Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Bacharelado em Ciência da Computação

Abordagens para variações de Problema do Caixeiro Viajante Clusterizado

Edson Lopes da Silva Junior

JUIZ DE FORA
JANEIRO, 2022

Abordagens para variações de Problema do Caixeiro Viajante Clusterizado

Edson Lopes da Silva Junior

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Bacharelado em Ciência da Computação

Orientador: Luciana Brugiolo Gonçalves

Coorientador: Lorenza Leão Oliveira Moreno

JUIZ DE FORA

JANEIRO, 2022

Abordagens para variações de Problema do Caixeiro Viajante Clusterizado

Edson Lopes da Silva Junior

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Luciana Brugiolo Gonçalves
D.Sc. em Ciência da Computação

Lorenza Leão Oliveira Moreno
D.Sc. em Informática

Stênio Sã Rosário Furtado Soares
D.Sc. em Ciência da Computação

Heder Soares Bernardino
D.Sc. em Modelagem Computacional

JUIZ DE FORA

31 DE JANEIRO, 2022

Resumo

O Problema do Caixeiro Viajante, ou *Traveling Salesman Problem* (TSP), é um problema clássico presente na literatura de Otimização Combinatória como um dos mais estudados do mundo. Naturalmente, variações do TSP surgiram para atender diversas aplicações. Este trabalho aborda duas variações do TSP, o *Clustered* TSP (CTSP) e o *Tabu Clustered* TSP (TCTSP). O trabalho apresenta um modelo de programação inteira para o TCTSP utilizado com o resolvedor CPLEX que encontra as soluções ótimas para 60% das instâncias da literatura, além de duas variações da meta-heurística GRASP que encontram valores iguais ou melhores em relação ao resolvedor e as literaturas existentes. O trabalho conta com uma versão de ILS para o CTSP que apresentou resultados competitivos com duas abordagens da literatura, entretanto com resultado superado posteriormente por uma abordagem evolutiva.

Palavras-chave: Problema do Caixeiro Viajante, Otimização Combinatória, Metaheurísticas.

Abstract

The Traveling Salesman Problem (TSP) is a well-studied problem in Combinatorial Optimization. Naturally, TSP variants arised to assign many applications. This work addresses two TSP variants, named Clustered TSP (CTSP) and Tabu Clustered TSP (TCTSP). We propose a integer programming model for the TCTSP using the CPLEX solver, which founds the optimal solutions of 60% of the literature instances, besides two versions of GRASP metaheuristic that are competitive in comparsion with the solver and the literature. The work also includes a ILS heuristic for the CTSP that presented competitive results compared to two approaches present in the literature. However those results were surpassed after by an evolutive approach.

Keywords: TSP, Combinatorial Optimization, Meta heuristics.

Agradecimentos

Quero começar agradecendo meu pai Edson, minha mãe Valéria e minha vó Maria Eulália por acreditar em mim e me apoiar de todas as formas durante minha graduação. Minhas conquistas só são possíveis por vocês estarem sempre presentes na minha vida.

À minha namorada Juliana, por ser esta companheira incrível que compartilhou tantos momentos igualmente incríveis comigo, compreensiva, por estar sempre ao meu lado me apoiando e conseguir tirar o meu melhor mesmo nas horas mais difíceis. A vida é melhor quando você está por perto.

Aos meus irmãos biológicos Jéssica, Hugo, meus primos Luana e Gabriel, que são meus irmãos de consideração, minhas tias Cecília, Gilmara, Patrícia, minha sogra Selma e sua mãe Eunice, amigos e demais familiares por proporcionar momentos felizes, contribuírem na formação do meu caráter e por me inspirar a ser uma pessoa melhor.

Aos professores Luciana, Stênio e Lorenza por terem me orientado com tanta dedicação durante toda a minha graduação, por toda a autonomia e confiança depositada no meu trabalho, por terem desenvolvido meu lado cientista e, claro, por todos esses anos de amizade.

Ao tutor do Grupo de Educação Tutorial Ciência da Computação (GETComp) Marcelo Caniato e meus companheiros GETianos por compartilhar as experiencias incríveis que vivi durante minha participação no grupo, fator determinante na minha formação pessoal e profissional e onde fiz amigos que vou levar pra vida toda.

Aos meus amigos de turma, com quem pude evoluir lado a lado, superando os momentos difíceis e obstáculos encontrados pelo caminho.

Aos professores do Departamento de Ciência da Computação da Universidade pela contribuição no meu desenvolvimento técnico e profissional.

"Nada melhor que um laço vermelho de fita... Fita que enlaça na fibra mais linda do abraço da casa ao lado".

Juliana Saleme (Linhas cernes)

Conteúdo

Lista de Figuras **6**

Lista de Tabelas **7**

Lista de Abreviações **8**

1 Introdução **9**

2 Descrição dos Problemas **11**
2.1 Caixeiro Viajante *Clusterizado* . . . 11
2.2 Caixeiro Viajante *Clusterizado* com conjuntos Tabu . . . 12

3 Fundamentação Teórica **13**
3.1 Trabalhos relacionados . . . 13
3.2 Programação Inteira (PI) . . . 14
3.3 Meta-heurísticas . . . 15
3.3.1 Reactive Greedy Random Adaptive Search Procedure . . . 15
3.3.2 Random Variable Neighborhood Descent . . . 17
3.3.3 Iterated Local Search . . . 18

4 Abordagens Propostas **20**
4.1 Algoritmo construtivo . . . 20
4.2 Busca Local Inter-*cluster* . . . 22
4.3 Busca Local Intra-*cluster* . . . 25
4.4 Busca Local Conjuntos Tabus . . . 26
4.5 2-OPT . . . 28
4.6 R-GRASP para o TCTSP . . . 29
4.7 GRASP-RVND para o TCTSP . . . 30
4.8 ILS-RVND para o CTSP . . . 31
4.9 Programação Inteira para o TCTSP . . . 33
4.9.1 Problema do subciclo . . . 33
4.9.2 Modelo inicial . . . 35
4.9.3 *Lazy Constraints* . . . 37

5 Experimentos Computacionais **38**
5.1 Instâncias . . . 38
5.2 Ambiente de testes . . . 38
5.3 Execução . . . 39
5.4 Resultados obtidos . . . 42
5.4.1 CTSP . . . 42
5.4.2 TCTSP . . . 44

6 Conclusão **52**

Bibliografia **54**

Lista de Figuras

2.1 Exemplo de solução para uma instância do CTSP. 11
2.2 Exemplo de solução para uma instância do TCTSP. 12

4.1 Visão geral dos componentes e como são utilizados 20
4.2 Criação de subciclos dos *clusters* (SILVA JUNIOR et al., 2019). 21
4.3 Processo construtivo da segunda etapa. 22
4.4 Iteração da Busca Local Inter-*cluster* melhorando conexão entre os *clusters* C_3 e C_4. Aresta (19,16) e $\gamma = 3$ arestas adicionais de cada lado são removidas e substituidas pelo caminho de menor custo de 2 ao 21. 23
4.5 Grafo $\bar{G}$ correspondente a Figura 4.4. 24
4.6 Exemplo de movimento presente na Busca Local intra-*cluster* 25
4.7 Exemplo de movimento presente no Refinamento de Conjuntos Tabus . . . 28
4.8 Solução viável para o TCTSP 30
4.9 Exemplo de subciclos em uma instância do TSP 34

Lista de Tabelas

5.1 Descrição das instâncias TCTSP 39
5.2 Conjunto de Instâncias pequenas CTSP 40
5.3 Conjunto de Instâncias médias CTSP 41
5.4 Conjunto de Instâncias grandes CTSP 41
5.5 Comparação dos resultados obtidos ILS-RNVD e (MESTRIA, 2018) para instâncias pequenas 43
5.6 Comparação dos resultados para o CTSP obtidos com ILS-RNVD e (LU; HAO; WU, 2020) para instâncias médias 46
5.7 Comparação dos resultados obtidos para o CTSP com ILS-RNVD e (LU; HAO; WU, 2020) para instâncias grandes 47
5.8 Comparação dos resultados estatísticos para o CTSP obtidos do ILS-RNVD e (LU; HAO; WU, 2020) 48
5.9 Comparação dos resultados para o TCTSP obtidos com GRASP-RNVD e (SILVA JUNIOR et al., 2019) para instâncias do conjunto 2 49
5.10 Comparação dos resultados para o TCTSP obtidos com GRASP-RNVD e (SILVA JUNIOR et al., 2019) para instâncias do conjunto 2 com a exclusão do valor PIC 50
5.11 Comparação dos resultados estatísticos para o TCTSP obtidos pelos métodos do TCTSP 51

Lista de Abreviações

DCC	Departamento de Ciência da Computução
UFJF	Universidade Federal de Juiz de Fora
TSP	*Traveling Salesman Problem*
GTSP	*Generalized Traveling Salesman Problem*
CTSP	*Clustered Traveling Salesman Problem*
TCTSP	*Tabu Clustered Traveling Salesman Problem*
CGTSP	*Clustered Generalized Traveling Salesman Problem*
GRASP	*Greedy Randomized Adaptative Search Procedure*
R-GRASP	*Reactive Greedy Randomized Adaptative Search Procedure*
VND	*Variable Neightborhood Descent*
RVND	*Random Variable Neightborhood Descent*
ILS	*Iterated Local Search*
PI	Programação Inteira

1 Introdução

O Problema do Caixeiro Viajante, ou *Traveling Salesman Problem* (TSP), é um problema clássico presente na literatura de Otimização Combinatória como um dos problemas mais estudados da área (HUERTA et al., 2022). Dado um conjunto de n cidades, o objetivo do problema é encontrar um ciclo de custo mínimo passando por cada cidade uma única vez. O estudo do problema foi motivado por diversas aplicações do mundo real (PUNNEN, 2007), como por exemplo escalonamento de tarefas, fabricação de celulares, e problema de atribuição de frequências.

Naturalmente, variações do TSP surgiram para atender diferentes aplicações. Neste trabalho são estudadas duas variações do TSP. A primeira variação está presente na literatura há décadas e pode-se considerar um clássico. Trata-se do Clustered Traveling Salesman Problem, ou Problema do Caixeiro Viajante *Clusterizado* (CTSP), proposto originalmente por (CHISMAN, 1975). Neste problema, as cidades são separadas em k grupos disjuntos, denominados *clusters* e todas as cidades pertencentes a um mesmo *cluster* devem ser visitadas consecutivamente na solução.

A segunda variação do TSP tratada neste trabalho foi proposta no ano de 2018 em (ZHANG et al., 2018), denominado Tabu Clustered Traveling Salesman Problem ou Problema do Caixeiro Viajante *Clusterizado* com conjuntos Tabu (TCTSP). O problema se assemelha ao CTSP por também agrupar as cidades em *clusters*, mas difere por ter um segundo agrupamento das cidades, estes grupos são intitulados como tabus. O objetivo do problema consiste em visitar uma única cidade de cada grupo tabu, além de respeitar a consecutividade na visitação das cidades de um mesmo *cluster*.

O CTSP possui uma vasta gama de aplicações associadas ao problema. O trabalho (MESTRIA; OCHI; MARTINS, 2013) lista roteamento automático de depósitos, despache de veículos de emergência, planejamento de produção, desfragmentação de discos rígidos de computadores, roteamento de veículos, fornecimento para lojas e mercados, entre outras.

O TCTSP foi proposto a partir de uma aplicação relacionada ao gerenciamento

de redes entre satélites e a comunicação entre eles com as bases terrestres (ZHANG et al., 2018). Outra aplicação presente em (BANIASADI et al., 2020) pode ser modelada para o problema do TCTSP. Trata-se de um problema de logística de entrega de pacotes auxiliadas por *drones*.

Além das aplicações existentes para ambos os problemas, a dificuldade de resolução dos problemas torna impeditivo encontrar solução ótima em tempo polinomial, desta forma torna-se necessário o uso de meta-heurísticas para encontrar boas soluções para estas aplicações em tempo desejável. Além disso, exite a possibilidade de encontrar limites inferiores para as instâncias presentes na literatura.

2 Descrição dos Problemas

Este capítulo define a modelagem utilizada para os problemas estudados neste trabalho.

2.1 Caixeiro Viajante *Clusterizado*

O Problema do Caixeiro Viajante *Clusterizado* pode ser modelado a partir de um grafo. Seja $G = (V, E)$ um grafo completo e ponderado nas arestas, onde $V = \{v_1, v_2, ..., v_n\}$ é o conjunto de vértices representando as cidades e $E = \{(v_i, v_j) | v_i, v_j \in V, i \neq j\}$ é o conjunto de arestas representando o caminho entre duas cidades, sendo a aresta ponderada pela distância entre as cidades. Os vértices são definidos como pontos no espaço $\mathbb{R}^2$. Assim, o peso c_{ij} de cada aresta (v_i, v_j), o custo pode ser dado pela distância euclidiana entre os vértices, por exemplo. Uma instância do problema pressupõe ainda o particionamento do conjunto de vértices em k *clusters*: $C_1, ..., C_k$. O problema consiste em obter um ciclo Hamiltoniano de custo mínimo sobre os k *clusters* de forma que os vértices que pertençam a um mesmo *cluster* sejam visitados consecutivamente.

A Figura 2.1 ilustra um exemplo de solução para uma instância de 7 cidades com 2 *clusters* contendo 3 nós e 1 *cluster* com 1 nó, totalizando 3 *clusters*.

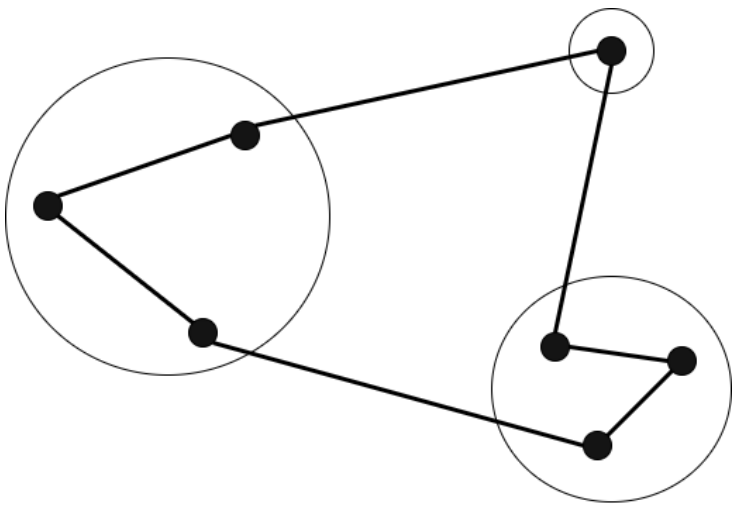

Figura 2.1: Exemplo de solução para uma instância do CTSP.

2.2 Caixeiro Viajante *Clusterizado* com conjuntos Tabu

O Problema do Caixeiro Viajante *Clusterizado* com Tabu pode ser modelado a partir de um grafo $G = (V, E)$ completo e ponderado nas arestas , onde $V = \{v_1, v_2, ..., v_n\}$ é o conjunto de vértices representando as cidades e $E = \{(v_i, v_j)|v_i, v_j \in V, i \neq j\}$ o conjunto de arestas representando conexão entre um par de cidades ponderada pela distância entre estas. Os vértices são definidos como pontos no espaço R^2. Assim, o peso c_{ij} de toda arestas (v_i, v_j) é dado pela distância euclidiana entre os vértices. Uma instância do problema pressupõe ainda o particionamento do conjunto de vértices em k *clusters*: $C_1, ..., C_k$; e em m subconjuntos tabus: $T_1, ..., T_m$. Os *clusters* são grupos disjuntos onde a união dos grupos resulta no conjunto V, isto implica que cada cidade está presente em exatamente um *cluster*. Os grupos tabus apresentam a mesma característica e portanto cada cidade está presente em exatamente um tabu.

O problema consiste em obter um ciclo de custo mínimo sobre os m subconjuntos tabus de forma que, em cada tabu um único vértice seja visitado. Além disso, os vértices selecionados para serem incluídos na rota, que pertençam a um mesmo *cluster*, devem ser visitados consecutivamente.

A Figura 2.2 ilustra um exemplo de solução. Nesta instâncias é possível observar um total de 7 cidades organizada em 3 *clusters*, dois *clusters* com 3 nós e um *cluster* com 1 único nó. Além disso, a instância possui 5 tabus, sendo dois contendo 2 nós e 3 tabus com apenas 1 nó.

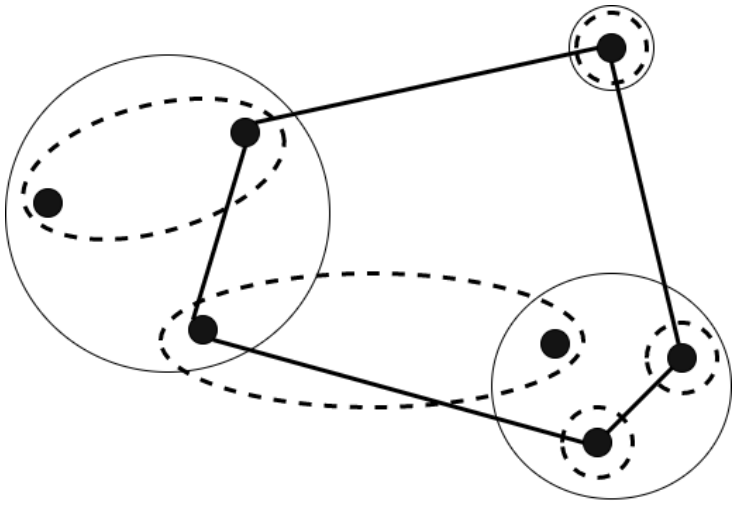

Figura 2.2: Exemplo de solução para uma instância do TCTSP.

3 Fundamentação Teórica

Este capítulo é dedicado a apresentação das referencias teóricas utilizadas como base para o desenvolvimento das técnicas presentes neste trabalho. A seção a seguir introduz os trabalhos relacionados que de alguma forma contribuíram na construção deste.

3.1 Trabalhos relacionados

O problema do Caixeiro Viajante *Clusterizado* com Tabu foi proposto por (ZHANG et al., 2018). Os autores apresentaram um Algoritmo Colônia de Formigas, ou Ant Colony Optimization (ACO), combinado com *Path Relinking* de três formas diferentes, resultando em abordagens distintas para resolver o problema. Com o intuito de criar instâncias com os conjuntos tabus, os autores adaptaram 10 instâncias de (MESTRIA; OCHI; MARTINS, 2013) aplicando o método de particionamento. Enfim, (ZHANG et al., 2018) comparou sua abordagem com a abordagem proposta em (MESTRIA; OCHI; MARTINS, 2013), que usa uma Busca Gulosa Randomizada Adaptativa, ou *Greedy Randomized Adaptative Search* (GRASP) para tratar o CTSP.

O Problema do Caixeiro Viajante *Clusterizado* está presente a mais tempo na literatura, proposto originalmente no trabalho (CHISMAN, 1975). O trabalho (MESTRIA; OCHI; MARTINS, 2013) utiliza o resolvedor Concorde (APPLEGATE; BIXBY; CHVÁTAL, 2007) com objetivo de obter melhores limites inferiores. Esses limites inferiores são usados como referência para outros trabalhos da literatura. O último trabalho relacionado diretamente ao problema (LU; HAO; WU, 2020) utiliza métodos consolidados de resolução do TSP, heurísticos e exatos para o CTSP e domina os resultados em qualidade e em tempo de execução. O trabalho (LU; HAO; WU, 2020) indica que existem diversos algoritmos propostos anteriormente dedicados a resolver o problema utilizando abordagens heurísticas, exatas e aproximadas.

Dos trabalhos anteriores a (LU; HAO; WU, 2020) presentes na literatura sobre o CTSP: (MESTRIA, 2016) propõe uma heurística híbrida baseada na Busca Local Iterada,

ou *Iterated Local Search* (ILS), GRASP e Descida de Vizinhança Variável, ou *Variable Neighboorhood Descend* (VND). No trabalho (MESTRIA, 2018), uma heurística híbrida composta por GRASP, ISL e um novo procedimento similar ao anterior chamado Descida de Vizinhança Variável Aleatória, ou *Variable Neighboorhood Random Descend* (VNRD) é descrito. As soluções encontradas por (MESTRIA, 2018) foram comparadas com as soluções obtidas nos trabalhos anteriores (MESTRIA; OCHI; MARTINS, 2013; MESTRIA, 2016) e o método exato Concorde, obteve as soluções ótimas na literatura para um grupo de instâncias do CTSP.

O Problema do Caixeiro Viajante Generalizado (GTSP) foi introduzido por (LAPORTE; MERCURE; NOBERT, 1987). (POP, 2007) introduz seis formulações matemáticas para resolver o GTSP de forma exata. Esses modelos são utilizados como inspirações para desenvolver formulações matemáticas para o TCTSP.

Uma nova combinação do CTSP e do GTSP é introduzida por (BANIASADI et al., 2020), nomeada por TSP *Clusterizado* Generalizado (CGTSP). Nesta variação, o conjunto de nós é subdividido em *clusters*, e os *clusters* contém *subclusters* de nós. O problema é similar ao GTSP já que a solução deve visitar apenas uma cidade por *subcluster*, e também muito similar ao CTSP já que dentro de cada *cluster* todas as cidades devem ser visitadas consecutivamente. Os autores descrevem uma aplicação real de logística modelada para o CGTSP e propõe uma técnica de transformação para resolvê-lo.

É importante não interpretar o problema do Caixeiro Viajante Clusterizado com Tabu (TCTSP) como o Problema do Caixeiro Viajante com Restrição de Tempo, (originalmente Time Constrained Traveling Salesman Problem), pois também é abreviado como TCTSP na literatura.

3.2 Programação Inteira (PI)

Pesquisa Operacional é definida como área de aplicação de modelos analíticos avançados, incluindo modelos matemáticos, com o intuito de melhorar as tomadas de decisões (ZACHARIAH et al., 2009). A Programação Linear Inteira é um ramo da Pesquisa Operacional em que são estudados modelos compostos apenas por funções lineares, com

variáveis de decisões inteiras.

Por se tratar de um método exato, a utilização de técnicas de Programação Inteira possibilitam encontrar a solução ótima dos problemas. Entretanto, a garantia de encontrar a melhor solução pode resultar num tempo de execução elevado que pode crescer de forma exponencial em relação ao tamanho da entrada. Por este motivo, trabalhos como (MESTRIA; OCHI; MARTINS, 2013) optaram por utilizar métodos inteiros na busca por limites inferiores, mesmo que isso seja possível apenas para um subconjunto das instâncias, para validar a qualidade de outras técnicas, como as heurísticas por exemplo que não garantem a otimalidade da solução obtida.

3.3 Meta-heurísticas

Como discutido na Seção 3.2, o tempo para encontrar a solução ótima pode ser inviável ou até mesmo intratável dependendo do tamanho da entrada. Uma opção prática para lidar com este problema de tempo é a utilização de heurísticas, que são técnicas que não garantem a otimalidade, entretanto possuem tempo de execução polinomial.

Meta-heurística é um *framework* de alto nível que independe do problema e provê um conjunto de diretrizes ou estratégias para desenvolver algoritmos de otimização heurísticos (SÖRENSEN; GLOVER, 2013). Utilizar uma meta-heurística para um problema específico fornece um certo nível de segurança, visto que a técnica já foi utilizada em distintos problemas com as mais diversas características. As subseções seguintes detalham algumas opções de meta-heurísticas adotadas para tratar os problemas abordados neste trabalho.

3.3.1 Reactive Greedy Random Adaptive Search Procedure

O *Greedy Random Adaptive Search Procedure* (GRASP) (FEO; RESENDE, 1995), ou Algoritmo de Busca Guloso Randomizado Adaptativo, trata-se de um método iterativo simples que alterna entre um algoritmo construtivo, que encontra novas soluções, seguido de uma busca local, utilizada para melhorar a solução encontrada previamente. Por fim, a solução da iteração é comparada com a melhor solução encontrada dentre as iterações

anteriores, e a substitui em caso de melhora.

As características Guloso Randomizado e Adaptativo devem estar presentes no algoritmo construtivo. Um algoritmo é dito guloso quando, durante o processo de construção de uma solução, as decisões são tomadas escolhendo a melhor opção considerando um critério de ranqueamento. Um algoritmo é guloso randomizado quando, no processo de escolha, há possibilidade de seleção para os melhores candidatos, não restringindo a escolha da melhor opção. Esta estratégia torna possível a geração de diferentes soluções a partir da execução de um mesmo algoritmo (TALBI, 2009). Por fim, a característica adaptativa está presente quando o critério de ranqueamento leva em conta as escolhas que foram feitas anteriormente no algoritmo.

A meta-heurística GRASP, proposta em (FEO; RESENDE, 1995) é amplamente utilizada na resolução de vários problemas de otimização distintos (RESENDE; RIBEIRO, 2014; FESTA; RESENDE, 2009). O Algoritmo 1 é um protótipo do funcionamento geral da meta-heurística. O parâmetro α indica o grau de aleatoriedade do construtivo, e a solução final S^* será a melhor entre as soluções obtidas durante todo o processo de busca.

Algoritmo 1: Pseudocódigo GRASP

Input : α
Output: $S*$

```
1 while criterio nao satisfeito do
2     S ← construtivo(α)
3     S ← buscaLocal(S)
4     if custo(S) < custo(S*) then
5         S* ← S
6     end if
7 end while
```

O GRASP Reativo, ou *Reactive* GRASP (RESENDE; RIBEIRO, 2010), é uma variação do GRASP clássico apresentado anteriormente. Nesta variante do GRASP, o valor de α não é fixo durante a execução. Neste caso, é utilizada uma lista de valores candidatos para α, com uma probabilidade associada a cada valor. Usando o GRASP Reativo, o algoritmo identifica durante a execução, de forma automática bons valores de α. Inicialmente, todas os valores presentes na lista de valores possíveis para α possuem a mesma probabilidade de seleção, mas esta probabilidade é ajustada ao longo das iterações, considerando o desempenho de cada α. Desta forma, existe uma tendência a valores α

que encontram melhores soluções serem escolhidos mais vezes durante a execução do algoritmo. O Algoritmo 2 é um protótipo do funcionamento geral da meta-heurística. Os detalhes sobre os ajustes das probabilidades pode ser visto no trabalho de (RESENDE; RIBEIRO, 2010).

Algoritmo 2: Pesudocódigo Reactive GRASP

Input : $L = [\alpha_1,\ \alpha_2,\ ...,\ \alpha_n],\ \beta$
Output: S^*

```
1  p ← iniciarProbabilidades(1/n, 1/n, ..., 1/n)
2  while criterio nao satisfeito do
3      α ← selecionarAlfa(p)
4      S ← construtivo(α)
5      S ← buscaLocal(S)
6      if cost(S) < cost(S*) then
7          S* ← S
8      end if
9      if i multiplo de β then
10         atualizarProbabilidades(p)
11     end if
12 end while
```

3.3.2 Random Variable Neighborhood Descent

O método Variable Neighborhood Descent (VND), ou Descida em vizinhança Variável (TALBI, 2009), se baseia no fato de movimentos distintos geralmente possuírem ótimos locais distintos. Desta forma, o método utiliza uma lista de estruturas de vizinhanças em uma ordem pré determinada. As vizinhanças são exploradas na ordem definida e, se alguma estrutura consegue encontrar uma solução melhor, o algoritmo recomeça utilizando a primeira estrutura da lista, caso contrário a próxima estrutura é utilizada.

Como mencionado anteriormente, o algoritmo exige que seja determinada a ordem de visitação das vizinhanças. A ordem influencia o resultado final obtido e determinar a ordem ideal não é uma tarefa trivial. Uma forma de evitar essa situação é utilizar uma variação do VND clássico, denominado por Random Variable Neighborhood Descent (RVND), ou Descida em vizinhança Variável Aleatória, que está presente na literatura aplicada em outros problemas (SUBRAMANIAN et al., 2010), (HADDAD et al., 2014). Nesta variação, a visitação das vizinhanças é determinada de forma aleatória. O Algo-

ritmo 3 ilustra este funcionamento em detalhes, onde há um conjunto $L = \{l_1 \dots l_k\}$ de vizinhanças que serão exploradas em diferentes ordens a cada chamada da função.

Algoritmo 3: Pseudocódigo RVND

```
   Input  : S
   Output: S*
 1 ordem ← embaralhar([l_1, ..., l_k])
 2 i ← 1
 3 while i ≤ k do
 4     movimento(s, ordem[i])
 5     i ← i + 1
 6     if melhorou then
 7         S* ← S
 8         i ← 1
 9     end if
10 end while
```

3.3.3 Iterated Local Search

A meta-heurística Iterated Local Search (ILS) presente em (LOURENÇO; MARTIN; STÜTZLE, 2003) intercala diversificação e intensificação durante o processo de busca no espaço de solução. A diversificação é feita utilizando o conceito chamado de perturbação, que modifica parte da solução corrente para então executar a intensificação, que aplica busca local na solução modificada. Desta forma, existe uma tendencia a evitar a estagnação em um ótimo local, ou seja, possibilitando a exploração de novos pontos no espaço de busca.

De forma mais detalhada, o algoritmo começa com a geração de uma solução inicial, seguido da aplicação da busca local, em seguida entra num processo iterativo envolvendo perturbação da solução, aplicação da busca local e a avaliação da solução encontrada utilizando um critério de aceitação. Todo este processo pode ser observado no Algoritmo 4.

Algoritmo 4: Pseudocódigo ILS

```
   Input  : S
   Output: S*
1  S ← solucaoInicial()
2  S ← buscaLocal()
3  while nao satisfeito do
4      S ← perturbacao(S)
5      S ← buscaLocal(S)
6      S' ← criterioAceitacao(S, S')
7      S ← S'
8  end while
```

4 Abordagens Propostas

Neste capítulo são definidos em detalhes como as técnicas apresentadas na seção anterior foram desenvolvidas, e quais decisões de projeto foram tomadas para tratar os problemas abordados neste trabalho. Este trabalho conta com 4 técnicas distintas que utilizam componentes em comum. Nas subseções seguintes esses componentes são abordados de forma independente. Para facilitar o entendimento de onde cada um destes componentes são empregados, a Figura 4.1 apresenta uma visão em alto nível de cada uma das técnicas, listando os componentes desenvolvidos neste trabalho juntamente dos métodos que os utilizam, separados por problema (TCTSP e CTSP).

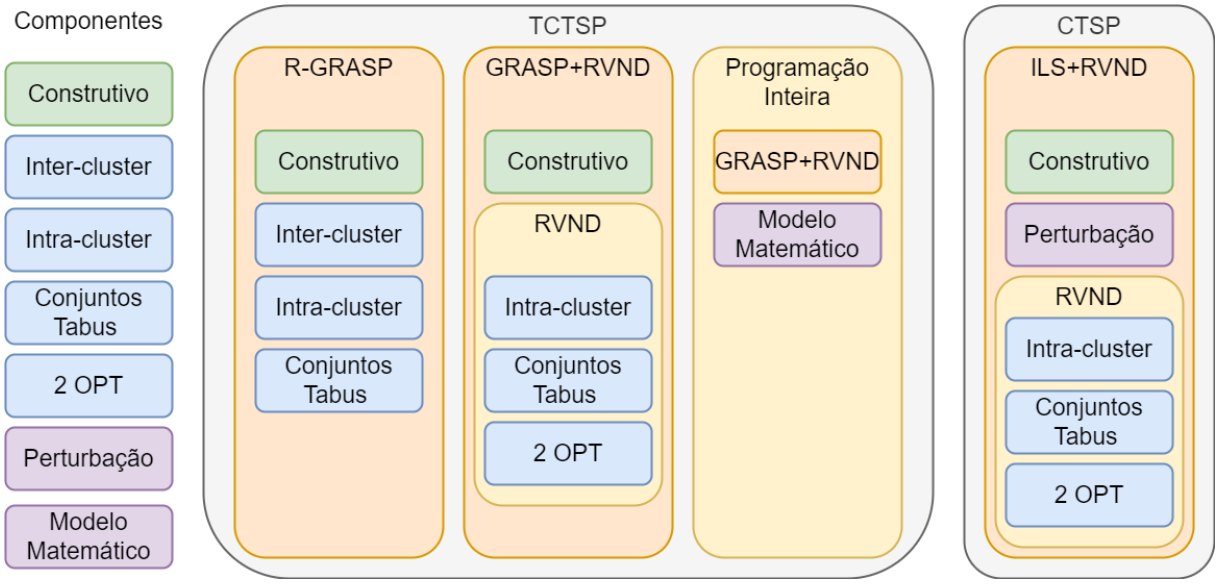

Figura 4.1: Visão geral dos componentes e como são utilizados

4.1 Algoritmo construtivo

Descrito originalmente em (SILVA JUNIOR et al., 2019), o algoritmo construtivo consiste em duas etapas: Criação de rotas intra-*cluster* e a união dessas rotas.

Na primeira etapa, para cada *cluster*, um subciclo é criado, utilizando um nó de cada tabu que intersecta com o *cluster* em questão. A rota é construída utilizando a heurística clássica da literatura chamada inserção mais barata (KINDERVATER; LENS-

TRA; SHMOYS, 1989). A heurística consiste em iniciar com um subciclo mínimo e, a cada iteração quebrar uma aresta e inserir um nó entre os dois nós das arestas. O processo completo está apresentado no pseudocódigo do Algoritmo 5. É importante ressaltar que o resultado da primeira etapa não produz uma solução válida, visto que vários subciclos estão presentes, além disso, a restrição de visitar exatamente 1 nó por tabu não é respeitada. A Figura 4.2 ilustra um resultado produzido pela primeira etapa, contendo 1 ciclo para cada *cluster*.

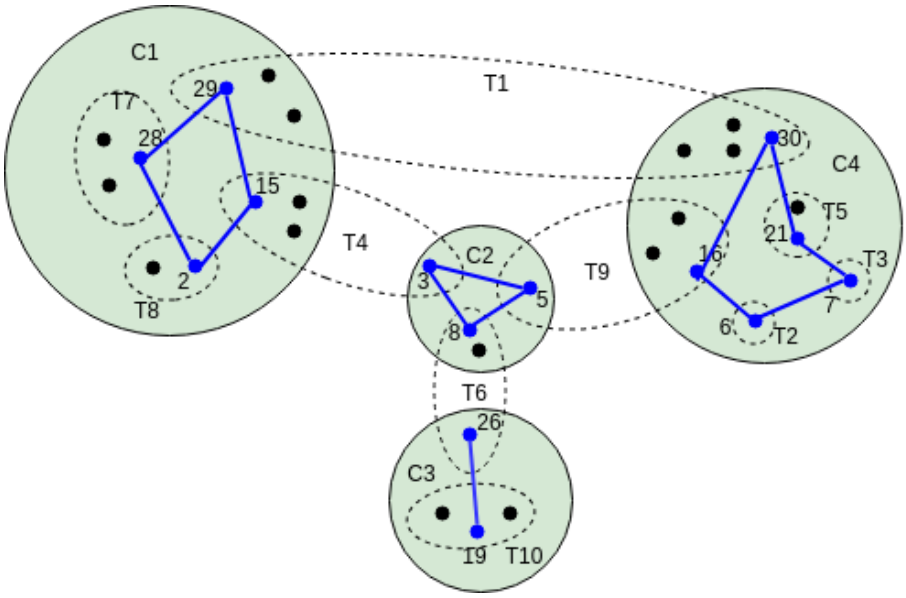

Figura 4.2: Criação de subciclos dos *clusters* (SILVA JUNIOR et al., 2019).

Algoritmo 5: Criação da lista de rotas

Input : I, α
Output: $listaRotas$

```
1  listaRotas ← ∅
2  foreach C_i ∈ V do
3      rota ← menorAresta(C_i)
4      T ← intersecaoTabuCluster(C_i)
5      LT ← ordenar(T)
6      while LT ≠ ∅ do
7          t ← tabuAleatorio(LT, α)
8          insercaoMaisBarata(route, t)
9          LT ← LT − {t}
10     end while
11     listaRotas ← listaRotas ∪ rota
12 end foreach
```

Na segunda etapa do construtivo, apresentado no Algoritmo 6, a aresta de maior

custo de cada ciclo é quebrada para que seja feita a união entre os subciclos, criando assim as denominadas rotas para cada *cluster*. Inicialmente, considera-se que nenhuma das rotas está incluída na solução (solução vazia), e a cada iteração uma dessas rotas é adicionada utilizando a heurística de inserção mais barata. Desta forma, a consecutividade entre os *clusters* não é violada. Após a inclusão de uma rota, as demais que ainda não foram incluídas na solução são atualizadas, isto é feito removendo-se os clientes de tabus que tiveram algum de seus clientes visitados. A Figura 4.3 ilustra todo o processo construtivo, inserindo-se as rotas de C_4, C_1, C_3, nesta ordem. Perceba que a atualização das rotas a cada inserção resulta na não inclusão dos nós do *cluster* de C_2, já que seus tabus tiveram clientes visitados em outros *clusters*.

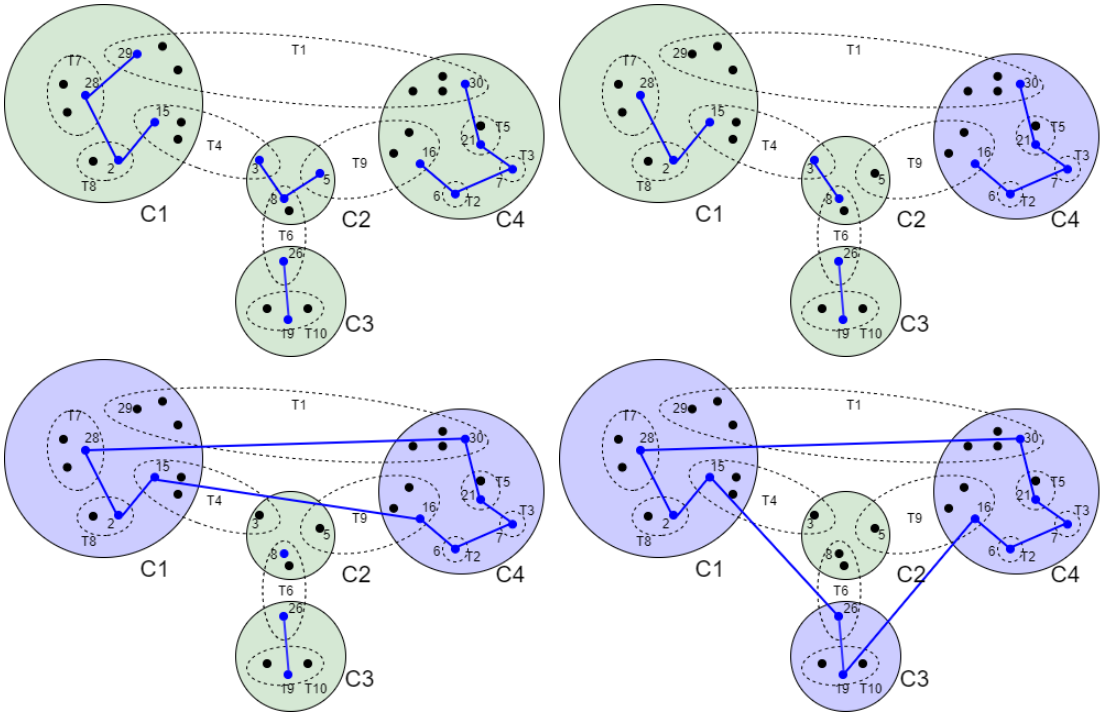

Figura 4.3: Processo construtivo da segunda etapa.

4.2 Busca Local Inter-*cluster*

Esta estratégia de refinamento foi descrita inicialmente em (SILVA JUNIOR et al., 2019). Dadas as características do algoritmo construtivo, pode ser interessante melhorar os trechos da solução próximos a conexão entre diferentes *clusters*. Esta estratégia de busca foi desenvolvida com este propósito.

A busca local inter-*cluster* consiste em remover uma aresta inter-*cluster* e algumas

Algoritmo 6: Estratégia para combinar as rotas geradas na etapa inicial

Input : I, α
Output: S

```
1 L ← criarListaRotas(I, α)
2 ordenar(L)
3 S ← ∅
4 while L ≠ ∅ do
5     rota ← rotaAleatoria(L, α)
6     insercaoMaisBarata(S, quebrarCiclo(rota))
7     L ← atualizarRotas(L, rota)
8 end while
```

das arestas adjacentes a ela, e então reconstruir a solução com o auxilio do algoritmo de *Dijkstra*. O objetivo dessa busca é melhorar o custo da área próxima à conexão entre dois *clusters* consecutivos, dado que as arestas inter-*cluster* tem valor diferenciado afetando mais significativamente o custo da solução.

Seja $S = (s_1, s_2, ..., s_p, s_{p+1}, ..., s_m)$ a representação de uma solução viável do TCTSP, com o ciclo de arestas $\bigcup_{i=1}^{m-1}(s_i, s_{i+1}) \cup (s_m, s_1)$. Cada iteração da busca local inicia com uma aresta inter-*cluster* (s_p, s_{p+1}).

Para um dado γ, o algoritmo remove da solução a aresta (s_p, s_{p+1}) além de γ arestas antecessoras e γ sucessoras a ela, i.e., remove cada aresta no caminho na solução entre os nós $s_{p-\gamma}$ e $s_{p+1+\gamma}$. A linha pontilhada vermelha na Figura 4.4 ilusta uma tentativa de otimizar a conexão entre os *clusters* C_3 e C_4, removendo a aresta $(19, 16)$ e $\gamma = 3$ arestas de cada direção.

Para reconstruir a o ciclo da solução, um grafo auxiliar é criado. Seja $\bar{G} = (\bar{V}, \bar{E})$

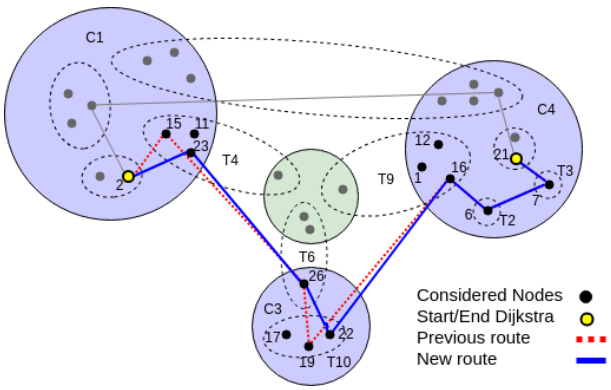

Figura 4.4: Iteração da Busca Local Inter-*cluster* melhorando conexão entre os *clusters* C_3 e C_4. Aresta (19,16) e $\gamma = 3$ arestas adicionais de cada lado são removidas e substituidas pelo caminho de menor custo de 2 ao 21.

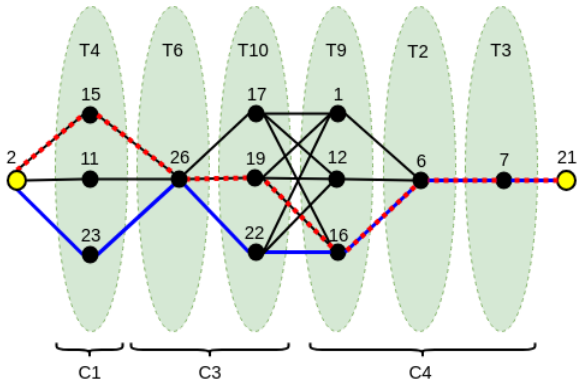

Figura 4.5: Grafo $\bar{G}$ correspondente a Figura 4.4.

a representação auxiliar de um grafo $(k+2)$-particionado, com $k = 2 \times \gamma$. Cada tabu no caminho de $s_{p-\gamma}$ até $s_{p+1+\gamma}$ gera uma partição em $\bar{V}$. A primeira e última partição ($\bar{V}_{p-\gamma}$ e $\bar{V}_{p+1+\gamma}$) são subconjuntos unitários com nós $s_{p-\gamma}$ e $s_{p+1+\gamma}$, respectivamente. Para cada nó s_i, $i = p-\gamma+1, \ldots, p+\gamma$, o conjunto $\bar{V}_i$ contém todos os nós que pertencem ao mesmo *cluster* e ao mesmo conjunto tabu que s_i. O conjunto de arestas consiste apenas em arestas de uma partição para a seguinte, i.e., $\bar{E} = \{(v_r, v_s) | v_r, v_s \in V, v_r \in \bar{V}_i, v_s \in \bar{V}_{i+1}, i = p-\gamma, \ldots, p+\gamma\}$.

Baseado no exemplo da Figura 4.4, o grafo auxiliar $\bar{G}$ é retratado na Figura 4.5. Percebe-se que as partições de $\bar{G}$ correspondem aos conjuntos tabus T_4, T_{10}, e T_9, contendo um nó do caminho da solução (15, 19, e 16, respectivamente) e os outros nós no mesmo conjunto tabu e no mesmo *cluster*.

Uma vez que o grafo $\bar{G}$ é construído, o algoritmo de *Dijkstra* é executado para encontrar o caminho mais curto de $s_{p-\gamma}$ até $s_{(p+1+\gamma)}$, o que pode alterar o caminho original entre estes nós. A linha azul na Figura 4.4 destaca o resultado obtido pelo algoritmo de *Dijkstra*.

O Algoritmo 7 detalha a busca local Inter-*cluster*. O parâmetro γ é definido com o valor 3 para evitar uma quantidade excessiva de permutações. Dada uma solução de entrada S, as arestas inter-*cluster* são selecionadas (em uma ordem particular) e o algoritmo computa todas as permutações viáveis (linha 5). Para cada permutação, constrói um grafo auxiliar $\bar{G}$ e encontra o caminho mais curto (linhas 7 e 8), dando origem a uma nova solução S_{new}. Se o custo é melhorado, a melhor solução é atualizada (linha 10).

Algoritmo 7: Busca Local Inter-cluster

Input : S
Output: S*

```
1  S* ← S
2  γ ← 3
3  foreach aresta (s_p, s_{p+1}) ∈ S do
4      if cluster(s_p) ≠ cluster(s_{p+1}) then
5          P ← enumerarPermutacoes(S, p, γ)
6          foreach S' ∈ P do
7              Ḡ ← construirGrafo(S', p, γ)
8              S_new ← dijkstra(S', Ḡ)
9              if custo(S_new) < custo(S*) then
10                 S* ← S_new
11             end if
12         end foreach
13     end if
14 end foreach
```

4.3 Busca Local Intra-*cluster*

Como abordado anteriormente, a fase de construção consiste em criar ciclos intra-*cluster* para então, quebrar os ciclos e unir as rotas resultantes. A estratégia adotada para unir as rotas a partir da quebra das arestas de maior custo pode não gerar as melhores soluções.

A Busca Local Intra-*cluster*, descrita inicialmente em (SILVA JUNIOR et al., 2019), foca justamente em reverter essas características ruins produzidas pela etapa construtiva. A cada iteração, um *cluster* é selecionado aleatoriamente e seus nós são removidos da solução, e então inseridos novamente no ciclo utilizando a heurística de inserção mais barata (KINDERVATER; LENSTRA; SHMOYS, 1989), descrita na etapa de construção. Vale ressaltar que o procedimento não altera quais nós serão visitados naquele *cluster*, modificando somente a ordem de visitação da solução. A Figura 4.6 exemplifica o funcionamento do movimento aplicado a um *cluster*.

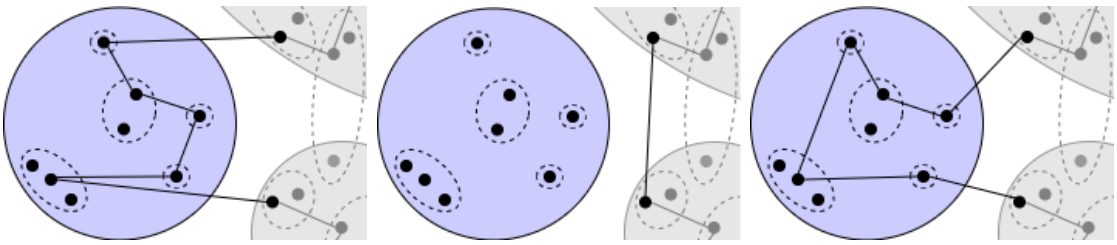

Figura 4.6: Exemplo de movimento presente na Busca Local intra-*cluster*

O Algoritmo 8 detalha a execução do procedimento. É iniciado um processo iterativo de remoção de cada *cluster* C_i entre as linhas 1 e 11. Os tabus que são visitados e possuem interseção com C_i são selecionados na linha 3 e em seguida todos os nós de C_i são removidos da solução parcial S' na linha 4. O algoritmo entra então no processo de reinserção dos tabus previamente removidos nas linhas 5 a 7 e por fim é avaliado se houve melhora da solução na reconstrução com a remoção do *cluster* C_i nas linhas 8 a 10.

Algoritmo 8: Busca Local Intra-cluster

Input : S
Output: S

```
1  foreach C_i ∈ S do
2      S' ← S
3      T ← tabusVisitadosNoCluster(S', C_i)
4      removerCluster(S', C_i)
5      foreach t ∈ T do
6          insercaoMaisBarata(S', t)
7      end foreach
8      if custo(S') < custo(S) then
9          S ← S'
10     end if
11 end foreach
```

4.4 Busca Local Conjuntos Tabus

Nenhuma das buscas locais anteriores permitem movimentos que mudam o *cluster* em que cada tabu é visitado. O Refinamento de Conjuntos Tabus, descrito originalmente em (SILVA JUNIOR et al., 2019), visa realizar movimentos para cobrir esse tipo de caso.

Nesta busca local, cada um dos tabus são avaliados como descrito no Algoritmo 9. A cada iteração o procedimento seleciona um conjunto tabu (esta seleção é feita de forma aleatória, Linha 1). Um segundo conjunto tabu, perto o suficiente do primeiro, também é selecionado para que ambos sejam removidos da solução (linha 3). Para verificar se um tabu está próximo o suficiente, utiliza-se um valor real $t = 0,3$ para calcular o número de tabus a diante do tabu i na solução. Por exemplo: considere $S = (s_1, ..., s_i, ..., s_{10})$ a solução para uma instância com 10 *tabus*, neste caso $10 \times 0,3 = 3$, então os 3 tabus posteriores a s_i na solução, $\{s_{i+1}, s_{i+2}, s_{i+3}\}$, são considerados tabus próximos de s_i.

Após a remoção dos dois tabus, o algoritmo entra em um processo iterativo (Linhas de 5 a 13) para buscar a melhor forma de reinserir o tabu T_i. Apenas os nós dos *clusters* que já estão sendo visitados são considerados (Linha 16), pois é prejudicial para a solução aumentar o número de *clusters* visitados. É realizado o processo da inserção mais barata com o nó da iteração (Linha 8). Na Linha 9 se garante que a melhor solução seja armazenada. Ao finalizar este processo, o mesmo é realizado levando em consideração a melhor solução encontrada no processo anterior para o tabu T_j (Linhas 15 a 23).

Algoritmo 9: Refinamento de Conjuntos Tabu

Input : S
Output: S

```
1  foreach T_i ∈ S do
2      T_j ← tabuProximo(T_i)
3      S' ← removerTabu(S', T_j)
4      S*_i ← S'
5      foreach v ∈ T_i do
6          S' ← removerTabu(S', T_i)
7          if clusterUtilizado(S', cluster(v)) then
8              insercaoMaisBarata(S', v)
9              if custo(S') < custo(S*_i) then
10                 S*_i ← S
11             end if
12         end if
13     end foreach
14     S' ← S*_i
15     foreach v ∈ T_j do
16         if clusterUtilizado(S', cluster(v)) then
17             insercaoMaisBarata(S', v)
18             if custo(S') < custo(S*_i) then
19                 S*_i ← S
20             end if
21             S' ← removerTabu(T_j)
22         end if
23     end foreach
24     S ← S*_i
25 end foreach
```

A Figura 4.7 exemplifica um movimento do refinamento. O primeiro tabu selecionado é o tabu destacado de vermelho, que possui interseção com os dois *clusters*. O segundo tabu selecionado é o destacado de azul e está contido no *cluster* inferior. Ambos são removidos da solução e são re-inseridos com a utilização do algoritmo de inserção mais

barata. Percebe-se que o algoritmo considera utilizar cidades pertencentes a um *cluster* diferente do que é visitado originalmente.

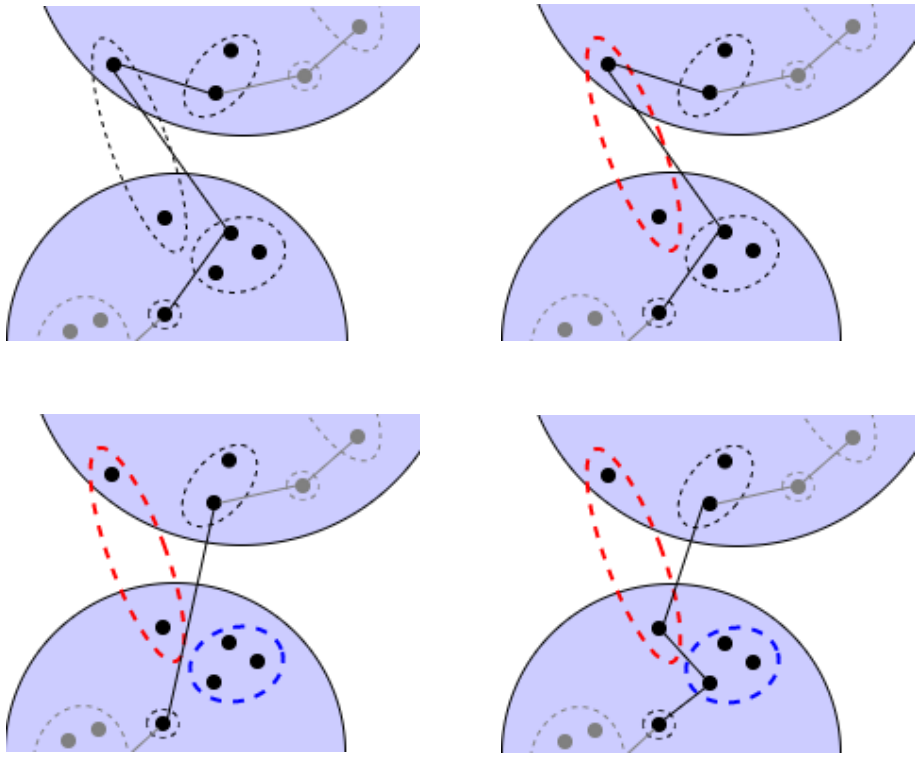

Figura 4.7: Exemplo de movimento presente no Refinamento de Conjuntos Tabus

4.5 2-OPT

A busca local 2-OPT é um clássico da literatura, vastamente utilizado em heurísticas para problemas de roteamento (YANG et al., 2008; THEYS et al., 2010). O método clássico quebra o ciclo em duas sequências de vértices e recria invertendo a ordem das aresta em uma sequência. Entretanto, nos problemas abordados neste trabalho o uso direto do 2-OPT pode causar inviabilidade por conta da restrição de visitação consecutiva dos *clusters*. Desta forma, o método foi adaptado para ser utilizado de forma restrita, envolvendo apenas vértices de um mesmo *cluster*.

O Algoritmo 10 detalha esse processo. O procedimento itera sobre os clusters (linha 2), na linha seguinte é selecionada a rota R a ser otimizada. As linhas 4 e 5 são responsáveis por verificar os movimentos envolvendo todos os pares de arestas de R. A Linha 6 verifica se a troca entre arestas possui valor inferior a solução original, caso

positivo a Linha 7 R^* armazena a melhor troca encontrada no momento atual, mas a solução só é alterada na linha 11, depois de verificar todas as possibilidades de troca.

Algoritmo 10: Busca Local 2-OPT

Input : S
Output: S*

```
1  while Melhorar do
2      foreach c ∈ C do
3          R ← subRota(S, c)
4          for i ← 1 to |V_c| − 1 do
5              for j ← i + 1 to |V_c| do
6                  if (r_i, r_{i+1}) + (r_j, r_{j+1}) > (r_i, r_j) + (r_{i+1}, r_{j+1}) then
7                      R* ← trocaDoisOpt(R, i, j)
8                  end if
9              end for
10         end for
11         S ← atualizarRota(S, R*)
12     end foreach
13 end while
```

4.6 R-GRASP para o TCTSP

A primeira heurística desenvolvida para o TCTSP é baseada na meta-heurística GRASP apresentada na subseção 3.3.1 e utiliza como componentes: algoritmo construtivo, as buscas intra-*cluster*, inter-*cluster* e refinamento de conjuntos tabus. Descrito originalmente em (SILVA JUNIOR et al., 2019).

O Algoritmo 11 apresenta detalhadamente o processo de execução. Os parâmetros de entrada são: I a instância do TCTSP; $L = [\alpha_1,\ \alpha_2,\ ...,\ \alpha_n]$ lista de valores para α; *iteracoes*, número de iterações do GRASP; β como o número de iterações antes de cada atualização das probabilidades.

A cada iteração (linhas 2 a 13), o algoritmo seleciona um valor α (linha 3), a etapa de construção cria uma solução S baseado no α escolhido (linha 4) e o procedimento das buscas locais melhora a solução S (linhas 5 a 7). Linhas de 8 a 10 garantem que S^* armazena a melhor solução. As probabilidades relacionadas à lista de α são atualizadas a cada β iterações (linhas 11 a 13). Por fim, o algoritmo retorna a melhor solução S^*.

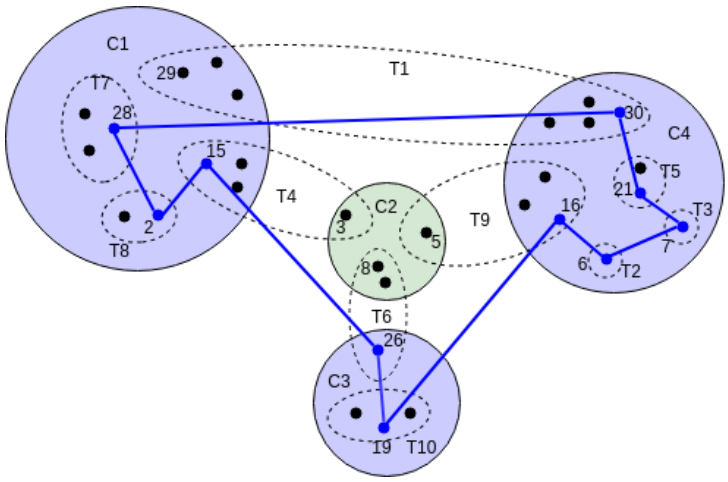

Figura 4.8: Solução viável para o TCTSP

Algoritmo 11: R-GRASP para o TCTSP

Input : I, $L = [\alpha_1,\ \alpha_2,\ ...,\ \alpha_n]$, $iteracoes$, β
Output: S^*

```
1  p ← iniciarProbabilidades(1/n, 1/n, ..., 1/n)
2  for i ← 1 to iteracoes do
3      α ← selecionarAlfa(p)
4      S ← gulosoRandomizado(α, I)
5      S ← intercluster(S)
6      S ← intracluster(S)
7      S ← tabusBL(S)
8      if cost(S) < cost(S*) then
9          S* ← S
10     end if
11     if i multiplo de β then
12         atualizarProbabilidades(p)
13     end if
14 end for
```

4.7 GRASP-RVND para o TCTSP

Esta heurística é uma versão adaptada da heurística da Subseção 4.6, e difere em relação a adição do parâmetro $itSemMelhora$, e o RVND como estratégia de melhora com os refinamentos 2-OPT, Conjuntos Tabus e Intra-*cluster* incluídos. O parâmetro $itSemMelhora$ é utilizado como segundo critério de parada para que o algoritmo termine se executar $itSemMelhora$ iterações consecutivas sem atualização da melhor solução S^*.

O Algoritmo 12 segue a mesma linha de funcionamento do Algoritmo 11. A estrutura de repetição principal do algoritmo na linha 4 utiliza o valor auxiliar j para controlar o número de iterações sem melhora. Para refinamento da solução, diferente do

GRASP apresentado na seção anterior, aqui as buscas são utilizadas conjuntamente em um RVND, linha 7.

Algoritmo 12: GRASP RVND para o TCTSP

Input : I, $L = [\alpha_1,\ \alpha_2,\ ...,\ \alpha_n]$, $iteracoes$, β, $itSemMelhora$
Output: S^*

```
1  p ← iniciarProbabilidades(1/n, 1/n, ..., 1/n)
2  i ← 1
3  j ← 1
4  while i ≤ iteracoes and j ≤ itSemMelhora do
5      α ← selecionarAlfa(p)
6      S ← gulosoRandomizado(α, I)
7      S ← rvnd(S)
8      if cost(S) < cost(S*) then
9          S* ← S
10         j ← 1
11     end if
12     if i multiplo de β then
13         atualizarProbabilidades(p)
14     end if
15     i ← i + 1
16     j ← j + 1
17 end while
```

4.8 ILS-RVND para o CTSP

Os bons resultados obtidos para o TCTSP com os componentes já apresentados são a principal motivação para o desenvolvimento do ILS para o CTSP. Como os algoritmos foram desenvolvidos originalmente para o TCTSP, é necessário adaptar as instâncias do CTSP, isto é feito considerando que existe um tabu para cada cidade presente originalmente na instância.

O ILS faz uso dos seguintes componentes: Algoritmo construtivo para a geração da solução inicial requerida pelo ILS e o RVND utilizando os refinamentos intra-*cluster*, refinamento de conjuntos tabus e 2-OPT. Perceba que a única busca não incluída é a inter-*cluster*, e isso se deve a natureza do funcionamento da busca em relação ao problema do CTSP, visto que a busca visa encontrar a melhor cidade dentre as que estão presentes em um dado conjunto tabu, e como todos os tabus possuem uma única cidade, não teria efeito neste contexto.

Além dos componentes já citados, o ILS traz o algoritmo de perturbação, combinado com uma estratégia que emprega taxa de perturbação crescente. A taxa é um valor real entre 0 e 1 utilizado para definir qual a porcentagem da solução que será perturbada durante o algoritmo. Essa taxa crescente se justifica pela dificuldade de utilizar um único valor fixo que seja adequado para cada uma das instâncias.

O algoritmo de perturbação é uma versão maior da busca intra-*cluster*. A perturbação é feita com a remoção de múltiplos *clusters* simultaneamente da solução, e não de apenas 1 *cluster* como é na busca local. O número de *clusters* removidos é ditado pela taxa de remoção abordada anteriormente, e.g., uma instância com 20 *clusters* e com a taxa de remoção de 20% terá 4 *clusters* removidos da solução inicial. Além disso, o número de *clusters* removido tem um valor mínimo de 3, esse valor é definido para que sempre exista a oportunidade de mudança na ordem de visitação dos *clusters*. Após a remoção dos *clusters*, a rota é reconstruída utilizando a inserção mais barata considerando sempre todos os nós que ainda não foram re-inseridos na solução.

Algoritmo 13: ILS RVND para o CTSP

Input : I, $itSemMelhora$, $taxa$, $taxaMax$, $taxaCrescimento$
Output: S^*

```
1  S ← gulosoRandomizado(1.0)
2  i ← 1
3  while taxa ≤ taxaMax do
4      S ← perturbacao(taxa)
5      S ← rvnd(S)
6      if custo(S) < custo(S*) then
7          S* ← S
8          i ← 1
9      end if
10     else
11         i ← 1 + 1
12         if i > itSemMelhora then
13             taxa ← taxa + taxaCrescimento
14         end if
15     end if
16 end while
```

4.9 Programação Inteira para o TCTSP

As heurísticas desenvolvidas neste trabalho para o TCTSP convergem rapidamente para resultados iguais ou melhores aos apresentados na literatura. Entretanto, apesar da existência do modelo proposto por (ZHANG et al., 2018) o mesmo não foi utilizado na prática, ou seja, não existem resultados originados de métodos exatos para resolver o TCTSP. Por consequência, a avaliação das heurísticas é prejudicada, visto não ser possível medir a distância das heurísticas e verificar possibilidade de melhora dos resultados obtidos.

As próximas seções têm como objetivo definir modelos que possam ser utilizados na prática para a resolução do problema, de todas ou ao menos parte das instâncias presentes na literatura. Para definir o modelo, é possível utilizar como referência trabalhos anteriores de problemas similares ao TCTSP, como, por exemplo, (KARA; GUDEN; KOC, 2012) e (MESTRIA; OCHI; MARTINS, 2013), que abordam os problemas GTSP e o CTSP. A seção a seguir apresenta os desafios presentes no desenvolvimento de modelos para problemas variantes do TSP clássico.

4.9.1 Problema do subciclo

O Problema do Caixeiro viajante pode ser considerado um problema fácil de se descrever, e, a princípio, igualmente fácil de se propor um modelo. Considere uma proposta inicial de modelo (LANGEVIN; SOUMIS; DESROSIERS, 1990) como apresentado a seguir onde x_{ij} é a variável de decisão que assume valor 1 se o caixeiro percorre a aresta da cidade i para a cidade j, 0 caso contrário, e c_{ij} é o custo associado à aresta ij:

$$min\ z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_{ij} \tag{4.1}$$

$$\sum_{i \in V, i \neq j} x_{ij} = 1 \qquad \forall j \in V \tag{4.2}$$

$$\sum_{j \in V, j \neq i} x_{ij} = 1 \qquad \forall i \in V \tag{4.3}$$

$$\sum_{j \in V, j \neq i} x_{ji} - \sum_{j \in V, j \neq i} x_{ij} = 0 \qquad \forall i \in V \tag{4.4}$$

Neste modelo, as Restrições 4.2 e 4.3 garantem que cada cidade seja visitada exatamente uma vez, já as Restrições 4.4 garantem que a aresta de entrada e a aresta de saída estão relacionadas ao mesmo nó. Este modelo garante a chegada e partida do caixeiro em cada cidade, entretanto não é o suficiente para a resolução do problema. Neste modelo é possível obter soluções contendo subciclos, i.e., ciclos com o número de cidades visitadas menor que o total de cidades da instância. A Figura 4.9 ilustra um caso de solução inviável devido à existência de subciclos.

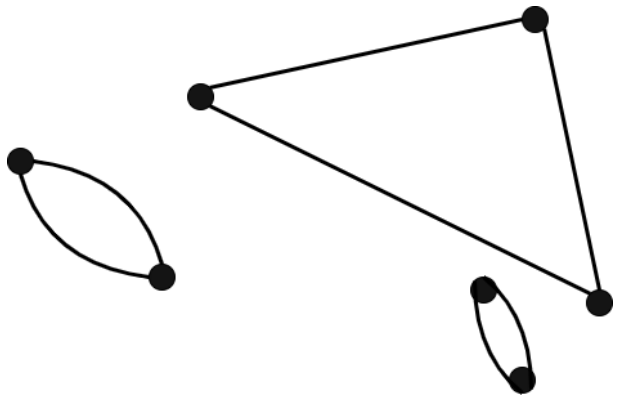

Figura 4.9: Exemplo de subciclos em uma instância do TSP

Para evitar a ocorrência de subciclos, uma alternativa é adicionar o seguinte conjunto de restrições:

$$\sum_{i \in Q} \sum_{j \in Q, j \neq i} x_{ij} \leq |Q| - 1 \qquad \forall Q \subsetneq \{1, \ldots, n\}, |Q| \geq 2 \tag{4.5}$$

Este conjunto de restrições garante que não exista nenhum ciclo com tamanho menor que o número de cidades, desta forma o ciclo final gerado é necessariamente uma solução para o problema. O problema prático presente na utilização deste modelo está na quantidade de restrições necessárias. Enquanto as restrições 4.2 e 4.3 crescem linearmente

em relação ao tamanho da entrada, a quantidade de Restrições 4.5 cresce exponencialmente, que torna o modelo impraticável para instâncias significativamente grandes. O desafio para resolver o TSP (e suas variações, como o TCTSP) utilizando os métodos exatos torna-se propor um modelo que não cresça exponencialmente, ou que de alguma forma não utilize todas as restrições na prática.

4.9.2 Modelo inicial

Nesta proposta de modelo, as restrições para tratar o problema do subciclo são inspiradas por modelos existentes na literatura para variações do TSP (KARA; GUDEN; KOC, 2012; MESTRIA; OCHI; MARTINS, 2013) referente aos problemas GTSP e CTSP, respectivamente.

Dados de Entrada

V Conjuntos de nós

C_k Conjuntos dos clusters

T_m Conjuntos dos tabus

c_{ij} Custo da aresta do nó i para o nó j

Variáveis

x_{ij} 1 se existe aresta do nó i para o nó j. 0 caso contrário

w_{pq} 1 se existe aresta do tabu p para o tabu q. 0 caso contrário

u_p Variável auxiliar - Define a ordem de visitação dos tabus e evita subciclos

$$min\ z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} \cdot x_{ij} \tag{4.6}$$

$$\sum_{i\in T_p}\sum_{j\in V\setminus T_p} x_{ij} = 1 \qquad p = 1\ ,...,\ m \tag{4.7}$$

$$\sum_{i\in V\setminus T_p}\sum_{j\in T_p} x_{ij} = 1 \qquad p = 1\ ,...,\ m \tag{4.8}$$

$$\sum_{j\in V\setminus\{i\}} x_{ji} - \sum_{j\in V\setminus\{i\}} x_{ij} = 0 \qquad \forall i \in V \tag{4.9}$$

$$\sum_{i\in T_p}\sum_{j\in T_q} x_{ij} = w_{pq} \qquad p \neq q; p, q = 1\ ,...,\ m \tag{4.10}$$

$$u_p - u_q + (m-1)w_{pq} + (m-3)w_{qp} \leq m-2 \qquad p \neq q; p, q = 2\ ,...,\ m \tag{4.11}$$

$$u_p - \sum_{\substack{q=2\\q\neq p}}^{k} w_{qp} \geq 1 \qquad p = 2\ ,...,\ m \tag{4.12}$$

$$u_p + (m-2)w_{1p} \leq m-1 \qquad p = 2\ ,...,\ m \tag{4.13}$$

$$\sum_{i\in C_r}\sum_{j\in V\setminus C_r} x_{ij} \leq 1 \qquad r = 1\ ,...,\ k \tag{4.14}$$

$$\sum_{i\in V\setminus C_r}\sum_{j\in C_r} x_{ij} \leq 1 \qquad r = 1\ ,...,\ k \tag{4.15}$$

$$\sum_{i\in C_r}\sum_{j\in V\setminus C_r} x_{ij} - \sum_{i\in V\setminus C_r}\sum_{j\in C_r} x_{ij} = 0 \qquad r = 1\ ,...,\ k \tag{4.16}$$

As equações 4.7 e 4.8 limitam o número de arestas de entrada e saída em um dado tabu = 1; A equação 4.9 define que a aresta de entrada e a aresta de saída devem estar relacionadas ao mesmo nó. As restrições 4.10 definem o valor de $w_{pq} = 1$ se existe uma aresta do tabu p para o tabu q; Restrições 4.12 e 4.13 são os limites superiores e inferiores das variáveis u_p; A restrição 4.11 elimina os subciclos com a ajuda das variáveis u_p (o valor assumido pela variável u_p é a posição do tabu p na solução. Exemplo: se $u_4 = 10$, significa que o tabu 4 foi o décimo tabu visitado); Equações 4.14, 4.15 e 4.16 garantem a consecutividade dos *clusters*; As restrições 4.14 e 4.15 limitam a no máximo uma aresta de entrada e uma de saída em um dado *cluster*, ou seja, cumprem o mesmo papel que as restrições 4.7 e 4.8 a nível de *cluster*; A restrição 4.16 define que se existe uma aresta de entrada deve haver uma aresta de saída no *cluster* em questão (desta forma, é possível

que um *cluster* não seja visitado).

4.9.3 *Lazy Constraints*

Apesar do número de restrições crescer exponencialmente no modelo inicial (restrições de 4.1 a 4.5), apenas um subconjunto dessas restrições são utilizadas na prática. A técnica de utilizar as *Lazy Constraints* torna praticável utilizar este modelo.

Ao utilizar *Lazy Constraints*, as restrições são adicionadas por meio do resolvedor somente quando necessárias. O resolvedor verifica a viabilidade das soluções inteiras encontradas constantemente. Utilizando as *Lazy Constraints*, caso a solução seja inviável, as restrições violadas são adicionadas de forma dinâmica. Dessa forma, restrições não violadas não são adicionadas e o total de restrições consideradas pelo resolvedor é muito menor.

O modelo utilizado é uma adaptação do modelo inicial definido na Seção 4.9.2, onde todas as restrições são utilizadas, com exceção das restrições 4.11, 4.12 e 4.13. Essas restrições são responsáveis por definir a ordem de visitação dos tabus, que por sua vez resolvem o problema do subciclo e não são necessárias nessa versão. Além dessas restrições originadas do modelo inicial, o resolvedor adiciona dinamicamente a restrição 4.5 quando uma solução inteira é encontrada e a restrição é violada.

5 Experimentos Computacionais

Este capítulo descreve os procedimentos, tecnologias, instâncias e parâmetros utilizados para realizar os testes dos algoritmos descritos anteriormente. A seção seguinte define quais instâncias foram utilizadas para cada problema, bem como a origem de cada uma delas.

5.1 Instâncias

O conjunto de instâncias utilizado para testar os algoritmos para o Problema do Caixeiro Viajante *Clusterizado* com conjuntos Tabu (TCTSP) é o mesmo utilizado por (ZHANG et al., 2018). Esse conjunto consiste em dois subconjuntos, onde o conjunto 1 possui instâncias com o número de nós em cada tabu sendo exatamente 1, visto que são instâncias originalmente do CTSP. O conjunto 2 possui o número de nós para cada tabu maior ou igual a 1, e são instâncias específicas para o TCTSP. As instâncias são descritas na Tabela 5.1, onde: *ID* é o identificador da instância; $\#vertices$ indica o número de vértices no grafo; $\#T$ é o número de conjuntos tabu; e $\#C$ é o número de *clusters*.

O conjunto de instâncias utilizado para testar o algoritmo para o Problema do Caixeiro Viajante *Clusterizado* (CTSP) é o mesmo utilizado pelos últimos trabalhos relacionados ao problema (MESTRIA, 2018; LU; HAO; WU, 2020). O trabalho (MESTRIA, 2018) apresenta os resultados para 3 conjuntos de instâncias: as instâncias de tamanho pequeno contando com 27 instâncias descritas na Tabela 5.2, 20 instâncias de tamanho médio descritas na Tabela 5.3 e 15 de tamanho grande descritas na Tabela 5.4.

5.2 Ambiente de testes

Os algoritmos apresentados foram implementados utilizando a linguagem C++ e os experimentos foram executados utilizando o sistema operacional Linux Ubuntu com o processador Intel Core i5, 8GB de memória RAM, executando a 2.2 GHz. O CPLEX foi

Nome da instância	**ID**	**#vertices**	$\#T$	$\#C$	$\#Conjunto$
10-lin318-1	I_1	318	318	10	1
10-pcb442-1	I_2	442	442	10	1
25-eil101-1	I_3	101	101	25	1
144-rat783-1	I_4	783	783	144	1
300-20-111-1	I_5	300	300	20	1
500-25-308-1	I_6	500	500	25	1
300-6-1	I_7	300	300	6	1
700-20-1	I_8	700	700	20	1
C1k-0-1	I_9	1000	1000	10	1
C1k-1-1	I_{10}	1000	1000	10	1
200-4-h-1	I_{11}	200	200	4	1
600-8-z-1	I_{12}	600	600	8	1
10-lin318-64-2	I_{13}	318	64	10	2
9-st70-14-2	I_{14}	70	14	9	2
10-kroB100-20-2	I_{15}	100	20	10	2
10-pcb442-89-2	I_{16}	442	89	10	2
10-pr439-88-2	I_{17}	439	88	10	2
25-eil101-21-2	I_{18}	101	21	25	2
25-gil262-53-2	I_{19}	262	53	25	2
25-kroA100-20-2	I_{20}	100	20	25	2
25-lin105-21-2	I_{21}	105	21	25	2
25-rat99-20-2	I_{22}	99	20	25	2

Tabela 5.1: Descrição das instâncias TCTSP

utilizado como resolvedor exato na sua versão 12.8.

5.3 Execução

Devido ao fator estocástico presente nos algoritmos, cada heurística implementada foi executada 30 vezes por instância. O parágrafo seguinte detalha as escolhas de valores para cada parâmetro de cada algoritmo.

Os parâmetros para o R-GRASP foram definidos de forma empírica como segue: $L = [0.1, 0.2, \ldots, 1.0]$ lista de valores para α, o número de iterações $iteracoes = 10000$, e a taxa de atualização da probabilidade de escolha de cada valor $\beta = 1000$ iterações.

Diferente do R-GRASP, os parâmetros para o GRASP-RVND foram definidos com a utilização do *Irace* (LóPEZ-IBáñEZ et al., 2016). O *Irace* ajusta de forma automática as configurações necessárias para a execução do algoritmo e possui sucesso em vários trabalhos presentes na literatura (SAMA et al., 2016) (BALI et al., 2019). Os parâmetros ajustados para o GRASP-RVND foram $iteracoes$ e $itSemMelhora$, e a faixa

Nome da instância	**#vertices**	**#C**
5-eil51	51	5
10-eil51	51	10
15-eil51	51	15
5-berlin52	52	5
10-berlin52	52	10
15-berlin52	52	15
5-st70	70	5
10-st70	70	10
15-st70	70	15
5-eil76	76	5
10-eil76	76	10
15-eil76	76	15
5-pr76	76	5
10-pr76	76	10
15-pr76	76	15
10-rat99	99	10
25-rat99	99	25
50-rat99	99	50
25-kroA100	100	25
50-kroA100	100	50
10-kroB100	100	10
50-kroB100	100	50
25-eil101	101	25
50-eil101	101	50
25-lin105	105	25
50-lin105	105	50
75-lin105	105	75

Tabela 5.2: Conjunto de Instâncias pequenas CTSP

Nome da instância	**#vertices**	**#C**
i-50-gil262	262	50
10-lin318	318	10
10-pcb442	442	10
C1k.0	1000	10
C1k.1	1000	10
C1k.2	1000	10
300-6	300	6
400-6	400	6
700-20	700	20
200-4-h	200	4
200-4-x1	200	4
600-8-z	600	8
600-8-x2	600	8
300-5-108	300	5
300-20-111	300	20
500-15-306	500	15
500-25-308	500	25
25-eil101	101	25
42-a280	280	42
144-rat783	783	144

Tabela 5.3: Conjunto de Instâncias médias CTSP

Nome da instância	**#vertices**	**#C**
49-pcb1173	1173	49
100-pcb1173	1173	100
144-pcb1173	1173	144
10-nrw1379	1379	10
12-nrw1379	1379	12
1500-10-503	1500	10
1500-20-504	1500	20
1500-50-505	1500	50
1500-100-506	1500	100
1500-150-507	1500	150
2000-10-a	2000	10
2000-10-h	2000	10
2000-10-z	2000	10
2000-10-x1	2000	10
2000-10-x2	2000	10

Tabela 5.4: Conjunto de Instâncias grandes CTSP

de valores possíveis para cada parâmetro foi definido para 5000 a 1500 *iteracoes* e 1000 a 5000 para $itSemMelhora$. A saída do Irace retornou 3 configurações equivalentes: $C_1 = (11281, 4275)$, $C_2 = (7372, 4827)$ e $C_3 = (7432, 4859)$ iterações e iterações sem melhoras respectivamente.

Para a execução do Modelo com *Lazy Constraints*, foi definido o parâmetro do CPLEX $TimeLimit = 7200$ segundos, $Empashis = 3$ definido para encontrar os melhores limites durante a execução do resolvedor, além disso foi dado como solução inicial uma solução encontrada pelo algoritmo GRASP-RVND.

Os parâmetros para o ILS-RVND foram definidos de forma empírica como segue: iterações sem melhora $itSemMelhora = 500$, taxa de perturbação $taxa = 0.2$, taxa de perturbação máxima $taxaMax = 0.4$ e crescimento da taxa de perturbação $taxaCrescimento = 0.05$.

5.4 Resultados obtidos

Esta seção apresenta os resultados obtidos para todos os métodos abordados anteriormente, com o objetivo de comparar com os resultados presentes na literatura.

5.4.1 CTSP

O experimento presente na Tabela 5.5 apresenta os melhores resultados obtidos por (MESTRIA, 2018) e pelo ILS-RVND apresentado neste trabalho para as instâncias pequenas presentes na Tabela 5.2. A primeira coluna apresenta o nome das instâncias. A segunda coluna são os valores ótimos obtidos pelo uso do resolvedor CPLEX (MESTRIA, 2018). A coluna VNRDGILS apresenta os melhores resultados obtidos pela heurística apresentada em (MESTRIA, 2018). Por fim, os resultados obtidos pelo ILS-RNVD na última coluna. Apesar do VNRDGILS possuir um gap médio menor de 0.05 quando comparado com o ILS com gap de 0.12, não fica claro a existência de uma dominância de um algoritmo em relação ao outro, visto que existem 17 valores empatados entre os algoritmos, e dos 10 desempates (indicados em negrito na tabela) 6 são para o VNRDGILS e 4 para o ILS-RVND.

	CPLEX	**VNRDGILS**		**ILS-RVND**		
Instância	custo	tempo (s)	gap	tempo (s)	custo	gap
10-berlin52	7896	3.70	0.52	0.49	7896	**0.00**
10-eil51	440	4.10	0.00	0.42	440	0.00
10-eil76	561	7.40	0.00	0.94	561	0.00
10-kroB100	22440	17.30	0.00	1.21	22440	0.00
10-pr76	109538	7.90	0.00	0.70	109538	0.00
10-rat99	1238	14.20	0.00	1.28	1238	0.00
10-st70	691	6.10	0.00	0.63	691	0.00
15-berlin52	8049	3.00	0.00	0.49	8049	0.00
15-eil51	437	4.10	0.00	0.44	437	0.00
15-eil76	565	7.40	0.35	0.80	565	**0.00**
15-pr76	110678	8.00	0.00	0.79	110678	0.00
15-st70	692	5.60	0.00	0.69	692	0.00
25-eil101	663	21.50	**0.00**	1.74	667	0.60
25-kroA100	21917	20.90	0.00	1.57	21917	0.00
25-lin105	14438	19.30	0.03	1.50	14438	**0.00**
25-rat99	1269	20.00	0.00	1.65	1269	0.00
5-berlin52	7991	3.30	0.00	0.50	7991	0.00
5-eil51	437	4.00	0.00	0.49	437	0.00
5-eil76	559	8.20	0.18	1.06	559	**0.00**
5-pr76	108590	9.00	0.00	0.83	108590	0.00
5-st70	695	7.30	0.00	0.80	695	0.00
50-eil101	644	23.20	**0.16**	2.38	647	0.46
50-kroA100	21453	23.30	**0.00**	2.15	21555	0.47
50-kroB100	22355	24.90	**0.00**	2.38	22476	0.54
50-lin105	14379	28.70	0.00	2.42	14379	0.00
50-rat99	1249	23.00	**0.00**	2.02	1254	0.40
75-lin105	14521	30.20	**0.00**	2.88	14626	0.72
Média	-	13.2	0.05	1.23	-	0.12

Tabela 5.5: Comparação dos resultados obtidos ILS-RNVD e (MESTRIA, 2018) para instâncias pequenas

O segundo experimento é dedicado a medir o desempenho do ILS-RNVD em relação as instâncias de médias e grandes, das Tabelas 5.3 e 5.4, respectivamente. Os resultados obtidos são apresentados na Tabelas 5.6 e 5.7. Os resultados do ILS-RVND são comparados aos presentes em (LU; HAO; WU, 2020). Entretanto, optou-se não incluir o algoritmo GPR1R2 pela falta de valores para algumas instâncias e por ter o resultado dominado pelos outros algoritmos presentes. Ambas as tabelas possuem a mesma estrutura, diferindo apenas das instâncias presentes. A primeira coluna apresenta o nome das instâncias referentes aos valores obtidos, as colunas seguintes são os melhores valores f_{best}, valores médios f_{avg}. Os algoritmos VNRDGILS e HHGILS possuem valor de tempo fixo de 720 segundos para as instâncias médias e 1080 segundos para as instâncias grandes e

por conta disso a coluna de t(s) não foi incluída para estes algoritmos. A Tabela 5.8 compila os resultados dos algoritmos. A linha de soluções ótimas indica o número de soluções ótimas encontradas. A linha de gap_{best}/gap_{avg} apresenta o gap dos melhores valores e valores médios obtidos em relação aos valores ótimos encontrados no trabalho (LU; HAO; WU, 2020). Por fim, a linha de tempo (s) representa o tempo médio utilizado para a obtenção dos valores por instância apresentados.

Observando-se a Tabela 5.8 é possível perceber a dominância do GA-EAX, que encontra 20/20 valores ótimos para as instâncias médias, enquanto o HHGILS e o ILS-RVND encontram apenas 1/20 e VNRDGILS encontra 0/20. Além disso, o GA-EAX gasta um tempo médio de 5.7 segundos, ou seja, ele executa mais rápido que os outros algoritmos. Este padrão é repetido para as instâncias grandes onde o GA-EAX encontra 14/15 soluções ótimas, enquanto os outros algoritmos não encontram nenhuma, utilizando aproximadamente 30 vezes menos tempo.

Apesar da dominância do GA-EAX, na comparação do ILS-RNVD com os demais algoritmos, VNRDGILS e HHGILS, os resultados são competitivos. Analisando gap_{best}/gap_{avg}, para instâncias médias os resultados são semelhantes, sendo que o ILS-RNVD utiliza uma quantia significativamente inferior de tempo. Para as instâncias grandes, o tempo médio é ligeiramente superior, entretanto é compensando com a qualidade da solução encontrada.

5.4.2 TCTSP

O terceiro experimento tem como objetivo apresentar os resultados obtidos pelos métodos dedicados ao TCTSP na Tabela 5.9 para as instâncias do conjunto 2 presentes na Tabela 5.1. A coluna ID refere-se ao identificador da instância na Tabela 5.1. A coluna CPLEX apresenta o melhor valor encontrado para cada instância, o tempo em segundos e o Gap indicado pelo solver ao final da execução. O gap está presente pois não foi possível encontrar a solução ótima para as 4 maiores instâncias (I_{13}, I_{16}, I_{17} e I_{19}). As outras colunas apresentam f_{best} o melhor valor encontrado entre todas as execuções e o valor médio dado por f_{avg}. A coluna de tempo t(s) para (ZHANG et al., 2018) não foi incluída pois possui tempo de execução fixo de 657,29 segundos.

Como abordado originalmente em (SILVA JUNIOR et al., 2019), as instâncias do TCTSP possuem uma característica que dificulta a comparação direta entre os algoritmos, que é a penalização das arestas entre *clusters* nomeado PIC. O valor PIC é calculado como $PIC = MaiorAresta \times 10 \times k$, onde k representa o número de *clusters* visitados na solução. Essa característica torna grande parte do valor das soluções constante, ou seja, soluções distintas possuem gap similar pois grande parte do valor é composto pelo PIC. A Tabela 5.10 apresenta os resultados presentes na Tabela 5.9 com a subtração do valor PIC.

A Tabela 5.11 compila os resultados presentes nas Tabelas 5.9 e 5.10. A Tabela segue no mesmo formato da Tabela 5.8 e apresenta o número de soluções ótimas encontradas por cada método, o tempo médio em segundos utilizado para a obtenção dos resultados por execução e o gap_{best}/gap_{avg} que apresenta o gap dos melhores valores e valores médios obtidos em relação aos valores encontrados pelo resolvedor CPLEX.

A Tabela 5.11 indica o domínio das duas versões do GRASP em relação a abordagem de (ZHANG et al., 2018), encontrando 6 soluções ótimas para um total de 10 instâncias, o mesmo número do CPLEX, enquanto a literatura encontra apenas 3/10. Além disso, obtém valores de gap_{best}/gap_{avg} com qualidade superior utilizando uma quantia significantemente inferior de tempo. A diferença de qualidade fica ainda mais acentuada quando o valor PIC é desconsiderado nos cálculos do gap.

	GA-EAX			**ILS-RVND**			**VNRDGILS**		**HHGILS**	
Instância	f_{best}	f_{avg}	**t(s)**	f_{best}	f_{avg}	**t(s)**	f_{best}	f_{avg}	f_{best}	f_{avg}
10-lin318	**529584**	529584.00	1.80	529975	530380.33	13.05	530604	530871.40	530283	530817.90
10-pcb442	**537419**	537419.00	6.30	537868	538832.43	34.68	538309	538903.40	538958	539988.30
144-rat783	**914228**	914228.00	9.40	914590	914764.40	256.57	915088	915179.80	915180	915363.20
200-4-h	**62777**	62777.00	0.90	63088	63190.97	13.25	62867	63058.30	62804	63058.30
200-4-x1	**60574**	60574.00	0.90	60799	60954.10	14.01	60637	60796.20	60931	61378.50
25-eil101	**23671**	23671.00	0.80	**23671**	23674.97	1.93	23673	23685.20	23678	23690.00
300-20-111	**309739**	309739.00	2.00	309828	310032.93	13.51	310146	310270.90	310209	310427.00
300-5-108	**67760**	67760.00	2.00	67920	67990.60	25.09	67766	67868.70	67814	67930.50
300-6	**8934**	8934.00	3.50	8941	8945.83	19.14	8935	8941.10	**8934**	8942.90
400-6	**9045**	9045.00	4.40	9063	9068.77	39.63	9053	9062.30	9051	9063.20
42-a280	**129645**	129645.00	1.70	129669	129738.40	15.68	129729	129755.20	129716	129833.20
500-15-306	**194818**	194818.00	5.20	195059	195159.27	39.94	194946	195201.50	195202	195438.10
500-25-308	**365447**	365447.00	5.40	365652	365789.27	46.74	365717	365937.80	365828	366085.00
600-8-x2	**128891**	128891.00	5.30	129672	129912.30	88.78	129246	129533.90	129246	129691.50
600-8-z	**128891**	128891.00	5.30	129688	129903.67	84.84	129468	129862.70	129416	129928.60
700-20	**41425**	41425.00	10.20	41448	41456.47	93.21	41456	41489.70	41452	41485.60
C1k.0	**132521027**	132521027.00	16.30	133002305	133081182.40	321.60	133260549	133490775.90	133287594	133776274.10
C1k.1	**129128125**	129128125.00	14.30	129586861	129651854.97	363.69	129877874	130035540.20	129825403	130206778.30
C1k.2	**142784000**	142784188.40	17.20	143112436	143165161.83	353.84	143321630	143481489.60	143278093	143525149.60
i-50-gil262	**135431**	135431.00	1.70	135437	135464.80	13.52	135483	135510.20	135510	135578.00

Tabela 5.6: Comparação dos resultados para o CTSP obtidos com ILS-RNVD e (LU; HAO; WU, 2020) para instâncias médias

	GA-EAX			**ILS-RVND**			**VNRDGILS**		**HHGILS**	
Instância	f_{best}	f_{avg}	**t(s)**	f_{best}	f_{avg}	**t(s)**	f_{best}	f_{avg}	f_{best}	f_{avg}
10-nrw1379	**58783**	58787.10	26.80	61242	61620.83	1074.83	63951	64895.90	63620	64643.90
100-pcb1173	**63382**	63382.80	32.50	65524	66157.57	484.49	68708	69383.20	68786	70640.80
12-nrw1379	**59129**	59129.40	27.60	61694	62032.23	895.95	62893	63532.30	63558	64741.60
144-pcb1173	**62142**	62142.00	18.60	65276	65946.00	587.44	68414	68941.40	66830	69084.30
1500-10-503	**11116**	11116.00	28.40	11563	11713.17	1173.52	11969	12103.00	11986	12109.50
1500-100-506	**29799**	29799.60	39.50	30849	31073.60	914.54	32474	32616.80	32260	33760.60
1500-150-507	**34068**	34068.00	32.30	35869	36156.13	1147.75	37357	38251.10	37658	38433.10
1500-20-504	**15698**	15700.70	34.50	16270	16394.13	695.42	16678	16867.40	17107	17315.70
1500-50-505	**22900**	22901.00	35.10	23578	23720.90	686.12	24631	24803.60	25264	25558.90
2000-10-a	105447	105483.00	45.30	111120	112021.20	1831.87	115779	116897.30	116254	116881.40
2000-10-h	**33708**	33708.00	35.60	35677	35868.00	1990.14	36806	38351.80	36447	37305.10
2000-10-x1	**33792**	33796.60	35.60	35654	35800.50	1971.05	36783	37488.60	36752	37704.00
2000-10-x2	**33509**	33509.00	39.60	35432	35591.10	1975.38	37132	38240.60	36660	37117.10
2000-10-z	**33509**	33509.10	37.30	35388	35572.73	2151.92	36815	38035.70	37059	37443.70
49-pcb1173	**61600**	61620.10	35.00	63272	63833.33	311.19	65750	66487.70	67043	68260.70

Tabela 5.7: Comparação dos resultados obtidos para o CTSP com ILS-RNVD e (LU; HAO; WU, 2020) para instâncias grandes

		GA-EAX	ILS-RVND	VNRDGILS	HHGILS
Instâncias médias	soluções ótimas	20/20	1/20	0/20	1/20
	$gap_{best}/gap_{avg}\%$	0.00/0.00	0.20/0.28	0.18/0.30	0.21/0.40
	tempo (s)	5.7	92.63	720	720
Instâncias grandes	soluções ótimas	14/15	0/15	0/15	0/15
	$gap_{best}/gap_{avg}\%$	0.00/0.01	4.28/4.98	8.61/10.39	8.92/11.04
	tempo (s)	33.6	1192.77	1080	1080

Tabela 5.8: Comparação dos resultados estatísticos para o CTSP obtidos do ILS-RNVD e (LU; HAO; WU, 2020)

	CPLEX			Zhang, 2018		R-GRASP			GRASP-RVND		
ID	f_{best}	**t (s)**	Gap	f_{best}	f_{avg}	f_{best}	f_{avg}	**t (s)**	f_{best}	f_{avg}	**t (s)**
I13	507795	7551.75	38.95%	508523	508671.10	507814	507855.40	49.73	507795	507795.33	3.85
I14	**9346**	3.41	0.00%	**9346**	9346.00	**9346**	9346.00	4.57	**9346**	9346.00	0.26
I15	**427028**	1409.62	0.00%	427113	427105.90	**427028**	427028.00	7.07	**427028**	427028.00	0.55
I16	506743	7522.85	97.13%	509411	509516.00	507044	507229.90	49.44	506553	506754.47	8.07
I17	1342647	7498.97	62.52%	1346312	1347246.80	1342667	1342777.00	55.77	1342647	1342648.63	6.88
I18	**10409**	39.23	0.00%	10411	10411.00	**10409**	10409.00	4.33	**10409**	10409.00	0.44
I19	54265	7592.42	41.75%	54290	54300.00	54265	54266.60	31.04	54260	54264.60	2.61
I20	**508238**	9.98	0.00%	508245	508260.90	**508238**	508238.00	3.68	**508238**	508238.00	0.42
I21	**518487**	23.96	0.00%	**518487**	518492.80	**518487**	518487.00	4.23	**518487**	518487.00	0.49
I22	**26674**	60.92	0.00%	**26674**	26674.10	**26674**	26674.00	3.44	**26674**	26674.00	0.43

Tabela 5.9: Comparação dos resultados para o TCTSP obtidos com GRASP-RNVD e (SILVA JUNIOR et al., 2019) para instâncias do conjunto 2

		CPLEX		Zhang, 2018		R-GRASP		GRASP-RVND	
ID	**PIC**	f_{best}	**t (s)**	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
I13	486600	21195	7551.75	219230	22071.10	21214.00	21255.40	21195	21195.33
I14	9030	316	3.40862	316	316.00	316.00	316.00	316	316.00
I15	416700	10328	1409.62	10413	10405.90	10328.00	10328.00	10328	10328.00
I16	484100	22643	7522.85	25311	25416.00	22944.00	23129.90	22453	22654.47
I17	1282000	60647	7498.97	64312	65246.80	60667.00	60777.00	60647	60648.63
I18	10120	289	39.23	291	291.00	289.00	289.00	289	289.00
I19	53200	1065	7592.42	1090	1100.00	1065.00	1066.60	1060	1064.60
I20	498000	10238	9.98	10245	10260.90	10238.00	10238.00	10238	10238.00
I21	510240	8247	23.96	8247	8252.80	8247.00	8247.00	8247	8247.00
I22	26160	514	60.9168	514	514.10	514.00	514.00	514	514.00

Tabela 5.10: Comparação dos resultados para o TCTSP obtidos com GRASP-RNVD e (SILVA JUNIOR et al., 2019) para instâncias do conjunto 2 com a exclusão do valor PIC

		CPLEX	Zhang, 2018	R-GRASP	GRASP-RVND
	soluções ótimas	6/10	3/10	6/10	6/10
	tempo (s)	3171.31	657.29	21.33	2.4
PIC considerado	gap_{best}/gap_{avg}	-	0.103/0.117	0.006/0.012	-0.005/0.000
PIC desconsiderado	gap_{best}/gap_{avg}	-	2.343/2.686	0.143/0.275	-0.132/0.002

Tabela 5.11: Comparação dos resultados estatísticos para o TCTSP obtidos pelos métodos do TCTSP

6 Conclusão

Este trabalho abordou duas variações do *Traveling Salesman Problem*, o *Clustered Traveling Salesman Problem* e o *Tabu Clustered Traveling Salesman Problem*. O trabalho apresenta 4 métodos distintos para buscar soluções para estes problemas: i) Uma heurística para o CTSP; ii) Duas heurísticas e uma formulação de Programação Inteira para o TCTSP. Estes métodos foram construídos com a junção de componentes independentes, como algoritmos construtivos e estratégias de refinamento.

Foi utilizado para a avaliação dos métodos propostos um total 27 instâncias, classificadas como pequenas, 20 instâncias médias e 15 instâncias grandes para o CTSP. Para o TCTSP, foram utilizadas as 10 instâncias do conjunto 2 que possuem tabus com número de nós maior que 1.

Os resultados obtidos para as instâncias pequenas do CTSP indicam que o ILS-RNVD e o VNRDGILS alcançam resultados com valores similares, com empate em várias das instâncias e melhora para ambos os algoritmos em diferentes situações. Não houve comparação com o GA-EAX pois os resultados não foram disponibilizados. Para as instâncias médias, os algoritmos ILS-RVND, HHGILS e VNRDGILS encontram valores gap_{best} e gap_{avg} similares, sem nenhuma diferença significativa entre eles, os três variando no intervalo de 0.18 a 0.40. Entretanto, o ILS-RVND se destaca por alcançar os resultados cerca de 7.8 vezes mais rápido em relação a HHGILS e VNRDGILS. Nas instâncias grandes o ILS-RVND encontra valores melhores que o VNRDGILS e HHGILS em tempo similar, cerca de 1.1 do tempo de VNRDGILS e HHGILS. É importante ressaltar que o algoritmo GA-EAX domina todos os outros, encontrando as soluções ótimas para 34 das 35 instâncias e utilizando um tempo médio dezenas a centenas de vezes mais rápido, enquanto o HHGILS encontra a solução ótima em uma única instância média e todos os outros métodos não encontram.

Para o TCTSP, a formulação de Programação Inteira foi desenvolvida para encontrar os valores ótimos das instâncias para que seja possível avaliar as heurísticas presentes e propostas na literatura. Este objetivo foi parcialmente atingido, no tempo limite de

2 horas foram resolvidas 6/10 instâncias, enquanto as 4 maiores instâncias ainda estão em aberto. Os métodos R-GRASP e GRASP-RVND dominam em tempo e em custo em relação a literatura, esta diferença fica mais expressiva com a avaliação que desconsidera o PIC. A diferença de custo entre o R-GRASP e o GRASP-RVND não é tão relevante quanto em relação a literatura, entretanto o GRASP-RNVD se destaca por encontrar valores melhores em, aproximadamente, 10 vezes mais rápido.

Um dos problemas enfrentado durante a execução do resolvedor para encontrar as soluções ótimas para o TCTSP foi a forma de inserção das *lazy constraints* no resolvedor, que não permite paralelização na execução do modelo. Como trabalhos futuros, sugere-se o estudo de alternativas que permita encontrar os valores ótimos para as instâncias restantes, como outro método na interface do resolvedor para a utilização das *lazy constraints* ou o teste em outros resolvedores presentes no mercado, como o Gurobi. É importante considerar a utilização o Irace para calibrar os parâmetros para o ILS-RVND como foi feito para o GRASP-RVND. Também é possível dar continuidade no estudo de similaridade entre as variações do TSP, como por exemplo a similaridade entre o TCTSP apresentado neste trabalho e o recém introduzido na literatura *Clustered Generalized TSP* (CGTSP).

Bibliografia

APPLEGATE, D.; BIXBY, R.; CHVÁTAL, V. *Concorde TSP Solver. William Cook, School of Industrial and Systems Engineering, Georgia Tech*. 2007.

BALI, K. K. et al. Multifactorial evolutionary algorithm with online transfer parameter estimation: Mfea-ii. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 24, n. 1, p. 69–83, 2019.

BANIASADI, P. et al. A transformation technique for the clustered generalized traveling salesman problem with applications to logistics. *European Journal of Operational Research*, Elsevier, 2020.

CHISMAN, J. A. The clustered traveling salesman problem. *Computers & Operations Research*, Elsevier, v. 2, n. 2, p. 115–119, 1975.

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. *Journal of global optimization*, Springer, v. 6, n. 2, p. 109–133, 1995.

FESTA, P.; RESENDE, M. G. An annotated bibliography of grasp–part ii: Applications. *International Transactions in Operational Research*, Wiley Online Library, v. 16, n. 2, p. 131–172, 2009.

HADDAD, M. N. et al. Aiv: A heuristic algorithm based on iterated local search and variable neighborhood descent for solving the unrelated parallel machine scheduling problem with setup times. In: *ICEIS (1)*. [S.l.: s.n.], 2014. p. 376–383.

HUERTA, I. I. et al. Improving the state-of-the-art in the traveling salesman problem: An anytime automatic algorithm selection. *Expert Systems with Applications*, Elsevier, v. 187, p. 115948, 2022.

KARA, I.; GUDEN, H.; KOC, O. N. New formulations for the generalized traveling salesman problem. In: *Proceedings of the 6th international conference on applied mathematics, simulation, modelling, ASM*. [S.l.: s.n.], 2012. v. 12, p. 60–65.

KINDERVATER, G. A.; LENSTRA, J. K.; SHMOYS, D. B. The parallel complexity of tsp heuristics. *Journal of Algorithms*, Elsevier, v. 10, n. 2, p. 249–270, 1989.

LANGEVIN, A.; SOUMIS, F.; DESROSIERS, J. Classification of travelling salesman problem formulations. *Operations Research Letters*, Elsevier, v. 9, n. 2, p. 127–132, 1990.

LAPORTE, G.; MERCURE, H.; NOBERT, Y. Generalized travelling salesman problem through n sets of nodes: the asymmetrical case. *Discrete Applied Mathematics*, Elsevier, v. 18, n. 2, p. 185–197, 1987.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2003. p. 320–353.

LU, Y.; HAO, J.-K.; WU, Q. Solving the clustered traveling salesman problem via tsp methods. *arXiv preprint arXiv:2007.05254*, 2020.

LóPEZ-IBáñEZ, M. et al. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, v. 3, p. 43–58, 2016.

MESTRIA, M. A hybrid heuristic algorithm for the clustered traveling salesman problem. *Pesquisa Operacional*, SciELO Brasil, v. 36, n. 1, p. 113–132, 2016.

MESTRIA, M. New hybrid heuristic algorithm for the clustered traveling salesman problem. *Computers & Industrial Engineering*, Elsevier, v. 116, p. 1–12, 2018.

MESTRIA, M.; OCHI, L. S.; MARTINS, S. de L. Grasp with path relinking for the symmetric euclidean clustered traveling salesman problem. *Computers & Operations Research*, Elsevier, v. 40, n. 12, p. 3218–3229, 2013.

POP, P. C. New integer programming formulations of the generalized traveling salesman problem. *American Journal of Applied Sciences*, Citeseer, v. 4, n. 11, p. 932–937, 2007.

PUNNEN, A. P. The traveling salesman problem: Applications, formulations and variations. In: *The traveling salesman problem and its variations*. [S.l.]: Springer, 2007. p. 1–28.

RESENDE, M. G.; RIBEIRO, C. C. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2010. p. 283–319.

RESENDE, M. G.; RIBEIRO, C. C. Grasp: Greedy randomized adaptive search procedures. In: *Search methodologies*. [S.l.]: Springer, 2014. p. 287–312.

SAMA, M. et al. Ant colony optimization for the real-time train routing selection problem. *Transportation Research Part B: Methodological*, Elsevier, v. 85, p. 89–108, 2016.

SILVA JUNIOR, E. L. da et al. An efficient algorithm for the tabu clustered traveling salesman problem. In: IEEE. *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*. [S.l.], 2019. p. 287–292.

SÖRENSEN, K.; GLOVER, F. Metaheuristics. *Encyclopedia of operations research and management science*, Springer New York, v. 62, p. 960–970, 2013.

SUBRAMANIAN, A. et al. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, Elsevier, v. 37, n. 11, p. 1899–1911, 2010.

TALBI, E.-G. *Metaheuristics: from design to implementation*. [S.l.]: John Wiley & Sons, 2009. v. 74.

THEYS, C. et al. Using a tsp heuristic for routing order pickers in warehouses. *European Journal of Operational Research*, Elsevier, v. 200, n. 3, p. 755–763, 2010.

YANG, J. et al. An ant colony optimization method for generalized tsp problem. *Progress in Natural Science*, Elsevier, v. 18, n. 11, p. 1417–1422, 2008.

ZACHARIAH, R. et al. Operational research in low-income countries: what, why, and how? *The Lancet infectious diseases*, Elsevier, v. 9, n. 11, p. 711–717, 2009.

ZHANG, T. et al. Metaheuristics for the tabu clustered traveling salesman problem. *Computers & Operations Research*, Elsevier, v. 89, p. 1–12, 2018.