias funções para leitura e análise de dados. Neste trabalho, ela foi utilizada para: leitura de dados a partir de um ou vários arquivos *csv*; calcular dados para a geração de tabelas; e gerar gráficos.

### 4.3 RESULTADOS E DISCUSSÃO

Dois experimentos foram realizados neste trabalho. O primeiro tinha a intenção de verificar qual componente apresentava o melhor resultado. Com os resultados coletados, foram levantadas hipóteses e foi realizado um segundo experimento para testar a hipótese em outro cenário.

#### 4.3.1 Experimento 1

Para a apresentação dos resultados foram criadas duas tabelas com dados das execuções das três variações dos algoritmos: *Crossover/Plasmídeo* (original de Goldberg et al. (2016)), *Path Relinking*, e *Crossover/Plasmídeo + Path Relinking*. Para avaliar o impacto dos componentes *Plasmídeo* e *Path Relinking* nos algoritmos, um outro experimento foi realizado desativando estes componentes e deixando apenas o componente *buscas locais*, descrito anteriormente na fundamentação teórica. Os resultados são apresentados nas tabelas 4 e 5. A coluna *instância* indica o nome da instância (problema) executada. A coluna *melhor* apresenta a qualidade (custo) da melhor solução encontrada no experimento realizado em (GOLDBARG et al., 2016). As demais colunas apresentam os resultados obtidos com cada variação dos algoritmos.

Em cada algoritmo, a coluna *média* indica a qualidade média da melhor solução encontrada (média da melhor solução encontrada em cada uma das trinta execuções), e a coluna *melhor* indica a qualidade da melhor solução encontrada dentre as trinta execuções. A ordem que as cidades aparecem nas tabelas seguem uma ordem aproximada de menor para maior no quesito complexidade.

Com relação aos resultados para as instâncias euclidianas (tabela 4), pode-se observar que em nenhuma instância os algoritmos chegaram no melhor resultado. As médias e os melhores ficaram relativamente longe da melhor solução. Outro ponto importante é que não houve muita diferença entre a qualidade dos resultados apresentado pelas quatro variações do algoritmo. As mesmas características podem ser observadas para as instâncias não euclidianas da tabela 5. Não houve casos em que algoritmo chegou no melhor resultado, e os valores de mínimo e média também não apresentaram grandes mudanças entre as variações de algoritmos.

O teste ANOVA foi realizado com os resultados para verificar se existia diferença entre os quatro grupos de dados. Todas as execuções de cada algoritmo foram utilizadas, apenas separando por euclidianas e não euclidianas. A intenção era verificar se um algoritmo tinha um desempenho geral melhor do que outro. Nas figuras 16 e 17 estão os *boxplots* que apresentam

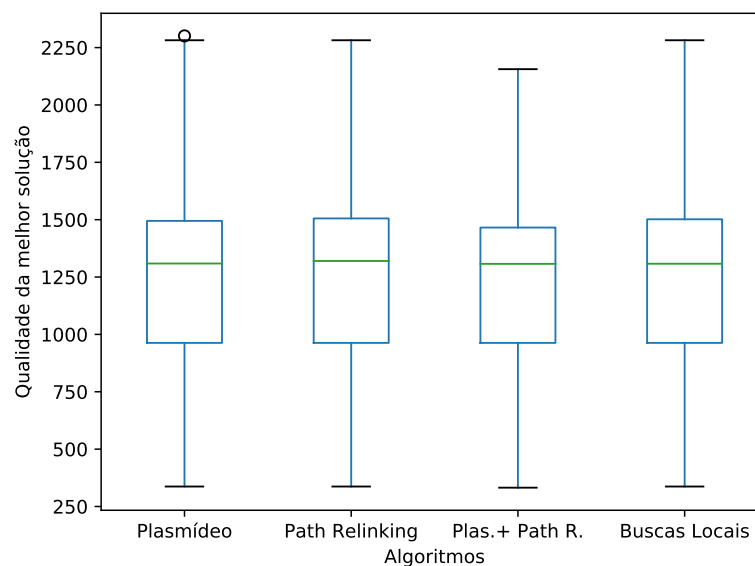
Tabela 4 – Resultados nas instâncias euclidianas no experimento 1

Problemas instância	melhor	MemPlas Plasmídeo		MemPlas Plasm. + Path Rel.		Somente Buscas Locais		Memplas Path Relinking	
		média	melhor	média	melhor	média	melhor	média	melhor
Bolivia10e	384	1302.0	1145	1258.6	1102	1289.2	1145	1299.9	1145
AfricaSul11e	402	1292.1	1107	1277.1	982	1301.9	1224	1302.2	1224
Niger12e	564	1441.9	1298	1408.5	1250	1420.7	1305	1440.3	1306
Mongolia13e	543	1372.5	1133	1395.5	1124	1404.6	1155	1407.1	1173
Indonesi14e	504	1301.6	1161	1257.3	1158	1261.8	1150	1294.4	1195
Argelia15e	487	1307.0	1175	1305.0	1110	1343.8	1168	1316.5	1175
India16e	705	1772.1	1749	1755.9	1608	1762.7	1749	1760.8	1749
China17e	728	1398.8	1291	1348.5	1291	1391.1	1309	1402.1	1303
Etiopia10e	283	893.9	779	841.8	777	939.3	779	894.4	807
Mali11e	428	1361.8	1240	1343.1	1179	1364.2	1204	1359.2	1285
Chade12e	655	1471.3	1345	1459.0	1312	1484.5	1351	1499.1	1345
Ira13e	532	1614.2	1456	1586.7	1477	1618.6	1445	1646.9	1489
Mexico14e	492	1321.0	1128	1235.8	1063	1279.5	1120	1325.7	1120
Sudao15e	422	1398.7	1355	1357.7	1164	1400.8	1331	1375.7	1273
Australia16e	682	1761.5	1576	1695.9	1509	1764.5	1586	1755.6	1625
Canada17e	783	2079.1	1920	1967.3	1864	2057.9	1864	2068.0	1927
Arabia14e	482	1601.6	1431	1563.4	1371	1609.0	1431	1615.5	1376
Cazaquistao15e	574	1469.8	1348	1431.4	1338	1474.4	1347	1455.2	1326
Brasil16e	619	1875.2	1782	1828.7	1626	1889.2	1665	1916.8	1782
Russia17e	750	1870.0	1616	1858.7	1551	1895.6	1748	1904.9	1865
BrasilAM26e	338	666.9	627	653.8	615	663.5	627	666.8	632
BrasilMG30e	368	714.7	672	703.0	635	713.6	671	718.2	694

Tabela 5 – Resultados nas instâncias não euclidianas no experimento 1

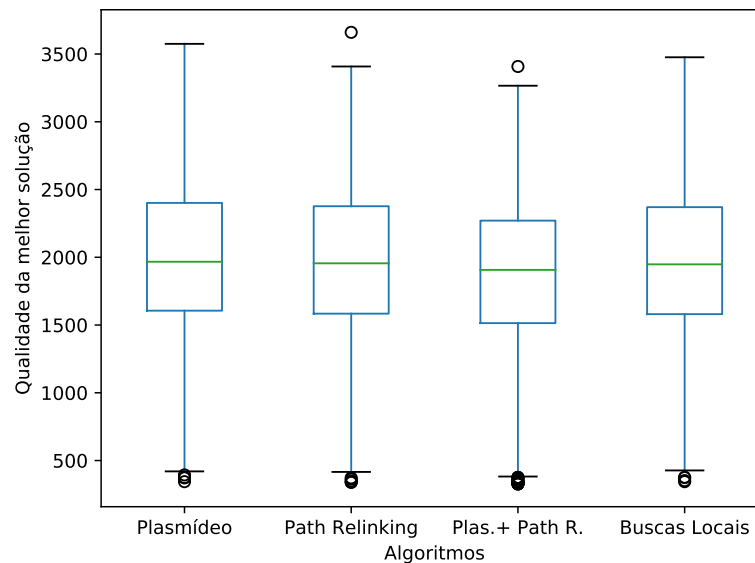
Problemas instância	melhor	MemPlas Plasmídeo		MemPlas Plasm. + Path Rel.		Somente Buscas Locais		MemPlas Path Relinking	
		média	melhor	média	melhor	média	melhor	média	melhor
Bolivia10n	448	1302.4	1247	1302.9	1247	1316.3	1247	1321.0	1247
AfricaSul11n	537	2030.9	1548	1961.0	1467	1952.5	1659	1976.8	1682
Niger12n	607	2243.8	1632	1945.2	1685	2166.7	1595	2203.8	1701
Mongolia13n	551	1866.0	1644	1792.0	1538	1873.1	1757	1890.0	1623
Indonesi14n	522	1621.6	1472	1546.0	1396	1591.7	1472	1608.1	1472
Argelia15n	616	2188.3	2016	2133.6	2016	2211.7	2016	2174.6	2016
India16n	723	2422.5	2171	2354.2	2114	2409.5	2161	2427.5	2182
China17n	638	1705.7	1624	1678.6	1624	1715.5	1624	1745.5	1624
Etiopia10n	403	1693.0	1336	1572.8	1310	1656.3	1314	1698.0	1335
Mali11n	494	2318.8	1977	2189.5	1883	2304.8	2012	2318.7	2093
Chade12n	649	2269.0	1929	2120.4	1871	2270.0	1996	2203.0	1917
Ira13n	625	2094.3	1899	2049.8	1899	2095.1	1899	2106.1	1905
Mexico14n	610	2099.1	1890	2032.8	1814	2068.1	1794	2126.4	1814
Sudao15n	769	2476.6	2152	2380.9	2036	2455.3	2039	2488.4	2135
Australia16n	525	2092.2	1967	2016.6	1771	2031.1	1785	2070.3	1831
Canada17n	824	3256.8	3056	3214.4	3056	3232.5	3104	3245.8	3056
Arabia14n	688	2564.2	2355	2507.6	2319	2585.1	2370	2602.8	2355
Cazaquistao15n	830	2629.1	2292	2593.7	2222	2583.9	2317	2624.4	2390
Brasil16n	742	2617.2	2597	2603.9	2329	2676.7	2597	2647.1	2597
Russia17n	778	2929.5	2551	2745.1	2357	2967.2	2655	2949.1	2561
BrasilAM26n	107	603.9	344	478.1	326	566.8	344	577.6	339
BrasilMG30n	160	582.1	487	564.9	466	546.2	480	583.1	483

Figura 16 – Boxplot com os resultados em instâncias euclidianas no experimento 1



Fonte: Autor (2019).

Figura 17 – Boxplot com os resultados em instâncias não euclidianas no experimento 1



Fonte: Autor (2019).

os resultados obtidos pelos algoritmos nas instâncias euclidianas e não euclidianas, respectivamente. O valor apresentado pelo teste ANOVA nas instâncias euclidianas foi de 37,51%, como este valor é superior aos 5%, a hipótese nula, que os dados são equivalentes, não foi rejeitada. Logo não existe diferença em pelo menos um par de grupos. Para as instâncias não euclidianas, o valor apresentado pelo teste ANOVA foi de 13,53%, também não rejeitando a hipótese nula.

Além do teste ANOVA para cada grupo de instâncias, foi realizado o teste para cada instância. A tabela 6 apresenta os resultados dos testes. As instâncias que estão com o seu valor  $p$  em negrito são as que tiveram um resultado menor que 5%. Assim, existe pelo menos diferença em um par de algoritmos. Ao todo 12 instâncias euclidianas e 10 instâncias não euclidianas apresentaram diferença entre ao menos um par de algoritmos.

Com os resultados do teste do ANOVA em mãos, foi realizado um teste de Tukey com as instâncias. Os resultados são apresentados na tabela 7. Os algoritmos foram comparados par a par com as seguintes siglas:  $m$  para o algoritmo MemPlas original;  $mpr$  para o algoritmo MemPlas com adição do componente *Path Relinking*;  $pr$  para o algoritmo que utiliza apenas o *Path Relinking*; e  $ols$  para o algoritmo que utiliza somente as buscas locais. Para cada par de algoritmos existe duas colunas na tabela. A coluna *Dif. Média* indica a diferença média entre os grupos comparados. Por exemplo, considerando a comparação *grupo1 vs. grupo 2*, a diferença entre a média é dada por  $grupo2 - grupo1$ . A coluna *Valor  $p$*  indica o valor de  $p$  resultante do teste. Antes de prosseguir com a análise, é importante dizer que durante o processo dos teste estatístico, algumas instâncias acabavam por rejeitar a hipótese nula do teste ANOVA (igualdade entre todos os grupos). Mas ao realizar o teste de Tukey, nenhum par de

Tabela 6 – Resultados do teste do ANOVA por instância no experimento 1

Euclidianas		Não Euclidianas	
Instância	p (%)	Instância	p (%)
AfricaSul11e	16.58	AfricaSul11n	23.69
Arabia14e	11.54	Arabia14n	7.06
Argelia15e	34.13	Argelia15n	<b>2.23</b>
Australia16e	<b>3.30</b>	Australia16n	<b>2.49</b>
Bolivia10e	<b>4.71</b>	Bolivia10n	70.98
Brasil16e	<b>0.78</b>	Brasil16n	<b>2.31</b>
BrasilAM26e	<b>4.16</b>	BrasilAM26n	<b>0.13</b>
BrasilMG30e	8.64	BrasilMG30n	15.75
Canada17e	<b>0.19</b>	Canada17n	20.43
Cazaquistao15e	14.26	Cazaquistao15n	68.33
Chade12e	36.70	Chade12n	<b>0.17</b>
China17e	<b>0.28</b>	China17n	<b>2.71</b>
Etiopia10e	<b>0.02</b>	Etiopia10n	5.76
India16e	52.22	India16n	37.27
Indonesi14e	<b>1.02</b>	Indonesi14n	9.00
Ira13e	<b>3.92</b>	Ira13n	37.94
Mali11e	53.13	Mali11n	<b>0.18</b>
Mexico14e	<b>1.73</b>	Mexico14n	8.61
Mongolia13e	62.03	Mongolia13n	<b>0.00</b>
Niger12e	31.34	Niger12n	<b>0.00</b>
Russia17e	<b>3.09</b>	Russia17n	<b>0.03</b>
Sudao15e	<b>1.63</b>	Sudao15n	11.68

grupos apresentava diferenças. Ambos os testes eram feitos com nível de confiança de 95%. Este fenômeno ocorre pois o teste ANOVA é mais sensível para indicar a diferença entre os grupos do que o teste Tukey (HANCOCK; KLOCKARS, 1996). Para contornar esta questão, adotou-se que o valor de  $p$  para o teste tukey, deve ser maior que 10% para que a hipótese nula seja mantida. Na tabela, todos os valores de  $p$  em negrito são aqueles abaixo dos 10%.



Tabela 7 – Resultados do teste de Tukey no experimento 1

Instância	m vs. mpr		m vs. ols		m vs. pr		mpr vs. ols		mpr vs. pr		ols vs. pr	
	Dif. Media	Valor p (%)	Dif. Media	Valor p (%)	Dif. Media	Valor p (%)	Dif. Media	Valor p (%)	Dif. Media	Valor p (%)	Dif. Media	Valor p (%)
Australia16e	-65.57	<b>6.95</b>	3.00	90.00	-5.87	90.00	68.57	<b>5.25</b>	59.70	11.56	-8.87	90.00
Bolivia10e	-43.40	<b>6.01</b>	-12.83	86.48	-2.13	90.00	30.57	28.61	41.27	<b>8.11</b>	10.70	90.00
Brasil16e	-46.47	26.98	13.97	90.00	41.63	36.67	60.43	<b>8.94</b>	88.10	<b>0.43</b>	27.67	67.85
BrasilAM26e	-13.07	<b>6.20</b>	-3.40	90.00	-0.07	90.00	9.67	24.88	13.00	<b>6.39</b>	3.33	90.00
Canada17e	-111.80	<b>0.32</b>	-21.23	90.00	-11.13	90.00	90.57	<b>2.51</b>	100.67	<b>0.99</b>	10.10	90.00
China17e	-50.27	<b>0.98</b>	-7.70	90.00	3.33	90.00	42.57	<b>3.94</b>	53.60	<b>0.51</b>	11.03	89.25
Etiopia10e	-52.17	<b>7.61</b>	45.33	15.38	0.50	90.00	97.50	<b>0.10</b>	52.67	<b>7.20</b>	-44.83	16.14
Indonésia14e	-44.30	<b>3.29</b>	-39.80	<b>6.75</b>	-7.20	90.00	4.50	90.00	37.10	<b>9.99</b>	32.60	18.07
Ira13e	-27.50	53.67	4.37	90.00	32.73	38.85	31.87	41.30	60.23	<b>2.11</b>	28.37	51.31
México14e	-85.17	<b>4.00</b>	-41.47	55.10	4.70	90.00	43.70	51.16	89.87	<b>2.68</b>	46.17	46.60
Rússia17e	-11.30	90.00	25.57	46.15	34.90	19.27	36.87	15.41	46.20	<b>4.46</b>	9.33	90.00
Sudão15e	-41.00	<b>4.19</b>	2.10	90.00	-22.97	44.29	43.10	<b>2.91</b>	18.03	62.61	-25.07	36.34
Argélia15n	-54.63	14.37	23.43	76.83	-13.70	90.00	78.07	<b>1.39</b>	40.93	37.75	-37.13	46.53
Australia16n	-75.67	<b>3.35</b>	-61.10	12.17	-21.90	83.71	14.57	90.00	53.77	20.88	39.20	48.39
Brasil16n	-13.23	90.00	59.53	<b>9.23</b>	29.93	62.28	72.77	<b>2.45</b>	43.17	32.54	-29.60	63.02
BrasilAM26n	-125.77	<b>0.11</b>	-37.13	64.93	-26.30	83.42	88.63	<b>3.83</b>	99.47	<b>1.52</b>	10.83	90.00
Chade12n	-148.60	<b>0.45</b>	1.03	90.00	-65.97	42.79	149.63	<b>0.42</b>	82.63	23.00	-67.00	41.39
China17n	-27.07	59.44	9.77	90.00	39.83	27.14	36.83	33.98	66.90	<b>1.49</b>	30.07	51.81
Mali11n	-129.30	<b>0.55</b>	-14.07	90.00	-0.17	90.00	115.23	<b>1.69</b>	129.13	<b>0.56</b>	13.90	90.00
Mongólia13n	-74.00	<b>0.36</b>	7.07	90.00	23.97	65.00	81.07	<b>0.12</b>	97.97	<b>0.10</b>	16.90	83.66
Níger12n	-298.67	<b>0.10</b>	-77.17	60.34	-40.00	90.00	221.50	<b>0.37</b>	258.67	<b>0.10</b>	37.17	90.00
Rússia17n	-184.33	<b>0.64</b>	37.73	90.00	19.60	90.00	222.07	<b>0.10</b>	203.93	<b>0.20</b>	-18.13	90.00

Quando o valor de  $p$  é menor que 10%, assim rejeitando a hipótese nula, é necessário indicar qual o melhor grupo do par. Considerando que o valor da coluna *Dif. Média* é calculado por  $grupo2 - grupo1$  (diferença entre as médias), então se  $grupo2$  é melhor que o  $grupo1$ , a diferença média é negativa (em um problema de minimização, melhor quer dizer que produz resultados numéricos menores). Caso  $grupo1$  seja melhor que o  $grupo2$ , a diferença média será positiva.

O par MemPlas original e MemPlas com Plasmídeo e *Path Relinking* (m vs. mpr) teve diferença em 18 instâncias, todas com diferença média negativa. Assim o MemPlas com *Path Relinking* apresentou resultado melhor que o MemPlas original nestas instâncias. O par MemPlas original com o algoritmo somente com as buscas locais (m vs. ols) teve um resultado parecido entre ambos. Apenas em dois casos o teste de Tukey apresentou diferenças. Na comparação MemPlas original com MemPlas utilizando apenas o *Path Relinking* (m vs. pr) não houve nenhuma instância com diferença significativa.

O par MemPlas com Plasmídeo e *Path Relinking* e algoritmo somente com as buscas locais (mpr vs. ols) apresentou diferença significativa em 16 instâncias, todas com diferenças de médias positivas, assim o MemPlas com plasmídeo e *Path Relinking* apresentou um resultado melhor. O par MemPlas com plasmídeo e *Path Relinking* e MemPlas utilizando apenas o *Path Relinking* (mpr vs. pr) apresentou 18 instâncias com diferença. Todas com diferença média positiva. Assim o algoritmo MemPlas com *Path Relinking* obteve um resultado melhor. O par algoritmo somente com buscas locais e MemPlas apenas com *Path Relinking* (ols vs pr) não apresentou nenhuma instância com diferença.

De modo geral, quando o teste ANOVA apontou haver diferença significativa entre os grupos, esta diferença foi favorável ao algoritmo MemPlas com plasmídeo e *Path relinking*. A hipótese deste trabalho, que o uso de uma estratégia de melhoria de soluções baseada em *Path Relinking* produziriam soluções de melhor qualidade que o algoritmo MemPlas com Plasmídeo, foi rejeita em um cenário geral. O teste estatístico ANOVA indicou que não existem diferença entre o algoritmo MemPlas e as outras duas variações. Mas ao realizar testes isolados para cada instância, observou-se que o algoritmo que utiliza o plasmídeo juntamente com *Path Relinking* apresentou um melhor desempenho que as outras variações do algoritmo. Assim, a hipótese não é rejeitada em todas as instâncias. Uma hipótese para o algoritmo MemPlas com plasmídeo juntamente com o *Path Relinking* é que, ao juntar outra estratégia de intensificação ao algoritmo houve uma exploração melhor das soluções geradas.

Os resultados colhidos com o algoritmo apenas executando os operadores de busca local também tiveram resultados muito parecidos com as três variações dos algoritmos para o cenário geral. Isto indica que o componente *Plasmídeo* e o componente *Path Relinking* não conseguiram melhorar as soluções. Nem os dois componentes juntos apresentaram melhores resultados. Mas

ao analisar os resultados de cada instância separadamente, nos casos que houve diferenças entre os algoritmos, a melhoria é por conta da combinação MemPlas com plasmídeo e *Path Relinking*.

Em se tratando de metaheurísticas, é importante lembrar que existem dois fatores importantes: intensificação e diversificação. Os operadores de busca local e o *Path Relinking* são estratégias de intensificação, ou seja, exploram uma região relativamente pequena do espaço de busca, concentrando a exploração em encontrar as melhores soluções desta região. Por outro lado, o *Plasmídeo* é uma estratégia de diversificação, que busca explorar áreas ainda não exploradas em busca de locais promissores. Uma metaheurística boa deve buscar um equilíbrio entre ambos fatores.

O algoritmo MemPlas busca esse equilíbrio utilizando buscas locais e o *Plasmídeo/crossover*. A hipótese considerada neste trabalho é que o uso de um componente *Path Relinking* em substituição ou em complementação ao componente plasmídeo/*crossover* poderia melhorar os resultados do algoritmo MemPlas. A substituição baseava-se no fato de ambas estratégias serem baseadas em populações de indivíduos e os componentes de intensificação poderiam ter resultados similares ou um poderiam ter resultados melhores que o outro. Já a complementação poderia ser interessante por ser mais uma estratégia de intensificação para o algoritmo.

Os resultados obtidos nos experimentos fornecem evidência preliminar de que os operadores de buscas locais são os componentes que têm maior impacto na melhoria dos algoritmos. Isto é baseado na visão geral dos dados. Ambas as estratégias não conseguiram melhorar o trabalho que os operadores de busca local fizeram durante a execução.

Uma possível ameaça a esta evidência preliminar é o fato de que a implementação do algoritmo MemPlas com *Plasmídeo* utilizada por Goldberg et al. (2016) não está disponível para uso. Portanto, neste trabalho o algoritmo foi reimplementado baseando-se no pseudocódigo e na descrição do artigo onde foi definido originalmente. Apesar do cuidado em realizar os testes unitários para verificar o funcionamento correto dos componentes do algoritmo, alguns aspectos não estavam claramente descritos em Goldberg et al. (2016)), como por exemplo, o funcionamento exato de cada componente.

O componente baseado em *Path Relinking* apresentou o problema já descrito no capítulo 3. Ele gerava muitos indivíduos inválidos durante o seu funcionamento. Estes não poderiam ser submetidos ao cálculo da sua aptidão, e por consequência, não poderiam ser considerados para o cálculo de melhor solução. Este trabalho não realizou um experimento para verificar a proporção de indivíduos inválidos gerados, mas durante a fase de implementação foi possível verificar este problema.

### 4.3.2 Experimento 2

Um segundo experimento foi realizado para verificar se a hipótese de que as buscas locais eram os únicos componentes que melhoravam a qualidade das soluções. O raciocínio é que ao remover as buscas locais dos algoritmos, deveria haver uma piora na qualidade das soluções. Assim, foram executadas três variações de algoritmos, todas sem as buscas locais. A primeira é o algoritmo MemPlas original. A segunda é o MemPlas com o *Path Relinking* e a terceira é MemPlas com ambos componentes.

A metodologia do primeiro experimento foi utilizado novamente. Calibração de parâmetros utilizando o iracee as mesmas instâncias de problemas QCaRS. Cada variação do algoritmo foi executada 30 vezes em cada instância e mesma máquina e sistema operacional foi utilizado.

Nas tabelas 8 e 9 estão os resultados obtidos para as instâncias euclidianas e não euclidianas, respectivamente. As explicações sobre as colunas são as mesmas das tabelas 4 e 5. Nos

Tabela 8 – Resultados nas instâncias euclidianas no experimento 2

Problemas		Memplas Apenas Plasmídeo		Memplas Apenas Path Relinking		Memplas Apenas Plasmídeo + Path Relinking	
instância	melhor	média	melhor	média	melhor	média	melhor
Bolivia10e	384	394.0	384	503.2	459	469.6	386
AfricaSul11e	402	415.4	402	417.0	417	417.0	417
Niger12e	564	640.2	582	693.9	656	693.8	614
Mongolia13e	543	621.8	581	691.7	617	670.2	615
Indonesia14e	504	610.6	559	755.6	679	731.9	587
Argelia15e	487	627.2	605	700.1	605	684.9	605
India16e	705	957.2	876	1060.0	1000	1059.3	996
China17e	728	806.4	764	854.6	834	847.6	812
Etiopia10e	283	296.8	283	312.5	308	308.0	308
Mali11e	428	486.5	440	551.4	544	542.5	532
Chade12e	655	698.0	672	801.9	672	763.5	672
Ira13e	532	701.5	596	830.8	734	858.5	749
Mexico14e	492	581.2	523	711.0	614	690.1	591
Sudao15e	422	652.2	604	719.9	642	700.3	629
Australia16e	682	845.8	773	977.4	822	959.3	830
Canada17e	783	1024.3	879	1203.4	970	1169.8	992
Arabia14e	482	710.4	594	844.6	734	823.2	678
Cazaquistao15e	574	772.1	653	879.7	783	892.6	750
Brasil16e	619	753.7	722	894.0	746	879.5	746
Russia17e	750	963.6	916	1068.4	988	1062.5	959
BrasilAM26e	338	404.8	373	438.0	411	431.9	408
BrasilMG30e	368	409.6	400	419.7	409	419.2	409

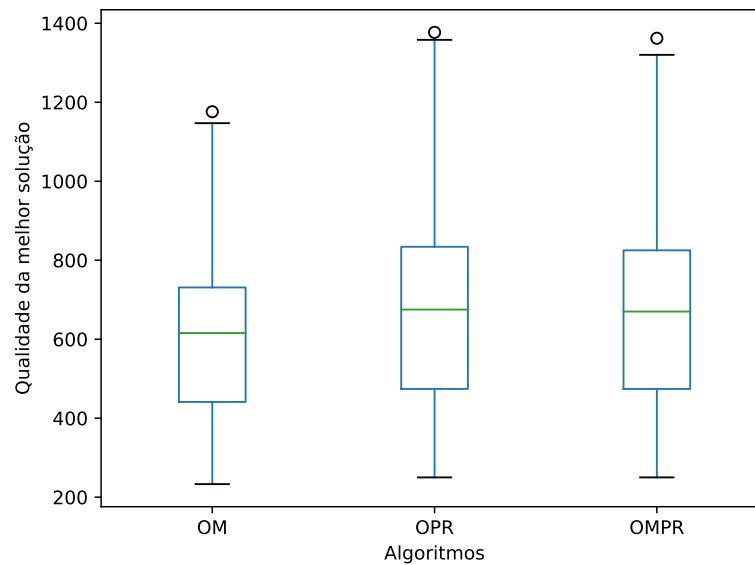
Tabela 9 – Resultados nas instâncias não euclidianas no experimento 2

Problemas		Memplas Apenas Plasmídeo		Memplas Apenas Path Relinking		Memplas Apenas Plasmídeo + Path Relinking	
instância	melhor	média	melhor	média	melhor	média	melhor
Bolivia10n	448	491.2	489	535.4	525	528.5	489
AfricaSul11n	537	609.4	557	729.9	664	697.0	621
Niger12n	607	631.1	618	688.1	667	674.0	621
Mongolia13n	551	583.2	555	666.9	621	647.2	584
Indonesias14n	522	614.9	573	697.1	661	698.2	615
Argelia15n	616	712.0	692	799.7	741	786.9	714
India16n	723	894.9	779	1075.2	885	1053.0	891
China17n	638	801.4	751	834.4	800	831.1	800
Etiopia10n	403	431.3	409	482.7	482	479.6	414
Mali11n	494	503.7	499	762.6	725	709.3	510
Chade12n	649	749.6	719	779.4	735	793.5	735
Ira13n	625	762.1	723	906.6	773	911.4	756
Mexico14n	610	755.3	703	992.9	777	971.3	777
Sudao15n	769	868.9	834	997.2	906	966.5	852
Australia16n	525	640.7	580	877.0	642	861.7	681
Canada17n	824	1075.5	924	1439.6	1033	1327.4	1091
Arabia14n	688	874.0	770	1157.7	960	1092.5	856
Cazaquistao15n	830	992.1	887	1304.9	1093	1206.7	995
Brasil16n	742	950.9	865	1120.6	891	1093.4	891
Russia17n	778	929.5	853	1103.8	890	1074.1	879
BrasilAM26n	107	151.4	136	199.3	158	187.9	159
BrasilMG30n	160	225.8	203	272.0	228	264.2	228

resultados para as euclidianas nota-se que a melhor e média ficaram próximos dos valores do estado da arte. Inclusive o MemPlasoriginal sem as busca locais conseguiu alcançar o melhor resultado em duas ocasiões. Os resultado para as instâncias não euclidianas também mostrou resultados próximos do estado da arte.

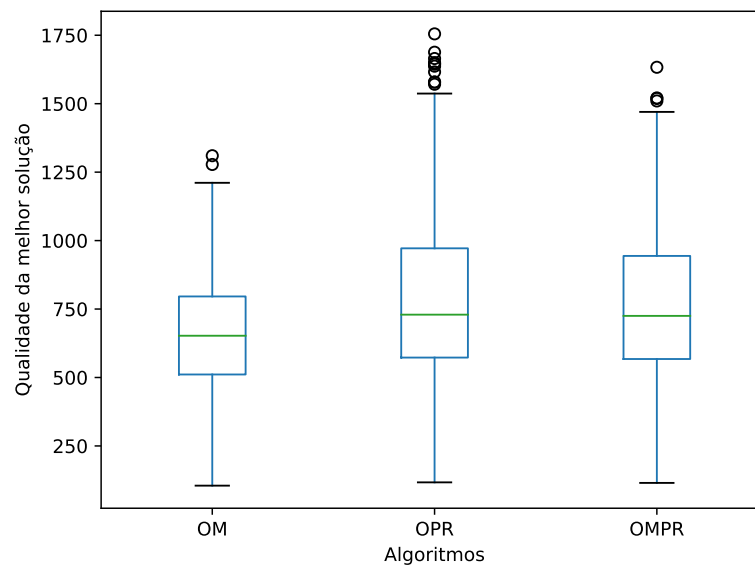
O teste ANOVA foi realizado com os resultados para verificar se existia diferença entre os três grupos de dados. Todas as execuções de cada algoritmo foram utilizadas, apenas separando por euclidianas e não euclidianas. A intenção era verificar se um algoritmo tinha um desempenho geral melhor do que outro. Nas figuras 18 e 19 estão os *boxplots* com os resultados apresentados pelos algoritmos nas instâncias euclidianas e não euclidianas, respectivamente. O valor apresentado pelo teste ANOVA nas instâncias euclidianas foi de aproximadamente 0%. Como este valor é inferior aos 5%, a hipótese nula, que os dados são equivalentes, foi rejeitada, logo existe diferença em pelo menos um par de grupos. Para as instâncias não euclidianas, o valor apresentado pelo teste ANOVA foi de aproximadamente 0%, também rejeitando a hipótese nula.

Figura 18 – Boxplot com os resultados em instâncias euclidianas no experimento 2



Fonte: Autor (2019).

Figura 19 – Boxplot com os resultados em instâncias não euclidianas no experimento 2



Fonte: Autor (2019).

A hipótese nula do teste do ANOVA foi rejeitada, assim foi feito o teste de Tukey para verificar qual algoritmo tinha o melhor desempenho. Na tabela 10 estão os resultados do teste. As siglas *om* indica o algoritmo MemPlas sem as buscas locais. A sigla *opr* indica o algoritmo MemPlas com *Path Relinking* sem as buscas locais. A sigla *ompr* indica o algoritmo MemPlas com ambos componentes sem as buscas locais.

Tanto para as euclidianas, como para as não euclidianas, o par *ompr-opr* não apresentou diferença entre si. Já os pares *om-ompr* e *om-opr* apresentou diferenças, pois rejeitaram a

hipótese nula do teste de Tukey. Em todos os casos o algoritmo MemPlas original apresentou melhores resultados.

Tabela 10 – Resultados do teste de Tukey agrupados por algoritmos no experimento 2

	Euclidianas			Não euclidianas		
	om-ompr	om-opr	ompr-opr	om-ompr	om-opr	ompr-opr
Dif. média	53.84	60.15	6.31	93.47	112.94	19.46
Valor $p(\%)$	<b>0.01</b>	<b>0.01</b>	77.75	<b>0.01</b>	<b>0.01</b>	29.41

Neste experimento também foi realizado a comparação instância por instância com o teste ANOVA. A tabela 11 apresenta os resultados dos testes. As instâncias que estão com o seu valor  $p$  em negrito são as que tiveram um resultado menor que 5%. Assim, existe pelo menos diferença em um par de algoritmos. Todas as 22 instâncias tanto euclidianas quanto não euclidianas apresentaram diferença entre ao menos um par de algoritmos.

Tabela 11 – Resultados do teste do ANOVA por instância para o experimento 2

Euclidianas		Não Euclidianas	
Instância	p (%)	Instância	p (%)
AfricaSul11e	<b>2.84</b>	AfricaSul11n	<b>0.00</b>
Arabia14e	<b>0.00</b>	Arabia14n	<b>0.00</b>
Argelia15e	<b>0.00</b>	Argelia15n	<b>0.00</b>
Australia16e	<b>0.00</b>	Australia16n	<b>0.00</b>
Bolivia10e	<b>0.00</b>	Bolivia10n	<b>0.00</b>
Brasil16e	<b>0.00</b>	Brasil16n	<b>0.00</b>
BrasilAM26e	<b>0.00</b>	BrasilAM26n	<b>0.00</b>
BrasilMG30e	<b>0.12</b>	BrasilMG30n	<b>0.00</b>
Canada17e	<b>0.00</b>	Canada17n	<b>0.00</b>
Cazaquistao15e	<b>0.00</b>	Cazaquistao15n	<b>0.00</b>
Chade12e	<b>0.00</b>	Chade12n	<b>1.51</b>
China17e	<b>0.00</b>	China17n	<b>0.00</b>
Etiopia10e	<b>0.08</b>	Etiopia10n	<b>0.00</b>
India16e	<b>0.00</b>	India16n	<b>0.00</b>
Indonesia14e	<b>0.00</b>	Indonesia14n	<b>0.00</b>
Ira13e	<b>0.00</b>	Ira13n	<b>0.00</b>
Mali11e	<b>0.00</b>	Mali11n	<b>0.00</b>
Mexico14e	<b>0.00</b>	Mexico14n	<b>0.00</b>
Mongolia13e	<b>0.00</b>	Mongolia13n	<b>0.00</b>
Niger12e	<b>0.00</b>	Niger12n	<b>0.00</b>
Russia17e	<b>0.00</b>	Russia17n	<b>0.00</b>
Sudao15e	<b>0.02</b>	Sudao15n	<b>0.00</b>

Foi realizado um teste de Tukey instância por instância. Os resultados estão nas tabelas 12 e 13. Aqui também foi adotado 10% para que a hipótese nula seja mantida. Na tabela, todos

Tabela 12 – Resultados do teste de Tukey no experimento 2 para instâncias euclidianas

Instancia	om vs. opr		om vs. ompr		opr vs. ompr	
	Dif. Media	p (%)	Dif. Media	p (%)	Dif. Media	p (%)
AfricaSul11e	1.57	<b>5.31</b>	1.57	<b>5.31</b>	0.00	90.00
Arabia14e	112.80	<b>0.10</b>	134.20	<b>0.10</b>	21.40	44.29
Argelia15e	57.70	<b>0.10</b>	72.93	<b>0.10</b>	15.23	30.74
Australia16e	113.57	<b>0.10</b>	131.60	<b>0.10</b>	18.03	39.27
Bolivia10e	75.57	<b>0.10</b>	109.23	<b>0.10</b>	33.67	<b>0.43</b>
Brasil16e	125.77	<b>0.10</b>	140.30	<b>0.10</b>	14.53	61.73
BrasilAM26e	27.13	<b>0.10</b>	33.23	<b>0.10</b>	6.10	22.79
BrasilMG30e	9.53	<b>0.54</b>	10.10	<b>0.30</b>	0.57	90.00
Canada17e	145.50	<b>0.10</b>	179.10	<b>0.10</b>	33.60	34.80
Cazaquistao15e	120.50	<b>0.10</b>	107.53	<b>0.10</b>	-12.97	71.63
Chade12e	65.53	<b>0.10</b>	103.87	<b>0.10</b>	38.33	<b>2.03</b>
China17e	41.17	<b>0.10</b>	48.20	<b>0.10</b>	7.03	38.25
Etiopia10e	11.21	<b>1.89</b>	15.75	<b>0.10</b>	4.53	34.96
India16e	102.17	<b>0.10</b>	102.80	<b>0.10</b>	0.63	90.00
Indonesia14e	121.27	<b>0.10</b>	144.97	<b>0.10</b>	23.70	<b>7.97</b>
Ira13e	156.93	<b>0.10</b>	129.23	<b>0.10</b>	-27.70	23.67
Mali11e	56.00	<b>0.10</b>	64.90	<b>0.10</b>	8.90	38.43
Mexico14e	108.93	<b>0.10</b>	129.83	<b>0.10</b>	20.90	<b>8.75</b>
Mongolia13e	48.37	<b>0.10</b>	69.87	<b>0.10</b>	21.50	<b>8.82</b>
Niger12e	53.57	<b>0.10</b>	53.67	<b>0.10</b>	0.10	90.00
Russia17e	98.90	<b>0.10</b>	104.83	<b>0.10</b>	5.93	75.97
Sudao15e	48.10	<b>1.03</b>	67.73	<b>0.10</b>	19.63	44.87

os valores de  $p$  em negrito, são valores que ficaram abaixo dos 10%. As explicações sobre a colunas são as mesmas da tabela 7.

O par MemPlas original sem as buscas locais e MemPlas com *Path Relinking* sem buscas locais (om vs. opr) teve diferença em todas as instâncias com diferença média positiva. Assim o MemPlas original apresentou o melhor resultado nestas instâncias. O par MemPlas original sem as buscas locais e o MemPlas com ambos componentes sem buscas locais (om vs. ompr) apresentou diferença em todas as instâncias. Todos com valor da diferença média positiva, que indica que o MemPlas original apresentou o melhor resultado nestas instâncias. O par MemPlas com *Path Relinking* sem buscas locais e MemPlas com ambos componentes sem busca locais (opr vs. ompr) apresentou diferença em 5 instâncias. Todos com diferença média positiva, indicando que o MemPlas com *Path Relinking* sem buscas locais teve o melhor desempenho nestas instâncias.

De modo geral, esse novo experimento apresentou indícios que o algoritmo MemPlas original sem buscas locais consegue fornecer melhores soluções em termos de qualidade de solução que os outros dois algoritmos. Assim, rejeitando a hipótese deste trabalho, que o compo-



Tabela 13 – Resultados do teste de Tukey no experimento 2 para instâncias não euclidianas

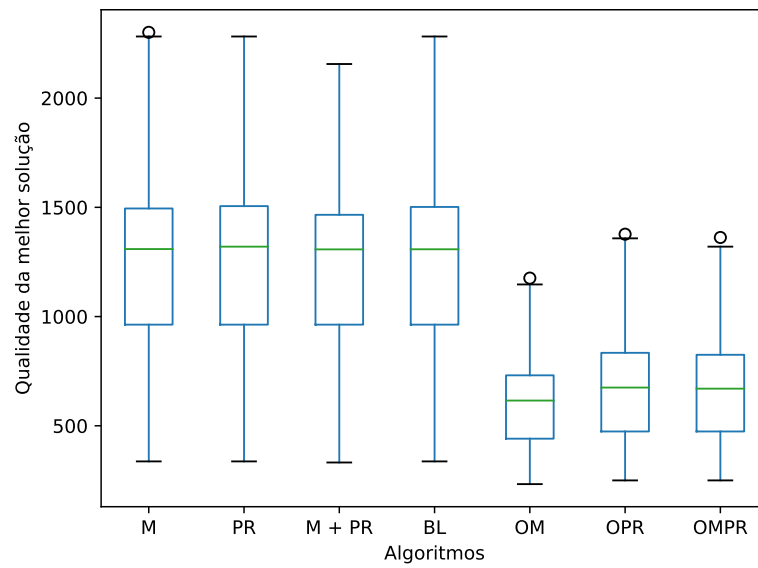
Instancia	om vs. opr		om vs. ompr		opr vs. ompr	
	Dif. Media	p (%)	Dif. Media	p (%)	Dif. Media	p (%)
AfricaSul11n	87.57	<b>0.10</b>	120.50	<b>0.10</b>	32.93	<b>0.10</b>
Arabia14n	218.43	<b>0.10</b>	283.70	<b>0.10</b>	65.27	<b>4.11</b>
Argelia15n	74.90	<b>0.10</b>	87.73	<b>0.10</b>	12.83	42.75
Australia16n	220.97	<b>0.10</b>	236.33	<b>0.10</b>	15.37	77.05
Bolivia10n	37.28	<b>0.10</b>	44.15	<b>0.10</b>	6.87	55.14
Brasil16n	142.50	<b>0.10</b>	169.73	<b>0.10</b>	27.23	68.40
BrasilAM26n	36.50	<b>0.10</b>	47.90	<b>0.10</b>	11.40	<b>0.57</b>
BrasilMG30n	38.43	<b>0.10</b>	46.20	<b>0.10</b>	7.77	19.71
Canada17n	251.90	<b>0.10</b>	364.03	<b>0.10</b>	112.13	<b>0.74</b>
Cazaquistao15n	214.60	<b>0.10</b>	312.77	<b>0.10</b>	98.17	<b>0.21</b>
Chade12n	43.90	<b>1.27</b>	29.80	12.50	-14.10	61.14
China17n	29.67	<b>0.10</b>	32.97	<b>0.10</b>	3.30	85.19
Etiopia10n	48.33	<b>0.10</b>	51.37	<b>0.10</b>	3.03	78.41
India16n	158.10	<b>0.10</b>	180.30	<b>0.10</b>	22.20	56.19
Indonesia14n	83.30	<b>0.10</b>	82.13	<b>0.10</b>	-1.17	90.00
Ira13n	149.33	<b>0.10</b>	144.50	<b>0.10</b>	-4.83	90.00
Mali11n	205.60	<b>0.10</b>	258.87	<b>0.10</b>	53.27	<b>0.10</b>
Mexico14n	216.00	<b>0.10</b>	237.63	<b>0.10</b>	21.63	53.19
Mongolia13n	63.93	<b>0.10</b>	83.67	<b>0.10</b>	19.73	<b>3.09</b>
Niger12n	42.87	<b>0.10</b>	56.97	<b>0.10</b>	14.10	23.03
Russia17n	144.63	<b>0.10</b>	174.37	<b>0.10</b>	29.73	54.00
Sudao15n	97.60	<b>0.10</b>	128.37	<b>0.10</b>	30.77	17.01

nente baseado em *Path Relinking* poderia melhorar as soluções apresentadas pelo algoritmo. A rejeição foi no cenário geral e também no cenário de separação por instâncias.

Nas figuras 20 e 21 são apresentados os *boxplots* com os valores de todos os resultados dos algoritmos apresentados nos experimentos 1 e 2, para as euclidianas e não euclidianas, respectivamente. Ao realizar o teste ANOVA com os sete grupos de dados, foi obtido aproximadamente 0% para instância euclidianas e não euclidianas. A partir destes *boxplots*, e observando as tabelas 4, 5, 8 e 9, pode-se notar que a remoção das buscas locais dos algoritmos fez com que os resultados melhorassem muito e aproximassem mais do estado da arte do que quando ainda era utilizado buscas locais nos algoritmos. Esse fato lança a hipótese contra-intuitiva que as buscas locais estavam priorando a qualidade da solução.

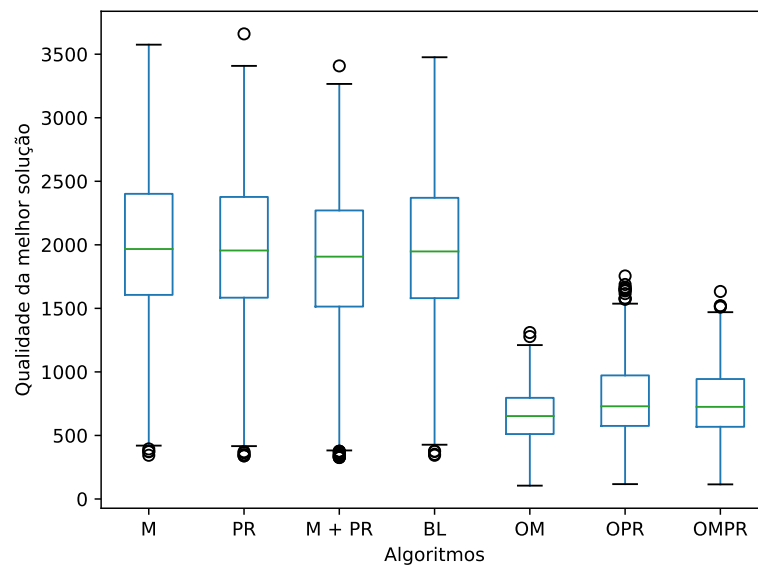
Os resultado obtidos nos experimentos fornecem evidências que os operadores de busca local não estavam melhorando as soluções durante o algoritmo. No experimento sem buscas locais, os algoritmos conseguiram um melhor desempenho e inclusive chegaram ao estado da arte em duas instâncias. A hipótese levantada no primeiro experimento, que o *plasmídeo/crossover* e o *Path Relinking* apresentavam problemas parece não mais fazer sentido. Agora essa hipótese

Figura 20 – Boxplot com os resultados em instâncias euclidianas para ambos experimentos



Fonte: Autor (2019).

Figura 21 – Boxplot com os resultados em instâncias não euclidianas para ambos experimentos



Fonte: Autor (2019).

parece ser o problema das buscas locais. Novamente o problema é a dificuldade em replicar o funcionamento do algoritmo sem o código fonte original, apenas se baseando no trabalho na qual foi definido.

## 5 CONCLUSÃO

Este trabalho estudou o problema Quota Car Traveing Selesman (QCaRS). Este problema consiste em minimizar o custo de uma viagem entre um grupo de cidades, visitando apenas um subgrupo, e garantindo a visita de cidades que cumpram com uma satisfação mínima para o viajante. Cada cidade possui uma quota, que indica a satisfação do viajante em visitá-la. O viajante precisa também minimizar o custo do percurso.

Este trabalho desenvolveu e avaliou um componente baseado em *Path Relinking* para o algoritmo MemPlas. O algoritmo MemPlas utiliza um componente baseado em *plasmídeo/crossover*. A hipótese considerada neste trabalho foi de que a substituição do componente *Plasmídeo* pelo *Path Relinking* poderia trazer resultados melhores considerando qualidade das soluções. Realizar a junção entre ambos os componentes também foi estudado neste trabalho.

Para avaliar o componente baseado em *Path Relinking* proposto, um experimento foi realizado em duas etapas. Na primeira, foi utilizado uma ferramenta para calibração de parâmetros para encontrar a melhor combinação que maximizava a qualidade da solução. Na segunda etapa, foi feito um experimento que executava os algoritmos trinta vezes em um grupo de 44 instâncias. Os testes estatísticos ANOVA indicaram que os resultados apresentados pelas três variações do algoritmo não apresentavam diferenças quando todas as instâncias são consideradas conjuntamente. Os resultados foram comparados com um algoritmo que não utilizava nenhum dos componentes para verificar se apresentavam alguma melhora. A hipótese foi rejeitada, pois todos os grupos apresentaram resultados parecidos em uma visão geral. A reimplementação do componente *plasmídeo/crossover* e a implementação do componente baseado em *Path Relinking* não conseguiram melhorar as soluções geradas. Ao realizar uma análise mais detalhada por instância, foi possível observar que o algoritmo que utilizava o *plasmídeo/crossover* juntamente com o *Path Relinking* apresentou melhores resultados para algumas instâncias.

Com base nos resultados desse primeiro experimento, foi realizado um segundo para testar a hipótese de que as buscas locais eram os únicos componentes que melhoravam as soluções. Neste experimento, as variações do algoritmo foram executadas retirando o componente das buscas locais. O experimento também foi dividido em duas etapas, uma para a calibração de parâmetros e outra para a execução dos algoritmos trinta vezes no mesmo grupo de 44 instâncias. Os testes estatísticos do segundo experimento mostrando que o algoritmo original apresentou o melhor resultado, tanto no cenário geral, como no cenário onde houve a avaliação por instância. Assim, a hipótese deste trabalho foi rejeitada neste segundo experimento.

Como trabalhos futuros, sugere-se verificar quatro pontos. O primeiro é reimplementar o *plasmídeo/crossover* do algoritmo MemPlas, estudando possíveis melhorias para o compo-

nente. O segundo seria identificar a proporção de indivíduos inválidos que o componente *Path Relinking* gera. O terceiro ponto a ser estudado é criar uma estratégia para contornar este problema e realizar um processamento para tornar o indivíduo válido novamente. O último ponto é realizar uma investigação do por que o algoritmo com *plasmídeo/crossover* juntamente com o *Path Relinking* conseguiu apresentar melhores resultados.

## REFERÊNCIAS

- CROES, G. A. A method for solving traveling-salesman problems. **Operations research**, 1958. INFORMS, v. 6, n. 6, p. 791–812, 1958.
- GENDREAU, M.; POTVIN, J.-Y. et al. **Handbook of metaheuristics**. United States: Springer, 2010.
- GOLDBARG, M. C.; GOLDBARG, E. F.; MENEZES, M. d. S.; LUNA, H. P. Quota traveling car renter problem: Model and evolutionary algorithm. **Information Sciences**, 2016. Elsevier, v. 367, p. 232–245, 2016.
- GOLDBARG, M. C.; GOLDBARG, E. F.; MENEZES, M. da S. Um algoritmo memético para o problema do caixeiro alugador com coleta de prêmios. **Anais 45 Simpósio Brasileiro de Pesquisa Operacional**, 2013. p. 2543–2554, 2013.
- GOOGLE INC. **Googletest - Google Testing and Mocking Framework**. 2019. Disponível em: <<https://github.com/google/googletest>>. Acesso em: 16 out. 2019.
- HANCOCK, G. R.; KLOCKARS, A. J. The quest for  $\alpha$ : developments in multiple comparison procedures in the quarter century since. **Review of Educational Research**, 1996. Sage Publications Sage CA: Thousand Oaks, CA, v. 66, n. 3, p. 269–306, 1996.
- LINDEN, R. **Algoritmos genéticos (2a edição)**. Brasil: Brasport, 2008.
- LÓPEZ-IBÁÑEZ, M.; DUBOIS-LACOSTE, J.; STÜTZLE, T.; BIRATTARI, M. **The irace package, iterated race for automatic algorithm configuration**. [S.l.], 2011.
- MENEZES, M. d. S.; GOLDBARG, M. C.; GOLDBARG, E. F.; FERREIRA, V. E. S.; COLAÇO, G. C. Abordagens GRASP aplicadas ao problema quota cars. **Anais do 49 Simpósio Brasileiro de Pesquisa Operacional**, 2017. p. 1807–1818, 2017.
- MENEZES, M. da S.; GOLDBARG, M. C.; GOLDBARG, E. F. A memetic algorithm for the prize-collecting traveling car renter problem. **2014 IEEE Congress on Evolutionary Computation (CEC)**, 2014. p. 3258–3265, 2014.
- NUMFOCUS. **Python Data Analysis Library**. 2019. Disponível em: <<https://pandas.pydata.org/>>. Acesso em: 16 out. 2019.
- PFLEEGER, S. L. **Engenharia de software: teoria e prática**. Brasil: Prentice Hall, 2004.
- TALBI, E.-G. **Metaheuristics: from design to implementation**. United States: John Wiley & Sons, 2009.
- WAZLAWICK, R. **Engenharia de software: conceitos e práticas**. Brasil: Elsevier Brasil, 2013.
- WOLFE, H. E. **Introduction to non-Euclidean geometry**. [S.l.]: Courier Corporation, 2012.